Global Information Assurance Certification Paper

GCIA Practical Version 3.4

David C. Love

February 24, 2004

TABLE OF CONTENTS

1. State of Intrusion Detection

Operating System Detection (OSD) is nothing new: Queso[1] has been able to use 7 packets to identify 100 systems since at least 1998. NMap[2] followed soon after and has been updated ever since. It uses 8 packets, and can now identify nearly 1,000 operating systems and devices.

This paper provides an in-depth view into how NMap's OSD works. It looks into the packets NMap sends, and how the response packetsare used to build an OS "fingerprint" to identify the system. The fingerprint format is detailed, showing exactly what kind of information NMap can use in its effort to detemine the OS in question.

NMap is then used to scan an OS currently unknown to it, and the resulting fingerprint compared with the actual packet trace, showing line-by-line how NMap works. Finally, the paper shows how to add the fingerprint to NMap's database, allowing the target system to be automatically recognized in the future.

1.1 Brief Review of Queso

The first OS fingerprinting to I'm aware of was Queso. Its ability to differentiate between more than 100 operating system was impressive, but only half the story. The amazing part of Queso is that it was able to do its job by sending only 7 packets to a given host. Then, by analysing the responses, it could pinpoint the target OS with amazing (for the time) accuracy.

It worked by playing games with the flags in the TCP header. It would send a group of six packets, identical except for their sequence numbers (random) and flags, which were set as follows: SYN, SYN|ACK, FIN, FIN|ACK, SYN|FIN, PUSH and SYN|XXX|YYY, where XXX and YYY correspond to the unused flag bits (now used for ECN).

The first packet is a standard part of a TCP handshake. The remainder, however, are invalid. Queso would send the packets, wait for the responses, then recorded four pieces of information for each: the sequence number, ack number, window size and TCP flags. It would then compare the results against a text file containing the known responses for various operating systems and print any matches it found. If no match was found, it would print out the configuration information it got back, allowing the operater to add it to the configuration file if the target OS was known.

That's all there was to it. What's really important however is not so much how it worked, but why it worked. Queso was designed to take advantage of the fact that while the correct behavior of an IP stack is strictly codified, the handling of abnormal traffic is not. For example, when an unexpected TCP packet is received by a host, it's required to send back a packet with the RESET flag set. No mention is made, however, about what should be done with the other fields in the packet; the RESET flag itself is sufficient to signal the sender that there's a problem, and as far as the RFCs are concerned, the other fields are insignificant.

Therein lies the secret. Different hosts respond differently to unexpected data. Some hosts will provide an ack number for a RESET, while others won't. Some hosts set the window size, while others copy it from the original packet. Some ignore the XXX and YYY flag bits altogether, treating Packet 7 as a standard SYN packet.

By tracking and codifying these responses, Queso was able identify 100 different operating systems.

1.2 NMap's OS Detection

Where Queso left off, NMap took over, and It's been adding new detection methods ever since. Here are just a few of the tricks used by NMap.

TCP Options

There's a wealth of information available in the way systems handle TCP options. Every TCP packet sent by NMap for OSD contains the same set of options set in the same order. NMap then checks the replies to see what options were returned and what order they were returned in. The variations here are amazing.

Additionally, NMap analizes the timestamp option[3] and the returned values to try and determine, in broad categories, how frequently the remote system updates its timestamps.

Closed vs. Open Ports

Sending an invalid packet to an open port often yields different results than when sending that same packet to a closed port.

Sequence Numbers

In addition to comparing sequence numbers with their resulting acks, NMap analyses the returned sequence numbers in an effort to determine how those number are

generated. This is useful not only in categorizing systems, but can be a significant aid to an attacker: sequence numbers that are trivially generated (incremented by 1, for example) can make it much easier for an attacker to guess the next number in series.

ICMP Errors

Some systems will set the IP TOS (type-of-service) field for these packets, even though it should be zero, according to the standard.[4] Other systems munge the data as they copy the original UDP packet into the ICMP response. Even the amount of data copied is revealing. As Fyodor says: "For a port unreachable message, almost all implementations send only the required IP header + 8 bytes back. However, Solaris sends back a bit more and Linux sends back even more than that. The beauty with this is it allows nmap to recognize Linux and Solaris hosts even if they don't have any ports listening." [5]

NMap tracks all of this information and more.

1.3 The Packets

NMap send the following 8 packets during its OSD phase:

#	Type	Port State	TCP Flags
1	TCP	Open	SYN
2	TCP	Open	NULL
3	TCP	Open	SYN, FIN, URG, PSH
4	TCP	Open	ACK
5	TCP	Closed	SYN
6	TCP	Closed	ACK
7	TCP	Closed	FIN, PSH, URG
8	UDP	Closed	N/A

All 7 TCP packets have the same TPC Options set in the same order: mss 1460, nop, wscale 0, nop, nop, timestamp 1061109567 0.

1.4 Scanning a Target

Here NMap is used to scan a system known to be running OpenBSD 3.4. At this point in time, NMap doesn't recognize that system, so it instead prints out the fingerprint it collected.

```
$ nmap -O 10.0.0.3

Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on beigebox (10.0.0.3):
(The 1653 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
113/tcp  open  auth
No exact OS matches for host (If you know what OS is running on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.48%P=powerpc-apple-darwin7.2.0%D=1/20%Time=403612B8%O=22%C=1)
TSeq(Class=TR%IPID=RD%TS=2HZ)
T1(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=Y%W=4000%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

Now what the heck does that mean?

1.5 Deciphering a Fingerprint

NMap's fingerprint format is only vaguely documented, but since the source is freely available, it's possible to decipher the language.

The 10 lines of output generated by NMap are actually straight-forward, once the codes are understood. Each line is in the format "type(key=value...)", and multiple key/value pairs can be listed by separating them with a percent sign ('%').

Values are either numbers or strings. Numbers are stored in hex unless otherwise noted. With the exception of SInfo's P parameter, all the strings are predefined.

Multiple values can be specified for a key by separating them with a pipe ('|'): The entry `"gcd=1000|2000|3000"` means that gcd can have a value of 1000, 2000 or 3000. Likewise, `"DF = Y|N"` means DF can by Y or N.

Numeric comparisons can be created using '<' for "less than" and '>' for "greater than". The entry `"gcd=<130"` means that gcd is less than 130. Remember that the equal sign is the key/value separator and not part of the separator. If you want to note that gcd can be less than or equal to 130, you'd have to say `"gcd=<130|130"`.

Finally, you can create numeric ranges by joining comparisons with '&'. The entry `"SI=<7A8F&>50"` means that SI must be less than 7A8F and greater than 50.

The next table shows all the available keys for NMap, which types they can belong to and the acceptable values for each.

Key	Types	Purpose	Values
V	SInfo	Version	
P	SInfo	System information	**String: powerpc-apple-darwin7.2.0**
D	SInfo	Date	DD/MM
T	SInfo	Time field (unix format)	Number (decimal)
O	SInfo	Open port used for test	Number (decimal)
C	SInfo	Closed port used for test	Number (decimal)
Class	TSeq	Classification predicting for initial sequence numbers, based on sampling.	64 - multiple of 64,000 TD - time-dependent (Microsoft) i800 - multiple of 800 TR - truly random C - constant RI - random incremental

IPID	TSeq	Classification for the way IP IDs are generated.	C - constant or duplicates I - simple increment (by 1) RD - random, up or down RPI - random, always increasing Z - every packet has an IPID of 0 BI - simple increment by 1, but Microsoft forgot network ordering so it increases by 256 on little-endian systems
TS	TSeq	Classification for how the TCP options timestamps received were incremented	0 - got back at least one zero 2HZ 100HZ 1000HZ U - none returned
SI	TSeq	Sequence number increment range if Class is TD or RI	Number
Val	TSeq	Sequence number if Class is C	Number
gcd	TSeq	Greatest common denominator for sequence numbers if Class is TD or RI	Number
Resp	T#, PU	Was a response received	Y or N
DF	T#	Was the Don't Fragment bit set?	Y, N, Y|N
W	T#	Window size	Number
ACK	T#		0 S - same as sent sequence S++ - same as sent sequence + 1
Flags	T#	TCP Flags set in response	Any of BUAPRSF
Ops	T#	TCP Options set in the order they appear in the response	N - nop M - mss E - echoed mss W - window scale T - timestamp
TOS	PU	IP Type of Service	Number

IPLEN	PU	Length from IP header	Number
RIPTL	PU	Length from IP header echoed back in icmp message	Number
RID	PU		
RIPCK	PU	UDP checksum echoed back in icmp message correct?	E = Yes F = No (Fouled up)
UCK	PU	UDP checksum correct?	E = Yes F = No
ULEN	PU	UDP Length	Number
DAT	PU	Data echoed back correctly?	E = Yes (or no data echoed) F = No

Given that information, we can decode the fingerprint.

```
SInfo(V=3.48%P=powerpc-apple-darwin7.2.0%D=1/20%Time=403612B8%O=22%C=1)
```

This entry is really only of interest when submitting a new fingerprint for inclusion with NMap's default distribution (see Adding a Fingerprint below). It shows the fingerprint was generated on Jan 20th (UNIX time 403612B8) using NMap V3.48, running on a platform that identified itself as powerpc-apple-darwin7.2.0 (in this case, Mac OS X 10.3.2). Port 22 was used for the open-port tests and port 1 was used for the closed port tests.

```
TSeq(Class=TR%IPID=RD%TS=2HZ)
```

This shows the information NMap was able to derive about the sequence numbers generated by the target. In this case, the Class shows the sequence numbers were truly random (TR), as were the IP ids (RD).

The TS field is based on the average of the ratios of the intervals between the timestamps recorded in the received packet's TCP Options, and the intervals between the times when the originating packets were actually sent. This gives NMap a view into the resolution of the clock used in the target's TCP stack. In this case, the target system falls in the 2Hz range.

1.6 A Fingerprint Trace

Two points need to be made here. First, every TCP response packet from the target resulted in a RESET packet from the local host. These RESET packets are not presented below. Second, a full trace of the NMap OSD is considerably longer than shown below, as NMap will resend queries if it doesn't receive answers back quickly enough. On a local 100Mbs network, NMap sent three complete sets of packets before it was satisified with the answers. This is by no means a silent scan.

The "interesting" portions -- those that relate directly to NMap's fingerprint -- are highlighted in the response. In the case of the TCP Options, only the option identifies themselves are highlighted, since the values received aren't directly stored in the fingerprint.

Packet 1: TCP SYN to Open Port

```
10.0.0.1.62864 > 10.0.0.3.22: SE 1131834306:1131834306(0) win 4096 <wscale
10,nop,mss 265,timestamp 1061109567 0,eol>
          4500 003c e566 0000 3b06 838b 0a00 0001
          0a00 0003 f590 0016 4376 6bc2 0000 0000
          a042 1000 fd4a 0000 0303 0a01 0204 0109
          080a 3f3f 3f3f 0000 0000 0000
10.0.0.3.22 > 10.0.0.1.62864: S 3104930414:3104930414(0) ack 1131834307 win
16445 <mss 1460,nop,wscale 0,nop,nop,timestamp 1722801641 1061109567> (DF)
          4500 003c 0270 4000 4006 2182 0a00 0003
          0a00 0001 0016 f590 b911 7a6e 4376 6bc3
          a012 403d 5878 0000 0204 05b4 0103 0300
          0101 080a 66af dde9 3f3f 3f3f 8dc8 4947
```

T1(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)

DF=Y: The DF bit is the middle bit of the IP header flags, which are stored in the first three bits of byte 6 of the IP header. Byte 6 has the value 40, which gives the flags the binary value '010', so DF = 1.

W=403D: The returned window size (4 bytes starting with byte 34) is 403d.

ACK=S++: The ack number (4 bytes starting at byte 28) is '43766bc3' which is one greater than the original sequence number of '43766bc2' (4 bytes starting at byte 24).

Flags=AS: The TCP flags (byte 33) are '42', indicating ACK and SYN are set.

Ops=MNWNNT: The returned TCP Options appeared in this order: 02 (mss, bytes 40-43) , 01 (nop, byte 44), 03 (window scale, bytes 45-47), 01 (nop, byte 48), 01 (nop, byte 49), 08 (timestamp, bytes 50-59).

Packet 2: TCP NULL to Open Port

```
10.0.0.1.62865 > 10.0.0.3.22: . win 1024 <wscale 10,nop,mss 265,timestamp
1061109567 0,eol>
           4500 003c 3258 0000 2c06 459a 0a00 0001
           0a00 0003 f591 0016 870c 674f 0000 0000
           a000 0400 ca68 0000 0303 0a01 0204 0109
           080a 3f3f 3f3f 0000 0000 0000
```

T2(Resp=N)

There was no response to this packet.

Packet 3: TCP SYN,FIN,URG,PSH to Open Port

```
10.0.0.1.62866 > 10.0.0.3.22: SFP 1131834306:1131834306(0) win 2048 urg 0
<wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
           4500 003c 85b0 0000 2906 f541 0a00 0001
           0a00 0003 f592 0016 4376 6bc2 0000 0000
           a02b 0800 0560 0000 0303 0a01 0204 0109
           080a 3f3f 3f3f 0000 0000 0000
10.0.0.3.22 > 10.0.0.1.62866: S 2996330388:2996330388(0) ack 1131834307 win
16445 <mss 1460,nop,wscale 0,nop,nop,timestamp 1722801641 1061109567> (DF)
           4500 003c 4324 4000 4006 e0cd 0a00 0003
           0a00 0001 0016 f592 b298 5f94 4376 6bc3
           a012 403d 79c9 0000 0204 05b4 0103 0300
           0101 080a 66af dde9 3f3f 3f3f f58b 7d89
```

T3(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)

Identical to packet 1, which is interesting. OpenBSD is completely ignoring the FIN flag and treating the packet as a valid SYN, returning a SYN|ACK.

Packet 4: TCP ACK to Open Port

```
10.0.0.1.62867 > 10.0.0.3.22: . ack 0 win 2048 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
           4500 003c 5f7c 0000 3106 1376 0a00 0001
           0a00 0003 f593 0016 4376 6bc2 0000 0000
           a010 0800 057a 0000 0303 0a01 0204 0109
           080a 3f3f 3f3f 0000 0000 0000
12:03:27.013308 10.0.0.3.22 > 10.0.0.1.62867: R 0:0(0) win 16384 (DF)
           4500 0028 4dbe 4000 4006 d647 0a00 0003
           0a00 0001 0016 f593 0000 0000 0000 0000
           5004 4000 636c 0000 0000 0000 0000 0f5c
           8142
```

T4(Resp=Y%DF=Y%W=4000%ACK=O%Flags=R%Ops=)

OpenBSD responds with a RESET, as expected. The ack number is cleared and no options are sent. Notice the returned window of 4000 and compare it to the results for Packet 6.

Packet 5: TCP SYN to Closed Port

```
10.0.0.1.62868 > 10.0.0.3.1: S 1131834306:1131834306(0) win 3072 <wscale
10,nop,mss 265,timestamp 1061109567 0,eol>
           4500 003c 0823 0000 3606 65cf 0a00 0001
           0a00 0003 f594 0001 4376 6bc2 0000 0000
           a002 0c00 019c 0000 0303 0a01 0204 0109
           080a 3f3f 3f3f 0000 0000 0000
12:03:27.013315 10.0.0.3.1 > 10.0.0.1.62868: R 0:0(0) ack 1131834307 win 0
(DF)
           4500 0028 48ef 4000 4006 db16 0a00 0003
           0a00 0001 0001 f594 0000 0000 4376 6bc3
           5014 0000 f436 0000 0000 0000 0000 066f
           6374
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
```

It's almost as though OpenBSD wanted to treat this as an open port, then rejected it at the last moment. Notice that the ACK is set, and the ack number is one greater than the original sequence number, just as if this were a normal SYN to an open port. As will be seen on all RESETs from this machine, the window size is 0 and no TCP options are included.

Packet 6: TCP ACK to Closed Port

```
10.0.0.1.62869 > 10.0.0.3.1: . ack 0 win 3072 <wscale 10,nop,mss
265,timestamp 1061109567 0,eol>
           4500 003c 0db5 0000 2a06 6c3d 0a00 0001
           0a00 0003 f595 0001 4376 6bc2 0000 0000
           a010 0c00 018d 0000 0303 0a01 0204 0109
           080a 3f3f 3f3f 0000 0000 0000
10.0.0.3.1 > 10.0.0.1.62869: R 0:0(0) win 0 (DF)
           4500 0028 4eb9 4000 4006 d54c 0a00 0003
           0a00 0001 0001 f595 0000 0000 0000 0000
           5004 0000 a37f 0000 0000 0000 0000 63a4
           250c
```

T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)

This is almost identical to Packet 4, which sent an ACK to an open port, except that the

returned windows size is set to 0.

Packet 7: TCP FIN,PSH,URG to Closed Port

```
10.0.0.1.62870 > 10.0.0.3.1: FP 1131834306:1131834306(0) win 3072 urg 0
<wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
          4500 003c 9195 0000 3a06 d85c 0a00 0001
          0a00 0003 f596 0001 4376 6bc2 0000 0000
          a029 0c00 0173 0000 0303 0a01 0204 0109
          080a 3f3f 3f3f 0000 0000 0000
12:03:27.013379 10.0.0.3.1 > 10.0.0.1.62870: R 0:0(0) ack 1131834306 win 0
(DF)
          4500 0028 140b 4000 4006 0ffb 0a00 0003
          0a00 0001 0001 f596 0000 0000 4376 6bc2
          5014 0000 f435 0000 0000 0000 0000 1e4e
          2666
T7(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
```

Similar to packet 5, except this time the ack number is set to the originating sequence number.

Packet 8: UDP to Closed Port

```
10.0.0.1.62857 > 10.0.0.3.1: udp 300
          4500 0148 18bf 0000 3811 521c 0a00 0001
          0a00 0003 f589 0001 0134 b4f4 6363 6363
          6363 .....
10.0.0.3 > 10.0.0.1: icmp 36: 10.0.0.3 udp port 1 unreachable
          4500 0038 1b17 0000 ff01 89e3 0a00 0003
          0a00 0001 0303 5149 0000 0000 4500 0148
          18bf 0000 3811 521c 0a00 0001 0a00 0003
          f589 0001 0134 b4f4 e0e6 a5f4
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

This packet tests the targets ICMP handling, checking the contents of the ICMP Port Unreachable message returned for the UDP packet. In the descriptions below, the word "echoed" refers to the data in the payload of the ICMP packet. It's a reference to how well that data echoes the original packet.

DF=N: For the first time, DF is false.

TOS=0: The IP Type-Of Service field (byte 2) is 0.

IPLEN=38: The IP Header Length field is (bytes 2-3) is 0038

RIPTL=148: The echoed IP Header Length field (bytes 30-31) is 0148.

RID=E: The echoed IP ID of 18bf (bytes 32-33) matches the original.
RIPCK=E: The echoed IP checksum (521c) matches the original.
UCK=E: The echoed UDP checksum of b4f4 (bytes 34-35) matches original.
ULEN=134: The echoed UDP length (0134) matches the original.
DAT=E: All the echoed data is consistent with the original.

1.7 Adding a fingerprint

Getting nmap to recognize a new fingerprint is easy. In a temporary working directory, copy the fingerprint into a file named nmap-os-fingerprints. Edit the file to remote the SInfo line, replacing it with the appropriate Fingerprint and Class lines. For OpenBSD 3.4, the entry looks like this:

```
Fingerprint OpenBSD 3.4
Class OpenBSD | OpenBSD | 3.X | general purpose
TSeq(Class=TR%IPID=RD%TS=2HZ)
T1(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=Y%W=4000%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

The Fingerprint line uniquely identifies this fingerprint from all the others NMap knows about. You'll need to scan NMap's nmap-os-fingerprints file to make certain your name doesn't conflict with an existing one. Also note that there can be multiple entries for the same system: different patches and installed applications (especially firewalls) can affect the fingerprint generated. This line still needs to be unique, however.

The Class line consists of four entries -- vendor, OS family, OS generation and device type -- separated by pipes ('|'). These are all text entries and some may be blank. Check NMap's default nmap-os-fingerprints file for examples.

Save the file and test it with nmap by setting the NMAPDIR variable to the current directory:

```
$ NMAPDIR=. nmap -O 10.0.1.103
Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
```

```
Interesting ports on beigebox (10.0.0.3):
(The 1652 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
113/tcp  open  auth
6000/tcp open  X11
Device type: general purpose
Running: OpenBSD 3.X
OS details: OpenBSD 3.4
```

New completed fingerprints should be submitted to http://www.insecure.org/cgi-bin/nmap-submit.cgi for inclusion in future versions of NMap.

1.8 References

Savage. QueSo source code.
http://www.l0t3k.net/tools/FingerPrinting/queso-980922.tar.gz (11 Feb 2003), September, 1998.

Fyodor. NMap source code.
http://www.insecure.org/nmap (11 Feb 2003).

Stevens, W. Richard, TCP/IP Illustrated, Volume 1 The Protocols,
Reading, MA: Addison-Wesley Pub Co., Jan. 1994.

Fyodor, Remote OS detection via TCP/IP Stack FingerPrinting.
http://www.insecure.org/nmap/nmap-fingerprinting-article.html (11 Feb 2003), June 2002.

V. Jacobosn, et. al. RFC1323: TCP Extensions for High Performance.
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1323.html (11 Feb 2003), May 1992.

2. Network Detects

2.1 Detect #1

Source

This detect was captured outside the firewall of a local network I'm responsible for. The firewall sits between the Internet, the local network and a samll service network. The only external services provided by the service network are smtp and dns. The local addresses have been sanitize.

Detect Generation

The data was in question collected using tcpdump between 12/15/03 - 12/17/03. In the process of checking on another issue, I noticed the occasional packet coming from source port 666. The packets were blocked automatically by the firewall, but they raised my interest. Over a three day period, I captured the following packets.

```
14:11:16.121378 64.156.39.12.666 > 10.0.0.1.1026: udp 525
02:24:14.800017 64.156.39.12.666 > 10.0.0.1.1026: udp 543
22:22:58.992665 64.156.39.12.666 > 10.0.0.1.1026: udp 476
04:56:10.757365 64.156.39.12.666 > 10.0.0.1.1026: udp 535
```

Probability of Spoofing

High. Given the source port and the nature of this attack, I've no reason to trust the source information. This is a single-packet UDP attack requiring no response, so as far as the attacker is concerned, the source is irrelavent.

There is a possibility that this was caused by an attack from an infected host. A heads-up to the owner's of that network block might be appropriate. A who-is on the source address 64.56.38.12 yields the following information.

```
Comcast Telecommunications, Inc. CTI-TEL (NET-64-56-32-0-1)
                                  64.56.32.0 - 64.56.63.255
Asia Star Broadcasting Company CMA1-ASIASTAR-1 (NET-64-56-39-0-1)
                                  64.56.39.0 - 64.56.39.31
```

Description of Attack

The attack comes from port 666 (frequently referred to as the Doom port, since id's game used this port) and ephemeral port 1026. A google search of port 666 gave no real light in the situation, but a search of port 1026 immediately turned up possible answers.

Port 1026 is frequently mentioned in association with Microsoft's Messenger Service. This service was originally intended to provide administrators a means of sending messages to users (similar to wall on Unix), but spammers had found they could use it just as easily to pop up spam on user's desktops.

A single UDP packet containing the desired message sent to the Messenger Service port (which is not always 1026) is all it takes to display an ad.

Attack Mechanism

Performing an ASCII-decode on one of the packets using 'tcpdump -X' quickly reveals the purpose of the packet.

```
04:56:10.757365 64.156.39.12.666 > 10.0.0.1.1026: udp 535
0x0000   4500 0233 718b 0000 7011 7e9b 409c 270c   E..3q...p.~.@.'.
0x0010   0a00 0001 029a 0402 021f 801b 0400 2800   C..G..........(.
0x0020   1000 0000 0000 0000 0000 0000 0000 0000   ................
0x0030   0000 0000 f891 7b5a 00ff d011 a9b2 00c0   ......{Z........
0x0040   4fb6 e6fc 3031 3030 3030 3131 3030 3031   O...010000110001
0x0050   3130 3031 0000 0000 0100 0000 0000 0000   1001............
0x0060   0000 ffff ffff c701 0000 0000 0600 0000   ................
0x0070   0000 0000 0600 0000 4164 6d69 6e00 0000   ........Admin...
0x0080   0400 0000 0000 0000 0400 0000 596f 7500   ............You.
0x0090   9701 0000 0000 0000 9701 0000 0d0d 2a2a   ..............**
0x00a0   2a2a 2a2a 2a2a 2a2a 2057 4152 4e49 4e47   ********.WARNING
0x00b0   3a20 5468 6973 206d 6573 7361 6765 2063   :.This.message.c
0x00c0   6f6e 6669 726d 7320 796f 7572 2073 7973   onfirms.your.sys
0x00d0   7465 6d20 6973 2056 554c 4e45 5241 424c   tem.is.VULNERABL
0x00e0   4520 746f 2061 7474 6163 6b73 202a 2a2a   E.to.attacks.***
0x00f0   2a2a 2a2a 2a2a 2a0d 0d0d 5468 6520 7265   *******...The.re
0x0100   6365 7074 696f 6e20 6f66 2061 6e6e 6f79   ception.of.annoy
0x0110   696e 6720 6d65 7373 656e 6765 7220 706f   ing.messenger.po
0x0120   702d 7570 2061 6473 2063 6f6e 6669 726d   p-up.ads.confirm
0x0130   730d 7468 6174 2079 6f75 7220 7379 7374   s.that.your.syst
0x0140   656d 2069 7320 7675 6c6e 6572 6162 6c65   em.is.vulnerable
0x0150   2074 6f20 7669 7275 7365 7320 616e 6420   .to.viruses.and.
0x0160   6174 7461 636b 7320 6279 206d 616c 6963   attacks.by.malic
```

```
0x0170   696f 7573 2068 6163 6b65 7273 2e0d 0d45   ious.hackers...E
0x0180   6c69 6d69 6e61 7465 2074 6865 2061 6e6e   liminate.the.ann
0x0190   6f79 696e 6720 706f 7020 7570 7320 616e   oying.pop.ups.an
0x01a0   6420 7365 6375 7265 2079 6f75 7220 636f   d.secure.your.co
0x01b0   6d70 7574 6572 0d77 6974 6820 6f6e 6520   mputer.with.one.
0x01c0   6561 7379 2d74 6f2d 7573 6520 7072 6f67   easy-to-use.prog
0x01d0   7261 6d20 2d20 4d65 7373 6167 6520 4465   ram.-.Message.De
0x01e0   6665 6e64 6572 2e0d 0d0d 2046 7265 6520   fender.....Free.
0x01f0   696e 666f 726d 6174 696f 6e20 6176 6169   information.avai
0x0200   6c61 626c 6520 2d20 0d0d 2047 6f20 746f   lable.-....Go.to
0x0210   3a20 7777 772e 4d65 7373 6167 6544 6566   :.www.MessageDef
0x0220   656e 6465 722e 636f 6d20 6e6f 7721 0d0d   ender.com.now!..
0x0230   0d0d 00
```

Decoding the other packets showed them to be identical to this one. An additional search of the collected data, this time for port 1026, yielded 7 additional messages from different hosts selling everything from beauty tips to Viagra. This particular add was had the most chutzpah, pointing to a site that sells software designed to block this type of ad.

A search of Microsoft web site turned up an explanation of how Messenger Service worked. In the article "Protecting Window RPC Traffic" [6], it's mentioned that Messenger Service is actually one of Window's RPC clients. The normal flow of traffic would be an incoming tcp query to port 135, which is the Window RPC Port Mapper. That query would as for and get the port number of a particular RPC service on that box. The calling program would then send queries directly to the desired service.

That raised another question, however, as there was no incoming traffic to port 135. The answer to that came from LURHQ[7]. It turns out, that by default, Messenger Service is almost always on port 1026, making the call to port 135 unnecessary.

Correlations

Johannes B. Ullrich of SANS published an excellent wrapup on this issue to the North American Network Operator's Group mailing list[8]. He was researching a similar ad for a company called PopAdStop.com and found that the software they sold to stop this type of popup ad was itself responsible for sending out these messages. PopAdStop.com is surprisingly no longer in business.

Also see the LURHQ article mentioned earlier.

Evidence of Active Targeting

None. This appears to be a very broad scan for any open hosts.

Severity

Criticality: 1. This attack was pointed at a firewall which hides the internal network via NAT. I take attacks against my firewalls very seriously, but because of the nature of these packets, I've lowered the criticality accordingly. The firewall is not Windows-based.

Lethality: 2. This is an annoyance for Windows users, but will not compromise their system.

System Countermeasures: 5. All machines are up to date in terms of patches and OS versions. All systems run anti-virus software and personal firewalls are enabled. Further, the site in question has to Windows machines, making this attack moot.

Network Countermeasures: 5. The firewall is the only means into or out of the local net, and it only allows traffic it expects. Inbound traffic is not allowed to pass unless it's part of an active conversation the firewall has been monitoring. The packets in question were not allowed in. Had the roles been reversed, traffic from port 666 would not have been allowed out.

Severity = (criticallity + lethality) - countermeasures(system + network)
= (1+2)-(5+5) = -7

This "attack" is not worth considering.

Security Recommendations

At the network level, block all inbound and outbound traffic from port 666. If possible, block all inbound network traffic that is not part of a recognized connection.

At the Windows level, make certain all patches are applied. This is especially important as there are known exploits that access system by the same mechanism.[9]

Disable "Messenger Service" if possible: Microsoft has instructions on how to do this for Window NT[10], 2000[11] and XP[12]. For older versions of Windows, consider upgrading. If that's not possible, try using personal firewall software on these machines to block udp traffic to at least port 1026, if not the wider range of 1025-1031. Note that this last piece of advice carries some risk. There's no guarantee as to which port Messenger Service resides on, and arbitrarily blocking all those ports might break some other service.

Multiple Choice Question

The most effective way to prevent the display of Messenger spam from the Internet

across all versions of Windows it to:

1. Disable "Messenger Service"
2. Use a properly configured external firewall
3. Make certain all Critical Updates have been applied.
4. All of the above.

Answer 4.

2.2 Detect #2

Source

The data for these detect was downloaded from www.incidents.org/logs/Raw. In all, 9 days worth of raw tcpdump files were considered. The files used were 2002.10.10, 2002.10.11, 2002.10.12, 2002.10.13, 2002.10.14, 2002.10.15, 2002.10.16, 2002.10.17 and 2002.10.18.

Detect Generation

The tcpdump files were processed by Snort on at a time, using the command line:

```
$ snort -c -r $file ./snort.conf -l logs -S HOME_NET=207.166.0.0/16 \
      EXTERNAL_NET=!207.166.0.0/16
```

The options used were:

-r read from file, where $file was the name of one of the tcpdump files.
-S use the provided values for HOME_NET and EXTERNAL_NET, overriding the values in snort.conf.
-c use the configuration file snort.con
-l logs to the logs/ directory

The snort configuration files was set to also load the alerts into a mysql database for subsequent use with ACID.
The following Snort rules was triggered:

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access";
flags:A+; dsize: >1;  reference:arachnids,203; sid:184;  classtype:misc-acti
vity; rev:3;)
```

The issued alerts looks like this:

```
[**] [1:184:3] BACKDOOR Q access [**]
```

```
[Classification: Misc activity] [Priority: 3]
11/13-15:55:13.576507 255.255.255.255:31337 -> 207.166.225.96:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS203]
```

The packet that caused the attack is here:

```
15:55:13.576507 255.255.255.255.31337 > 207.166.225.96.515: R 0:3(3) ack 0
win 0
                         4500 002b 0000 0000 0f06 4511 ffff ffff
                         cfa6 e160 7a69 0203 0000 0000 0000 0000
                         5014 0000 fa38 0000 636b 6f00 0000
```

Probability of Spoofing

There is no question the source address (255.255.255.255) was spoofed. This is the limited broadcast address and replies to it would never be forwarded by a router.[4]

Description of Attack

This alert appers to indicate an attack by the Q trojan. Its author, Mixter, calls it a "Remote shell and admin tool with strong encryption." [13] It consists of two pieces, a client which gets installed on the remote host, and a server which controls the program. Once the client's installed, the server can use it to open a remote shell, or execute any command on the host. Later versions can even direct the client to listen for traffic on a specific port, then automatically "bounce" (retransmit) it to yet another remote machine.

Attack Mechanism

The Q program is not an attack program, so it relies on some other method to get its client installed on its target. Once the client is installed, it sniffs all incoming packets to its host, looking for any messages from a Q server. This technique renders it transparent to port-scanners, since it has no associated ports. Commands can be hidden in TCP, UDP or ICMP packets, and can be completely self-contained within a single packet, rendering communication back to the server unnecessary unless the server requests it.

The server has options to spoof any source address, making the Snort rule above ineffective. In fact, as far back as Q 1.0, the default was to use random number for source ip and port; I've been unable to determine what attack led to this exact Snort rule. It does appear to be geared at a single instance, or a small subset, of the attacks this tool is capable of.

The Q client/server is written to run on UNIX hosts as command-line tools. The client, qd, goes to great lengths to hide itself from casual observance by system administrators and make itself difficult to purge.

Correlations

A quick scan of the Q source code for versions 1, 2 and the current 2.04 show that it could be the culprit. All three versions are available at PacketStorm Security[14].

A review of arachNIDS Intrusion Event Database[15] show agreement that the rule is not generic, but targeted at a particular exploit.' The Snort signature listed is slightly more targeted than the one in Snort 2.0, looking for source 255.255.255.255/32, instead of 255.255.255.0/24.

Pete Storm, in his GCIA Practical[16], agrees with this analysis. He's assigned a slightly different Severity level (lower Criticality and Countermeasure level) than I have, but our assumptions about the unknown network are different. Pete also pointed to the next source, which turned out to be the best page on this trojan I've seen yet.

Les Gordon's SANS article "What is the Q Trojan? [17] is the most in-depth look into the Q client/server software. He actually installed multiple versions and ran them through their paces, collecting packet traces throughout. The results were both fascinating and terrifying. He also ran across this alert while looking at the logs from SANS, and he agrees the snort rule which generated this detect is likely to be ineffective.

Evidence of Active Targeting

A review of the destination hosts using ACID revealed that none of them were involved in any other alerts. This appears to be a fairly broadbased scan looking for infected hosts.

Severity

Criticality: 4.
This attack is is scanning for Unix machines infected with the Q client.
Given that the majority of Unix machines tend to be servers (not necessarily true, given the advent on Linux on the desktop and Mac OS X), I've increased the criticality accordingly. The destination port is of no use in determining the true target, since the Q client ignores it.

Lethality: 5.

If the attacker finds an infected host, things can't get much worse. That host would be under the complete control of the attacker.

Network Countermeasures: 3.
This is difficult to assess without knowing more information on the network. Using "tcpdump -e" on the raw files, it's easy to see that only two MAC addresses are listed, meaning this traffic was sniffed between 0:0:c:4:b2:33 on the local side and 0:3:e3:d9:26:c0 on the Internet side. Looking up the company prefixes (0:0:c and 0:3:e3) at IEEE[18] shows that both devices are made by Cisco. Beyond that, nothing is known of the network in question.

System Countermeasures: 3.
Again, difficult to assess without knowing anything about the network. I'm making the assumption that the site is middle-of-the-road in terms of its security practices.

Severity = (criticallity + lethality) - countermeasures(system + network)
= (4+5)-(3+3) = 3

Security Recommendations

Due to the nature and adaptability of the Q server/client software, prevent these attacks will involve several levels.

At the network level, the firewall needs to be configured

- Block packets to or from invalid or reserved addresses. A list of these is available from IANA[19]. This would, in itself, have stopped these alerts from triggering. It would not have necessarily stopped Q, however.
- Block any packets that aren't part of a valid connection. The packets in question all had ACK and RESET set, but weren't part of a connection.
- Allow inbound traffic to only selected hosts, if possible. Depending on your network and needs, hiding internal machines behind a NAT'd firewall, for example, would render these inbound scans useless.
- Depending on how strictly you can control outbound traffic, only opening ports for a small set of approved services is a very good idea. It's not a panacea, since a Q server inside the firewall could always choose an "approved" port, but it can make it more difficult for the attacker.

At the host level, this attack targets UNIX-based machines, and requires a process to run with root priveleges in order to operate.

- Install tripwire or some equivalent software, keep it up to date and run it regularly. It

won't prevent the application from getting installed or being run, but it can be an excellent early-warning indicator that there are problems with the machine in question.

- It's important to note that on many operating systems, using a software-based firewall on a box with the Q client wouldn't be effective, since the Q-client will see the packets at the same time as the firewall software. It could, however, help prevent the system from being broken into in the first place, before the Q-client is installed.

Finally, users need to be educated about the dangers involved in installing these types of applications (where "these types" include GoToMyPC.com, PCAnywhere, etc.) on their PCs. Many do it so they can access their PCs from home, never realizing the jeopardy they're putting the internal network in. This obviously won't stop the app from being installed surreptitiously, but it might help prevent the casual use of remote-control software. Every little bit helps.

Multiple Choice Question

Which of the following statements is true:

A One function of the Q server is to install the Q clients on compromised machines.
B. NMap is the best tool for finding compromised machines running Q clients
C. A Q client can be used to control other Q clients.
D. Q servers always use the source address 255.255.255.255.

The answer is C. A Q server can put a client in "bounce" mode, then use that client to forward packets to another machine. The chain can go on indefinitely.

Online Response

This detect was posted the the incidents.org mailing list on February 22nd. There were no responses.

2.3 Detect #3

Source

The data for these detect was downloaded from www.incidents.org/logs/Raw. In all, 9 days worth of raw tcpdump files were considered. The files used were 2002.10.10, 2002.10.11, 2002.10.12, 2002.10.13, 2002.10.14, 2002.10.15, 2002.10.16, 2002.10.17 and 2002.10.18.

Using tcpdump to scan the files, it is apparent that the home network is 207.166.0.0/16. There are also only two MAC addresses are listed in the logs, meaning this traffic was sniffed between 0:0:c:4:b2:33 on the local side and 0:3:e3:d9:26:c0 on the Internet side. Looking up the company prefixes (0:0:c and 0:3:e3) at IEEE shows that both devices are made by Cisco. Beyond that, nothing is known of the network in question.

Detect Generation

The tcpdump files were processed by Snort on at a time, using the command line:

```
$ snort -c -r $file ./snort.conf -l logs -S HOME_NET=207.166.0.0/16 \
      EXTERNAL_NET=!207.166.0.0/16
```

The options used were:

-r read from file, where $file was the name of one of the tcpdump files.
-S use the provided values for HOME_NET and EXTERNAL_NET, overriding the values in snort.conf.
-c use the configuration file snort.con
-l logs to the logs/ directory

The snort configuration files was set to also load the alerts into a mysql database for subsequent use with ACID.

The following Snort rules was triggered:

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC bad frag bits";
fragbits:MD; sid:1322; classtype:misc-activity; rev:4;)
```

The issued alerts looks like this:

```
[**] [1:1322:4] BAD TRAFFIC bad frag bits [**]
[Classification: Misc activity] [Priority: 3]
11/09-17:24:47.786507 81.97.214.13 -> 207.166.84.248
TCP TTL:110 TOS:0x0 ID:36529 IpLen:20 DgmLen:1468 DF MF
Frag Offset: 0x0000   Frag Size: 0x0014
```

The packet that caused the alert is here:

```
17:24:47.786507 81.97.214.13.4746 > 207.166.84.248.80: .
691024427:691025855(1428) ack 3604160229 win 17520 (frag 36529:1448@0+) (DF)
  0000: 4500 05bc 8eb1 6000 6e06 56c7 5161 d60d  E..?.?`.n.V?Qa?.
  0010: cfa6 54f8 128a 0050 2930 322b d6d3 1ee5  ?T?...P)02+??.?
  0020: 5010 4470 dfc7 0000 feff ff69 d28d 66f0  P.Dp??..???i?.f?
```

```
0030: 5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t???.E...P???
0040: 8948 108b f48d 952c feff ff52 6a00 8d85  .H..?..,???Rj...
0050: 4cfe ffff 508d 4e4e 4e4e 4e4e 4e4e 4e4e  L???P.NNNNNNNNNN
0060: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
0070: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
0080: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
0090: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00a0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00b0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00c0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00d0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00e0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
00f0: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
0100: 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e  NNNNNNNNNNNNNNNN
0110: 4e4e 4e4e 4e4e 4e4e 4e25 7539 3039 3025  NNNNNNNNN%u9090%
0120: 7536 3835 3825 7563 6264 3325 7537 3830  u6858%ucbd3%u780
0130: 3125 7539 3039 3025 7536 3835 3825 7563  1%u9090%u6858%uc
0140: 6264 3325 7537 3830 3125 7539 3039 3025  bd3%u7801%u9090%
0150: 7536 3835 3825 7563 6264 3325 7537 3830  u6858%ucbd3%u780
0160: 3125 7539 3039 3025 7539 3039 3025 7538  1%u9090%u9090%u8
0170: 3139 3025 7530 3063 3325 7530 3030 3325  190%u00c3%u0003%
0180: 7538 6230 3025 7535 3331 6225 7535 3366  u8b00%u531b%u53f
0190: 6625 7530 3037 3825 7530 3030 3025 7530  f%u0078%u0000%u0
01a0: 303d 6120 2048 5454 502f 312e 300d 0a43  0=a  HTTP/1.0..C
01b0: 6f6e 7465 6e74 2d74 7970 653a 2074 6578  ontent-type: tex
01c0: 742f 786d 6c0a 484f 5354 3a77 7777 2e77  t/xml.HOST:www.w
01d0: 6f72 6d2e 636f 6d0a 2041 6363 6570 743a  orm.com. Accept:
01e0: 202a 2f2a 0a43 6f6e 7465 6e74 2d6c 656e   */*.Content-len
01f0: 6774 683a 2033 3536 3920 0d0a 0d0a 558b  gth: 3569 ....U.
0200: ec81 ec18 0200 0053 5657 8dbd e8fd ffff  ?.?....SVW.?????
...
```

Probability of Spoofing

Possible but unlikely. This is a an ack packet, meaning it would have to be part of a TCP conversation before the host would do anything with it.

The hosts resolves to spr1-seve2-4-0-cust13.lond.broadband.ntl.com, which speaks for itself. Most likely a broadbad customer with London's NTL. A whois doesn't shed much light on the situation, but does give an abuse address:

The whois information for this host points to Great Britian:

```
inetnum:      81.97.208.0 - 81.97.223.255
```

```
netname:      NTL
descr:        NTL Infrastructure - Waltham Park
country:      GB
...
role:         NTLI Network Management Centre
address:      NTL Internet
address:      Crawley Court
address:      Winchester
address:      Hampshire
address:      SO21 2QA
trouble:      ---------------------------------------------
trouble:      For abuse notifications please -
trouble:      email : abuse@ntlworld.com
trouble:      telephone : +44 2920 305142
trouble:      ---------------------------------------------
...
```

Description of Attack

This alert was triggered because the packet had both the "Don't Fragment" and "More Fragments" bits set, something that can happen in valid IP traffic[20]. Take a close look at the beginning of the IP header of the packet in question:

```
691024427:691025855(1428) ack 3604160229 win 17520 (frag 36529:1448@0+) (DF)
  0000: 4500 05bc 8eb1 6000 6e06 56c7 5161 d60d  E..?.?`.n.V?Qa?.
```

The fragment information is stored in bytes 6 and 7 of the IP header and is layed out like this:

R	DF	MF	13-bit fragment offset

The R bit is reserved and should always be 0. The DF bit, when set, states the packet should never be fragmented, under any circumstances. The MF bit indicates if there are more fragments to follow, where MF=1 means yes. The fragment offset is used to tell the recieved where in the packet this fragment belongs (the offset is measured in 8-octets, so the actual offset is 8 times the fragment offset). In the above packet, the value of the high byte is 6, meaning both MF and DF are set. That is non-sensical and disallowed by the standard.

Those bits are the reason for the alert, but they're not what piqued my interest in the packet. Take a look at an ASCII reprentation of the payload, slightly reformatted from the detect above:

```
        NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u
6858%ucbd3%u7801%u9090%u6858%ucbd3%u780
1%u9090%u9090%u8190%u00c3%u0003%u8b00%u
531b%u53ff%u0078%u0000%u00=a
```

Now compare it to this one:

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u
6858%ucbd3%u7801%u9090%u6858%ucbd3%u780
1%u9090%u9090%u8190%u00c3%u0003%u8b00%u
531b%u53ff%u0078%u0000%u00=a
```

The second payload comes from CERT's advisory about the Code Red worm[21]. Our packet appears to be corrupted Code Red attack.

Attack Mechanism

The "Code Red" virus spreads itself by exploiting a vulnerability in Microsoft's IIS server. It sends a carefully crafted HTTP request which causes a buffer overflow in unpatched versions of IIS, allowing the execution of the code embedded in the request. The compromised IIS server would then begin attacking other servers using the same mechanism.

There are four distinct differences between our code fragment and a typical Code Red attack.

First, our packets are fragmented. That proves nothing, however, other than the packet we're looking at got fragmented somewhere along its journey to us.

Second, our packet has both the Don't Fragment and More Fragments bits set. This is what triggered the alert.

Third, the "N-sled," or sequence of consecutive Ns, is slightly longer in our packet than in the standard Code Red attack. That could mean the attacker used a modified version of Code Red for this attack, of which several are available (as detailed in the SANS paper "Code and Code Red II: Double Dragons" [22]). The Ns are used to force the buffer overflow and won't interfere with the overall attack.

Finally, the lead-in to the N-sled in our packet contains garbage. Where the standard Code Red attack contains the phrase "GET /default.ida?", our packet begins with a random sequence of bytes. Specifically, it doesn't begin with a standard HTTP request, and so should be rejected out-of-hand by the server, rendering the attack moot. There's always the possibility this is some new form of attack, but I've been unable to find any corroborating evidence that such an attack exists.

There were a total of five such packets sent from the same source to the same destination over a 3 minute period. Neither the source nor destination figured in any other alerts. All the alerts were of the "bad frag" type, all contained a fragment offset of zero and only two contained N-sleds at the start.

My guess is that this is a standard Code Red (or variant) attack that was corrupted somewhere in transit. It's possible that the same device or code that corrupted the Code Red signature corrupted the fragmentation bits and offsets as well.

Correlations

Todd Williams posted a similar detect on the intrustions.org mailing list in December, 2003[23]. He analyzed different data from different days, but he came to the same conclusions. His packets actually contained the complete Code Red signature, making detection slightly easier.

He noted however that Snort triggered the "bad frags" alert and not the "Code Red" alert because the "bad frags" alert appeared first in the snort rules. That finding is a perfect example of how the ordering of rules in Snort is critical. Though it didn't happen in our case, it's possible for a low-level alert to mask higher level one, if the low-level rule is checked first. This is because Snort will only alert once for any given packet. Also note the order in which Snort applies rules is not necessarily the order in which they're listed in the rules files. Anyone interested in as to why should consult section 3.16, "How does rule ordering work?", in the Snort FAQ[24].

Evidence of Active Targeting

This appears to be a random attack. Five fragmented packets over three minutes constitute the entirety of the attack.

Severity

Criticality: 3. This attack, assuming it's Code-Red related, targets IIS web servers, which slightly increases its severity. The scale of the attack however keeps me from assigning it anything higher than a 3.

Lethality: 3. The consequence to an infected machine are severe, but offset by the fact that Code Red is an extremely old attack, and patches for it have been available from Microsoft for nearly as long.

Network Countermeasures: 3.
This is difficult to assess without knowing more information on the network. I've assigned an average value.

System Countermeasures: 3.
Again, difficult to assess without knowing anything about the network. I'm making the assumption that the site is middle-of-the-road in terms of its security practices.

Severity = (criticallity + lethality) - countermeasures(system + network)
= (3+3)-(3+3) = 0

Security Recommendations

At the network level, the firewall needs to be configured

- Block packets with invalid flag settings. That would reduce this particular alert to nothing more than a mild curiosity.
- Implement a proxy for all web-based traffic. Should this turn out to be a corrupted Code Red attack, the attacks would be caught and nullified before hitting the internal network.
- If feasible, defragment packets at the firewall, rejecting any with problems (e.g., gaps, overlaps, invalid offsets, etc.). Depending on traffic, this might place too much load on the firewall, as well as open it to denial-of-service attacks.

At the host level, these were ACK packets heading for the http port of the host in question:

- Verify if the host in question is running an IIS server. If so, it should be checked for infection (all current anti-virus software can detect Code Red and variants). If the machine is infected, best practices dictate reloading the machine from a known-good backup or re-installing it from scratch. That's particularly important if this is a new variation on the Code Red virus. Also verify that the host actually needs to be

running IIS. Many times it's installed "because I wanted everything," not for any specific purpose.

- Make certain all patches have been applied, and that the AV software is up to date. Microsoft provides several free tools -- HFNetCheck[25], MBSA[26] and others -- which can be used to verify exactly which patches have been installed on a given machine. The Windows Update feature can also be used, though its website is not available to non-Windows hosts, so I'm unable to verity what's available there.
- Tripwire and/or a similar package, when installed properly, can help identify the creation of, or changes to, executables. This can be invaluable in determining if a system has been compromised. It's not alway possible to get users to use such a system on the desktop, but it should be strongly recommended. It should be mandatory on servers.

Multiple Choice Question

A value of 2 in byte 6 of the IP header indicates:

A. This is the first part of a fragmented packet.
B. This packet should never be fragmented.
C. There are additional fragments following this one.
D. This is the final fragment of a fragmented packet.

Answer: C

The MF flag is set when there are more fragments following this one. The flag is cleared if the packet is not fragmented, or if it's the last fragment for a packet.

3. Analyze This

The following report provides an in-depth analysis of five days worth of Snort IDS logfiles provided by the University. Please note that this analysis assumes that a given IP has not been shared amongst several machines (e.g., DHCP) during this period.

3.1 Executive Summary

Numerous internal systems have been compromised by remote hosts and are actively being used to attack other machines. Fixing these machines needs to be your highest priority.

Several Windows viruses are running rampant throughout the network. It will require a major effort to isolate and disinfect these machines, but it must be done. The University needs to take a pro-active role in helping its users protect and secure their system. Providing such services is not without cost, but it will, in the end, be cheaper than continually dealing with these same issues.

A list of infected systems requiring immediate attention appears in the Insights section, along with the hundreds of potentially-infected local systems that need to be checked as well.

This document includes several defensive recommendations that should be implemented as soon as possible. Also, note that implementing these procedures is necessarily an iterative process. Once the major items on the recommendations are dealt with, it's quite possible that new scans will show other new high-priority items that will need to be addresses before the lower-priority items listed below. Over time, however, the number of recurring incidents will lessen as systems are isolated, disinfected and patched to current levels. It is recommended that administrators begin with the firewall and network recommendations, as these are "one-shot" items which won't need to be repeated in the future.

3.2 Analyzed Files

Log files, collected by the Snort IDS, were provided for analysis. These files, covering the period between July 24th, 2003 and July 28th, 2003, were downloaded from www.incidents.org/logs.

alert.0300724	scans.030724	OOS_Report_2003_07_25_2435
alert.0300725	scans.030725	OOS_Report_2003_07_25_2435
alert.0300726	scans.030726	OOS_Report_2003_07_27_18859
alert.0300727	scans.030727	OOS_Report_2003_07_28_29050
alert.0300728	scans.030728	OOS_Report_2003_07_29_23718

Table 1: Files used in analysis

3.3 Machine Overview

The provided logfiles masked the internal network address behind the string "MY.NET", which is reflected in the remainder of the document. It's worth pointing out that the scan files were not masked, so we were able to determine that "MY.NET" is really 130.85.

Address	**Purpose**	**Notes**
MY.NET.100.165	External Web Server	CS Web Server possible ftp
MY.NET.53.29	FTP server	HelpDesk
MY.NET.24.15	LPD server	
MY.NET.24.27	FTP server	
MY.NET.24.47	FTP server	
MY.NET.113.207	Web Server/IMAP	
MY.NET.1.3	DNS Server (?)	If not, see Scan section!
MY.NET.25.21	Mail server (pop and imap)	
MY.NET.25.22	Mail server (pop and imap)	
MY.NET.25.23	Mail server (pop and imap)	
MY.NET.25.24	Mail server (pop and imap)	
MY.NET.30.3	Web and Novell server	
MY.NET.30.3	Web and Novell server	

Table 2: Machines Discovered

3.4 Summary of Detects

The alert files contained 1,210,386 separate events, recorded one per line using the following format:

date-time [**] alert type [**] source:port -> destination:port

Any records which didn't meet the above format were considered invalid and were ignored for this analysis. These records all appear to be the result of two or more valid alerts becoming intermingled during collection. All together, less that .2% of the records were invalid.

The remaining records fell into two main categeries. The "Portscan Notification" records, which all have an alert type starting with "spp_portscan" actually augment the information in the scan files and are discussed in the Scan section below. The remaining records, which I've tagged as "IDS Alerts", have been sorted as to severity and are addressed in the following Alert sections.

The final breakdown of alerts is show in the following table:

IDS Alerts	534,756
Portscan Notifications	673,772
Invalid Records	1,858

Table 3: Alert breakdown

3.4.1 Alerts

The following table lists the alerts in order of quantity received, one alert per row.

The Lv column contains the level I've assigned to the alert in question, based on an analysis of the systems involved in that alert. High severity alerts (H) demand immediate attention, noting compromised systems and/or active attacks. Moderate severity (M) alerts require attention, but the corresponding threat isn't nearly as severe, or there is a high probability of false-alarms. Low-severity alerts should eventually be reviewed, but the threat is minimal. Alerts flagged with a question mark are apparantly information and it was impossible to assign a severity without knowing the reason behind the alert type. Alerts with a blank level represent what amounts to noise.

The Inbound and Outbound columns each contain 3 values in the format Q/S/D, where Q is the quantity of alerts received, S is the number of unique source addresses for those alerts, and D is the number of unique destinations. There were no alerts for packets between local systems (MY.NET.x.x -> MY.NET.x.x), so there is no corresponding column.

The comment column is used to for any additional information regarding the alerts (time of day, spikes, etc.).

Each alert type is individually detailed beginning immediately after the chart.

Lv	Alert	Qty	Inbound	Outbound	Comments
H	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	174070	174070/57/53	0/0/0	81%: 7/25 bus. hours 18%: 7/25 bus. hours
?	CS WEBSERVER - external web traffic	141208	141208/20550/1	0/0/0	
	SMB Name Wildcard	58207	58207/953/1299	0/0/0	Fairly uniform spread.
H	High port 65535 udp - possible Red Worm - traffic	46797	26402/145/37	20393/15/134	92% on 7/24 between 4 and 5pm
H	spp_http_decode: IIS Unicode attack detected	37093	3528/191/362	33565/345/770	Fairly uniform spread
	Queso fingerprint	12849	12849/349/81	0/0/0	
H	High port 65535 tcp - possible Red Worm - traffic	10018	4473/57/105	5545/39/77	73% on 7/25 at 2am
?	MY.NET.30.4 activity	9916	9916/420/1	0/0/0	
?	MY.NET.30.3 activity	9512	9511/88/1	0/0/0	
H	spp_http_decode: CGI Null Byte attack detected	8428	102/7/14	8326/99/106	58% on 7/27 at 9pm
M	EXPLOIT x86 NOOP	8327	8327/62/93	0/0/0	Most on 7/24, 26, 27
H	Tiny Fragments - Possible Hostile Activity	3338	787/11/10	255/1/1	
	SYN-FIN scan!	2552	2552/2/2552	0/0/0	All but one on 7/28 at 3pm
	connect to 515 from outside	1982	1982/2/2	0/0/0	90% of trarric between 8-9am and 10-11pm on the 24th, 25th and 28th.
H	Possible trojan server activity	1660	1274/38/935	386/53/38	Hugh spike 7/24 at 2am
	IDS552/web-iis_IIS ISAPI Overflow ida nosize	1368	1368/761/492	0/0/0	Uniform spread

L	TCP SRC and DST outside network	1186	0/0/0	0/0/0	Evenly distributed throughout week.
M	TFTP - External TCP connection to internal tftp server	1031	894/2/394	137/60/1	Peak on 7/26
L	External RPC call	931	931/2/682	0/0/0	Spikes 7/25 and 7/28
M	SUNRPC highport access!	716	716/14/14	0/0/0	Spike 7/25
	Null scan!	671	671/43/37	0/0/0	Peak 7/25 4-6am
	NMAP TCP ping!	669	669/126/65	0/0/0	Uniform spread
L	Incomplete Packet Fragments Discarded	350	200/39/35	150/1/5	Uniform spread
M	EXPLOIT x86 stealth noop	288	288/8/7	0/0/0	Spike 7/25 1-2pm
?	CS WEBSERVER - external ftp traffic	201	201/14/1	0/0/0	Spikes 724, 7/26
L	SNMP public access	173	173/1/1	0/0/0	7/24, 25, 28: bus hours
L	SMB C access	150	150/75/8	0/0/0	
H	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	131	0/0/0	131/2/107	7/24 bus. hours
H	NIMDA - Attempt to execute cmd from campus host	110	0/0/0	110/9/81	Spikes 7/24 at 8-9am and 2-3pm
?	Notify Brian B. 3.54 tcp	95	95/50/1	0/0/0	
L	TFTP - Internal TCP connection to external tftp server	80	48/6/4	32/2/4	Spike 7/28 8pm
	IRC evil - running XDCC	79	0/0/0	79/2/2	Uniform spread
?	Notify Brian B. 3.56 tcp	65	65/38/1	0/0/0	
L	FTP DoS ftpd globbing	61	61/9/1	0/0/0	Spike 7/27
L	FTP passwd attempt	57	57/39/1	0/0/0	Uniform spread
M	Attempted Sun RPC high port access	54	54/6/5	0/0/0	Spike 7/25
	EXPLOIT x86 setuid 0	53	53/39/38	0/0/0	Uniform spread
M	RFB - Possible WinVNC - 010708-1	39	20/8/8	19/7/7	
	EXPLOIT x86 setgid 0	35	35/29/26	0/0/0	Uniform spread
	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	27	27/2/2	0/0/0	
H	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	18	0/0/0	18/13/2	
	NETBIOS NT NULL session	18	18/5/7	0/0/0	Spike 7/25 4-6am
H	TCP SMTP Source Port traffic	15	15/1/2	0/0/0	7/25 10pm
M	TFTP - Internal UDP connection to external tftp server	15	9/6/7	6/2/2	Spike 7/26 5pm

H	PHF attempt	14	14/2/8	0/0/0	7/25,7/26 after bus. hours
?	External FTP to HelpDesk MY.NET.70.50	13	13/4/1	0/0/0	
H	EXPLOIT NTPDX buffer overflow	13	13/7/5	0/0/0	Spike 7/28
?	External POP to HelpDesk MY.NET.70.49	11	11/1/1	0/0/0	
	TFTP - External UDP connection to internal tftp server	10	10/4/5	0/0/0	
?	External POP to HelpDesk MY.NET.70.50	8	8/1/1	0/0/0	
?	External FTP to HelpDesk MY.NET.70.49	8	8/2/1	0/0/0	
H	[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.	5	5/4/5	0/0/0	
	Probable NMAP fingerprint attempt	5	5/4/4	0/0/0	
	Traffic from port 53 to port 123	5	5/2/1	0/0/0	
?	External FTP to HelpDesk MY.NET.53.29	4	4/2/1	0/0/0	
	ICMP SRC and DST outside network	3	0/0/0	0/0/0	
M	Back Orifice	3	3/1/2	0/0/0	
H	[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	2	0/0/0	2/2/1	
	DDOS shaft client to handler	2	2/2/2	0/0/0	7/24
H	DDOS mstream client to handler	2	2/2/2	0/0/0	7/27
H	[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	1	1/1/1	0/0/0	
H	NIMDA - Attempt to execute root from campus host	1	0/0/0	1/1/1	
	Fragmentation Overflow Attack	1	1/1/1	0/0/0	
H	[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	1	1/1/1	0/0/0	
?	HelpDesk MY.NET.70.49 to External FTP	1	0/0/0	1/1/1	

Table 4: Summary of Alerts

[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.

This alert triggers on the presence of a /kill command in an IRC conversation. This command was originally meant to be used by the IRC sysop to forcefully terminate a

user's connection. It's proper usage is therefore innocuous. Unfortunately, many attack tools use IRC to communicate with their controlling machines and/or victims. In this case, the presence of a "/kill" command could signal a remote user is terminating a connection with a compromised host.

It's also worth noting that the /kill command has been removed from many of the latest IRC clients: as irchelp.org[27] states: "With the advent of auto-reconnecting clients KILL is almost totally worthless as a tool for punishment." It's usage is now relegated to old IRC clients and viruses.

Recommendation: The 53 internal system targeted by these alerts need to be checked immediately. Many are already compromised.

Correlations:

I could find no direct correlations for this event, since it's specific to the University. However, by combining irchelp.org's explanation of /kill, along with viruses which communicate via IRC (see sdbot below), the above explanation seems correct.

SMB Name Wildcard

This alert indicate a NETBIOS name query[28] against a given host. The query could be used by an attacker to gain such information as the workstation's name, domain, and the ids of logged-in users. In an of itself, this alert is not very serious. However, the information gained could be of great use to an attacker, helping to target future attacks against known users and systems.

These is part of Microsoft's SMB and NetBIOS services which operation on UDP ports 137, 138 and 139, as well as TCP ports 139 and 445. These parts a critical to Windows networking, but should not be allowed to cross the firewall into or out of the local network. Additionally, once blocked, it might be advisable to disable the inbound alerts on these ports. As can be seen, they generate a tremendous amount of alert data than can hide other, more significant, attacks.

Correlations:

Terry MacDonald's GCIA Practical Assignment[29], p.48 agrees with the overall assessment of this being an extremely low severity item.

Microsoft has a wealth of information on these ports, including an excellent article on their security site for "Hardening Systems and Servers." [30]

High port 65535 udp - possible Red Worm - traffic
High port 65535 tcp - possible Red Worm - traffic
Red Worm (also called "Adore") is a particularly invasive unix-based virus. It infects systems via holes in any of several services -- BIND, LPRng, rpc-statd -- and automatically installs itself on the system by modifying or replacing numerous system files. Once installed, the virus signals the owner that it's installed and waits for a crafted ICMP packet. When that packet arrives, the virus opens port 65535 and allows incoming telnet session to connect to that port directly as root. In particular, hosts MY.NET.70.207 (18,492 packets) and MY.NET.114.89 (4,322 packets) should be checked immediately.

This worm will replace several system executables, including ps, 0anacron and klogd. The latter is used to collect information on local users and process, which it subsequently emails to a remote address (long since closed). The worm will also begin outbound attacks, looking for other victims to infect.

Once infected, hosts will need to be reloaded from known-good versions, or completely reinstalled, since all files and executables left on the system are suspect.

Correlations:
Though they covered different dates, this assessment matches with those provided in Terry MacDonald's GCIA Practical V3.3[29] and Les Gordon's GCIA Practical V3.3[31].

J. Anthone Dell provides an excellent analysis of the Adore Worm virus[32], from which much of worm's attack characteristics were drawn.

spp_http_decode: IIS Unicode attack detected
These alerts point to a variety of attacks against Microsoft's IIS server. These exploits take advantage of an IIS flaw in handling Unicode-encoded queries, allowing the attacker to bypass IIS security mechanisms and execute commands that would normally be rejected. For example, the ASCII string "\" can be represented in unicode as "%c1%9c". Whereas an unpatched IIS server would reject this query:

```
http://www.example.com/scripts/..\../winnt/system32/cmd.exe?/c+dir
```

it would accept the the same query with the "\" in unicode:

```
http://www.example.com/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir
```

Using such a carefully constructed query allows attackers (sadmind, Nimda, Code Red

and others) to execute arbitrary commands on the web server.

This allows the attacker to take complete control of the system in question. There are a variety of widely-available attacks -- sadmind, Nimda, Code Red and others -- which can trigger this alert.

Two actual issues are indicated here. The first is that there are incoming attacks looking for IIS servers. The second is more frightening, in that attacks are going out of the local network to remote sites. This implies that numerous local hosts have already been infected. Disinfecting these hosts, and disabling unnecessary IIS installations, is highly recommended.

Correlations:
SANS Intrustion Detection FAQ: What are the vulnerabilities on Internet Information Server (IIS)[33]
Donald Gregory's GCIA Practical V3.2, p. 26-27[34]

Queso fingerprint
SYN-FIN scan!
Null scan!
NMAP TCP ping!
Probable NMAP fingerprint attempt
These are all remote scans of machines on the internal network. Queso[1] is an old OS fingerprinting tool that is used for reconnaisance. It works by sending six packets -- one normal and 5 malformed -- to a remote system, then anaylizing the packets it gets back. This works because while the handling of proper packets is carefully standardized, the responses to invalid packets it no. Different operating systems will leave different flags and/or options set when they reject the packet.

The other scans also involve sending invalid packets: the SYN-FIN scan has those two flags set, hoping firewalls will see the FIN and let the packet through; the Null scan has no flags set; and the NMAP TCP ping as the ACK flag set but isn't part of an existing conversation. The NMAP fingerprint technique is discussed indepth and will not be covered again here.

Blocking invalid packets at the border router will stop the first three remote scans cold. A stateful firewall should eliminate the TCP ping.

spp_http_decode: CGI Null Byte attack detected
This alert is triggered then the scanner sees a Unicode-encoded NULL (%00) in a URL begin sent in an http query to a web server.

The attack works by exploiting the difference between how a web server sees a request, and how the web server's operating system sees it. When a web server sees a request for page.html, it eventually hands the request off to the operating system to open the file (a separate CGI language might be involved here, but eventually the request gets to the OS). The OS opens the file and returns the contents, which the web server can process and return to the client. If the request lacks the appropriate extension, the web-server will append it before passing the request along to the OS, so a request for "page" will converted to "page.html" by the web server before it gets passed to the OS. The server will also convert any Unicode-embedded sequences to their ASCII counterparts before handing off the request, since most OS's don't deal with Unicode.

Therein lies the issue. Someone figured out that by appending a Unicode-encoded nul ("%00"), they could get the web server and operating system to disagree on which file was being asked for. An attacker would request `/etc/passwd%00`, the web server would change that to `/etc/passwd\0.html`, where "\0" is an actual zero byte. The OS would treat that zero as the end of the string, because that's how strings are represented in the C programming language. The `.html` ending would be ignored completely, and the OS would return the contents of the password file `/etc/passwd`. The OS is happy, the server is happy, and the attacker is very happy.

This attack is now checked for by all modern web servers, but older systems will need to be updated. Additionally, any CGI programs being used by the web server will need to be inspected to verify that they also treat such names correctly.

The local systems that triggered this attack need to be examined and disinfected, if necessary. The local systems that were attacked using this technique also need to be checked to ensure any software they're running is either patched or not vulnerable.

Correlations:
This mal-feature is discussed in depth in the Phrack #55[35] article "Poison Null Byte".

Donald Gregory's GCIA Practical also covers this item, agreeing with the above analysis.

EXPLOIT x86 NOOP
EXPLOIT x86 stealth noop
EXPLOIT x86 setuid 0
EXPLOIT x86 setgid 0
Each of these alerts works by scanning packets for a particular series of bytes, triggering when those series are found. The first alert looks for a string of 10 bytes of 0x90 (the x86 no-op instruction), the second looks for NOOP looks for the sequence "eb 02 eb 02 eb 02", the third looks for "b0 17 cd 80", while the fourth looks for "b0 b5 cd 80". These strings are common in overflow attacks against x86-based hosts.

The problem here is that these strings also appear frequently in a wide variety of non-hostile data (jpegs, gifs, mpegs, etc.), all of which will trigger an alert. By way of example, I can trigger these regularly on an closed all-Macintosh network, where the probability of this being a threatening x86-based overflow attack is nil. Further, there is no way to diagnose which alerts are valid without access to the actual raw packet data, where the analysis is both time-consuming and, usually, unrewarding.

The best that can be said is that these packets *might* represent a threat, and the actual packet data will need to be checked. If that data is not available, then the targeted systems should be manually checked. In the absence of any hard data, I'm assigning this a moderate threat.

Correlations:
Terry MacDonald[29] disagrees with this analysis and rates this item as 'Noise'. I agree with Terry that there are an incredible amount of false-positives for this alert, but I disagree they can all simply be dismissed without any corroborating data (e.g., packet traces).

The SmashGuard Group at Purdue University has a "Buffer Overflow Page" which is an excellent source of information on these types of attacks.[36]

Tiny Fragments - Possible Hostile Activity
Tiny fragments are frequently used to subvert firewall rules, since many of those rules require the entire packet to work correctly. If the firewall doesn't re-assemble the incoming packets before applying the rules, there usually isn't enough information in a given fragment to do anything other than accept or reject it outright.

The outbound scan from MY.NET.97.91 is worrisome and should be checked immediately, since there's no reason for internal hosts to fragment outbound traffic.

Inbound fragments need to either be reassembled by the firewall before being passed, or at the vary least, scrubbed to ensure no overlaps and/or dupicates pass (the latter options can significantly decrease the overhead on the firewall). Outbound fragmented packets should be logged and dropped, unless there is some known reason as to why they need to exist.

Correlations:
Terry MacDonald (p.45) agrees with this assessment, but also recommends setting an upper limit to fragment-alert sizes to 512 bytes[29]. He bases that on an interesting note in the Snort documention: "Generally speaking, there is no piece of commercial networking equipment that fragments in sizes smaller than 512 bytes."

connect to 515 from outside
Port 515 is normally reserved for the lpd printer service. Further, all but one packet came from a single host (131.118.229.7) to a single host (MY.NET.24.15). Neither host is the subject of any other alerts. MY.NET.24.15 was the subject of multiple scans, but none on port 515. I believe this to be the result of someone running an lpd server on MY.NET.24.15 and printing to it remotely, usually in the mornings and late in the evenings. While there are known lpd exploits[37], this traffic appears to be innocuous.

Possible trojan server activity
These alerts are for connections to and from port 27374. Numerous attack tools for Windows -- Bad Blood, Ramen, SubSeven, etc. -- are known to use this port. Many of these alerts appear to be the result of innocent traffic: connections to imap, pop, smtp and web servers from the ephemeral port 27374.

Ramen is a worm that atacks several UNIX servers, including w-ftpd, rpc.statd and LPRng (the "next-generation" print server). Once it breached a remote machine, it would download it's own source code, compile it and then begin outbound attacks from that machine. It modified the ftp configuration to allow anonymous connections, and spawned a web-server on port 27374, from which other machines could get copies of its source.

Sub-Seven is a trojan for Windows systems. It works by getting the server installed on a victim's machine, which can be accomplished through a wide variety of methods (buffer overflow attacks, social engineering, etc.). Once executed, the server makes certain it's run at startup by adding itself to the system's win.ini and system.ini, as well as the registry under the key `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows` (subkeys `Current\Version\Run` and `CurrentVersion\RunServices`). It also places copies of itself in a half-dozen locations under a variety of filenames. The compromised

system then contacts the attacker's client whenever the victim connects to the Internet. The attacker then has complete control over the victim's machine using whatever port the server opened, including 27374.

There was, however, one large scan from 68.68.110.48 which got responses from 37 local machines. These machines need to be examined and disinfected as soon as possible.

Blocking inbound SYNs to port 27374 at the firewall would stop these inbound scans and/or attacks at the firewall.

Correlations:
William Sterns ha a script available from SANS[38] which can help find systems infected with Ramen. The same source contains quite a bit of information on what the worm is and how it can be removed.

SubSeven is discussed in detail in the SAN's FAQ[39]. The article also lists directions for removing the trojan, though all current anti-virus software can remove it as well. However, it's best to reload the infected systems, or restore them from a known-good backup, since removing the server doesn't guarantee the attacker hasn't planted other viruses or trojans on the machine in question.

Simovits Consulting has a handy page[40] of trojan port number complete with short summaries on all the trojans that use port 27374. The complete list is considerably longer than the few listed above.

IDS552/web-iis_IIS ISAPI Overflow ida nosize

These are attempts to exploit yet another vulnerability in Microsoft's IIS. In unpatched systems, a buffer-overflow could be used to take control of the targeted machine[41]. This was the principle attack vector of the Code Red virus.

Any internal hosts running IIS need to be patched to the current level at all times.

Correlations:

This attack is discussed in-depth in Detect #3 in Section 2. That analysis would be included here for the University.

TCP SRC and DST outside network

This alert was caused by 93 hosts, the vast majority of which are either reserved addresses (10.x.x.x, 192.168.x.x) or dialup carriers like AOL. These are most likely caused by users plugging in laptops that were configured for their home networks. There is no apparent danger in these packets.

These types of packets should be dropped at the firewall.

TFTP - External TCP connection to internal tftp server

TFTP - External UDP connection to internal tftp server

These alerts were triggered when a remote system (68.55.195.92) scanned the MY.NET.70 and MY.NET.71 subnets for tftp servers. These servers are generally used to provide boot images for diskless clients, and can provide a treasure trove in data to attackers. Port 69/tcp is also used by an Windows attack tool called BackGate[42]. Once installed, it allows the attacker to open new accounts, enable multiple proxy servers -- ftp, telnet, www, SOCKS, and others -- and also use the compromised box to launch other attacks.

Of concern among these queries are the 60 internal hosts that responded to TCP queries. These hosts need to be checked to insure they aren't infected, and that they really do need to provide tftp services.

The firewall should be configured to block all access, inbound or outbound, to port 69.

External RPC call

These alerts were caused by two remote systems scanning internal hosts for machines running portmapper (a unix utility that handles remote procedure calls). There are several well-known buffer-overflow bugs in various versions of portmapper which a well-crafted RPC (remote procedure call) can compromise. The result is usually root access for the attacker.

The scanned system need to be checked to see if they are running portmapper and to see if they've been compromised. The RPC port 111 needs to be blocked at the firewall.

SUNRPC highport access!

Attempted Sun RPC high port access

These alerts are both caused by connections to internal hosts on port 32771. This port is normally used by Sun for RPC calls, and can therefore can represent an attack vector

if the host is running Sun's portmapper.

These alerts are no longer provided with Snort, and without the snort rule, it's difficult to assess the severity of these alerts. They may represent attempted RPC attacks and/or scans, or they may be part of a valid, normal conversation (all but 64 originate from http, domain, imap and pop ports).

Without knowing the exact rules that triggered these events, I'm assigning a moderate priority and recommending the internal hosts be examined for possible compromise. Updating snort and installing the latest rules will generate much more targetted RPC alerts, eliminating much of the confusion here.

Incomplete Packet Fragments Discarded

This alert is caused by Snort's old defrag filter. This filter was regarded as unstable, generating many false positives, and was replaced by the frag2 filter in Snort 1.8.

The only system that appears to be of concern here is MY.NET.83.98 which appears to be sending fragmented packets. Unless there is some overriding reason for those packets to be fragmented, it is recommended the system be checked.

It is also strongly recommended that the version of Snort being using be upgraded as soon as possible!

SNMP public access

These alerts were triggered when an external host (134.192.79.87) tried to access the 'public' SNMP community on MY.NET.190.13. There's no indication as to whether the connection was successful or not.

Unless there's a reason for allowing remote monitoring of SNMP clients, SNMP should be blocked at the border. Also, the SNMP server should be checked to verify the default 'public' and 'private' communities are not being used.

SMB C access

These alerts show attempted access to target machines C: drive. If successful, the attacker will be able to read , and possible write(!), files on that drive. These packets can represent potential reconnaissance by an attacker and, if access is successful, represent a high-severity threat.

The quantity and duration of alerts from individual hosts is very low, amounting to at most an access attempt and a retry from any host to any other host, so I'm rating the

threat here as low. There doesn't appear to be any sustained effort to compromise a host here. It is highly recommended that all SMB-related traffic be blocked at the border.

IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize

This signature captures outbound attacks by hosts infected with the Code Red virus (or one of its variants). Two internal host , MY.NET.97.51 and MY.NET.97.225, are presently infected and actively attacking remote hosts. This machine needs to be isolated and disinfected immediately. It also needs to be patched with the latest patches, or have IIS removed, if it isn't needed.

NIMDA - Attempt to execute cmd from campus host

I'm unable to determine the rule which triggers this alert, but it's apparent that the source machines are likely infected with the NIMDA[43] virus and are actively seeking IIS servers to exploit.

These nine internal hosts need to be examined and disinfected immediately.

TFTP - Internal TCP connection to external tftp server

The vast majority of these were caused by internal host MY.NET.97.217 which had conversation with port 69/tcp on three remote machines. It's recommended that this machine, as well as the other four machines that used this port, be checked immediately to see if it's infected, or if it's controlling a BackGate virus on the remote machines.

Port 69 should be block, both inbound and outbound, at the firewall.

IRC evil - running XDCC

[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.

These alerts were caused by two internal hosts (MY.NET.74.216 and MY.NET.198.221) contacting two remote hosts on port 6667 from port 1026. This port is used for IRC, which fits because X-DCC[44] is an IRC-based file server, frequently used for "warez". The "evil" alerts were probably fired by the string ``:Total Offered:" in the payload, while the second appears to indicate the local users are downloading files.

Without knowing the University's policies on XDCC and IRC, it's not possible to assign a severity to these alerts.

FTP DoS ftpd globbing

This rule is no longer included in Snort, but it appears to have checked for an attack against the Washington University FTP server (WU-FTPD) which could give the attacker complete access to the attacked server, with whatever privileges the wu-ftpd server was running at (usually root). The attack took advantage of wu-ftpd's handling of wildcard characters, "globbing" being the term used for expanding and matching wildcard strings. The server failed to handle the strings "~{" and "~[" and could crash if the appropriate string was sent[45].

The server MY.NET.24.27 should be checked to make certain it isn't running wuftpd, or that it's running a properly patched version. Given the age of this vulnerability, I've rating this a low priority.

FTP passwd attempt

These alerts are triggered when a user attempts to download the a file with the string ``passwd'' in its name from our ftp server[46]. If there's no such file under the publicly-available ftp directories on the server in question, these are likely attempts to pull down the system's password file. Once retrieved, any number of tools can be used to crack passwords in the file (assuming passwords are stored in the file).

Most modern ftp servers, at least on UNIX-based platforms, can be run in a chroot'ed envorinment, and may not even require a password file in the chrooted directory structure.

Correlations:

The proftpd documentation[47] shows an example of setting up an ftp server in a chrooted environment. OpenBSD can chroot ftp connections for users by listing them in the /etc/ftpchroot file. A /etc/passwd file must present for anonymous ftp, but it is used only for user id in the 'ls' command, not for resolving passwords.

RFB - Possible WinVNC - 010708-1

These alerts are all for connections to or from port 5900. That is normally associated with VNC, or Virtual Network Computing, an AT&T research project that allows remote control over a machine[48]. Older versions had no support for security and so posed a considerable security risk.

The local machines should be checked for a VNC server running on port 5900 and reconfigured if necessary, asVNC can now run securely over ssh and/or IPSec. Access to VNC-server related ports (5800-5809,5900-5909) can be blocked at the firewall to

eliminate remote access to these servers.

[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC

This alert appears to check for hosts trying to access machines infected by the sdbot trojan[49]. This trojan, once installed, operates over IRC, and allows the attacker compete control of the infected machine. Multiple internal machines are infected with this trojan and are being actively exploited to attack remote systems. All the local machines listed in these alerts need to be immediately checked, isolated and disinfected.

NETBIOS NT NULL session

This is a simple reconnaisance. By initiating a Netbios session using a blank (NULL) name and password, it's possible to get a list of the available shared drives and user names[50]. This is the way Windows browses the "Network Neighborhood".

This can be disabled in NT, 2000 and XP by setting the registry key /System/ CurrentControlSet/Control/LSA/RestrictAnonymous to 1. Better still is to block port 139 at the firewall and not allow this traffic into or out of the network.

TCP SMTP Source Port traffic

This is really an informational message flagging inbound traffic to point 25 (presumably on non-smtp hosts, since there are only 15 alerts).

I've rated this a High priority not for this alert, but for the two local systems -- MY.NET.12.6 and MY.NET.25.68 -- that triggered it. Both have been the subject of multiple Queso scans and Red Worm alerts and should be check immediately.

TFTP - Internal UDP connection to external tftp server

At first glance, these are internal hosts that are trying to access a remote tftp server, as the alert implies. Also included, however, is external tftp servers connecting to local hosts. Additionally, one of the remote servers, 12.129.72.202, figures prominently in several other alerts -- Attempted Sun RPC high port access, possible Red Worm, and External tftp accesses -- leading me to conclude more is going on here.

Port 69, both inbound and outbound, should be blocked at the firewall. Additionally, the internal hosts targeted by this alert (MY.NET.84.145 in particular) should be immediately examined for possible infection.

PHF attempt

This item is no longer flagged by Snort, which now contains two different PHF alerts.

Presumably this matches on of them. The first of these exploited a buffer overflow in a CGI command to gain control of the remote server[51], while the second allows for remote command execution through the use of meta-characters[52]. Either attack will give the attacker control of the system in question.

One remote system is responsible for all but two of these alerts, and five of the six machines it contacted had subsequently contacted that same remote machine from port 69/tcp. There is an excellent chance these machines have been compromised. They should be examined immediately.

EXPLOIT NTPDX buffer overflow

These alerts indicate attempts to trigger a buffer overflow in the network time daemon ntpd[53]. If successful, the attacker gains root access to the machine question. The alerts were caused by four external systems targeting four internal machines.

At least one host, MY.NET.97.175, appears to be infected and has been used to launch numerous IIS and Code Red attacks. This system need to be investigated immediately.

Traffic from port 53 to port 123

These alerts were cause by two remote systems sending 5 packets to MY.NET.1.3, two from port 123 to port 53 and three from port 53 to port 123. Port 53 is used for DNS while port 123 is used for NTP. I do not understand the significance of these alerts.

[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan

I believe this is similar to this rule I found on the web[54]:

```
alert tcp $EXTERNAL_NET 6660:7000 -> $HOME_NET any\
(content: " 465 "; \
msg: "K\:line'd user detected, possible trojan.";\
classtype:misc-activity;)
```

This seems to imply that the recipient of the packet had been intentionally rejected by the IRC server. The name come from the IRC server configuration file where it's possible to ban a user/host by listing them on a line beginning with K:.

In this case, all five of the local systems -- MY.NET.97.53, 153.76, 76.116, 97.222 and 97.162) appear to be infected, as all have launched IIS Unicode attacks against remote machines. All of these hosts need to be isolated and disinfected immediately.

Back Orifice

These alerts were cause by a single remote system contacting port 31337 on two local system. That port is used to control the Back Orifice software, essentially giving the remote user control over the local PC. The same remote host also tried a "Red Worm" connection to one of the hosts, in addition to a SYN-FIN scan, so it's definltely hostile.

The two local systems -- MY.NET.152.167 and MY.NET.69.169 -- should be checked for infections. Port 31337 may be blocked at the border for some additional protection, though it might occasionally interfere with valid traffic.

ICMP SRC and DST outside network

Two hosts sent a total of three packets to three different systems. None of the source or destination addresses appeared in any other alerts or scans. This may have been due to a user plugging a laptop into the local network while it was configured for a remote network.

These packets should be automatically blocked at the border router (both inbound and outbound).

[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC

These alerts were caused by two internal hosts -- MY.NET.97.119 and MY.NET.97.231 -- sending IRC requests to 215.152.66.71. The alert itself is difficult to classify since it's particular to the University and the IRC policy is unknown. However, both internal hosts are the source of several IIS Unicode and CGI Null Byte alerts, so I'm rating this as High. These hosts should be checked immediately.

DDOS mstream client to handler

These alerts indicate the hosts in question have been compromised and are infected with the mstream DDoS handler[55]. These handlers are used by remote clients to launch attacks against remote targets. Two internal systems -- MY.NET.97.82 and MY.NET.60.11 -- appear to be infected and are actively attacking remote machines. These hosts need to be isolated and disinfected immediately.

DDOS shaft client to handler

The DDOS shaft client[56] communicates to its handler via port 20432, so any traffic to that port will trigger this event. Since this is a valid port for normal use, this alert can lead to a lot of false-positives, as in this case. These were triggered by conversations with a web-server and an smtp server and do not appear to be hostile.

Fragmentation Overflow Attack

This alert is presumably created by the Snort defrag filter, presumably because a reassembled packet was too large to be valid. It's difficult to tell, since this filter is no longer included with Snort. However, since the source host in question send two NULL scan packets and two other scan packets (SYN-PUSH-URG) to the same destination (MY.NET.150.41), subsequent to this packet, it's likely the intent was hostile. All packets arrived in a 19-minute period.

The firewall should be configure to reassemble fragments before allowing them to pass. That would stop this in its tracks.

NIMDA - Attempt to execute root from campus host

Host MY.NET.97.225 attempted to use one of NIMDA's attacks against a remote host. This particular host is definitely infected (it's responsible for 127 other alerts, including NIMDA, CGI NULL Byte attacks, ISAPI Overflows and others). This host should be isolated and disinfected immediately.

[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot

Both of these alerts are due to a an incoming packet (one each) from port 6667 to a single host (MY.NET.53.208). That host is involved in numerous other alerts, including "IRC usr /kill detected" and "Red Worm" and should be assumed compromised and examined immediately.

MY.NET.30.4 activity
MY.NET.30.3 activity

There purpose of these in unknown. The majority of the traffic headed to these addresses is destined for ports 80 (http) and 524 (Novell directory server?).

CS WEBSERVER - external web traffic
CS WEBSERVER - external ftp traffic
Notify Brian B. 3.56 tcp
Notify Brian B. 3.54 tcp
External FTP to HelpDesk MY.NET.70.50
External POP to HelpDesk MY.NET.70.49
External FTP to HelpDesk MY.NET.70.49
External POP to HelpDesk MY.NET.70.50
External FTP to HelpDesk MY.NET.53.29

These are all apparantly informational in nature. More information regarding their purpose is required before they can be correctly classified.

3.4.2 Out-Of-Spec Packets

There were 31,180 packets in the out-of-spec files. These main differentiating characteristic for the majority of these packets appears to be the flag fields, as shown in the table below.

********	607	NULL scans - Matches logs
******SF	7645	SYN-FIN scan - Matches scan logs
****P***	218	(Kazaa: 212)(Gnutella: 6)
12****S*	22612	
12***R**	52	
12UAPRSF	3	

Table 5: Out-Of-Spec Packet Breakdown

In addition to the above, packets with the following flag settings appeared only once or twice during the period in question, for a cumulative total of 44 packets :

```
***A*RSF, **U*****, **UA**SF, **UAPRSF, *2****SF, *2**P*SF, *2U*P*SF,
1*****SF, 1****RSF, 1**A*RSF, 1*U**RSF, 12****S*, 12***R**, 12***RS*,
12**P*SF, 12**PR*F, 12**PRS*, 12*A*R*F, 12*AP*S*, 12*APRS*, 12U**R*F,
12U**RS*, 12U**RSF, 12U*P**F, 12U*P*S*, 12U*P*SF, 12U*PR**, 12U*PR*F,
12U*PRSF, 12UA****, 12UA**SF, 12UA*RS*, 12UAP***, 12UAP*S*
```

These all came from random source to random destination, so no overriding purpose can be attributed to them. About 25% were involved in apparant Kazaa, EDonkey and/ or Napster usage.

Many of the remaining records are the result of scans against the internal systems and are also recorded in the alert and scan logs. For example, host 66.82.245.45 launch a broad SYN-FIN scan of the local network, yielding entries in all three types of logs:

From Alerts:

```
07/28-15:56:03.012314  [**] SYN-FIN scan! [**] 66.82.245.45:21 -> MY.NET.199.170:21
07/28-15:56:03.063396  [**] SYN-FIN scan! [**] 66.82.245.45:21 -> MY.NET.199.171:21
07/28-15:56:03.063406  [**] SYN-FIN scan! [**] 66.82.245.45:21 -> MY.NET.199.172:21
```

From Scans:

```
07/28-15:56:03 [**] SCAN SYNFIN ******SF  [**] 66.82.245.45:21 -> 130.85.199.170:21
07/28-15:56:03 [**] SCAN SYNFIN ******SF  [**] 66.82.245.45:21 -> 130.85.199.171:21
```

```
07/28-15:56:03 [**] SCAN SYNFIN ******SF  [**] 66.82.245.45:21 -> 130.85.199.172:21
```

From OOS:

```
07/28-15:56:03.012311 66.82.245.45:21 -> MY.NET.199.170:21
TCP TTL:31 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x5E65D991  Ack: 0xA5D7970  Win: 0x404  TcpLen: 20
```

The advantage of recording this information three separate times in three separate locations escapes me.

Only five of these types of packets rise above the level of noise: ********, ******SF, ****P***, and 12****S*. The other 44 packets can be attributed to garbled communications. Given the quantity of data during the period, I'm not overly worried about a 3-packet scan.

The first two types of packets of interest -- ******** and ******SF -- are due to remote scans (NULL and SYN-FIN, respectively) of the internal hosts. This is born out by confirmations in the scan and alert logs.

The ****P*** packets are a characteristic of the Kazaa and Gnutella file-sharing networks, and if fact, the payloads of those packets confirm this.

The last type of interest -- 12****S* -- represent the bulk of the OOS packets. An analysis of these packets show them to or from normal service ports (smtp, pop, http) as well as some lesser-known ports (edonkey). None appear to be part of a systematic scan. The alert logs flag many, but not all, of the 12****S* packets as a "QUESO scan". I believe that to be incorrect.

The first two flags (cleverly labeled 1 and 2) used to be reserved for future use and were always supposed to be set to zero. That changed several years ago with the advent of ECN. ECN uses those two bits for Explicit Congestion Control, allowing hosts to use those two bits, now labeld ECE (ECN-Echo) and CWR (Congestion Window Reduced), to signal the other when one host is unable to keep up.

Correlations:
RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP[57]
Pete Storm's GCIA Practical Assignment V3.3[16]
Donald Gregory's GCIA Practical Assignment V3.2[34] (p. 44)

I was able to confirm my ECN suppositions using both Donald's [p. 44] and Pete's [p.

61] works, as well as RFC3168.

3.4.3 Scans

The Scan table lists the top 11 scans found in the scan logs. There were a further 160 different scan types recorded, but the averaged less than 4 packets per type, and so have been ignored for this analysis.

Type	Quantity	Sources	Comment
UDP	6,055,805	505	
SYN ******S*	2,497,910	700	
SYN 12****S* RESERVEDBITS	11,708	350	ECN (see OOS section)
SYNFIN ******SF	2,558	2	from 66.82.245.45 port 21 to port 21
NULL ********	679	46	
INVALIDACK ***A*R*F	217	54	
VECNA ****P***	153	8	
UNKNOWN 1****R** RESERVEDBITS	115	36	All but five from known services (http, imap, etc.) . ECN
INVALIDACK ***AP*S*	105	5	
UNKNOWN *2*A**** RESERVEDBITS	66	3	All from a web server. ECN
UNKNOWN *2***R** RESERVEDBITS	64	5	All from or to known services. ECN

Table 6: Scan Breakdown

These logs, when combined with the OOS logs, raise significant questions about the accuracy of one or both sets of logs. Several scans using invalid flag settings -- 1****R**, ***A*R*F, *2*A**** and *2***R** -- don't appear in the OOS logs at all.

All but one of the SYNFIN scans came from a single host, 66.82.245.45, all to port 21. That host also performed a small (18-packet) SYN scan also to port 21, but to different targets. These scans all occured on the afternoon of the last day of logs, so it's probably worth adding new rules to track activity from this host for the near future.

The "***AP*S*" are all from web or imap servers to low ephemeral ports (1025-1036) on 4 machines (3 remote). The targeted system don't appear in any other alerts, and no known services normally run on those ports. It's possible these are actual responses to invalid SYN packets sent out "*****P*S*", but no such packets were logged. The local

webserver sending many of these packets, MY.NET.113.207, was previously the target of several IIS ISAPI attacks, so it would be worthwhile to examine that system and check for a compromise.

The "***A*R*F" packets are an unknown, but appear harmless. Many come from ident servers, and all are destined for ephemeral ports. There doesn't appear to be any pattern.

The majority of the NULL scans ("********") were targeted against the pop3 port of MY.NET.12.4, MY.NET.25.21, MY.NET.25.22, MY.NET.25.23 and MY.NET.25.24. This matches the data from the OOS and alert logs. All of these systems were also mentioned in Red Worm alerts and should be checked.

The SYN scans (******S*) were all over the map. Three internal systems topped the list (MY.NET.97.51, MY.NET.97.225 and MY.NET.100.230) with more that 1/2 million combined alerts. That points out a major problem with these alerts in general. Short of saying someone needs to look at these systems, the sheer quantity of alerts becomes quickly overwhelming. The first two systems here are already high on the "must be checked" list with NIMDA, so no real knowledge is here.

Nearly 1/2 of the UDP scans were originated my MY.NET.1.3, and 80% of those where targeted at apparent DNS servers (port 53).

3.5 Top Talkers

My initial take on this was to simply run counts of alerts and scans from various systems and record the totals here. While easy to obtain (a simple perl script can generate the appropriate source and target breakdowns for alerts, scans and OOS packets), the results were neither illuminating nor particularly interesting.

In the end, I took a different tack: My definition for "top talkers" are those internal systems involved in the most high-priority attacks against remote machines. It's important to note that this definition doesn't simply rely on alert/OOS/scan counts, but rather looks at both the frequency and types of attacks being launched by the systems in question. This admittedly weights the scale towards high-priority alerts over OOS and/or scan records, but my belief here is that a system actively launching 100 IIS attacks is far more important than a system sending 1,000 NMap scan packets. The scans are important, but the attacks are critical and demand immediate attention.

That said, MY.NET.1.3 made the list due solely to the sheer number of scan packets it

sent out (nearly 2.5 million). If this is your DNS server, do you really want to wade through 2.5 million alerts?

MY.NET.97.51	189,967	SYN scan
	123,270	UDP scan
	56	NIMDA - Attempt to execute cmd from campus host
	54	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
MY.NET.97.225	192,138	SYN scan
	77	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
	46	NIMDA - Attempt to execute cmd from campus host
	4	spp_http_decode: CGI Null Byte attack detected
	1	UDP scan
MY.NET.97.49	176,065	UDP scan
	1,173	SYN scan
	213	spp_http_decode: IIS Unicode attack detected
	124	spp_http_decode: CGI Null Byte attack detected
MY.NET.70.207	76,719	UDP scan
	18,492	High port 65535 udp - possible Red Worm - traffic
	2	TFTP - External TCP connection to internal tftp server
	1	NIMDA - Attempt to execute cmd from campus host
MY.NET.97.88	58,365	UDP scan
	412	SYN scan
	345	spp_http_decode: IIS Unicode attack detected
	7	spp_http_decode: CGI Null Byte attack detected
MY.NET.97.79	45,281	UDP scan
	917	SYN scan
	193	spp_http_decode: IIS Unicode attack detected
	26	spp_http_decode: CGI Null Byte attack detected
MY.NET.97.83	128,324	UDP scan
	716	SYN scan
	24	spp_http_decode: IIS Unicode attack detected
	1	spp_http_decode: CGI Null Byte attack detected
MY.NET.97.77	95,410	UDP scan
	584	SYN scan
	169	spp_http_decode: IIS Unicode attack detected
	4	spp_http_decode: CGI Null Byte attack detected
MY.NET.97.21	76.473	UDP scan
	528	SYN scan
	123	spp_http_decode: IIS Unicode attack detected

MY.NET.1.3	2,469,977 UDP Scans 98 SYN Scans

Table 7: Top Talkers

3.6 External Sources

The five external sources examined here all did something to stand out in the provided logs.

3.6.1 216.152.64.155

This host has compromised and is actively controlling several internal machines, as detailed in the linkgraph section below. This host is only directly involved in two alerts, "IRC user /kill detected" as source, and "Possible sdbot floodnet" as destination, but it's indirectly responsible for numerous others, including "Red Worm", "IIS Unicode" and NIMDA alerts, through hosts it's controlling.

```
whois 216.152.64.155

OrgName:    WebMaster, Incorporated
OrgID:      WBMR
Address:    1601 Civic Center Drive, Suite 101
City:       Santa Clara
StateProv:  CA
PostalCode: 95050
Country:    US

NetRange:   216.152.64.0 - 216.152.79.255
CIDR:       216.152.64.0/20
NetName:    WEBMASTER-BLK-1
NetHandle:  NET-216-152-64-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.WEBMASTER.COM
NameServer: NS1.WEBCHAT.ORG
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-07-18
Updated:    2003-09-05

TechHandle: MO21-ARIN
TechName:   Owen, Mark
TechPhone:  +1-408-345-1800
TechEmail:  mark@webmaster.com
```

```
# ARIN WHOIS database, last updated 2004-02-17 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

3.6.2 68.155.195.92

This is the remote host that generated the most scan records (116,889) over the period in question. This is apparently a BellSouth user.

```
whois 68.155.195.92

OrgName:    BellSouth.net Inc.
OrgID:      BELL
Address:    575 Morosgo Drive
City:       Atlanta
StateProv:  GA
PostalCode: 30324
Country:    US

ReferralServer: rwhois://rwhois.eng.bellsouth.net:4321

NetRange:   68.152.0.0 - 68.159.255.255
CIDR:       68.152.0.0/13
NetName:    BELLSNET-BLK14
NetHandle:  NET-68-152-0-0-1
Parent:     NET-68-0-0-0-0
NetType:    Direct Allocation
NameServer: NS.BELLSOUTH.NET
NameServer: NS.ATL.BELLSOUTH.NET
Comment:
Comment:    For Abuse Issues, email abuse@bellsouth.net. NO ATTACHMENTS. Include IP
Comment:    address, time/date, message header, and attack logs.
Comment:    For Subpoena Request, email ipoperations@bellsouth.net with "SUBPOENA" in
Comment:    the subject line. Law Enforcement Agencies ONLY, please.
RegDate:    2002-07-22
Updated:    2003-05-05

AbuseHandle: ABUSE81-ARIN
AbuseName:   Abuse Group
AbusePhone:  +1-404-499-5224
AbuseEmail:  abuse@bellsouth.net

TechHandle: JG726-ARIN
TechName:   Geurin, Joe
TechPhone:  +1-404-499-5240
TechEmail:  ipoperations@bellsouth.net
```

```
OrgAbuseHandle: ABUSE81-ARIN
OrgAbuseName:   Abuse Group
OrgAbusePhone:  +1-404-499-5224
OrgAbuseEmail:  abuse@bellsouth.net

OrgTechHandle: JG726-ARIN
OrgTechName:   Geurin, Joe
OrgTechPhone:  +1-404-499-5240
OrgTechEmail:  ipoperations@bellsouth.net

# ARIN WHOIS database, last updated 2004-02-17 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

3.6.3 200.51.212.201

This is the remote host that generated the most Out-Of-Spec records (342) over the period in question. This user appears to be a dial-up user in Buenos Aires, Argentina. This was all traffice to eDonkey file-sharing servers, so not necessarily harmful, but prevalent none-the-less.

```
whois 200.51.212.201

OrgName:    Latin American and Caribbean IP address Regional Registry
OrgID:      LACNIC
Address:    Potosi 1517
City:       Montevideo
StateProv:
PostalCode: 11500
Country:    UY

ReferralServer: whois://whois.lacnic.net

NetRange:   200.0.0.0 - 200.255.255.255
CIDR:       200.0.0.0/8
NetName:    LACNIC-200
NetHandle:  NET-200-0-0-0-1
Parent:
NetType:    Allocated to LACNIC
NameServer: TINNIE.ARIN.NET
NameServer: NS.LACNIC.ORG
NameServer: NS.DNS.BR
NameServer: NS2.DNS.BR
```

```
Comment:    This IP address range is under LACNIC responsibility for further
Comment:    allocations to users in LACNIC region.
Comment:    Please see http://www.lacnic.net/ for further details, or check the
Comment:    WHOIS server located at whois.lacnic.net
RegDate:    2002-07-27
Updated:    2003-06-12

TechHandle: LACNIC-ARIN
TechName:   LACNIC Hostmaster
TechPhone:  (+55) 11 5509-3522
TechEmail:  abuse@lacnic.net

OrgTechHandle: LACNIC-ARIN
OrgTechName:   LACNIC Hostmaster
OrgTechPhone:  (+55) 11 5509-3522
OrgTechEmail:  abuse@lacnic.net

# ARIN WHOIS database, last updated 2004-02-17 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.

% Copyright LACNIC lacnic.net
%  The data below is provided for information purposes
%  and to assist persons in obtaining information about or
%  related to AS and IP numbers registrations
%  By submitting a whois query, you agree to use this data
%  only for lawful purposes.
%  2004-02-18 19:38:30 (BRT -03:00)

inetnum:     200.51.212/22
status:      reallocated
owner:       Telefonica de Argentina
ownerid:     AR-TEAR7-LACNIC
responsible: Marcelo A. MuÒoz
address:     Defensa, 390, Piso 5
address:     1065 - Buenos Aires - CF
country:     AR
phone:       +54 11 4-3335509 []
owner-c:     TEA
tech-c:      TEA
created:     20030916
changed:     20030916
inetnum-up:  200.51.208/21
inetnum-up:  200.51/16

nic-hdl:     TEA
person:      TELEFONICA DE ARGENTINA
e-mail:      tasamail@TELEFONICA.COM.AR
address:     H. Yrigoyen 1556 - 8th floor, 1556,
```

```
address:     1089 - Capital Federal - BA
country:     AR
phone:       +54 11 4332-2364 []
created:     20030618
changed:     20030915

% whois.lacnic.net accepts only direct match queries.
% Types of queries are: POCs, ownerid, CIDR blocks, IP
% and AS numbers.
# ARIN WHOIS database, last updated 2004-02-17 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

3.6.4 199.29.143.28

This is the remote host that generated the most alert records (173,208) over the period in question, all of type "IRC user /kill detected".

```
whois 199.29.143.28
Performance Systems International Inc. NETBLK-PSINET-CBLK4 (NET-199-29-0-0-1)
                                199.29.0.0 - 199.29.255.255
L.H. Rosenberg & Associates NET-LHRASSO (NET-199-29-143-0-1)
                                199.29.143.0 - 199.29.143.255
```

This isn't overly helpful. A web search identifies L.H Rosenberg as a Leonard Rosenberg of www.lhrasso.com. That address is no longer valid, so this will take a call to PSI to get further information.

3.6.5 216.231.173.71

In light of the lack of success in finding information on the prior address, this is the remote host that generated the second most alert records (24,476) over the period in question, all of type "possible Red Worm".

```
whois 216.231.173.71

OrgName:    Gulf Telephone Company
OrgID:      GTCO
Address:    316 South McKenzie St
City:       Foley
StateProv:  AL
PostalCode: 36535
Country:    US
```

```
NetRange:   216.231.160.0 - 216.231.191.255
CIDR:       216.231.160.0/19
NetName:    GTCO
NetHandle:  NET-216-231-160-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: NS.GULFTEL.COM
NameServer: NS2.GULFTEL.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-06-22
Updated:    2002-10-30

TechHandle: ZM158-ARIN
TechName:   Madison River Communications
TechPhone:  +1-919-563-1500
TechEmail:  hostmaster@madisonriver.net

# ARIN WHOIS database, last updated 2004-02-17 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

3.7 Link Graph and Analysis

The following link graph clearly demonstrates the havoc a remote hacker can cause. Seven systems on the MY.NET.97 subnet -- 35, 55, 59, 73, 107, 160, and 161 -- have been compromised and are all being remotely controlled by the machine at 216.152.64.155.

The infected hosts all appear to be running the sdbot trojan, and are communicating with the master using IRC, as show in the following alerts:

```
07/26-13:32:38.020867  [**] [UMBC NIDS IRC Alert] Possible sdbot floodnet detected
attempting to IRC [**] MY.NET.97.161:1332 -> 216.152.64.155:6660
07/26-14:20:28.068166  [**] [UMBC NIDS IRC Alert] IRC user /kill detected, possible
trojan. [**] 216.152.64.155:6660 -> MY.NET.97.161:1332
```

The master has complete control over the infected machines, including the ability to download and run executables. As shown in the diagram, the infected machines have been used to launch NIMDA and IIS Unicode attacks, as well as performing hundreds of scans on remote machines. In total, these machines sent out 4,991 SYN scan packets and 13,416 scan packets in the 5-day period in question (the scans aren't represented on the linkgraph for clarity).

The WinVNC does not appear to be the result of actions by the controlling host, but is included for completeness.

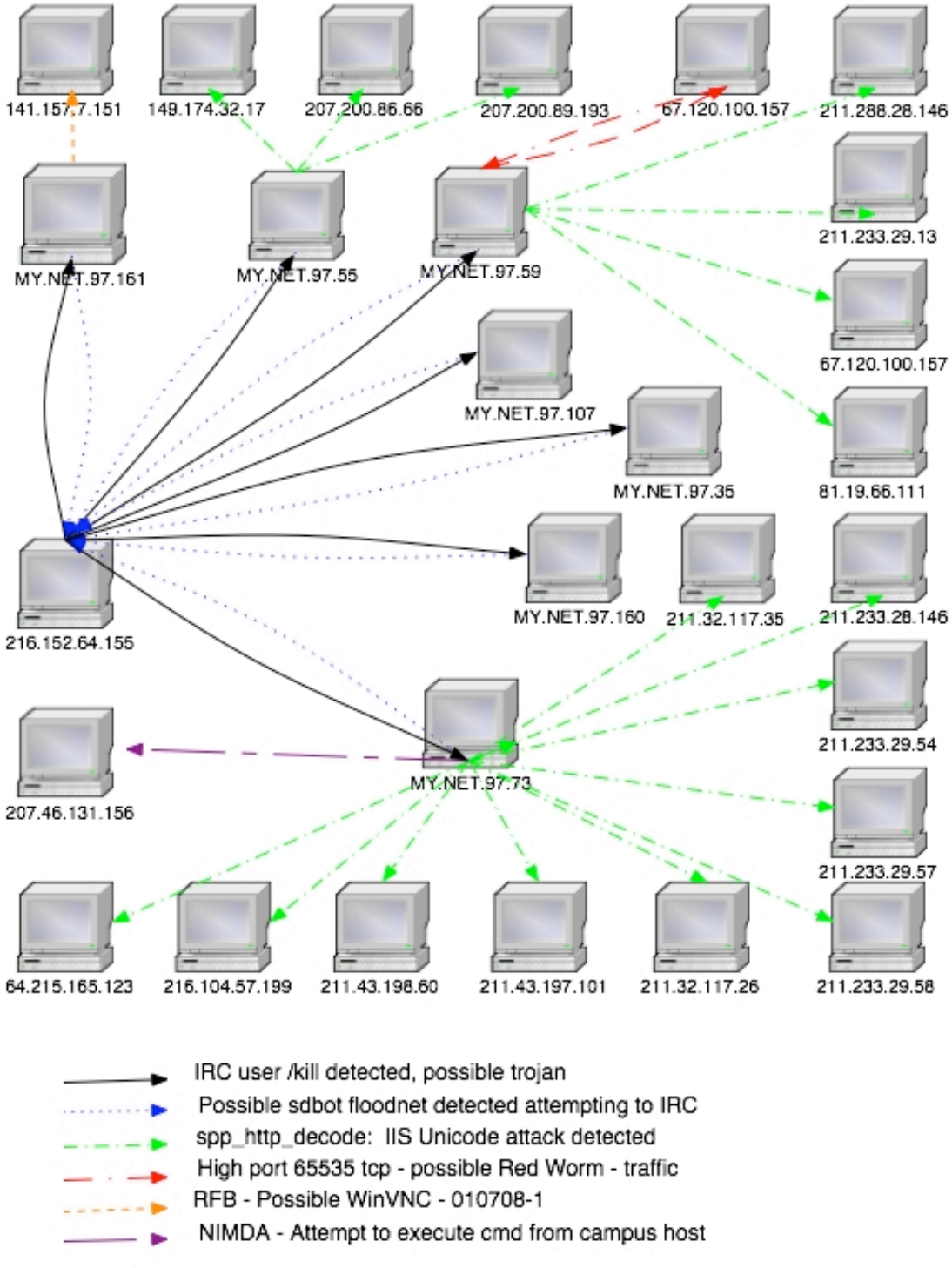

Table 8: Linkgraph of Compromised Systems

3.8 Insights

The following systems are known to be infected and need to be dealt with immediately.

Disinfect

The following hosts are either known to be, or have strongly indicated that they are, infected by one or more viruses. Many are actively under control of remote systems. These should be your top priority.

MY.NET.53.208	IRC /kill, XDCC, Warez, Red Worm
MY.NET.60.11	DDOS mstream client to handler
MY.NET.70.41	PHF Attempt, TFTP - External TCP connection
MY.NET.70.118	PHF Attempt, TFTP - External TCP connection
MY.NET.70.172	PHF Attempt, TFTP - External TCP connection
MY.NET.70.191	PHF Attempt, TFTP - External TCP connection
MY.NET.70.203	PHF Attempt, TFTP - External TCP connection
MY.NET.70.207	NIMDA, Red Worm
MY.NET.82.2	NIMDA
MY.NET.82.26	NIMDA
MY.NET.82.53	NIMDA
MY.NET.97.11	sdbot
MY.NET.97.29	NIMDA
MY.NET.97.35	sdbot
MY.NET.97.51	IIS ISAPI Overflow ida (Code Red), NIMDA
MY.NET.97.53	K:\line'd user, IIS Unicode
MY.NET.97.55	sdbot
MY.NET.97.59	sdbot, IIS Unicode, Red Worm
MY.NET.97.64	sdbot
MY.NET.97.73	sdbot, IIS Unicode, NIMDA
MY.NET.97.82	DDOS mstream client to handler
MY.NET.97.107	sdbot
MY.NET.97.116	K:\line'd user, IIS Unicode
MY.NET.97.160	sdbot
MY.NET.97.161	sdbot

MY.NET.97.162	sdbot
MY.NET.97.162	K:\line'd user, IIS Unicode
MY.NET.97.175	EXPLOIT NTPDX buffer overflow, Code Red
MY.NET.97.183	sdbot
MY.NET.97.188	sdbot
MY.NET.97.219	sdbot
MY.NET.97.222	K:\line'd user, IIS Unicode
MY.NET.97.225	IIS ISAPI Overflow ida (Code Red), NIMDA
MY.NET.97.227	sdbot
MY.NET.114.89	Red Worm
MY.NET.132.42	NIMDA
MY.NET.153.76	K:\line'd user, IIS Unicode

Table 9: Systems to be Disinfected

Check for infection

The next table is a short summary of hosts that need to be checked for infection. The table is followed by a much larger list of hosts that need to be checked. Rather than listing those hundreds of hosts individually, I've tried to note how to identify them in the logs.

MY.NET.5.92	TFTP - Internal UDP connection
MY.NET.12.4	target of NULL scan and Red Worm alerts
MY.NET.12.6	Queso, Red Worm, TCP SMTP Source Port traffic
MY.NET.24.27	FTP DoS ftpd globbing
MY.NET.25.21	target of NULL scan and Red Worm alerts
MY.NET.25.22	target of NULL scan and Red Worm alerts
MY.NET.25.23	target of NULL scan and Red Worm alerts
MY.NET.25.24	target of NULL scan and Red Worm alerts
MY.NET.25.68	Queso, Red Worm, TCP SMTP Source Port traffic
MY.NET.69.169	Back Orifice
MY.NET.70.125	TFTP - Internal UDP connection
MY.NET.83.98	Sending fragmented packets
MY.NET.84.145	TFTP - Internal UDP connection
MY.NET.97.20	TFTP - Internal TCP connection

MY.NET.97.91	Sending NULL scans, sending fragments
MY.NET.97.217	TFTP - Internal TCP connection
MY.NET.113.207	subjected to "IIS ISAPI" attacks and is now sending packets with invalid flags ("***AP*S*").
MY.NET.152.167	Back Orifice
MY.NET.190.13	SNMP public access

Table 10: Systems to be checked

69 hosts on the MY.NET.70 and MY.NET.71 networks answered a scan that came in on port 69. These hosts need to be checked to verify they're not infected and are running a valid, required tftp server.

37 hosts responded to a scan on port 27274 from 66.68.110.48. These need to be checked for SubSeven, Ramen and/or other infection.

53 hosts recieved "IRC user /kill" commands and need to be checked.

98 hosts launched CGI Null Byte attacks and need to be checked.

344 systems were detected sending IIS Unicode attacks and need to be checked.

Systems to Watch

66.82.245.45 performed extensive scan for ftp servers

3.9 Defensive Recommendation

I've several recommendations for reducing risk and lowering the cost of doing normal business for the University and its administrators. I've divided those recommendations below, based on their intended audience. There is, however, considerable overlap between them.

Recommendations for the University

- Consider a policy allowing the administrators to immediately quarantine infected machines, removing them from the network until such time as they are deemed safe. This is the fastest way to halt the spread of a new virus.

- Look into providing reduced-cost subscriptions to current anti-virus software, and

require such software on all internal machines.

Recommendations for Local Users:

- Use current anti-virus software on all machines. The University might be able to acquire such software at a much reduced cost by negotiating with vendors on behalf of the students.

- Use personal firewalls, if available (they should be for most systems). This won't replace the need for a border firewall, but it will help reduce the damage done when/if a user's machine is infected.

Recommendations for the border firewall:

- Drop all mis-configured packets. Dropping a SYN-FIN scan at the firewall, for example, will not only help eliminate these types of scans, but will also significantly reduce the logged traffic. As it stands, these scans generate an incredible amount of noise that the administrators need to weed through.

- Block truly unneeded services. TFTP, for example, can be safely blocked at the firewall without affecting any legitimate traffic.

- Provide local sources for standard types of traffic, where possible, then block those ports for any other sources. For example, providing a local ntp server for your users allows you to block the ntp ports for any other hosts.

- Block SMB traffic at the firewall. This might be an inconvenience to the local users, but allowing external access to these ports is simply asking for trouble. Users should be encourage to use alternate methods to share files (personal space on University-supplied web server, ssh, etc.).

- Investigate the use of transparent proxies for services that support them. An http proxy could easily stop known IIS attacks cold.

- Require the use of current anti-virus software on all machines, where possible. Many of the incoming attacks are ancient (in Internet time), and would be readil

Recommendation for administrators:

- See if there is another way to gather the data provided by the "informational" alerts. For example, the second most frequent alert was for external traffic to the CS web server. Not only is that an incredible amount of chaff, it almost certainly duplicates data already collected in the web server logs.

- Update your version of Snort and its associated rules to the latest versions. Many of the alerts in the provided logs are no longer current.

- If MY.NET.1.3 is your DNS server, remove it from the logs by adding the following line to your snort.conf file:

```
preprocessor portscan-ignorehosts: MY.NET.1.3
```

This alone would prevent 2.5 million alerts.

3.10 Analysis Process

I began my analysis by first using a script to find and remove invalid records from the source files: these were mostly either incomplete or inter-mixed alerts that couldn't be correctly deciphered. This allowed me to concentrate future scripts on parsing the valid data without worrying about formatting errors.

I then normalized the alert, scan and OOS records into a common format (actually, the standard Alert format). Each OOS record spanned multiple lines, so for each record I output a single line that summarize the records basic information (date, time, flags, and source and destination hosts and ports). I also slightly reformatted the Scan records so that the date formats matched the Alert records, and alter the addresses so that 130.85 showed MY.NET. The resulting data could then be quickly sorted (using a standard ASCII sort) into a time-ordered view of the available data. This made it trivial to note the correlation between certain OOS records and a Queso scan, for instance.

I then wrote another script to summarize the standardized records by alert-type. This script yeilded counts by type, further sub-divided between inbound and outbound alerts. The script also kept track of the number of unique hosts for each alert.

Additionaly scripts were used to create and operate on subsets of the alert data, since processing the entire large collection was fairly time-consuming. Files and summaries were created for each alert type, greatly decreasing the time involved in further study.

At this point, I need to give special thanks to Glenn Larratt, whose GCIA Practical[58] reminded me of the necessity for generating attack timelines. I created a perl script that summarized alerts by type, day and time, then plotted histograms for each. That was invaluable in spotting trends that deserved special attention. The "Red Worm" alert, for example, yielded the following histogram. The three sets of numbers per day show the total number of alerts for the day, the number during business hours (presumed to be 8-5), and out-of-hours, along with their respective percentages of the total.

```
Alert: High port 65535 tcp - possible Red Worm - traffic:  10018
07/24 :   373 (03.72)    124 (01.24%)    249 (02.49%)  **
07/25 :  7416 (74.03)     79 (00.79%)   7337 (73.24%)  ************************************
07/26 :   892 (08.90)    120 (01.20%)    772 (07.71%)  *****
07/27 :   726 (07.25)    444 (04.43%)    282 (02.81%)  ****
07/28 :   611 (06.10)    318 (03.17%)    293 (02.92%)  ****

By Hour
      07/24 00:     60 (00.60%)%  *
      07/24 01:     96 (00.96%)%  *
      07/24 02:      3 (00.03%)%  *
      07/24 03:     27 (00.27%)%  *
...
      07/25 02:   7228 (72.15%)%  ************************************
...
```

This helped to highlight blocks of time requiring special attention during analysis.

I also based the layout of my Alert Summary table on Gary's, which I found to be among the clearest available. I did not, however, include port number in my table, as they didn't add significantly to the overall understanding of the alerts. I instead noted port numbers in the individual alert breakdowns, where appropriate.

One particularly interesting script I wrote looked for chains of events between the combined alerts. It began by looking for alert sources that had themselves been the destination of previous alerts. It recorded those alerts and began looking for subsequent alerts from those destinations as well. It was this script that allowed the creation of the linkgraph, as it cleared showed connections not easily seen in the mass of provided data.

Once I had defined my "top talkers" criteria, I generated another script which looked for alerts, OOS records and scans from internal hosts, and sorted the results by quantity. I then manually reviewed the results, choosing the first 10 I found to be "interesting."

Scan data were the most painful to deal with, of only for the sheer quantity of data to go through. Several perl scripts were used to generate overall summaries of the data (unique sources, frequency of destinations IPs, ports, etc.). That helped pinpoint that MY.NET.1.3 was probably a DNS server and Snort was merely misconfigured.

4. References

1. Savage. "Queso Source Code." http://www.l0t3k.net/tools/FingerPrinting/queso-980922.tar.gz
2. Fyodor. "NMap v3.48." (2002). URL: http://www.insecure.org/nmap
3. Jacobson, V., R. Braden, and D. Borman. "RFC1323: TCP Extensions for High Performance." (1992). http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1323.html 4. Richard, W. Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison Wesley Longman, Inc, 1994.
5. Fyodor. "Remote OS detection via TCP/IP Stack FingerPrinting." (2002). http://www.insecure.org/nmap/nmap-fingerprinting-article.html
6. Sakellariadis, Spyros. "Protecting Windows RPC traffic." (2002). http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/isa/maintain/rpcwisa.asp (Feb 11, 2004)
7. LURHQ. "Windows Messenger Popup Spam on UDP Port 1026." http://www.lurhq.com/popup_spam.html (2/11/03)
8. Ullrich, Johannes B. "popupad spam wrapup." (2003). http://www.merit.edu/mail.archives/nanog/2003-12/msg00181.html
9. Microsoft. "Buffer Overrun in Messenger Service Could Allow Code Execution (828035)." (2003). http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-043.asp (Feb 13, 2004)
10. Microsoft. "Managing System Services." http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winntas/maintain/monitor/03wntpcb.asp (Feb 11, 2004)
11. Microsoft. "Stopping Advertisements with Messenger Service Titles." (2003). http://www.microsoft.com/windows2000/techinfo/administration/communications/msgrspam.asp (Feb 13 2004)
12. Microsoft. "Disabling Messenger Service in Windows XP." (2004). http://www.microsoft.com/windowsxp/pro/using/howto/communicate/stopspam.asp
13. Mixter. ".mixter security home page." http://mixter.void.ru/code.html (Feb 13 2004)
14. PacketStormSecurity. "groups / mixter /." http://packetstormsecurity.org/groups/mixter/ (Feb 13, 2004)
15. whitehats.com. "IDS203 TROJAN-ACTIVE-Q-TCP." http://www.whitehats.com/info/IDS203 (Feb 13, 2004)
16. Storm, Pete. "GCIA Practical Assignment V3.3." (2003). http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf (Feb 11, 2004)
17. Gordon, Les. "What is the Q Trojan?" http://www.sans.org/resources/idfaq/qtrojan.php

18. IEEE. "IEEE OUI & Company ID Assignments." (2/1/04). http://standards.ieee.org/regauth/oui/index.shtml
19. IANA. "Internet Protocol V4 Address Space." (2002). http://www.aian.org/assignments/ipv4-address-space (IANA:IPV4)
20. IETF. "INTERNET PROTOCOL." (1981). http://www.ietf.org/rfc/rfc0791.txt
21. CERT. "CERT® Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL." (2002). http://www.cert.org/advisories/CA-2001-19.html (Feb 4, 2004)
22. Teeraruangchaisri, Kittipong. "Code and Code Red II: Double Dragons." (2001). http://www.sans.org/rr/papers/36/88.pdf (Feb 4, 2002)
23. Williams, Todd. "LOGS: GIAC GCIA Version 3.3 Practical - Todd Williams." (2003). http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00176.html (Feb 8, 2004)
24. Snort.org. "The Snort FAQ." (2/15/04). http://www.snort.org/docs/FAQ.txt (Feb 11, 2002)
25. Microsoft. "Q303215: Microsoft Network Security Hotfix Checker (Hfnetchk.exe) Tool Is Available." (2003). http://support.microsoft.com/default.aspx?kbid=303215 (Feb 11 2004)
26. Microsoft. "Q206460: Microsoft Baseline Security Analyzer (MBSA)." http://support.microsoft.com/default.aspx?scid=kb;en-us;306460 (Feb 11, 2004)
27. Anonymous. "IRC KILL command." (2004), http://www.irchelp.org/irchelp/ircii/commands/KILL
28. www.whitehats.com. "IDS177 NETBIOS-NAME-QUERY." (2003). http://www.whitehats.com/info/IDS177 (Feb 11, 2004)
29. MacDonald, Terry. "GCIA Practical Assignment V3.3." (2003). http://www.giac.org/practical/GCIA/Terry_MacDonald_GCIA.pdf
30. Microsoft. "Chapter 11 - Additional Member Server Hardening Procedures." (2/7/04). http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/topics/hardsys/TCG/TCGCH11.asp
31. Gordon, Les. "GCIA Practical Assignment V3.3." http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc (Feb 4, 2004)
32. J., Anthony Dell. "Adore Worm -- Another Mutation." (2001). http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf (Mar 2, 2002)
33. Rodriguez, Tom. "What are unicode vulnerabilities on Internet Information Server." (2001). http://www.sans.org/resources/idfaq/iis_unicode.php
34. Gregory, Donald. "GCIA Practical Assignment V3.2." (2003). http://www.giac.org/practical/GCIA/Donald_Gregory_GCIA.pdf (Feb 7, 2004)
35. rain.forest.puppy. "Perl CGI problems.", http://www.phrack.org/how.php?p=55\&a=7 (Feb 1, 2004)
36. Brodley, Carla, T.N. Vijaykumar, Hilmi Ozdoganoglu, and Ankit Jalote. "The Buffer Overflow Page." (2004). http://min.ecn.purdue.edu/~cyprian/BoF_Page.html#x86
37. CERT. "CA-2001-30 Multiple Vulnerabilities in lpd." http://www.cert.org/advisories/CA-2001-30.html (Feb 7, 2004)

38. Stearns, William. "Ramen Worm." (2/1/04). http://www.sans.org/y2k/ramen.htm
39. SANS. "Intrusion Detection FAQ: SubSeven Trojan." (2/1/04).
http://www.sans.org/resources/idfaq/subseven.php
40. Consulting, Simovits. "Trojan list sorted on trojan port." (2/1/04).
http://www.simovits.com/trojans/trojans.html (Feb 1, 2004)
41. www.whitehats.com. "IDS552 IIS ISAPI OVERFLOW IDA."
http://www.whitehats.com/info/IDS552 (Feb 4 2002)
42. Associates, Computer. "BackGate Kit." (2002):
http://www3.ca.com/virusinfo/virus.aspx?ID=9739 (Jan 12, 2004)
43. CERT. "CERT Advisory CA-2001-26: Nimda."
http://www.cert.org/advisories/CA-2001-26.html (Oct 11, 2002)
44. Dittrich, Dave. "World-wide distributed DoS and ``warez" bot networks." (2002).
http://staff.washington.edu/dittrich/talks/core02/xdcc-analysis.txt (Feb 11, 2004)
45. CVE. "CVE-2001-0550." (2003).
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0550 (Feb. 4, 2004)
46. www.whitehats.com. "IDS213 FTP-PASSWS-RETRIEVAL-RETR."
http://www.whitehats.com/info/IDS213 (Feb 11, 2004)
47. Proftpd.org. "ProFTPD Logins and Authentication." (2/17/04)
http://www.castaglia.org/proftpd/doc/contrib/ProFTPD-mini-HOWTO-Authentication.html
48. Seifried, Kurt. "Port 5900 TCP, UDP."
http://www.seifried.org/security/ports/5000/5900.html (Feb. 11, 2004)
49. Anonymous. "sdbot 0.5b source code."
http://www.manshadow.org/tools/windows/sdbot05b.zip (Feb 17, 2004)
50. Snort.org. "NETBIOS NT NULL session." (2/11/04).
http://www.snort.org/snort-db/sid.html?sid=530 (Feb 11, 2004)
51. CVE. "CAN-2000-1186." (2/11/04).
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-1186 (Feb 12, 2004)
52. CVE. "CVE-1999-0067." (2004).
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0067 (Feb 4, 2004)
53. www.whitehats.com. "IDS492 NTPDX-BUFFER-OVERFLOW."
http://www.whitehats.com/info/IDS492 (11 Feb 2003)
54. Lorier, Perry. "Snort rules for IRC." http://coders.meta.net.nz/~perry/irc.rules (Feb 4 2002)
55. CVE. "CAN-2000-0138." (2/11/04).
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0138 (Feb 4, 2004)
56. www.whitehats.com. "IDS254 DDOS-SHAFT-CLIENT-TO-HANDLER." (2003).
http://www.whitehats.com/info/IDS254 (Feb 11, 2004)
57. Ramakrishnan, K., S. Floyd, and D. Black. "RFC3168: The Addition of Explicit Congestion Notification (ECN) to IP." (2001). http://www.ietf.org/rfc/rfc3168.txt (Feb 3, 2004)
58. Larratt, Gary. "GCIA Practical Assignment V3.0."
http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html (Feb 4, 2004)

Last Updated: October 17th, 2017

Upcoming Training

SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
Community SANS Pensacola SEC503	Pensacola, FL	Nov 27, 2017 - Dec 02, 2017	Community SANS
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Munich December 2017	Munich, Germany	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
Community SANS Nashville SEC401^	Nashville, TN	Jan 08, 2018 - Jan 13, 2018	Community SANS
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
Las Vegas 2018 - SEC503: Intrusion Detection In-Depth	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	vLive
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
SANS London February 2018	London, United Kingdom	Feb 05, 2018 - Feb 10, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Northern VA Spring - Tysons 2018	McLean, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS Secure Canberra 2018	Canberra, Australia	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
SANS Security West 2018	San Diego, CA	May 11, 2018 - May 18, 2018	Live Event
Community SANS Columbia SEC503	Columbia, MD	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced