Global Information Assurance Certification Paper

Task-Optimized Operating Systems for Intrusion Detection

Benjamin Allen
SANS New Orleans, Louisiana, USA
November 13-19, 2003
GIAC GCIA Practical Assignment (version 3.4)
Submitted: 30 April 2004

Abstract

This paper is presented in pursuit of the GIAC GCIA certification, using version 3.4 of the assignment.

The first section gives an introduction to a few open source customized operating system distributions focused on intrusion detection. These distributions offer analysts a functional snort system, often with an analysis console included, which can be installed in well under an hour. These systems provide analysts a tool for rapidly deploying additional IDS sensors on demand. This could prove invaluable when responding to an incident.

In the second section, three network detects are presented based on logs from SANS and from an employer network. These detects cover a CWD buffer overflow attack aimed at the Vermilion FTPd server, NULL scanning, and some common warez-related FTP activities detected on a printer.

In the final section, 5 days of logs from SANS are analyzed, revealing virus infected hosts, peer to peer file sharing, and some anomalous scanning behavior. Also, a novel example for visualizing IDS data using free tools is presented.

1.State of Intrusion Detection

Task-Optimized Operating Systems for Intrusion Detection

As open source operating systems like Linux, FreeBSD, NetBSD, and OpenBSD have become more popular and more mature, the community around them has created tools to enable customization of the operating system and of how it's installed. Because of the nature of open source licenses and the growth of the tools required for customization, task oriented operating system (OS) distributions have flourished. The site http://www.distrowatch.com provides a place to search for all manner of Linux distributions, and according to their statistics page, there are nearly 300 cataloged on the site.

Considering the mind boggling number of OS distributions, there must be some that are designed with the needs of intrusion detection analysts in mind. Searching distrowatch.com for “intrusion detection” lists some of the distributions I will summarize below, and some others.

Benefits

Regardless of which system you choose, there are several benefits to working with a task-optimized version of your favorite operating system. The first is rapid deployment. When the CD-ROM in your hand has everything you need to get the OS installed, configure snort with default rules, and start logging traffic, you can create a usable system quickly. This is critical when responding to a security incident. When the need arises to do extra monitoring for a portion of the network, spending a couple of hours to install a generic OS, all of the necessary packages and their prerequisites, and then configuring everything prevents you from collecting data and starting analysis. When it comes to computer security, time is always critical.

Another advantage to a customized OS distribution is repeatability. As many of the customized Linux distributions are designed to run directly from CD, you can be assured that you will get the same software, installed the same way every time you boot that CD. The other IDS oriented distributions that I've experimented with ask a minimal set of questions, and install and configure everything. This simplicity enables you to keep the list of your answers on a slip of paper in the CD case, ready for the next time you need it, or to walk someone through setting up an IDS sensor at a remote office over the phone.

Finally, an intrusion detection oriented OS distribution can be an educational tool. Using a bootable CD distribution, a user can test an unfamiliar operating system without needing a dedicated machine and without risking the data on their primary system. Also, as these OS's are focused on security work, the creators take care to provide default settings which mesh

well with industry best practices. This allows an analyst to focus his or her time on learning the analysis tools, not on closing gaping security holes left by a generic, “user friendly” installation of the operating system.

Features

The table below gives an overview of the four distributions with which I've experimented. The table lists the underlying operating system and kernel version, the versions of snort and ACID which are included, whether the OS supports multiprocessor systems or SCSI, where the OS runs from, and a short note about rule management capabilities that are installed by default.

Distro	Base OS	Snort	ACID	SMP	SCSI	Run From	Rule Management
OpenIDS v1.1	OpenBSD 3.4	2.0.6	0.9.6 b23	NO	NO	HDD	SSH. Since the OS is installed on a disk, can add other s/w.
Sentinix v1.0 Beta	Linux 2.4.22	2.0.6	0.9.6 b23	YES	YES	HDD	SnortCenter
IPCop v1.3.0	Linux 2.4.20	2.0.0	NO	NO	NO	HDD	OS upgrades, SSH, install something.
Knoppix-STD v0.1	Linux 2.4.21	2.1.0	0.9.6 b20	YES	N/A	CD	Oinkmaster. New rules lost at reboot.

OpenIDS

This distribution is based on OpenBSD version 3.4, and installs to a hard disk. Somehow, the authors have managed to streamline the OpenBSD's already minimalist installer. Installation is quick, and upon reboot the system starts snort and ACID, and provides SSH and https access on the management interface. As the default configuration of the firewall prohibits all traffic on the management interface except SSH and https, using one interface for traffic sniffing and management does not work well. During the installation, you are asked for the parameters to create a self-signed certificate for the SSL-enabled web server.

Sentinix

In addition to snort, ACID, and SnortCenter, Sentinix provides tools for monitoring network performance and service availability, fighting spam, and performing vulnerability assessments of your network. The installer lets you choose which services you would like to install, but exposes some services to the network unnecessarily (MySQL, http). This is the only distribution I've seen which includes SnortCenter for managing snort's rules and config files

via the web (Dens). Finally, Sentinix supports SMP on a single system, and clustering by way of the openMosix kernel extension (Bar).

The installer should add at least one dependency check so that enabling Snort and ACID requires MySQL to be enabled also. This configuration is valid to the installer, but packets don't get logged to the database when it is not installed. One other drawback with the system is that snort needs to be started from the web GUI at each system boot. However, since Sentinix is installed to the hard disk, this can be fixed easily.

IPCop

Designed to be a firewall and gateway system, IPCop also provides snort as part of its tool set. One feature about the IPCop installer that stood out to me was the clarity of the questions it asked. This makes IPCop more accessible to those unfamiliar with security, increasing the likelihood that it will be used. IPCop configures snort to log to syslog, and does not provide direct tools for updating snort or its rules. However, the system upgrades include updates to snort and snort's rules.

IPCop explicitly does not support SCSI, but when it attempts to find a hard disk for installation, at least it tells you “No IDE harddisk found.” One other nice feature would be occasional “roll-up” upgrades. When I looked, there were 9 updates on the IPCop site, each of which needed to be applied individually. Not all of them require a reboot, but it is an annoyance at installation, and barrier to rapid deployment.

Knoppix-STD

By employing on-the-fly decompression, Klaus Knopper raised the bar for bootable CD's when KNOPPIX was first released. The compression technique allows for roughly 2GB of applications to be accessible from an OS booted via a 650MB CD-ROM. There are many customizations to KNOPPIX, ranging from language localizations to cluster-oriented to bioinformatics (Bodnar).

With Knoppix-STD, booting from the CD gives you a full graphical desktop, with access to tools including the Nessus vulnerability scanner, snort and ACID, and Prelude Hybrid IDS (Vandoorselaere). Some of the startup scripts seem to have glitches when called from the GUI, but running them by hand seems to solve them. This, unfortunately, is true of the script which is supposed to start snort, barnyard, and MySQL to get ACID running. For some reason, barnyard does not start properly. This may be a timing issue where barnyard tries to start before MySQL is ready to receive any alerts.

Prelude NIDS is included in the distribution, however it failed to start due to read-only filesystem errors. Having access to prelude on a bootable CD would be an excellent opportunity to explore some of the features it has to offer.

Downfalls

Now that we've seen some of the features and benefits of IDS specific OS distributions, we should give some consideration to the drawbacks to these tools. Especially with the CD-ROM based distributions, making updates to the OS and applications is more difficult. Since all of these OS's are taking steps to make installation, package selection, and configuration much easier for end users, they would be doing a disservice to their users to expect them to download and compile updates to the OS itself. IPCop was the only distro I looked at which provided binary updates that were designed to be easy for end users to apply. One relief with CD-ROM based distributions is that it is also more difficult for attackers to make persistent changes.

Another concern is performance. Since these distributions are targeting a wide range of hardware, optimizations for more recent hardware are not available. Fortunately, in the scope of rapidly getting an IDS system online, performance is not necessarily measured in packet per second, but rather in minutes of traffic not analyzed.

While Sentinix provides a comprehensive method for managing snort's rules and configuration, the others do not. This can be a problem when custom rules need to be deployed and for keeping standard rules up to date. Since OpenIDS and IPCop both install to a hard disk, additional software could be loaded to address rule management, however, that software would not get updated with any patches to the distribution's software.

One final shortcoming, especially for an environment which is already set up for centralized logging, is that central logging is not typically supported. Since SnortCenter can be used on a Sentinix installation to configure an output plugin, a user would be able to configure a new output plugin from the web interface. However, this is no less work than editing the snort configuration file directly. Separating the sensor and alert aggregation roles is on the “to do” list for OpenIDS (Keri).

Although these and other IDS oriented distributions are still maturing, they provide a valuable set of tools for IDS analysts urgently needing an additional sensor. Learning to use these tools can provide insight into unfamiliar operating systems, experience with new tools, and readiness to respond to the next emergency.

References

Bar, Moshe. "openMosix, an Open Source Linux Cluster Project". <http://openmosix.sourceforge.net>. 2002-2004 Moshe Bar. Visited 28 April 2004.

Belinger, Jack. IPCop. “Twiki. IPCop. WebHome”. <http://www.ipcop.org>. 2001. 17 April 2004. Visited 26 April 2004.

Bodnar, Ladislav. “DistroWatch.com: Put the fun back into computing. Use Linux, BSD.”. <http://www.distrowatch.com>. Visited 28 April 2004.

Blomgren, Michel. “SENTINIX GNU/Linux distribution”. <http://www.sentinix.org/>. 2003. Visited 26 April 2004.

Dens, Setfan. "Snort Center - Snort management console". <http://users.pandora.be/larc>. 2002 Stefan Dens. Visited 28 April 2004.

Keri, Mikael. “OpenIDS”. <http://www.prowling.nu/main/openids/openids.html>. 3 June 2003. 1 March 2004. Visited 26 April 2004.

Knpooer, Klaus. "KNOPPIX - Live Linux Filesystem On CD". <http://knopper.net/knoppix-info/index-en.html>. Visited 28 April 2004.

Knoppix-STD. “Knoppix STD 0.1 security tools distribution”. <http://www.knoppix-std.org/>. 3 February 2004. Visited 26 April 2004.

Roesch, Martin. “Snort™ Users Manual v2.1.1”. <http://www.snort.org/docs/snort_manual.pdf> The Snort Project. 25 February 2004.

Vandoorselaere, Yoann. "Prelude Hybrid IDS". <http://www.prelude-ids.org>. Visited 28 April 2004.

2.Network Detects

Detect #1 – Vermilion FTPd CWD attack

This detect was posted to the intrusions@incidents.org email list on 11 February 2004, and is archived at:

http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00099.html
There were no replies.

1. Source of trace

This trace is based on logs obtained from http://www.incidents.org/logs/Raw/2002/ . Mergecap [1] was used to combine logs from sets 9, and 10 dated Dec 2, 2002 into one large capture file.

```
% mergecap -w goodset 2002*
```

2. Detect was generated by

I used a 2-pass method with snort (v2.1.0 [3])... first, a pass with all rules enabled (rule set from 28 Jan 2004), then a second pass looking for specific alert types. For both passes, binary logging is enabled (-b), logs go to a specified directory (-l dirname), "Full" alert mode is used (-A full), no checksums are verified (-k none), hex & ascii are shown in any text logs (-X), and ethernet headers are included (-e).

First pass:

```
% snort -c snort/snort.conf -r goodset -A full -l log200 -beXk none
% cd log200
% cat alert | grep -E "^\[\*\*\]" | sort | uniq -c | sort -n > alert.summary
```

Use IpAudit's total [2] to count the number of alerts (- => no grouping, 1s => sum of column 1) :

```
% total - 1s alert.summary
215898
```

Since dealing with 215898 alerts is not realistic, I choose to examine the “SHELLCODE” alerts more closely.

```
% cat alert.summary | grep SHELLCODE
   1 [**] [1:651:6] SHELLCODE x86 stealth NOOP [**]
   1 [**] [1:653:6] SHELLCODE x86 unicode NOOP [**]
   2 [**] [1:2314:1] SHELLCODE x86 0x90 NOOP unicode [**]
   7 [**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
   9 [**] [1:649:6] SHELLCODE x86 setgid 0 [**]
  16 [**] [1:650:6] SHELLCODE x86 setuid 0 [**]
  64 [**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
  73 [**] [1:1394:4] SHELLCODE x86 NOOP [**]
 263 [**] [1:648:6] SHELLCODE x86 NOOP [**]
```

Second pass: run snort again, with binary logging enabled, using only the SHELLCODE rules, and placing output into a new directory

```
% snort -c snort/snort.conf.shellcode -r goodset -A full -l log201 -beXk none
```

Use tcpdump to determine the time span of events:

```
% tcpdump -nttttr log201/snort.log | cut -d' ' -f1,2
10/09/2002 11:46:18.486507
....
11/17/2002 12:54:27.776507
```

Create a summary of the alerts triggered by snort.

```
% cat alert | grep -E "^\[\*\*\]" | sort | uniq -c | sort -n > alert.summary
% cat alert.summary
   1 [**] [116:54:1] (snort_decoder): Tcp Options found with bad lengths [**]
   1 [**] [121:3:1] Portscan detected from 32.245.166.236 Talker(fixed: 30 sliding: 4)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 170.129.50.120 Talker(fixed: 28 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 202.108.254.200 Talker(fixed: 10 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 207.166.87.157 Talker(fixed: 22 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 216.77.219.165 Talker(fixed: 2 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 217.228.210.78 Talker(fixed: 30 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 24.101.114.84 Talker(fixed: 10 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 24.154.202.158 Talker(fixed: 30 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 24.190.48.235 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 24.220.227.20 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 24.90.122.137 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 66.123.116.234 Talker(fixed: 30 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 66.250.114.252 Talker(fixed: 30 sliding:
30) Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [121:4:1] Portscan detected from 66.28.100.206 Talker(fixed: 6 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   1 [**] [1:651:6] SHELLCODE x86 stealth NOOP [**]
   1 [**] [1:653:6] SHELLCODE x86 unicode NOOP [**]
   2 [**] [1:2314:1] SHELLCODE x86 0x90 NOOP unicode [**]
   7 [**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
   8 [**] [116:97:1] (snort_decoder): Short UDP packet, length field > payload length
[**]
   9 [**] [1:649:6] SHELLCODE x86 setgid 0 [**]
  16 [**] [1:650:6] SHELLCODE x86 setuid 0 [**]
  42 [**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
  64 [**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
  73 [**] [1:1394:4] SHELLCODE x86 NOOP [**]
```

```
 263 [**] [1:648:6] SHELLCODE x86 NOOP [**]
```

Take a glance at snort.log with ethereal to see if any of the protocol decodes jump out as interesting... here's a block of

```
"FTP    CWD 00000000000000000000000000000000000000000000000000000000000000"
```

lines that are consecutive.

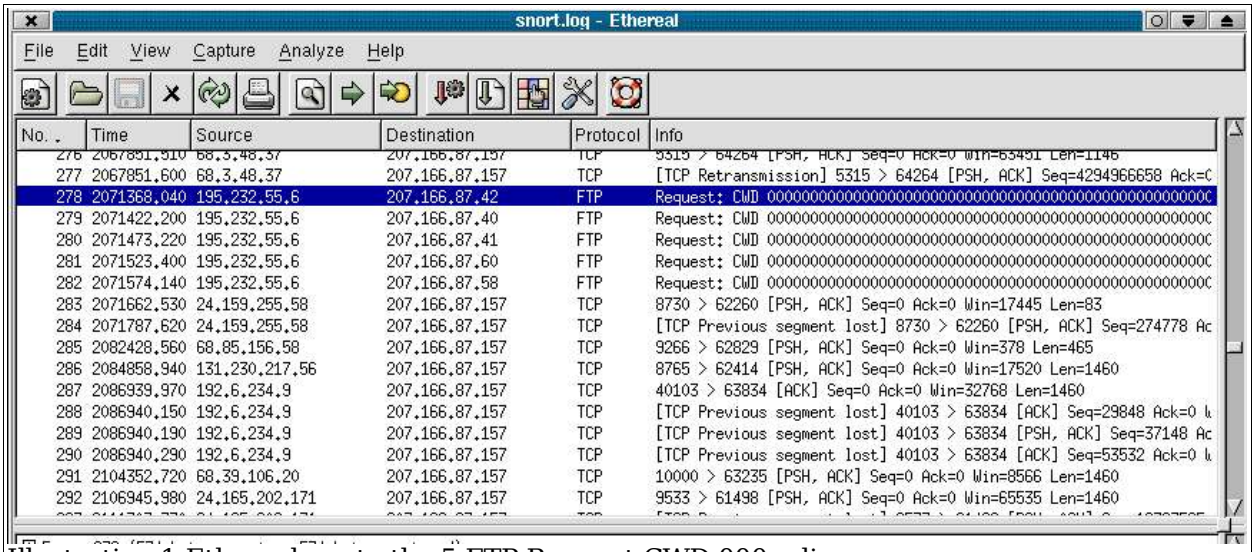


Illustration 1 Ethereal - note the 5 FTP Request:CWD 000... lines.

Sorting packets by ethereal's "Info" field, and finding the CWD 00000000000000000, we find that there are 7 such packets. Looking at the summary, we see that there are exactly 7 instances of only one snort rule: sid=1424 "SHELLCODE x86 0xEB0C NOOP".

Finding that rule in shellcode.rules, we see that it is (edited for readability):

```
  alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (
       msg:"SHELLCODE x86 0xEB0C NOOP";
       content:"|EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C|";
       classtype:shellcode-detect; sid:1424; rev:6;)
```

The relevant info from the snort alert output is:

```
% grep -A 6 "1:1424:6" alert
[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/02-05:09:06.526507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
195.232.55.6:1701 -> 207.166.87.42:21 TCP TTL:45 TOS:0x0 ID:55450 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0x82340FD5  Ack: 0x4333A866  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1040178 3948516

[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/02-05:10:00.686507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
```

```
195.232.55.6:1710 -> 207.166.87.40:21 TCP TTL:45 TOS:0x0 ID:37808 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0x866C7E2B  Ack: 0x45F91C5E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1045592 3953929

[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/02-05:10:51.706507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
195.232.55.6:1737 -> 207.166.87.41:21 TCP TTL:45 TOS:0x0 ID:11897 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0x8971D6B0  Ack: 0x48EDC37E  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1050693 3959031

[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/02-05:11:41.886507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
195.232.55.6:1756 -> 207.166.87.60:21 TCP TTL:45 TOS:0x0 ID:29061 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0x8CFA6343  Ack: 0x4BEAE351  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1055711 3964041

[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/02-05:12:32.626507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
195.232.55.6:1763 -> 207.166.87.58:21 TCP TTL:45 TOS:0x0 ID:14521 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0x8F862074  Ack: 0x4F49B33B  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 1060783 3969121

--
[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/16-09:41:40.176507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
163.24.239.8:2377 -> 170.129.50.5:21 TCP TTL:44 TOS:0x0 ID:35386 IpLen:20 DgmLen:560
DF
***AP*** Seq: 0xAB7BA6BD  Ack: 0xA5C3AABB  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675704 5583989

--
[**] [1:1424:6] SHELLCODE x86 0xEB0C NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/17-06:54:27.776507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x23E
165.154.7.2:1982 -> 170.129.50.4:21 TCP TTL:46 TOS:0x0 ID:35277 IpLen:20 DgmLen:560 DF
***AP*** Seq: 0x473ADE51  Ack: 0x719E0279  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 648881732 4580808
```

Looking at one of these packets with tcpdump, we can verify that it matches the snort rule:

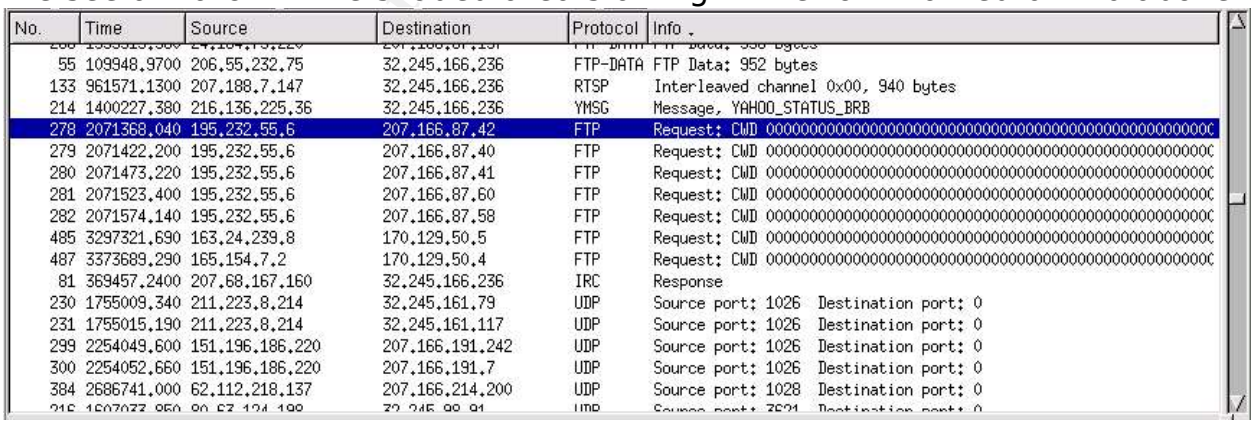
```
  alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (
       msg:"SHELLCODE x86 0xEB0C NOOP";
       content:"|EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C EB 0C|";
classtype:shellcode-detect; sid:1424; rev:6;)



% tcpdump -nvXr snort.log 'dst host 170.129.50.5 and dst port 21'
09:41:40.176507 163.24.239.8.2377 > 170.129.50.5.21: P [bad tcp cksum f6fe!]
```

```
2877007549:2877008057(508) ack 2781063867 win 32120 <nop,nop,timestamp 4675704
5583989> (DF) (ttl 44, id 35386, len 560)
0x0000   4500 0230 8a3a 4000 2c06 53e6 a318 ef08        E..0.:@.,.S.....
0x0010   aa81 3205 0949 0015 ab7b a6bd a5c3 aabb        ..2..I...{......
0x0020   8018 7d78 dd79 0000 0101 080a 0047 5878        ..}x.y.......GXx
0x0030   0055 3475 4357 4420 3030 3030 3030 3030        .U4uCWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0fc 4031 0708 985f        00000000..@1..._
0x0140   0808 eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0150   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0160   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0170   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0180   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0190   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01a0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01b0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01c0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01d0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01e0   eb0c eb0c eb0c eb0c 9090 9090 9090 9090        ................
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
```

We see a match in the shaded area starting in the row marked 0x140 above.



No.	Time	Source	Destination	Protocol	Info
55	109948.9700	206.55.232.75	32.245.166.236	FTP-DATA	FTP Data: 952 bytes
133	961571.1300	207.188.7.147	32.245.166.236	RTSP	Interleaved channel 0x00, 940 bytes
214	1400227.380	216.136.225.36	32.245.166.236	YMSG	Message, YAHOO_STATUS_BRB
278	2071368.040	195.232.55.6	207.166.87.42	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
279	2071422.200	195.232.55.6	207.166.87.40	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
280	2071473.220	195.232.55.6	207.166.87.41	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
281	2071523.400	195.232.55.6	207.166.87.60	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
282	2071574.140	195.232.55.6	207.166.87.58	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
485	3297321.690	163.24.239.8	170.129.50.5	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
487	3373689.290	165.154.7.2	170.129.50.4	FTP	Request: CWD 0000000000000000000000000000000000000000000000000000000000000000
81	369457.2400	207.68.167.160	32.245.166.236	IRC	Response
230	1755009.340	211.223.8.214	32.245.161.79	UDP	Source port: 1026 Destination port: 0
231	1755015.190	211.223.8.214	32.245.161.117	UDP	Source port: 1026 Destination port: 0
299	2254049.600	151.196.186.220	207.166.191.242	UDP	Source port: 1026 Destination port: 0
300	2254052.660	151.196.186.220	207.166.191.7	UDP	Source port: 1026 Destination port: 0
384	2686741.000	62.112.218.137	207.166.214.200	UDP	Source port: 1028 Destination port: 0

Illustration 2 Ethereal: 7 alerts for CWD 0000, 3 sources, 7 destinations.

We can see in ethereal, and in the alerts above, that 3 sources cause these 7 alerts. Since one source has triggered this alert 5 times [195.232.55.6], we look back at what hosts it has talked to in the large, merged capture file.

```
% tcpdump -nr goodset -w log201/host-195.232.55.6 'host 195.232.55.6'
```

And to see what's going on we look at the file with tcpdump again:

```
% tcpdump -nvXr host-195.232.55.6
05:09:06.526507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum acb4!]
2184450005:2184450513(508) ack 1127458918 win 5840 <nop,nop,timestamp 1040178 3948516>
(DF) (ttl 45, id 55450, len 560, bad cksum 9bb8!)
0x0000   4500 0230 d89a 4000 2d06 9bb8 c3e8 3706        E..0..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 0fd5 4333 a866        ..W*....4..C3.f
0x0020   8018 16d0 7135 0000 0101 080a 000f df32        ....q5.........2
0x0030   003c 3fe4 4357 4420 3030 3030 3030 3030        .<?.CWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0fc 4031 0708 985f        00000000..@1..._
0x0140   0808 eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0150   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0160   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0170   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0180   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0190   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01a0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01b0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01c0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01d0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01e0   eb0c eb0c eb0c eb0c 9090 9090 9090 9090        ................
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
05:09:06.976507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum b5b5!] 508:524
(16) ack 522 win 6432 <nop,nop,timestamp 1040231 3948557> (DF) (ttl 45, id 55451, len
68, bad cksum 9da3!)
0x0000   4500 0044 d89b 4000 2d06 9da3 c3e8 3706        E..D..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 11d1 4333 aa6f        ..W*.....4..C3.o
0x0020   8018 1920 5b83 0000 0101 080a 000f df67        ....[..........g
0x0030   003c 400d 4357 4420 7e2f 7b2e 2c2e 2c2e        .<@.CWD.~/{.,.,.
0x0040   2c2e 7d0a                                      ,.}.
05:09:09.716507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum b5b5!] 612:619
(7) ack 833 win 6432
```

```
<nop,nop,timestamp 1040506 3948842> (DF) (ttl 45, id 55459, len 59, bad cksum 9da4!)
0x0000   4500 003b d8a3 4000 2d06 9da4 c3e8 3706        E..;..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 1239 4333 aba6        ..W*.....4.9C3..
0x0020   8018 1920 ca34 0000 0101 080a 000f e07a        .....4.........z
0x0030   003c 412a 4357 4420 7e7b 0a                    .<A*CWD.~{.
```

... 4 more of these sets, 1 set for each destination host attacked by this source.

3. Probability the source address was spoofed

Since the attacker is issuing commands to an FTP server, using an established TCP connection, the source address is nearly impossible to spoof. Based on the rate at which the alerts are being triggered and the content of the packets, we can assume that this attack is being performed by a tool.

For the attack against the Vermilion FTP server to work, 3 CWD commands longer than 504 characters each. Since we only see one of the 3, we can imply that the other packets were sent earlier in the session, and were crafted so as not to trigger the snort alerts.

Also, if the exploit is successful, the attacker will need to have information from the compromised machine returned to him/her.

The TTL is consistently 45. Doing a traceroute to an address in the net block, I observed 21 hops from the central US to France before traceroute starts breaking. 45+21=65 so the attacker's host likely has a TTL of 64, and is slightly closer to the victim than to my test machine. This would fit with many variants of UNIX, based on the work of the Honeynet Project [4] and p0f [5].

Looking up the WHOIS info for this host [details in Section 7 "Active Targeting"], we see that this IP is part of a Compuserve Europe dial-up pool. As the address on a dial-up tends to change between each connection, the source address is not likely spoofed.

4. Description of attack

A search of Google for "FTP CWD overflow" yields:

- Vermillion FTP CWD overflow DoS

  http://xforce.iss.net/xforce/xfdb/3543

  from http://xforce.iss.net/xforce/xfdb/3543 [whitespace condensed]

  vermillion-ftp-cwd-overflow (3543) Medium Risk

  Description: Vermillion is an FTP is a Windows based FTP daemon. It

contains a buffer overflow in the CWD command. If a remote or local user overflows the CWD command with 504 characters 3 times in a row, the server will crash and will have to be restarted.

Platforms Affected:

Arcane Software, Inc. Vermillion FTP Daemon 1.23

on Microsoft Corporation Windows Any version

Remedy: Upgrade to the latest version of Vermillion (1.31 or later), available from the Arcane Software Web site. See References.

Consequences: Denial of Service

The xforce.iss.net site also had referneces to the following sites:

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-1058

  The MITRE group's Common Vulnerabilities & Exposures list has this attack listed as a candidate for inclusion in the list (CAN).

- http://www.securityfocus.com/bid/0818

  Security Focus classifies this activity as a "Boundary Condition Error" which local and remote users can attack, and credits USSR Labs with discovery or the buffer overflow.

- http://www.ussrback.com/labs14.html

  USSR Labs has a write-up of the buffer overflow problem, and code which is able to demonstrate the impact of the buffer overflow.

- http://www.iss.net/security_center/advice/Intrusions/2001308/default.htm

  Lists several different buffer overflows against different FTP servers.

Also 2 others came up from the intrusions@incidents.org email list.

- http://cert.uni-stuttgart.de/archive/intrusions/2002/06/msg00175.html

  Shows snort alerts for "FTP large CWD packet" and others as an example to prompt discussion on the intrusions list.

- http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00080.html

  Is part of a discussion about alerts on large FTP commands.

Finally, 2 other students did detects on FTP CWD attacks:

- http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00186.html

- http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00195.html

5. Attack mechanism

As mentioned previously, this attack was likely preformed with a tool. Nessus has a plugin to test for this vulnerability [6,7]. Another tool is available from USSR Labs [9].

Based on the alerts seen from this dataset, code analysis of the USSR exploit, and tests with the nessus plugin, neither of these tools was the one used to trigger these alerts. The tool used attempts to exploit multiple buffer overflows for various ftp servers on each ftp connection.

To exploit the denial of service bug in the Vermilion FTP server, 3 packets are sent with CWD commands longer than 504 bytes in length. At this point, the server crashes [10]. As the third large CWD command triggers the buffer overflow, it would be pointless to put SHELLCODE in the first two CWD commands issued by the tool, since they tend to trigger IDS alerts.

6. Correlations

Searching http://www.dshield.org turned up nothing current about 195.232.0.0/16 scanning for ftp.

http://www.incidents.org/archives/intrusions/msg11827.html - shows 195.232.25.5 scanning a host for any open ports on Wed, 07 May 2002.

http://www.giac.org/practical/GCIA/Jared_McLaren_GCIA.pdf shows 195.232.60.24 scanning for the KDM window manager in analysis of the logs from http://www.incidents.org/logs/Raw/2002.5.10

Beyond that, there seem to be no leads. If the ISP's acceptable use policy were readily available, it may have given some insight into how the network is managed, and what policies are in place. However, the AUP would likely be different now than it was then.

7. Evidence of active targeting

This is the only traffic seen from this source host. It is very likely targeted.

```
% tcpdump -nr ../goodset 'host 195.232.55.6' | wc -l
 15
```

3 alert packets * 5 destination hosts = 15 packets => no other traffic from this host.

Also, the TCP sessions are already established before we see the alerts, but we can tell that the first 2 packets are in sequence by their IP IDs. As we're seeing multiple FTP buffer overflow alerts per system, it would seem that the attacker knows that these systems are ftp servers.

We can also see that each attack set spans nearly the same amount of time

per host (just over 3 seconds), and the time spacing between the hosts is consistently near 50 seconds. Since we know that the destination addresses have been mangled to protect the innocent, we cannot tell whether the attacker was originally scanning them in sequential order by IP. However, with the consistent timings, we can assume that the attacker knew the addresses in advance.

Here we can see the timestamps in Month/Day/Year Hour: Minute: Second. fraction

```
% tcpdump -nttttr ../goodset 'host 195.232.55.6' | cut -d: -f1-3
11/02/2002 11:09:06.526507 195.232.55.6.1701 > 207.166.87.42.21
11/02/2002 11:09:06.976507 195.232.55.6.1701 > 207.166.87.42.21
11/02/2002 11:09:09.716507 195.232.55.6.1701 > 207.166.87.42.21
11/02/2002 11:10:00.686507 195.232.55.6.1710 > 207.166.87.40.21
11/02/2002 11:10:01.146507 195.232.55.6.1710 > 207.166.87.40.21
11/02/2002 11:10:03.846507 195.232.55.6.1710 > 207.166.87.40.21
11/02/2002 11:10:51.706507 195.232.55.6.1737 > 207.166.87.41.21
11/02/2002 11:10:52.156507 195.232.55.6.1737 > 207.166.87.41.21
11/02/2002 11:10:54.846507 195.232.55.6.1737 > 207.166.87.41.21
11/02/2002 11:11:41.886507 195.232.55.6.1756 > 207.166.87.60.21
11/02/2002 11:11:42.356507 195.232.55.6.1756 > 207.166.87.60.21
11/02/2002 11:11:45.016507 195.232.55.6.1756 > 207.166.87.60.21
11/02/2002 11:12:32.626507 195.232.55.6.1763 > 207.166.87.58.21
11/02/2002 11:12:33.056507 195.232.55.6.1763 > 207.166.87.58.21
11/02/2002 11:12:36.046507 195.232.55.6.1763 > 207.166.87.58.21
```

We know that these were the only alerts from the attacker's address, and nearly the only ftp traffic that generated alerts. The other ftp traffic on the standard ports can be found with tcpdump and (t)ethereal. Since all 5 hosts that were targeted by 195.232.55.6 are on one subnet, we will limit our search for other ftp traffic to the subnet 207.166.87.0/24.

```
% tcpdump -nr ../goodset -w other-ftp-traffic \
  '(port 20 or port 21) and (host not 195.232.55.6 and net 207.166.87)'

% tethereal -r other-ftp-traffic
  1   0.000000 24.184.79.220 -> 207.166.87.157 FTP-DATA FTP Data: 536 bytes
  2 9652.670000 207.166.87.157 -> 194.213.194.37 FTP Request: GNUTELLA CONNECT/0.6
  3 246536.840000 207.166.87.157 -> 194.213.194.37 FTP Request: GNUTELLA CONNECT/0.6
  4 416334.200000 208.61.206.232 -> 207.166.87.40 FTP Request: PASS
  5 995601.770000 207.166.87.40 -> 218.147.119.184 FTP-DATA FTP Data: 1402 bytes
  6 995601.770000 207.166.87.40 -> 218.147.119.184 FTP-DATA FTP Data: 1402 bytes
  7 995602.020000 207.166.87.40 -> 218.147.119.184 FTP-DATA FTP Data: 1402 bytes
  8 995602.020000 207.166.87.40 -> 218.147.119.184 FTP-DATA FTP Data: 723 bytes
```

So we can see that there is an FTP-DATA transfer from 207.166.87.40 to 218.147.119.184, a login from 208.61.206.232 to 207.166.87.40, and some GNUTELLA traffic.

Using tcpflow [8], we can rebuild the FTP-DATA stream, and see what's inside:

```
% tcpflow -cr other-ftp-traffic 'host 207.166.87.40 and 218.147.119.184'
207.166.087.040.00020-218.147.119.184.05032: total 79336
-rw-r--r--   1 root     root      1381971 Oct  9 14:37 10n268.pdf
-rw-rw-r--   1 root     504        360759 Jan  6  2002 20019.pdf
-rw-rw-r--   1 root     504        380407 Jan  6  2002 20020.pdf
-rw-rw-r--   1 root     root       371140 Jun 11  2002 200203v.pdf
[snip - many lines omitted for brevity.]
-rw-rw-r--   1 root     504        316849 Jan  6  2002 ircc20.pdf
-rw-rw-r--   1 root     504         55645 Jan  6  2002 victorybx66.pdf
```

We see a directory listing for an FTP server, showing the owner of the files to be root, the UNIX superuser. We also notice that there appear to be no other legitimate ftp servers on the subnet of interest.

To find out more about the attacking host, we consult WHOIS (and trim out the excess)

```
% whois 195.232.55.6

[snipped for brevity]
inetnum:      195.232.48.0 - 195.232.63.255
netname:      UUNET-HIL-PPP-POOL
descr:        Frankfurt PPP Client Pool
descr:        Dynamic Dial-Up
descr:        Security Incident reports should be send to: abuse@wcom.net
country:      DE
admin-c:      HC3-ORG
tech-c:       HC3-ORG
status:       ASSIGNED PA
mnt-by:       RIPE-NCC-NONE-MNT
changed:      hostmaster@wcom.net 19991220
changed:      hostmaster@wcom.net 20001206
source:       RIPE

route:        195.232.0.0/17
descr:        Compuserve Europe
origin:       AS5621
notify:       wan@ipeng.wcom.net
mnt-by:       AS5621-MNT
changed:      jpinnow@ipeng.wcom.net 19990827
source:       RIPE
[snipped for brevity]
```

Noting the "descr: Dynamic Dial-Up", this is likely a modem, DSL, or cable-modem address, and the address may have changed between reconnaissance & attack.

Let's look for other traffic from this net block

```
% tcpdump -nr ../goodset -w recon-srcs 'net 195.232.48.0/20'
```

If the only traffic from the 195.232.48.0/20 network block is from our attacker, then the two files below will be the same:

```
% diff -s recon-srcs host-195.232.55.6
```

```
Files recon-srcs and host-195.232.55.6 are identical
```

Thus, only 195.232.55.6 connected to the monitored from the 195.232.48.0/20 net block.

Based on the consistent time sequencing, definitive verification of one ftp server, multiple exploit attempts per connect, and lack of other traffic from this attacking host, I believe that this attacker/tool was targeting the ftp servers on this network actively. This does not rule out the possibility that the attacker's tool has scanned for ftp servers previously, and we have not been alerted to the scanning.

As we are unable to see the initial logins in the snort alerts, we can assume that the attacker was able to do any needed reconnaissance without being detected.

8. Severity

Using the SANS formula for severity found in the GCIA practical assignment:

severity = (criticality + lethality)(system +network countermeasures) with 1 being the lowest value, and 5 the highest, we have the following:

Criticality: 5

Based on the listing of documents from the ftp data stream, and searching Google for documents with the same name, it would seem that this system houses public documentation which customers need access to. This is a "first impression" system: if it is down, customers will get a poor first impression of the company, and the company can loose customers.

Also this host does a significant amount of http traffic (403 errors outbound).

Lethality: 5

Based on other traffic from 207.166.87.40 [below], the web server claims to be running Red Hat Linux with Apache & FrontPage.

```
% tcpdump -nr ../goodset -w net-207.166.87-targets 'host ( 207.166.87.42 or \
  207.166.87.40 or 207.166.87.41 or 207.166.87.60 or 207.166.87.58 )'

% tcpflow -cr net-207.166.87-targets 'host 213.240.6.14'
207.166.087.040.00080-213.240.006.014.01120: HTTP/1.1 403 Forbidden
Date: Sat, 02 Nov 2002 10:55:17 GMT
Server: Apache/1.3.12 (Unix)  (Red Hat/Linux) FrontPage/4.0.4.3
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
X-Pad: avoid browser bug

10b
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
```

```
<TITLE>403 Forbidden</TITLE>
</HEAD><BODY>
<H1>Forbidden</H1>
You don't have permission to access /
on this server.<P>
<HR>
<ADDRESS>Apache/1.3.12 Server at www.XXXXXXXX Port 80</ADDRESS>
</BODY></HTML>

0
[snipped more that followed]
```

Also, we saw root listed as the owner of the files in the FTP directory listing previously obtained with tcpflow. This system is likely running a flavor of UNIX or Linux.

As this exploit is designed for the Windows-based Vermilion FTP server, it is not likely to be lethal. However, there are other ftp servers for Linux which have been susceptible to buffer overflows as well. Also, this tool seems to be checking for multiple different buffer overflows on each system it visits.

Perhaps we can tell wether the host or the ftp service was crashed by looking at the timing of the alerts.

```
% tcpdump -nttttr ../goodset 'host 207.166.87.40' | head -220 | tail -20 |cut -d: -f1-
3
11/02/2002 03:57:03.626507 4.63.215.135.2009 > 207.166.87.40.80
11/02/2002 04:11:03.796507 4.65.196.108.2564 > 207.166.87.40.80
11/02/2002 06:05:11.466507 207.166.87.40.80 > 213.240.6.14.1120
11/02/2002 06:05:19.936507 207.166.87.40.80 > 213.240.6.14.1120
11/02/2002 06:05:29.526507 207.166.87.40.80 > 213.240.6.14.1120
11/02/2002 07:26:05.156507 218.227.137.78.60476 > 207.166.87.40.80
11/02/2002 08:06:40.296507 64.51.141.2.3466 > 207.166.87.40.80
11/02/2002 08:37:56.386507 62.103.210.66.1216 > 207.166.87.40.80
11/02/2002 08:37:57.816507 62.103.210.66.1220 > 207.166.87.40.80
11/02/2002 10:08:59.756507 207.166.87.40.80 > 213.240.6.20.2336
11/02/2002 11:10:00.686507 195.232.55.6.1710 > 207.166.87.40.21
11/02/2002 11:10:01.146507 195.232.55.6.1710 > 207.166.87.40.21
11/02/2002 11:10:03.846507 195.232.55.6.1710 > 207.166.87.40.21
[assumed crash & reboot]
11/02/2002 11:45:56.206507 213.99.232.219.1159 > 207.166.87.40.80
11/02/2002 11:45:58.346507 213.99.232.219.1162 > 207.166.87.40.80
11/02/2002 15:28:44.536507 68.152.32.167.3509 > 207.166.87.40.80
11/02/2002 15:39:16.446507 68.152.32.167.4609 > 207.166.87.40.80
11/02/2002 17:09:59.836507 66.43.149.135.4672 > 207.166.87.40.80
11/02/2002 19:54:40.266507 24.247.223.2.1294 > 207.166.87.40.80
11/02/2002 22:28:22.066507 217.66.199.241.18871 > 207.166.87.40.80
```

Based on the irregularity of the alerts, we cannot conclude whether the system has crashed.

For purposes of analysis, we will assume that the machine crashed, and was rebooted during the 35 minutes following the alert at 11:10:03 on 11/02/2002.

System Countermeasures: 1

Based on the file listings from http://archive.apache.org/dist/httpd/old/ Apache v1.3.12 was released on 25-Feb-2000. Based on the date in the HTTP/403 response and the packet capture logs, the web server software is over 2 years old. At the time & date listed in the HTTP/403 response, the current versions of Apache appear to have been 1.3.27 or 2.0.43.

Also, the presence of the FrontPage extensions implies to me that the organization strongly prefers to work from Windows. These 2 things together indicate that this system is not being actively maintained by a security-conscious administrator.

Network Countermeasures: 4

As this is a customer-oriented web & ftp server, those services are required to be open to the world at large. There was one packet of traffic detected which was not to/from port 20, 21, or 80. Searching snort.org's port list for the ports involved (40195 and 33709) turned up no listings; this packet could have been caused by a passive FTP session. Also, the lack of packets destined for other ports implies that there is ingress filtering in front of the network. Without ingress filtering, the snort instance would have likely tripped some port-based, scanning, or fingerprinting rules, if the rules were enabled.

The existence of a NIDS on the network indicates a desire to know what is happening on the network.

Severity = (5 + 5) (1 + 4) = 50

Using the formula from http://www.giac.org/ID_assignment_guidelines.php

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

9. Defensive recommendations

Upgrade the web and ftp server software to current versions.

If practical, split the web & ftp services between 2 separate hosts.

Use a non-x86 architecture to raise the bar for writing the assembly code used in the buffer overflow. For example, fewer people have access to sparc- and mips-based systems to test buffer overflow code on.

Remove FrontPage extensions on the web server, if they are not used. If they are required, add a virtual server to the apache instance, ideally using a dedicated network interface on the web server, and only allow FrontPage access via the dedicated interface, and/or only from explicitly enumerated hosts.

Recompile or reconfigure Apache to give out less information (for example, in the "Server: " blocks seen in the HTTP/403 replies). Reducing the amount

of information an attacker can determine about your system makes it more difficult to attack.

Set up a reverse web proxy to filter requests coming in to the web server.

Make sure the operating system is up to date (patches, etc.).

Make sure that logging is configured and working, preferably logging to a loghost as well as the machine itself.

Keep the snort rules up to date, and keep monitoring the alerts.

10. Multiple choice test question

```
% tcpdump -nvXr ftp-attack.tcpdump
05:09:06.526507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum acb4!]
2184450005:2184450513(508) ack 1127458918 win 5840 <nop,nop,timestamp 1040178 3948516>
(DF) (ttl 45, id 55450, len 560, bad cksum 9bb8!)
0x0000   4500 0230 d89a 4000 2d06 9bb8 c3e8 3706        E..0..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 0fd5 4333 a866        ..W*.....4..C3.f
0x0020   8018 16d0 7135 0000 0101 080a 000f df32        ....q5.........2
0x0030   003c 3fe4 4357 4420 3030 3030 3030 3030        .<?.CWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0fc 4031 0708 985f        00000000..@1..._
0x0140   0808 eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0150   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0160   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0170   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0180   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0190   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01a0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01b0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01c0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01d0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01e0   eb0c eb0c eb0c eb0c 9090 9090 9090 9090        ................
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
05:09:06.976507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum b5b5!] 508:524
(16) ack 522 win 6432 <nop,nop,timestamp 1040231 3948557> (DF) (ttl 45, id 55451, len
68, bad cksum 9da3!)
0x0000   4500 0044 d89b 4000 2d06 9da3 c3e8 3706        E..D..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 11d1 4333 aa6f        ..W*.....4..C3.o
```

```
0x0020   8018 1920 5b83 0000 0101 080a 000f df67        ....[..........g
0x0030   003c 400d 4357 4420 7e2f 7b2e 2c2e 2c2e        .<@.CWD.~/{.,.,.
0x0040   2c2e 7d0a                                      ,.}.
05:09:09.716507 195.232.55.6.1701 > 207.166.87.42.21: P [bad tcp cksum b5b5!] 612:619
(7) ack 833 win 6432
<nop,nop,timestamp 1040506 3948842> (DF) (ttl 45, id 55459, len 59, bad cksum 9da4!)
0x0000   4500 003b d8a3 4000 2d06 9da4 c3e8 3706        E..;..@.-.....7.
0x0010   cfa6 572a 06a5 0015 8234 1239 4333 aba6        ..W*.....4.9C3..
0x0020   8018 1920 ca34 0000 0101 080a 000f e07a        .....4.........z
0x0030   003c 412a 4357 4420 7e7b 0a                    .<A*CWD.~{.
```

In the tcpdump output above, generated by a snort instance logging in binary mode, the approximately 3 second pause between the last 2 alerts is:

A) A TCP retransmit timeout

B) A symptom of a system that crashed

C) An indication that several packets in the stream did not trigger any snort rules

D) A delay while the attacker typed in the next command

Answer: C

A: If it had been a retransmit, the packet would have been identical to the one previous (note the sequence numbers changing.)

B: If the system had crashed, we would be likely see TCP retransmits.

D: Based on the spacing between and content of the first 2 packets, we can tell that this attack was not done by hand. [Try to type "CWD.~/{.,.,.,.}" in less than 0.5 seconds!]

References for Detect #1

[1] mergecap - http://www.ethereal.com/mergecap.1.html included with ethereal.

[2] ipaudit - http://ipaudit.sourceforge.net/

[3] snort - http://www.snort.org/

[4] Project Honeynet - http://project.honeynet.org/papers/finger/traces.txt

[5] p0f - http://www.stearns.org/p0f/

[6] nessus - http://www.nessus.org

[7] nessus vftpd plugin - http://cgi.nessus.org/plugins/dump.php3?id=10293

[8] tcpflow - http://www.circlemud.org/~jelson/software/tcpflow

[9] USSR Labs exploit - http://www.ussrback.com/labs14.html

[10] Security Focus Discussion - http://www.securityfocus.com/bid/0818/discussion/

Detect #2 – NULL Scans

This detect was posted to the intrusions@incidents.org email list on 27 February 2004, and is archived at:

http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00166.html
There were no replies.

1. Source of trace

This analysis is based on logs from http://http://www.incidents.org/logs/Raw/2003.12.15.tgz , concatenated with mergecap[6].

[Network Architecture]

Based on the source addresses (IP and ethernet/MAC), we can determine the topology of the network of interest using tcpdump [1]

```
% tcpdump -ner host-10.10.10.113 | cut -d' ' -f2,6 | cut -d. -f1-4 | sort | uniq
0:50:56:40:0:64 10.10.10.2
0:50:56:40:0:64 arp
0:50:56:40:0:6d 192.168.17.129
0:50:56:40:0:6d 192.168.17.135
0:50:56:40:0:6d 192.168.17.68
0:50:56:40:0:6d arp
0:a:95:7c:24:0 10.10.10.113
0:a:95:7c:24:0 arp
```

This leads us to the network diagram shown below:

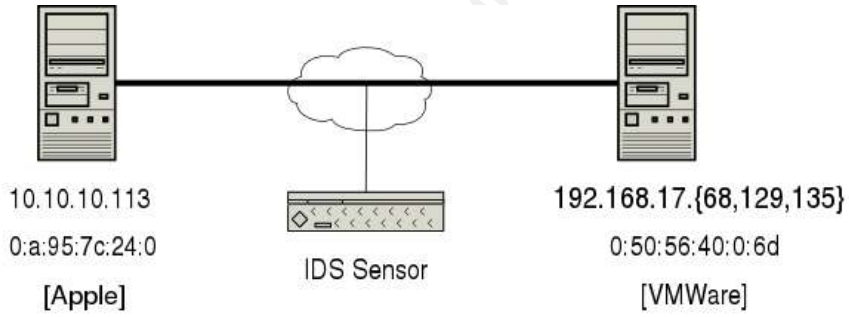


Illustration 3 Network Topology

The “cloud” in the illustration above could be a switch with a span port, a hub, or an IDS sensor running on a VMWare “Host” operating system and monitoring all traffic to the virtual machines [15].

2. Detect was generated by

This detect was generated using snort [9] v2.1.0, making 3 passes thru the

data:

a first pass with all of the rules in the 28 Jan 2004 set,

a second pass concentrating on a single host's traffic, and

a third pass with a tuned rule set, looking only at the specified host's traffic.

For all three passes, binary logging is enabled (-b), logs go to a specified directory (-l dirname), "Full" alert mode is used (-A full), no checksums are verified (-k none), hex & ascii are shown in any text logs (-X), and ethernet headers are included (-e).

```
% snort -c snort-full -r 2003.12.15-all -l log100 -A full -beXk none
% cd log100
% sfa-summary alert
[source for sfa-summary in References - given "alert", creates "alert.summary"]
% cat alert.summary | tail -2
5050 [**] [1:465:1] ICMP ISS Pinger [**]
18130 [**] [1:623:1] SCAN NULL [**]
% mkdir ss && snortsnarf -d ss alert
```

The snortsnarf [10] index.html shows that 1 host (10.10.10.113) was responsible for all of the SCAN NULL rules, triggered on connections to 3 hosts: 192.168.17.{68,129,135}.

```
% lynx -dump -nolist -width 100 index.html | grep -E "SCAN NULL|Priority"
   Priority Signature (click for sig info) # Alerts # Sources # Dests Detail link
   2 SCAN NULL [sid] [arachNIDS] 18130 1 3 Summary
```

For passes 2 & 3, filter the tcpdump file to contain only communications with 10.10.10.113.

```
% tcpdump -nr 2003.12.15-all -w host-10.10.10.113 'host 10.10.10.113'
```

Run snort again, this time on the dump for 10.10.10.113, to see what other activity snort catches:

```
% snort -c snort-full -r host-10.10.10.113 -l log101 -A full -beXk none
% cd log101
% sfa-summary alert
% cat alert.summary
   1 [**] [121:4:1] Portscan detected from 10.10.10.113 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   4 [**] [1:1228:2] SCAN nmap XMAS [**]
   4 [**] [1:628:2] SCAN nmap TCP [**]
  10 [**] [1:1418:2] SNMP request tcp [**]
  10 [**] [1:1420:2] SNMP trap tcp [**]
  12 [**] [1:1421:2] SNMP AgentX/tcp request [**]
18130 [**] [1:623:1] SCAN NULL [**]
```

The rule for "SCAN NULL" is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN NULL"; flags:0;
```

```
       seq:0; ack:0; reference:arachnids,4; classtype:attempted-recon;
       sid:623; rev:1;)
```

Examining the rules for sid 1418, 1420, 1421 we see that all 3 are in snmp.rules, and all 3 are port based. Since these rules will also trigger with a "SCAN NULL" we want to ignore them, in favor of the “SCAN NULL” rule.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP request tcp";
       reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1418;
       rev:2; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 162 (msg:"SNMP trap tcp";\
       reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1420;
       rev:2; classtype:attempted-recon;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 705 (msg:"SNMP AgentX/tcp request";
       reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1421;
       rev:2; classtype:attempted-recon;)
```

Running snort again, with the SNMP rules disabled,

```
% snort -c snort.conf.no-snmp -r host-10.10.10.113 -l log102 -A full -beXk none
% cd log102 && sfa-summary alert && cat alert.summary
   1 [**] [121:4:1] Portscan detected from 10.10.10.113 Talker(fixed: 30 sliding: 30)
Scanner(fixed: 0 sliding: 0) [**]
   4 [**] [1:1228:2] SCAN nmap XMAS [**]
   4 [**] [1:628:2] SCAN nmap TCP [**]
18162 [**] [1:623:1] SCAN NULL [**]
```

We see that all 32 of the packets that triggered the SNMP rules, also match on the "SCAN NULL" rule.

To find out how many packets our attacker is sending to each of our targets, we use tcpdump:

```
% tcpdump -nr host-10.10.10.113 'src 10.10.10.113 and tcp' | cut -d' ' -f4 | cut -d.
-f1-4 | sort | uniq -c | sort -n
5291 192.168.17.135
6315 192.168.17.68
6624 192.168.17.129
```

A total of 1657 different ports were scanned, ranging from port 1 to 65301 [see section 7 for details]. There is a notable amount of traffic setting different flags and options being sent to port 1 of each of the 3 target hosts. As none of these scans against port 1 responded, they will be ignored for the rest of this detect.

To get an idea of how aggressively this scan was done, we can use tcpdump to find the number of packets sent during each minute of the attack. We include the date in the output to ensure that the sorting does not lump events from different days together.

```
% tcpdump -nttttr host-10.10.10.113 | cut -d: -f1,2 | uniq -c
1234 11/18/2003 18:57
2122 11/18/2003 18:58
```

```
1950 11/18/2003 18:59
1168 11/18/2003 19:00
1719 11/18/2003 19:01
2035 11/18/2003 19:02
1940 11/18/2003 19:03
 824 11/18/2003 19:04
   4 11/18/2003 19:07
   4 11/18/2003 19:12
1037 11/18/2003 19:14
1463 11/18/2003 19:15
1572 11/18/2003 19:16
1234 11/18/2003 19:17
```

The scan lasts 20 minutes, and generates nearly 20,000 packets. This gives us a rate of 1000 packets per minute, or roughly 16.7 packets per second. This is a fairly aggressive scan.

3. Probability the source address was spoofed

The attacker is trying to find out which ports are open by sending an invalid TCP packet in an attempt to elicit a TCP RESET packet from the host. As the attacker wants to see this packet return, spoofing the source address is highly unlikely.

4. Description of attack

The attacker is attempting to enumerate the TCP ports which are open on the target hosts by sending invalid packets and waiting for the response.

From snort.org's info on sid=623: [11]

> "A tcp packet with none of it's control bits (URG, ACK, PSH, RST, SYN, FIN) was detected. Additionally, both the sequence number and acknowledgment number were set to 0. An open port will generally not respond at all, whereas a closed port will generally respond with an ACK RST. The particular response varies between operating systems, and is also governed by any filtering that may be done between the two hosts."

From White Hats: [12]

> “This event indicates that a TCP frame has been seen with a sequence number of zero and all control bits are set to zero. This frame should never be seen in normal TCP operation. An attacker may be scanning your system by sending these specially formatted frames to see what services are available."

From the NMAP man page: [3]

> "The idea is that closed ports are required to reply to your probe packet with an RST, while open ports must ignore the packets in question (see RFC 793 pp 64). The FIN scan uses a bare (surprise) FIN packet as

the probe, while the Xmas tree scan turns on the FIN, URG, and PUSH flags. The Null scan turns off all flags."

From RFC 793, page 64: [4]

```
"SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

    all data in the incoming segment is discarded.  An incoming
    segment containing a RST is discarded.  An incoming segment not
    containing a RST causes a RST to be sent in response.  The
    acknowledgment and sequence field values are selected to make the
    reset sequence acceptable to the TCP that sent the offending
    segment."
```

Thus, for a target system that is RFC 793-compliant, the attacker is expecting to see RST packets being sent back for every port which is closed.

5. Attack mechanism

As from above, this attack is looking for RST's from the host to find out which ports are listening.

Here's a representative packet from tcpdump, with 10.10.10.113 as the source.

```
% tcpdump -nvXr host-10.10.10.113 -c 1 'src host 10.10.10.113'
12:57:23.130647 10.10.10.113.59194 > 192.168.17.68.1027: . [tcp sum ok] win 4096 (ttl
51, id 1556, len 40)
0x0000   4500 0028 0614 0000 3306 9b55 0a0a 0a71        E..(....3..U...q
0x0010   c0a8 1144 e73a 0403 0000 0000 0000 0000        ...D.:..........
0x0020   5000 1000 ce3f 0000 5555 5555 5555             P....?..UUUUUU
```

All TCP packets in this trace from 10.10.10.113 have the string "UUUUU" in their payload. This string may be helpful in identifying the tool which created the packets, however, I have not run across the specific tool.

Also, the source port of the TCP packets sent by 10.10.10.113 indicates that a tool was used to craft & send the packets. As you can see below, 2 source ports are associated over 18000 packets, and the remaining 64 packets are spread across 12 other source ports. Also, the range of source ports is contiguous.

```
% tcpdump -nr host-10.10.10.113 'src host 10.10.10.113 and tcp' | cut -d' ' -f2 | cut
-d. -f5 | sort | uniq -c | sort -n
   4 59196
   4 59197
   4 59198
   4 59199
   4 59200
   4 59205
   4 59206
   4 59207
   8 59201
   8 59202
```

```
   8 59203
   8 59204
9076 59195
9090 59194
```

By looking at packets to 10.10.10.113, we should be able to get an idea of which ports respond with a RST packet.

```
%  tcpdump -nvr host-10.10.10.113 'dst host 10.10.10.113 and tcp'
12:57:23.146706 192.168.17.68.443 > 10.10.10.113.59194: R [tcp sum ok] 0:0(0) ack 0
win 0 (ttl 125, id 51235, len 40)
12:57:40.925834 192.168.17.68.22 > 10.10.10.113.59194: R [tcp sum ok] 0:0(0) ack 0 win
0 (ttl 125, id 51241, len 40)
[snip - many lines omitted for brevity]
13:15:51.854980 192.168.17.135.53 > 10.10.10.113.59194: R [tcp sum ok] 0:0(0) ack 0
win 0 (DF) (ttl 252, id 0, len 40)
13:16:10.132772 192.168.17.135.20 > 10.10.10.113.59194: R [tcp sum ok] 0:0(0) ack 0
win 0 (DF) (ttl 252, id 0, len 40)
```

We can see by the "R" in tcpdump's Flags position, that these packets are all RSTs sent from the hosts being scanned back to 10.10.10.113.

Filtering this down a bit, we find out which ports are involved:

```
% tcpdump -nvr host-10.10.10.113 'dst host 10.10.10.113 and tcp' | cut -d' ' -f2 |cut
-d. -f5 | sort -u -n
20
21
22
23
25
53
80
443
```

These are the ports commonly associated with ftp-data(20), ftp(21), ssh(22), telnet(23), smtp(25), dns(53), http(80) and https(443) [8]. This doesn't seem right... the only ports sending RST's are the one that we would expect to be open on a server. Looking at securityfocus [7] we find out one possible reason:

> "When a Microsoft system receives an xmas packet, it will respond with a RST/ACK, regardless of whether or not the port is open or closed. Since an xmas scan deems that a response from the remote host indicates a closed port, a Microsoft system is invulnerable to the xmas tree scan, as it will show all ports closed regardless of their state. [...]
>
> So in the light, it also falls prey to the same limitations that its predecessor does, [FIN and NULL Scans] cannot be used against non RFC compliant systems, such as Microsoft Windows."

Using NMAP[13], lets do a test in a lab environment (FreeBSD, port 22 open, port 80 closed).

```
target% tcpdump -ni fxp0 -w /tmp/nmap-sN-3.5 'host 192.168.77.69' &
```

```
attack% nmap -sN -P0 -p 22,80 192.168.77.69
target% tcpdump -nvr /tmp/nmap-sN-3.5 'port (22 or 80)'
13:45:30.689375 192.168.77.77.56182 > 192.168.77.69.80: . [tcp sum ok] ack 2925499678
win 3072 (ttl 42, id 30489)
13:45:30.689399 192.168.77.69.80 > 192.168.77.77.56182: R [tcp sum ok]
2925499678:2925499678(0) win 0 (DF) (ttl 64, id 46020)
13:45:38.945517 192.168.77.77.56162 > 192.168.77.69.22: . [tcp sum ok] win 1024 (ttl
56, id 40610)
13:45:39.265398 192.168.77.77.56163 > 192.168.77.69.22: . [tcp sum ok] win 1024 (ttl
48, id 23517)
```

We see that the FreeBSD TCP/IP stack sends back RST's for port 80, but not for 22. This correctly indicates that port 80 is closed, and port 22 is open on this host, per RFC 793.

Further testing indicates that a Windows host will respond with a RST when a NULL SCAN packet is sent to a port which is OPEN, for example port 445.

```
20:40:30.929918 192.168.77.77.37262 > 192.168.77.68.445: . [tcp sum ok] ack 2425501278
win 2048 (ttl 37, id 49779, len 40)
20:40:30.930890 192.168.77.68.445 > 192.168.77.77.37262: R [tcp sum ok]
2425501278:2425501278(0) win 0 (ttl 124, id 64938, len 40)
```

Based on the large number of packets being sent to such a wide variety of ports, and on the limited number of RST packets being sent out, it is very likely that these hosts are running some flavor of Windows, or another non-RFC compliant operating system. To confirm this, we can look at traffic with 192.168.17.68 as the source, and the RST flag not set. This should tell us wether the ports sending RST's are doing so for all communications or only for 10.10.10.113 .

```
% tcpdump -nr ../2003.12.15-all 'src host 192.168.17.68 and tcp[13] & 0x04 != 0x04' |
wc -l
      42
% tcpdump -nr ../2003.12.15-all 'src host 192.168.17.68 and tcp[13] & 0x04 != 0x04'
12:59:53.671905 192.168.17.68.80 > 10.10.10.165.1085: . ack 3841034003 win 0 (DF)
13:03:24.715660 192.168.17.68.443 > 10.10.10.112.43586: S 676105396:676105396(0) ack
656953651 win 17520 <mss 1460,nop,wscale 0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
13:03:27.907965 192.168.17.68.25 > 10.10.10.112.43740: S 677075485:677075485(0) ack
657908839 win 17520 <mss 1460,nop,wscale 0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
[snipped for brevity]
13:19:30.888709 192.168.17.68.21 > 10.10.10.165.3829: S 918526590:918526590(0) ack
1007629249 win 17520 <mss 1460,nop,nop,sackOK> (DF)
13:19:40.979699 192.168.17.68.21 > 10.10.10.165.3829: . ack 2 win 17520 (DF)
13:20:21.688109 192.168.17.68.80 > 10.10.10.165.1085: . ack 1 win 0 (DF)
[snipped again]
```

Based on this, we can see that ports 80, 443, 25, and 21 are all being used to communicate with other hosts. Thus, this host is explicitly denying 10.10.10.113 access to the ports listed above, by sending RST packets.

Using tcpdump, we can get an idea of how much these 3 hosts send to other hosts, ignoring RST packets.

```
%       tcpdump -nr ../2003.12.15-all 'src host 192.168.17.135 and tcp[13] & 0x04 !=
```

```
0x04' | cut -d' ' -f2 | sort | uniq -c | sort -n
   1 192.168.17.135.23
   1 192.168.17.135.443
   1 192.168.17.135.48486
   1 192.168.17.135.48489
[snip 18 similar lines, all for port 48xxx]
   4 192.168.17.135.48512
  13 192.168.17.135.80
  45 192.168.17.135.20
  82 192.168.17.135.25
 144 192.168.17.135.21
% tcpdump -nr ../2003.12.15-all 'src host 192.168.17.68 and tcp[13] & 0x04 != 0x04' |
cut -d' ' -f2 | sort | uniq -c | sort -n
   2 192.168.17.68.21
   2 192.168.17.68.53
   4 192.168.17.68.25
  10 192.168.17.68.80
  24 192.168.17.68.443
% tcpdump -nr ../2003.12.15-all 'src host 192.168.17.129 and tcp[13] & 0x04 != 0x04' |
cut -d' ' -f2 | sort | uniq -c | sort -n
   9 192.168.17.129.22
```

The large number of connections from high number ports on 192.168.17.135 is consistent with an FTP server in "passive" mode - when the client requests data, the server sends a message to the client via the existing TCP connection telling the client to connect to some ephemeral port. The client will then connect to the server again, this time on the previously agreed upon high-numbered port. [14]

Trying to determine the operating system of the hosts involved, we use p0f [3] to examine the RST signatures.

-R => look at RST signatures

-U => ignore Unknown hosts

-N => No details (distances, link media)

-l => one line per hosts,

-s [tcpdump snapshot] => read packets from the specified file

% p0f -RUNls host-10.10.10.113 | sort -u

```
p0f - passive os fingerprinting utility, version 2.0.3
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (RST+) on 'host-10.10.10.113', 14 sigs (0 generic), rule: 'all'.
[+] End of input file.
192.168.17.129:20 - Linux recent 2.4 (refused)
192.168.17.68:20 - Windows XP/2000 (refused)
```

Since we did not find any information in the RST packets to identify 192.168.17.135, we run p0f again, this time using SYN mode and looking at the full dataset:

```
% p0f -UNls ../2003.12.15-all | sort -u | grep 192.168.17.135
p0f - passive os fingerprinting utility, version 2.0.3
```

```
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on '../2003.12.15-all', 206 sigs (12 generic), rule: 'all'.
[+] End of input file.
192.168.17.135:20 - Linux 2.4/2.6 [low delay] (up: 14 hrs)
192.168.17.135:48486 - Linux 2.4/2.6 (up: 13 hrs)
192.168.17.135:48511 - Linux 2.4/2.6 (up: 13 hrs)
192.168.17.135:48513 - Linux 2.4/2.6 (up: 14 hrs)
192.168.17.135:48517 - Linux 2.4/2.6 (up: 14 hrs)
192.168.17.135:48519 - Linux 2.4/2.6 (up: 14 hrs)
192.168.17.135:48520 - Linux 2.4/2.6 (up: 14 hrs)
192.168.17.135:48521 - Linux 2.4/2.6 (up: 14 hrs)
192.168.17.135:48522 - Linux 2.4/2.6 (up: 14 hrs)
```

Connecting all of the pieces together, we have 3 hosts;

- p0f identifies 2 as Linux, 1 as Windows XP/2000;
- All 3 send RST packets for ports which are OPEN; and
- All 3 use the same ethernet MAC address, in a range assigned to VMWare.

This is likely a set of 3 Virtual Machines ("guests"), running on a single VMWare "host". Further, it appears that the "host" has a firewall enabled, and configured to send RST packets in response to disallowed connections. The behavior of the firewall does not give away the "host" OS in this case.

6. Correlations

Since the IP addresses have been obfuscated by placing them into address blocks which are reserved for special purposes, information from whois will not give us any help in linking this attacker to other attacks seen on the internet[5].

Another detect done by Antony Gummery discusses another type of NMAP scan and can be found at:

http://cert.uni-tuttgart.de/archive/intrusions/2003/03/msg00300.html

The Snort website describes the snort alert in more detail at: http://www.snort.org/snort-db/sid.html?sid=623

The arachNIDS "Intrusion Event Database" rehashes much of what's in the snort.org signature description at http://www.whitehats.com/info/IDS4

In an article at Security Focus [7] Randy Williams discusses various types of scanning techniques, and their downfalls. From the article:

> "One disadvantage to this technique was partially discussed in the above paragraph, the system's tcp/ip implementation MUST correspond with RFC standard 793. As such, this method will not work against any current version of Microsoft Windows. [...] When a Microsoft system receives an xmas packet, it will respond with a RST/ACK, regardless of whether or not the port is open or closed. "

Randy also explains that FIN and NULL scans are also limited by the same problems with non-RFC-compliant TCP stacks. Finally, the article mentions that these scans are typically ineffective against firewalled hosts, as the firewalls often drop the packets silently before the targeted host ever sees them.

7. Evidence of active targeting

Since the attacking host (10.10.10.113) only attempts to make TCP connections to 3 hosts, on a large number of ports, I consider this to be active targeting. Also, since the services that appear to be running on the 3 target hosts are those likely to be exposed to the internet for a company to make information available to its customers, the attacker seems to have some information about this target to begin with.

How many packets does 10.10.10.113 send to each of the targeted hosts?

```
% tcpdump -nr host-10.10.10.113 'src 10.10.10.113 and tcp' | cut -d' ' -f4 | cut -d.
-f1-4 | sort | uniq -c | sort -n
5291 192.168.17.135
6315 192.168.17.68
6624 192.168.17.129
```

Using tcpdump, we see that the attacker sweeps across a large number of ports per host:

```
% tcpdump -nr ../2003.12.15-all 'src 10.10.10.113 and tcp and dst 192.168.17.129' |
cut -d' ' -f4 | cut -d. -f5 | sort | uniq | wc -l
    1657
% tcpdump -nr ../2003.12.15-all 'src 10.10.10.113 and tcp and dst 192.168.17.135' |
cut -d' ' -f4 | cut -d. -f5 | sort | uniq | wc -l
    1657
% tcpdump -nr ../2003.12.15-all 'src 10.10.10.113 and tcp and dst 192.168.17.68' | cut
-d' ' -f4 | cut -d. -f5 | sort | uniq | wc -l
    1657
```

Comparing the output of the 3 commands above (before cutting & sorting) reveals that the same 1657 ports were attacked on each host, spread fairly evenly between hosts and ports.

8. Severity

Per the SANS guidelines, the formula for severity is:

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

1 is lowest, 5 highest.

Criticality: 5

Based on the types of services the 3 target hosts appear to be running, the 3 targets appear to be offering key services for the organization.

Lethality: 1

Based on the few RST's that were sent out, and the large number of probe packets sent in, the only information the attacker has implies that the ports are closed. Also, since we see that the attacker made the scans repeatedly over a short period of time, the hosts being attacked were not crashed by these attacks.

System countermeasures: 5

All three hosts which were scanned by 10.10.10.113 appear to have firewalls which prevent this attacker from connecting. To me, this implies that someone responsible for these systems is pro-actively thinking about how to secure the systems, and taking necessary actions.

Network countermeasures: 4

This network appears to have IDS & firewalls in use, and policies regarding what resources are available to whom have been configured, and are being enforced.

Severity = (5 + 1)(5 + 4) = 54

Using the formula from http://www.giac.org/ID_assignment_guidelines.php

Severity = (5 + 1) - (5 + 4) = -3

9. Defensive recommendations

Keep up the good work. Keep monitoring the IDS logs, and any firewall logs. Keep the operating system and application software on the systems up to date.

Test your network filtering devices on a regular basis to ensure that their configuration is correct, and to detect changes in the configuration. It is feasible that this attack could have been part of this type of testing.

10. Multiple choice test question

When examining RST packets leaving you network, which of the following is safe to assume:

(a) RST packets are only sent to end existing connections.

(b) Only Windows will send RST packets for ports that are open.

(c) Outside attackers can gain no information about your network from RST packets.

(d) None of the above.

Answer: (d)

Incorrect:

(a) - Windows will respond to a NULL SCAN packet sent to an open port with a RST, before a TCP 3-way handshake has completed.

(b) - Many firewalls can be configured to send RSTs in response to packets which are being blocked.

(c) - Due to the many, varied implementations of TCP, any stimulus - response test that sends packets out of your network can give an attacker information about your network.

References for Detect #2

[1] tcpdump - http://www.tcpdump.org/

[2] nmap - http://www.insecure.org/nmap/

[3] p0f - http://lcamtuf.coredump.cx/p0f.shtml

[4] "Transmission Control Protocol" - http://www.faqs.org/rfcs/rfc793.html

[5] "Address Allocation for Private Internets" - http://www.faqs.org/rfcs/rfc1918.html

[6] mergecap - comes with ethereal - http://www.ethereal.com/

[7] "Low-Level Enumeration With TCP/IP" - http://www.securityfocus.com/guest/24226

[8] PORT NUMBERS - http://www.iana.org/assignments/port-numbers

[9] Snort - http://www.snort.org

[10] SnortSnarf - http://www.silicondefense.com/software/snortsnarf/

[11] http://www.snort.org/snort-db/sid.html?sid=623

[12] http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids4

[13] NMap - http://www.insecure.org/nmap/

[14] FreeBSD Man Pages - http://www.freebsd.org/cgi/man.cgi?query=ftp

[15] VMWare – http://www.vmware.com/products/desktop/ws_features.html

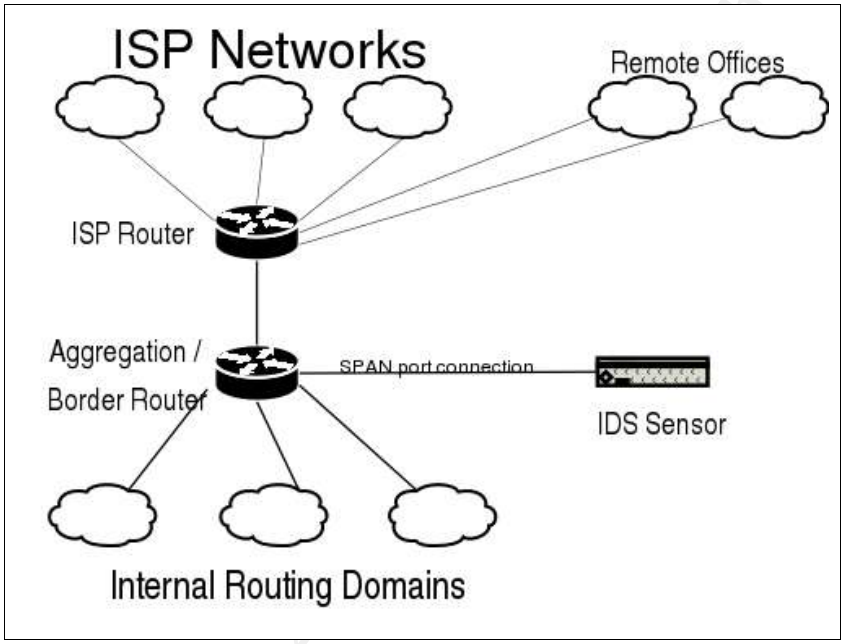

```
[sfa-summary]
#!/bin/tcsh
#
# generate a summary of the snort alert file passed
# as argv[1]
#
setenv ALERT $1
cat $ALERT | grep -E "^\[\*\*\]" | sort | uniq -c | sort -n > $ALERT.summary
# EOF
```

Detect #3 – Warez Printer

This detect was not posted to the intrusions@incidents.org email list.

1. Source of trace

Data used for this trace came from a snort+ACID instance monitoring the border of a network similar to the one shown below. In the simplified diagram below, the snort sensor is connected to the aggregation router, on a port mirroring the link to the ISP router.



2. Detect was generated by

The detect was generated by snort version 2.0.0, logging to a database (Caswel). We use a mix of signatures hand-written for our environment, and signatures from the snort development team. The sensor launches snort using the command:

```
% /usr/pkg/bin/snort -c /path/to/snort.conf -NoDd -i wm1
```

to run snort with logging disabled (-N), with “pass” rules evaluated first (-o), as a daemon or background process (-D), with application layer data dumped (-d), and listening to interface wm1 (-i wm1).

For analyzing the large number of alerts generated by snort, we use Roman Danyliw's ACID, version v0.9.6b24 from sourceforge.net's CVS server. Logging into ACID, and clicking the link to look at the most recent 50 alerts showed that 5 of the 7 most recent alerts were related to various FTP abuses commonly found when warez sites are created.

Signature	Classification	Total #	Sensor #	Src. Addr.	Dest. Addr.
[snort] POLICY FTP 'MKD / ' possible warez site	misc-activity	3677	1	121	16
[snort] POLICY FTP 'CWD / ' possible warez site	misc-activity	5844	1	251	1396
HTA MIME Content in HTML	*unclassified*	34638	1	354	3842
POSSIBLE NEW IE EXPLOIT: MHTML	attempted-admin	567	1	31	376
[snort] POLICY FTP 'CWD ' possible warez site	misc-activity	36	1	15	11
[snort] POLICY FTP 'MKD ' possible warez site	misc-activity	8	1	5	3
[snort] POLICY FTP 'MKD .' possible warez site	misc-activity	274	1	13	7
[snort] (snort_decoder): Experimental Tcp Options found	*unclassified*	2651	1	712	834

This list of alerts is listed by time (not shown in the screenshot), with the most recent first. The “POLICY FTP 'MKD / ' possible warez site” alert (top) is not the most frequent, but the ratio of total alerts (3677) to destination addresses (16) stands out. To look at this one in more depth, we can click on the link for the total number for this alert, and get a list of all of the alerts matching the specific signature. From this page, we can choose to show a summary by destination address, similar to the one in the screenshot below.

ACID **Unique Destination Address(es)**

Home | Search | AG Maintenance

[Back]

Queried DB on : Sun April 11, 2004 18:37:24

Meta Criteria	Signature "[snort] POLICY FTP 'MKD / ' possible warez site" ...clear...
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Displaying 1-16 of 16 total

Dest IP address	FQDN	Sensor #	Total #	Unique Alerts	Src. Addr.
[illegible].4	*Unable to resolve address*	1	8	1	1
[illegible].9	[illegible]	1	2712	1	3
[illegible].13	[illegible]	1	298	1	1
[illegible].197	[illegible]	1	1	1	1
[illegible].248	[illegible]	1	1	1	1
[illegible].158	*Unable to resolve address*	1	33	1	28
[illegible].4	[illegible]	1	8	1	6
[illegible].10	[illegible]	1	30	1	22
[illegible].39	[illegible]	1	28	1	25
[illegible].185	[illegible]	1	1	1	1
[illegible].25	[illegible]	1	36	1	1
[illegible].24	[illegible]	1	10	1	10
[illegible].60	[illegible]	1	115	1	3
[illegible].52	*Unable to resolve address*	1	23	1	2
[illegible].64	*Unable to resolve address*	1	221	1	10
[illegible].249	*Unable to resolve address*	1	152	1	5

Further investigation, predominately with a generic FTP client and ping, revealed that most of these are legitimate FTP servers, or are machines which are dynamically configured. One, however, was a printer.

```
% ftp MY.NET.IP.248
Connected to MY.NET.IP.248.
220 JD FTP Server Ready
Name (MY.NET.IP.248:nouser): anonymous
331 Username OK, send identity (email address) as password.
Password:
230- Hewlett-Packard J3113A FTP Server Version 1.0
```

```
Directory:     Description:
---------------------------
PORT1         Print to port 1 HP LaserJet 4050 Series

To print a file, use the command: put <filename> [portx]
or 'cd' to a desired port and use: put <filename>.

Ready to print to PORT1

230 User logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
500 Command unrecognized or unimplemented
227 Entering Passive Mode (MY.NET.IP,248,20,2)
150 Openining data connection.
d-w--w--w-   2 JetDirect  public         512 Aug  1  1999 PORT1
226 Transfer complete.
Ftp>
```

Not only is it an FTP server, but it is world-writeable!

3. Probability the source address was spoofed

As this alert is generated by an established FTP session using TCP, the liklihood that the source address was spoofed is near zero. The other alerts which have this destination are all FTP related, and appear to be related to various activities related to pub-scanning().

4. Description of attack

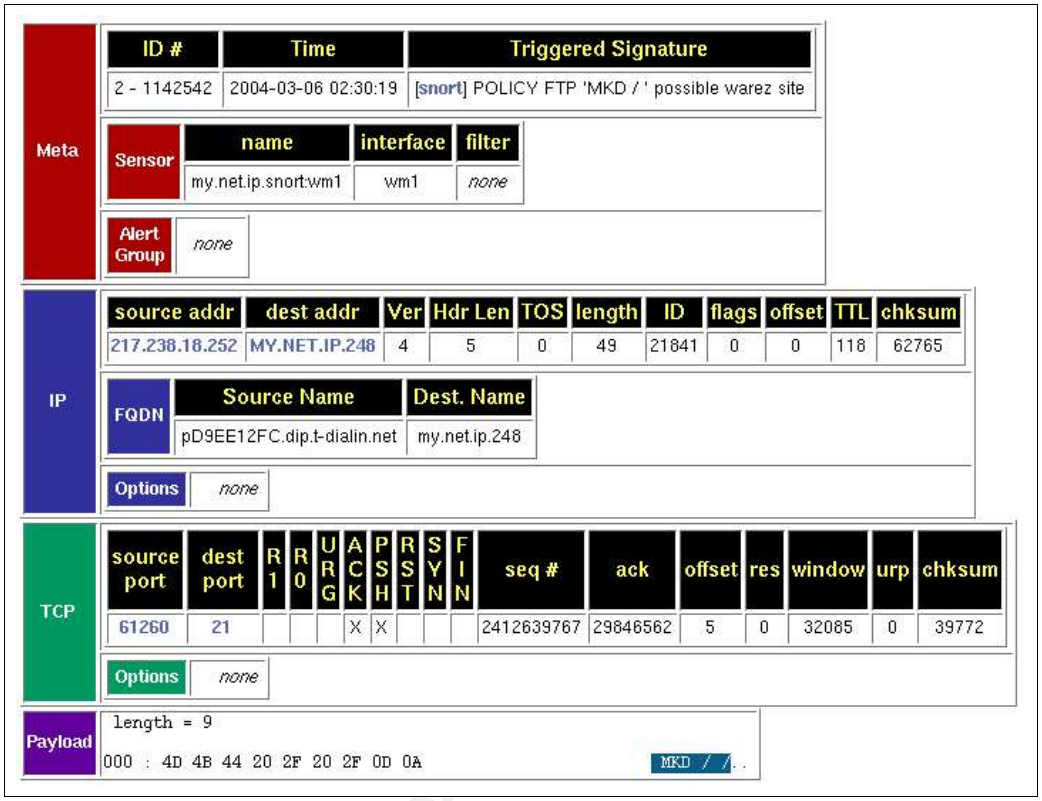
The most basic description of the attack can be found by looking at the rule which generated the alert. The snort signature is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
       (msg:"POLICY FTP 'MKD / ' possible warez site";
       flow:to_server,established; content:"MKD"; nocase; content:"/ ";
       distance:1; classtype:misc-activity; sid:554; rev:6;)
```

according to http://www.snort.org/snort-db/sid.html?sid=554 and is found in the warez.rules file on our installation.

The first line directs snort to use this signature only when examining TCP packets destined for FTP servers on $HOME_NET, and to generate an alert for any packet that matches the remaining criteria. The second line defines the message to display when a packet does match. The final 2 line direct snort to look for the content “MKD”, ignoring case, and to look for “/ “ at lease one byte after the end of “MKD”. Also, for this rule to generate an alert, the packet must be part on an established TCP connection, and must be traveling to the server. In other words, snort will generate an alert anytime an FTP client directs the server to create a directory with leading spaces in the name, for example “/pub/ /imK00L/” or “/ /onespaceb4/”.

Using ACID to look at the packet that actually generated the alert, we can find out what the attacker tried to name the directory.



Meta

ID #	Time	Triggered Signature
2 - 1142542	2004-03-06 02:30:19	[snort] POLICY FTP 'MKD / ' possible warez site

Sensor	name	interface	filter
	my.net.ip.snort:wm1	wm1	*none*

Alert Group	*none*

IP

source addr	dest addr	Ver	Hdr Len	TOS	length	ID	flags	offset	TTL	chksum
217.238.18.252	MY.NET.IP.248	4	5	0	49	21841	0	0	118	62765

FQDN	Source Name	Dest. Name
	pD9EE12FC.dip.t-dialin.net	my.net.ip.248

Options	*none*

TCP

source port	dest port	R1	R0	URG	ACK	PSH	RST	SYN	FIN	seq #	ack	offset	res	window	urp	chksum
61260	21				X	X				2412639767	29846562	5	0	32085	0	39772

Options	*none*

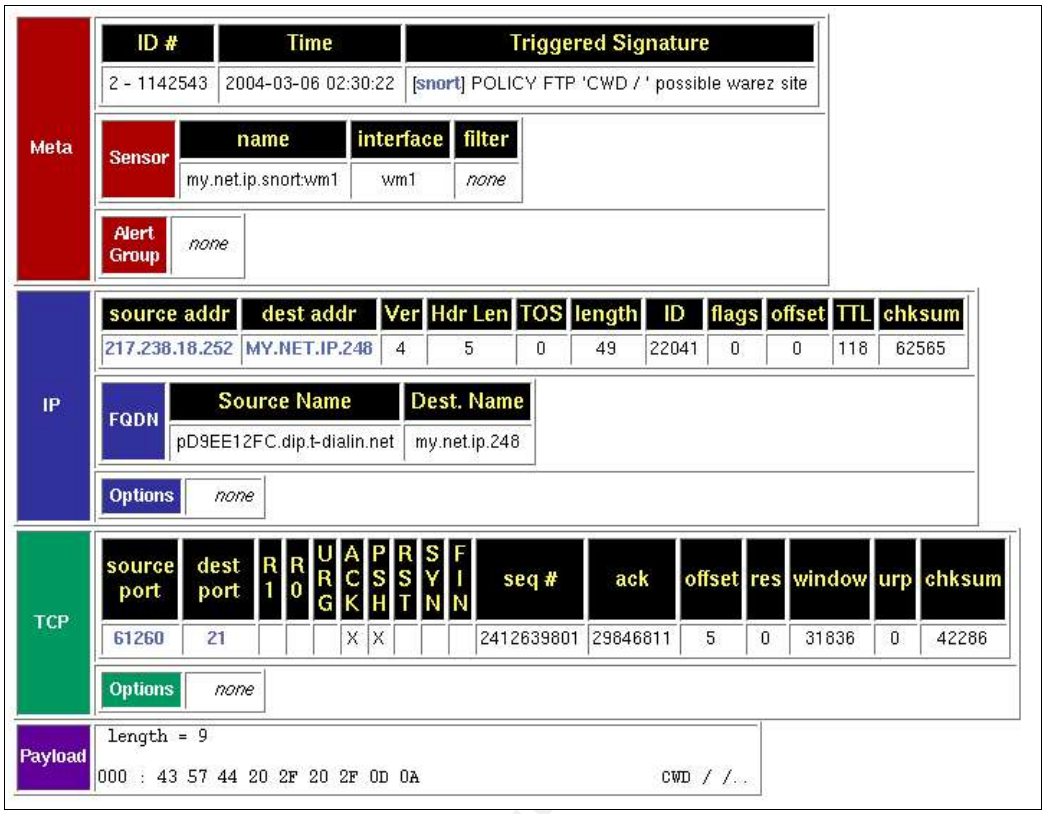
Payload

```
length = 9
000 : 4D 4B 44 20 2F 20 2F 0D 0A                MKD / /..
```

In the “Payload” area, highlighted in blue, we see that the attacker issued a “MKD / /” command to the FTP server. ACID marks the two characters that follow as non-printable characters by displaying them as “.”'s. The area to the left shows the hexadecimal values to be 0x0D and 0x0A, representing the carriage return and line feed, respectively, in ASCII (foldoc).

As the attacker tried only to create a single directory named as a single space, we can infer that the creation of the directory failed. Had it succeeded, the attacker would likely have created other directories below the new directory to hide various files, and each of those following directories would have triggered the same rule.

The only other alert generated by this attacker was generated when they attempted to change directory into the directory “/ “ which they had just attempted to create.



Meta	ID #	Time	Triggered Signature
	2 - 1142543	2004-03-06 02:30:22	[snort] POLICY FTP 'CWD / ' possible warez site

Sensor	name	interface	filter
	my.net.ip.snort:wm1	wm1	none

Alert Group	none

IP	source addr	dest addr	Ver	Hdr Len	TOS	length	ID	flags	offset	TTL	chksum
	217.238.18.252	MY.NET.IP.248	4	5	0	49	22041	0	0	118	62565

FQDN	Source Name	Dest. Name
	pD9EE12FC.dip.t-dialin.net	my.net.ip.248

Options	none

TCP	source port	dest port	R1	R0	URG	ACK	PSH	RST	SYN	FIN	seq #	ack	offset	res	window	urp	chksum
	61260	21				X	X				2412639801	29846811	5	0	31836	0	42286

Options	none

Payload	length = 9
	000 : 43 57 44 20 2F 20 2F 0D 0A CWD / /..

Also, we can see that the attacker's source port did not change. This reinforces the earlier statement that the source address is not being spoofed, as it indicates the the MKD and CWD commands are both part of the same TCP session.

Further, based on the sequence (SEQ) numbers, and the acknowledgment (ACK) numbers, we can determine that there were other packets sent by this host, as part of this session, that snort did not generate alerts for. The payload length of the first packet is 9 bytes. Adding this to the SEQ number of the first packet (2412639767) gives us 2412639776. Since the SEQ number of the second packet is 2412639801, we can infer that there was 25 bytes of data transmitted which snort did not see (Bryant).

Doing a similar calculations on the ACK numbers, we find that the printer sent a total of 249 bytes in response to the first packet and any unseen packets. Finally, based on the difference between the IP ID's of the packets, the attacker transmitted 199 other packets, across all IP connections, in the 3 seconds between the 2 packets logged by snort.

5. Attack mechanism

Starting the name of a directory with a space is likely done to make it difficult for the directory to be accessed or deleted. Operating systems that reserve certain names can be abused further by tricking the FTP server to

use reserved names for the directories (Hawk). The normal tools will often refuse to delete the directories on these hosts because of the reserved name.

Once this type of directory has been created, and the attacker has verified that they are able to change into the new directory, they know that they will be able to store data there. Likely next steps would be to determine the performance of the FTP server & its network connection, and to load data onto the server (Obscure). Also, this type of attacker is likely to try to find out how much space is available on the FTP server. It has been my experience that a misconfigured FTP server, with large disks (hundreds of gigabytes) and a fast connection, can be found by a disk full error in a surprisingly short time.

6. Correlations

Security Focus lists a vulnerability for HP Jetdirect printers in bid 1491 from July 2001 which will cause the printer to become unusable until being powered off & back on by issuing a specific FTP command. Also, MITRE has assigned that same issue the identifier CVE-2000-0636, and updated it last on April 2 of 2003.

Also, according to HP Jetdirect Firmware History,

> "There was a couple of denial of service attacks that were successful against HP Jetdirect's FTP implementation that were fixed. Also, a malicious user could launch proxy attacks via HP Jetdirect's FTP server. "

Both of these were fixed in version x.08.32 of the firmware.

As this printer is obviously a potential target for resource abuse, and denial of service of the printer, the level of external access needs to be evaluated. Attempting to connect from an external computer, I was able to reach the printer using FTP, implying that anyone in the world could as well.

Also we must determine what other alerts have been triggered for this host. Using ACID, from where we left off last, we click on the IP address we're interested in. On the resulting page, we click on the number in the "Occurrences as Dest." column. Then, by clicking on the "... clear ..." link in the "Meta Criteria" box, we get a list of all of the alerts with this IP as the destination, similar to the one below.

Displaying alerts 1-13 of 13 total

ID	Signature	Timestamp	Source Address	Dest. Address	Layer 4 Proto
#0-(2-1221857)	[arachnids][snort] INFO FTP No Password	2004-04-05 10:32:15	213.182.119.29:3173	[redacted]:21	TCP
#1-(2-1196104)	[snort] POLICY FTP 'CWD / ' possible warez site	2004-03-23 05:54:02	83.112.16.164:4646	[redacted]:21	TCP
#2-(2-1176723)	[snort] POLICY FTP 'CWD / ' possible warez site	2004-03-16 04:31:28	81.251.91.89:2207	[redacted]:21	TCP
#3-(2-1182267)	[arachnids][snort] INFO FTP No Password	2004-03-18 03:31:35	213.54.15.202:1447	[redacted]:21	TCP
#4-(2-1154554)	[arachnids][snort] INFO FTP No Password	2004-03-09 04:17:04	194.130.97.162:1725	[redacted]:21	TCP
#5-(2-1154553)	[arachnids][snort] INFO FTP No Password	2004-03-09 04:17:04	194.130.97.162:1722	[redacted]:21	TCP
#6-(2-1154551)	[arachnids][snort] INFO FTP No Password	2004-03-09 04:17:01	194.130.97.162:1720	[redacted]:21	TCP
#7-(2-1154550)	[arachnids][snort] INFO FTP No Password	2004-03-09 04:17:01	194.130.97.162:1718	[redacted]:21	TCP
#8-(2-1146523)	[arachnids][snort] INFO FTP No Password	2004-03-07 11:12:12	217.224.135.163:2216	[redacted]:21	TCP
#9-(2-1142543)	[snort] POLICY FTP 'CWD / ' possible warez site	2004-03-06 02:30:22	217.238.18.252:61260	[redacted]:21	TCP
#10-(2-1142542)	[snort] POLICY FTP 'MKD / ' possible warez site	2004-03-06 02:30:19	217.238.18.252:61260	[redacted]:21	TCP
#11-(2-1131333)	[arachnids][snort] INFO FTP No Password	2004-02-29 16:27:39	80.145.124.57:12531	[redacted]:21	TCP
#12-(2-506914)	[snort] POLICY FTP 'CWD / ' possible warez site	2004-01-07 17:06:12	82.65.184.49:3972	[redacted]:21	TCP

As we can see from this list, there has been suspicious FTP traffic to this device from a small number of hosts since the earliest alert in the database. Also, considering the alerts span nearly 3 months, and the device has still not been secured properly, I would guess that it is unlikely that this printer has been being abused regularly.

In a thread on the intrusions@incidents.org mailing list, one member posted some logs relating to a printer that was being scanned for ftp vulnerabilities in August of 2002 from a host, named “p508F7435.dip.t-dialin.net”, in the same domain as the attacker seen by our printer (Carey).

Searching dshield.org for 217.238.18.252 reveals that the IP is owned by “Deutsche Telekom AG, Internet service provider” and is part of the 217.224.0.0/11 network (dshield). There were no reports on dshield fieled against this specific address, but the “dialin” in the host name implies that the addresses for any given customer may change frequently.

7. Evidence of active targeting

Based on the small number of hosts attaching to this IP, I don't belive that it is actively being targeted. However, all of the alerts generated for this destination host are related to the FTP service. The source IP also generated alerts for attempting to enumerate accounts on 28 other systems, likely using the command “finger 0@{ipaddress}” or an equivalent script. This could be an information gathering process to find possible targets, as discussed in http://cgi.nessus.org/plugins/dump.php3?id=10069 (*Deraison*) and http://xforce.iss.net/xforce/xfdb/8378 (ISS) among others.

8. Severity

Per the SANS assignment page, the formula for severity is:

severity = (criticality + lethality)(system countermeasures + network countermeasures) with 1 being the lowest value, and 5 the highest.

Criticality: 1

As the attacker was targeting a workgroup printer, the criticality of this attack is very low in scope of the entire organization. However, large,

specialty, or high-volume printers might indicate a higher criticality.

Lethality: 3

As there was no indication of warez being stored on the printer's FTP site, the attack was not successful, and thus a low score for lethality. Since the FTP service is available to anyone with an IP connection, a paper & toner denial of service could be accomplished trivially by sending random documents to the printer at times when people are not likely to be around.

System countermeasures: 2

The printer is designed to be sufficient for printing, while not being attractive for use as a general purpose FTP server. However, this system had not been secured in any way... telnet was enabled, without a password; the web management interface was also enabled, without a password, and the FTP service had been turned ON. Fortunately, the printer has the latest firmware, which fixes some of the worst bugs (proxy abuse, crashing the printer).

Network countermeasures: 1

The network area in which this printer is connected has not implemented any access control mechanisms. Controlling the printer through a departmental print server would provide a mechanism for controlling access and doing accounting if needed. Configuring passwords for the management accounts would help prevent an external party reconfiguring the printer.

Severity = (1 + 3)(2 +1) = 12

Using the formula from http://www.giac.org/ID_assignment_guidelines.php

Severity = (1 + 3) – (2 +1) = 1

9. Defensive recommendations

There are many things which should be done on this printer:

- disable FTP, if operationally possible;
- limit access to the printer to either the local subnet by removing the default gateway configuration, or by using a firewall;
- move the printer to a non-routed network segment, such as one using addresses in the RFC-1918 space;
- keep the printer's firmware up to date;
- set passwords on the management accounts.

While additional passwords are considered by some to be a strain, writing the password down on the inside of the printer (toner door, paper tray) is an easy way to find it again, while not preventing unauthorized access to the

printer. Since the password can often be reset from the printer's control display, there is no additional risk having the password available to anyone who has physical access.

10. Multiple choice test question

The following log snippet came from the SANS practical logs located at http://www.incidents.org/logs/oos/oos_report_040303.

```
03/07-04:39:49.032184 68.54.84.49:56552 -> MY.NET.6.7:110
TCP TTL:51 TOS:0x0 ID:59855 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x486126FF  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 605736752 0 NOP WS: 0
--
03/07-04:41:56.525619 68.54.84.49:56554 -> MY.NET.6.7:110
TCP TTL:51 TOS:0x0 ID:56764 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x506DDD42  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 605749502 0 NOP WS: 0
--
03/07-04:44:04.577181 68.54.84.49:56556 -> MY.NET.6.7:110
TCP TTL:51 TOS:0x0 ID:33495 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x5881B1D5  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 605762306 0 NOP WS: 0
```

What is likely happening?

A – 68.54.84.49 is attempting to open a connection to MY.NET.6.7 with ECN enabled.

B – 68.54.84.49 is responding to a portscan by MY.NET.6.7.

C – 68.54.84.49 is checking for email on MY.NET.6.7 using IMAP.

D – 68.54.84.49 is signaling MY.NET.6.7 to cut it's congestion window in half, and stating that it will cut it's congestion window in half as well.

Answer: A. Per RFC 3168, this type of packet is used by hosts to indicate that they wish to negotiate ECN for the connection.

B – A proper response to a port scan could include a SYN/ACK packet, or a RESET packet. These do not match.

C – Port 110 is commonly associated with POP, port 143 IMAP.

D – As this is likely an initial SYN (Ack: 0x0), ECN has not been fully negotiated, and thus neither side can indicate congestion.

References for Detect #3

Bryant, Randal E. “Internetworking”. <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15347-s98/public/lectures/lect25.pdf> 21 April 1998. visited 13 April 2004. Carnegie Mellon School of Computer Science. Pittsburgh, PA, USA.

Carey, Steve T. “FTP Scan Variation (printers)”. <http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00177.html>. 16 August 2002.

Caswel, Brian and Roesch, Marty. “snort.org”. <http://www.snort.org/snort-db/sid.html?sid=554>. 2002. visited 12 April 2004.

Danyliw, Roman. “Analysis Console for Intrusion Detection (ACID)”. <http://acidlab.sourceforge.net/>. 3 March 2002. visited 12 April 2004.

*Deraison,Renaud. “*Finger zero at host feature*”. <http://cgi.nessus.org/plugins/dump.php3?id=10069> 1999. visited 12 April 2004.*

Dshield. “DShield - IP info”. <http://www.dshield.org/ipinfo.php?ip=217.238.018.252&summary=Y>. visited 12 April 2004. Euclidian Consulting.

Foldoc. “ASCII character table from FOLDOC”. <http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?ASCII+character+table>. 1993. visited 12 April 2004. supported by Imperial College Department of Computing.

Grundl, Peter. “HP JetDirect Invalid FTP Command DoS Vulnerability”. <http://www.securityfocus.com/bid/1491>. 19 July 2000. visited 12 April 2004.

Hawk. “Net Knowledge Base Forums”. http://www.netknowledgebase.com/tutorials/locking.html visited 12 April 2004.

HP. “Jetdirect Firmware History Part II (Current Products)”. <http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?objectID=bpj02930>. Visited 12 April 2004.

ISS. “Finger service lists unused accounts”. <http://xforce.iss.net/xforce/xfdb/8378> 1994-2004. visited 12 April 2004. Internet Security Systems, Inc.

MITRE. “CVE-2000-0636“. <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0636>. 13 October 2000. visited 12 April 2004.

Obsecure. “When your server ends up a Warez site”. http://eyeonsecurity.org/papers/pubscanning.pdf . 21 June 2001. obscure@eyeonsecurity.net

3.Analyze This!

Executive Summary

When presented with analyzing 5 days of logs containing more than 5 million alert records, I prepared the worst. Fortunately, a large majority of the records represented benign traffic from busy DNS servers doing their job. This presents a challenge for the analysis staff – adjusting the thresholds for these events too high can cause real events to be missed, but leaving them as they are makes finding other events more difficult. These adjustments need to be made cautiously.

Usage of peer to peer file sharing programs on the institution's network presents a significant and imminent threat. The aggressive copyright enforcement tactics employed by the RIAA, MPAA, and their international equivalents encourage technical measures to be taken to prevent copyrighted material from being transferred to or from hosts on MY.NET, or risk legal battles. Additionally, this traffic consumes bandwidth resources which are likely put to better use elsewhere.

Finally, there are hosts showing signs of virus or worm infections, and need to be cleaned up before they spread to the rest of MY.NET or outside.

Files Analyzed

This analysis is based on logs from http://www.incidents.org/logs dated March 3-7, 2004, and named as below:

Directory Date	http://www.incidents.org/ logs/oos	http://www.incidents.org/ logs/scans	http://www.incidents.org/ logs/alerts
3 March	oos_report_040303	scans.040303.gz	alert.040303.gz
4 March	oos_report_040304	scans.040304.gz	alert.040304.gz
5 March	oos_report_040305	scans.040305.gz	alert.040305.gz
6 March	oos_report_040306	scans.040306.gz	alert.040306.gz
7 March	oos_report_040307	scans.040307.gz	alert.040307.gz

The incidents.org logs page states that the IP address obfuscation is performed consistently for batches of logs with the same date in the file's timestamp. To simplify analysis, I will assume that all 5 logs have had the MY.NET addresses generated from the real addresses in the same way, despite the 5 day span of the logfiles' timestamps. As the original data for one log class contained no MY.NET addresses, an initial correlation was made based on the frequency of the first 2 octets across the other log files to determine which address range(s) to sanitize. After selecting the range(s),

the appropriate data was obfuscated by converting the first 2 octets to MY.NET. Other information in the data was used to validate the range selection using DNS and whois queries. The precise details of this process are left ambiguous intentionally.

As events of interest arise, they will be analyzed. An summary will be provided at the end.

Top Talkers

By loading the 3 sets of logs into a database, summary reports become easy to generate. This section looks at the "Top Talkers" in each of the log categories, for different parameters. Since this technique immediately highlights certain types of anomalies, those are investigated as they become evident.

To help future analysts, the SQL queries are included with every table generated from the database, and the database schema is included as an appendix.

Scans Data Set

The first section we will cover is the "scans" dataset, as it is the largest. This scan data is generated by snort's "portscan" preprocessor.

Top 10 Source Addresses by Count

To determine the IP's that are responsible for the most scan alerts, we query the database grouping the results by source IP, and sorting by the number of records in the database for each source address. Since every row in the database has a value in every field, it does not matter which column we use for counting "hits."

```
select distinct src_ip, count(distinct(src_port)) as sports, count(distinct(dest_ip)) as destips,
count(distinct(dest_port)) as dports, count(dest_ip) as hits from scans2 group by src_ip order by
hits desc limit 10
```

src_ip	sports	destips	dports	hits
MY.NET.1.3	87	89756	835	2200824
MY.NET.110.72	5	12924	7473	246700
MY.NET.1.4	11	25458	190	237477
MY.NET.53.169	3684	29405	13878	236868
MY.NET.34.14	31331	1191	2	144770
MY.NET.81.39	3970	141159	28	141836
MY.NET.80.224	3975	112461	3	112616
MY.NET.112.216	464	20299	3412	63536
MY.NET.153.79	3963	10084	1771	55901
MY.NET.97.74	15417	17037	2207	48812

The three lines in the table above are highlighted because of their high ratio of hits to ports. Looking into these further will reveal what is happening on

these 3 systems.

Detect 1

The host MY.NET.110.72 has 5 source ports responsible for roughly a quarter of a million portscan alerts.

```
select distinct src_ip,src_port,count(distinct(dest_port)) as dports, count(distinct(dest_ip)) as
destips, protocol, count(src_port) as hits from scans2 where src_ip='MY.NET.110.72' group by
src_port order by hits desc limit 10
```

src_ip	src_port	dports	destips	protocol	hits
MY.NET.110.72	8767	138	45	UDP	153959
MY.NET.110.72	12203	140	101	UDP	73593
MY.NET.110.72	12300	7327	12836	UDP	18694
MY.NET.110.72	32808	1	1	UDP	413
MY.NET.110.72	32773	1	1	UDP	41

Examining each of these, starting with the least frequent, we find that port 32773 is used by FileNet's <www.filenet.com> Content Manager service, according to portsdb.org (FileNet, Payne). Also, Kurt Seifried lists the port as used for the database server for ToolTalk. FileNet could be using ToolTalk in their product, and both sources would be correct. Since the overall volume is low, it may not matter much. The destination IP for this port, 213.202.254.116, is assigned to TeamSpeak.org – a group that makes software for online game players to talk to each other over the internet.

```
% host 213.202.254.116
116.254.202.213.IN-ADDR.ARPA domain name pointer 213.202.254.116.unitedcolo.de

% whois 213.202.254.116
[snip]
inetnum:      213.202.254.112   - 213.202.254.119
netname:      ETTEL-TEAMSPEAK-NET
descr:        TeamSpeak.org
country:      DE
[snip]
```

The TeamSpeak forum also contains a thread about the destination port for this row – 45647 – explaining that these packets are used to communicate information from various TeamSpeak servers to the primary server to provide improved service for all of the users on the TeamSpeak network (TeamSpeak who).

Port 32808 also has one destination IP: 207.38.8.34, which resolves to master.gamespy.com. The gamespy.com website has reviews, cheats, forums, and other information about a wide variety of games on numerous platforms. The single destination port is 29910, which Microsoft lists as one of the ports needed by the game “Rise of Nations” for multi-player mode using “GameSpy Internet Matchmaking” (Microsoft 820877).

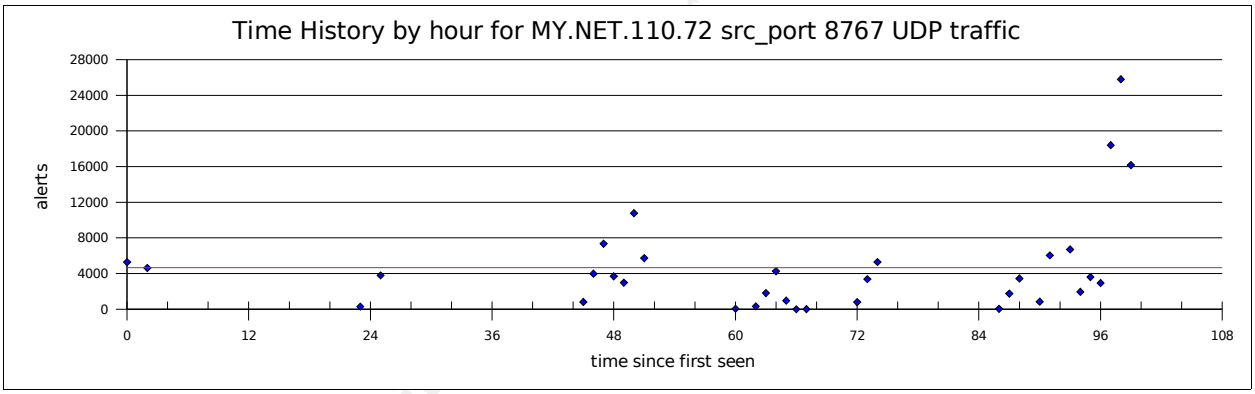
The Castanet Communications site lists ports 12300 and 12203 as the

minimum 2 ports required to play “Medal of Honor” online. It describes port 12300 as a “GameSpy Monitoring” port.

According to GameRanger, port 12203/udp is used by the game “Medal of Honor” for playing over the network (Kevill). Additionally, a GamingForums user states that 12300/udp is required for “Medal of Honor: Spearhead” multi-player modes to work (Sniperstein).

According to the TeamSpeak Forums, port 8767 UDP is the primary port for carrying voice data to & from the server (TeamSpeak Which). Based on their documentation, this host is running a TeamSpeak server. This should be checked against the organization's Acceptable Use Policy, and any needed actions taken. Also, based on the number of alerts, this host is doing a fair amount of traffic on this port.

The chart below shows a time history for this source IP & source port pair, broken down by hour. The horizontal axis shows the number of hours after the first event. The usage pattern is intermittent, with an average rate of under 5000 events per hour, and a maximum over 25000 events per hour.



Detect 2

The ports for MY.NET.34.14 are commonly associated with SMTP (25) and ident (113). Both of these protocols are commonly used by mail servers, and this traffic is likely innocuous, provided MY.NET.34.14 is supposed to be transmitting email. This host needs to be validated as a legitimate SMTP server, and the snort rules adjusted accordingly.

```
select distinct src_ip, count(distinct(src_port)) as sports, count(distinct(dest_ip)) as destips,
dest_port, protocol, count(src_ip) as hits from scans2 WHERE src_ip='MY.NET.34.14' GROUP BY
dest_port order by hits desc
```

src_ip	sports	destips	dest_port	protocol	hits
MY.NET.34.14	31281	509	25	SYN	143788
MY.NET.34.14	882	688	113	SYN	982

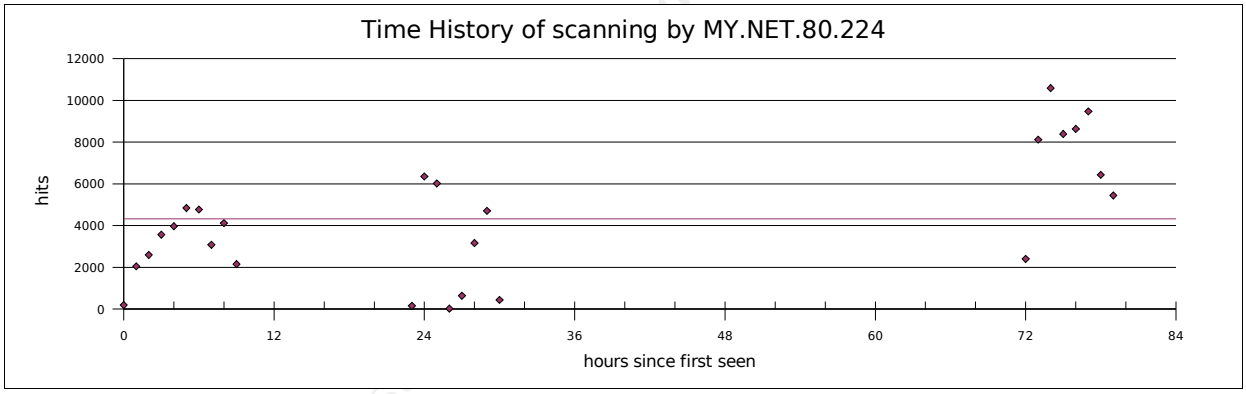
Detect 3

The host MY.NET.80.224 appears to be infected with a worm or trojan of some sort which is scanning for Windows systems with vulnerable RPC implementations, as described in Microsoft's security bulletins MS03-026. According to Symantec, this could be variant of GaoBot (Shannon). This host should be visited with anti-virus repair tools, or should have it's operating system reinstalled. This host is classified as scanning & infected based on the fact that it attempted to connect to 112360 different hosts, without retries, over the 5 days in the logs, as shown in the table below.

```
select distinct src_ip, count(distinct(src_port)) as sports, count(distinct(dest_ip)) as destips,
dest_port, protocol, count(src_ip) as hits from scans2 WHERE src_ip='MY.NET.80.224' GROUP BY
dest_port order by hits desc
```

src_ip	sports	destips	dest_port	protocol	hits
MY.NET.80.224	3975	112360	135	SYN	112360
MY.NET.80.224	246	99	80	SYN	250
MY.NET.80.224	6	3	443	SYN	6

To determine if this host is scanning continuously, we graph some data from the scans database to show a time history.



The graph shows a trend of intermittent spikes of activity. One pattern which stands out to me is the alignment with an 8-hour work schedule. The first burst lasts for about 8 hours. The second burst starts around 24 hours after the first alert, and also lasts about 8 hours. Then there's a 48 hour gap, and activity resumes again.

A quick check of a calendar for March 2004, and the scans data show that the activity occurred on Wednesday March 3, Thursday March 4, and Saturday March 6, if the timestamps are accurate. The table below shows the date, the hour of the first alert for that day, and the total number of alerts see for that day.

```
select distinct DATE_FORMAT(stamp, '%e') as t_date, MIN(DATE_FORMAT(stamp, '%H')) as t_hr, count
(stamp) as hits from scans2 WHERE src_ip='MY.NET.80.224' and dest_port=135 GROUP BY t_date ORDER
BY t_date asc;
```

t_date	t_hr	hits
3	11	31347
4	10	21510
6	11	59503

This indicates that MY.NET.80.224 was being turned on around 10 or 11 AM on the 3 days in question, and that the host was not active on Friday, March 5th or on Sunday, the 7th of March.

Top 10 Destination Addresses by Count

Now we look into the hosts which appear most often in the scans database as destinations. In the table below, several things stand out: (1) the number of hits only spans a factor of 2; (2) the typical number of destination ports is 1 or 2; (3) one host has vastly more destination ports; and (4) there is only one internal host listed. These characteristics differ significantly from the behavior of the top 10 sources, and likely indicate normal traffic getting flagged by snort.

```
select distinct dest_ip, count(distinct(dest_port)) as dports, count(distinct(src_ip)) as srcips,
count(distinct(src_port)) as sports, count(src_ip) as hits from scans2 group by dest_ip order by
hits desc limit 10
```

dest_ip	dports	srcips	sports	hits
69.6.68.10	2	11	819	44336
69.6.68.11	1	2	2	43764
192.26.92.30	1	3	3	39665
MY.NET.25.70	30749	73	79	36816
192.48.79.30	1	2	2	33882
203.20.52.5	1	2	2	30701
4.13.52.66	27	2	2	27228
192.5.6.30	1	2	2	25387
192.52.178.30	1	2	2	24271
216.109.116.17	1	2	2	22906

Detect 4

To examine the single internal host, which is also the host with a large number of destination ports scanned, we build a table showing the source ports used to scan this host.

```
select distinct src_port, src_ip, count(distinct(dest_port)) as dports, dest_ip, protocol, count
(src_port) as hits from scans2 WHERE dest_ip='MY.NET.25.70' group by src_port,src_ip order by
hits desc limit 15
```

src_port	src_ip	dports	dest_ip	protocol	hits
55746	204.152.186.189	15825	MY.NET.25.70	SYN	15825
55748	204.152.186.189	7538	MY.NET.25.70	SYN	7538
55747	204.152.186.189	6680	MY.NET.25.70	SYN	6680
55749	204.152.186.189	3167	MY.NET.25.70	SYN	3167
55750	204.152.186.189	1315	MY.NET.25.70	SYN	1315
55751	204.152.186.189	1290	MY.NET.25.70	SYN	1290
55752	204.152.186.189	576	MY.NET.25.70	SYN	576
55753	204.152.186.189	187	MY.NET.25.70	SYN	187
55754	204.152.186.189	77	MY.NET.25.70	SYN	77
55755	204.152.186.189	45	MY.NET.25.70	SYN	45
25	200.180.79.54	8	MY.NET.25.70	INVALIDA	8
55756	204.152.186.189	7	MY.NET.25.70	SYN	7
113	218.57.116.19	2	MY.NET.25.70	INVALIDA	3
113	80.224.1.68	3	MY.NET.25.70	INVALIDA	3
113	218.59.108.134	3	MY.NET.25.70	INVALIDA	3

The vast majority of these scan alerts are from traffic used to communicate with 204.152.186.189 on ports 55746 through 55756. Looking up host information for 204.152.186.189 shows that it is called www.dnsbl.us.sorbs.net . Since the site resolved to a “www” address, I checked the website, and found information about a DNS based black list aimed at helping to limit spam (Sullivan). This would imply that MY.NET.25.70 is a mail server, or one of the SORBS “feeder” servers. Based on either role, rules should be tuned to account for this server's “normal” traffic.

One of the other oddities mentioned is that each host listed in the top 10 by destination address only seems to be using one destination port. Which one?

```
select distinct dest_ip, dest_port, count(distinct(src_ip)) as srcips, count(distinct(src_port))
as sports, protocol, count(src_ip) as hits from scans2 group by dest_ip,dest_port,protocol order
by hits desc limit 10
```

dest_ip	dest_port	srcips	sports	protocol	hits
69.6.68.11	53	2	2	UDP	43764
69.6.68.10	53	2	2	UDP	42854
192.26.92.30	53	3	3	UDP	39665
192.48.79.30	53	2	2	UDP	33882
203.20.52.5	53	2	2	UDP	30701
192.5.6.30	53	2	2	UDP	25387
192.52.178.30	53	2	2	UDP	24271
216.109.116.17	53	2	2	UDP	22906
128.194.254.5	53	2	2	UDP	22180
128.194.254.4	53	2	2	UDP	22065

This all appears to be likely DNS traffic, so let's take a quick look at the what hostname is associated with each of the destination IP's. For any hosts without proper records, the whois domain name for the owner is listed.

dest_ip	hostname
69.6.68.11	Not found - whois => wholesalebandwidth.com
69.6.68.10	Not found - whois => wholesalebandwidth.com
192.26.92.30	c.gtld-servers.net
192.48.79.30	j.gtld-servers.net
203.20.52.5	Not found – whois => cluevoid.net
192.5.6.30	a.gtld-servers.net
192.52.178.30	k.gtld-servers.net
216.109.116.17	ns5.yahoo.com
128.194.254.5	ns2.tamu.edu
128.194.254.4	ns1.tamu.edu

Based on this information, this appears to be normal DNS traffic. There is likely an opportunity to tune the snort rules to reduce the rate of these alerts, however care must be taken to avoid muting valid alerts. Building a list of internal DNS servers and commonly referenced external DNS servers, and ignoring traffic between the 2 sets using port 53 UDP & TCP would likely reduce the noise level significantly. This will likely involve using snort's portscan-ignorehosts preprocessor to enumerate the hosts that the portscan preprocessor is to ignore, or configuring snort's event suppression (Roesch, User's Manual).

Top 10 Destination Ports

Now that we've examined some of the most frequently listed addresses of scanners & scan targets, we take a quick look at the most frequently seen port and protocol pairs.

```
select distinct dest_port, count(distinct(dest_ip)) as destips, count(distinct(src_port)) as
sports, count(distinct(src_ip)) as srcips, protocol, count(src_ip) as hits from scans2 group by
dest_port,protocol order by hits desc limit 10
```

dest_port	destips	sports	srcips	protocol	hits
53	92901	33	38	UDP	2427461
135	253623	4857	122	SYN	261592
25	14309	35664	502	SYN	180972
6129	15738	13849	27	SYN	139290
80	28341	7293	271	SYN	110332
443	15615	20247	34	SYN	71199
4899	15524	3977	16	SYN	66993
20168	15533	3976	8	SYN	64631
4000	15682	18085	9	SYN	62953
21	15523	9510	32	SYN	52509

As is evident from the table above, UDP connections to external servers on the port commonly associated with DNS are the most frequent events. One notable item is that there were 38 hosts responsible for making direct external DNS requests. As few organizations use such a large number of DNS servers for redundancy, we should look at those hosts to see if there is

anything of interest in the data.

```
select distinct dest_port, count(distinct(dest_ip)) as destips, count(distinct(src_port)) as
sports, src_ip, protocol, count(src_ip) as hits from scans2 WHERE dest_port=53 group by
src_ip,protocol order by hits desc limit 10
```

dest_port	destips	sports	src_ip	protocol	hits
53	89691	1	MY.NET.1.3	UDP	2192255
53	25430	1	MY.NET.1.4	UDP	235145
53	26	85	MY.NET.1.3	SYN	86
53	3	9	MY.NET.1.4	SYN	9
53	1	7	MY.NET.153.79	SYN	7
53	1	1	68.33.206.62	UDP	5
53	3	1	63.251.32.198	UDP	4
53	3	1	67.114.19.186	UDP	4
53	4	1	MY.NET.111.140	UDP	4
53	1	4	MY.NET.60.16	SYN	4

We see that the top 4 entries are for MY.NET.1.3 and MY.NET.1.4 using UDP and TCP for DNS communications, and that they cover the overwhelming majority of the entries. This behavior is typical of a site with 2 main DNS servers for load balancing and failover. This is also consistent with the top 10 destination addresses largely indicating 2 source IP's and 2 source ports.

Detect 5

The next item highlighted in this section's first table shows connections to more than a quarter million hosts on port 135/TCP. This activity is common on hosts infected with malware, as described earlier for MY.NET.80.244. A quick database query will show the hosts responsible for this traffic.

```
select distinct dest_port, count(distinct(dest_ip)) as destips, count(distinct(src_port)) as
sports, src_ip, protocol, count(src_ip) as hits from scans2 WHERE protocol='SYN' and
dest_port=135 group by src_ip order by hits desc limit 10
```

dest_port	destips	sports	src_ip	protocol	hits
135	141006	3969	MY.NET.81.39	SYN	141006
135	112360	3975	MY.NET.80.224	SYN	112360
135	191	269	24.242.151.1	SYN	280
135	192	192	82.64.189.13	SYN	277
135	239	239	141.151.118.17	SYN	260
135	178	178	219.211.164.63	SYN	242
135	215	215	165.121.228.227	SYN	239
135	186	227	67.61.195.107	SYN	227
135	217	217	61.206.42.130	SYN	226
135	196	201	62.37.22.226	SYN	201

As shown above, MY.NET.80.224, a host we previously determined to be infected with a trojan of some sort, is responsible for nearly half of the traffic. The other primary contributor, MY.NET.81.39, is also likely infected with some virus or trojan which is scanning for other vulnerable hosts running a Microsoft operating system (Shannon). This machine needs to be

disconnected from the network and cleaned up.

Detect 6

Next on the list is port 6129. This port is used for "DameWare" and "Cisco AUX/TTY/VTY ports," according to Payne's portsdb.org. DameWare recently had a remote-user gains total control exploit (Rafail), which several viruii have made use of as an infection vector (Canavan). Again, we consult the database for more detailed information.

```
select distinct dest_port, count(distinct(dest_ip)) as destips, count(distinct(src_port)) as
sports, src_ip, protocol, count(src_ip) as hits from scans2 WHERE protocol='SYN' and
dest_port=6129  group by src_ip order by hits desc limit 10
```

dest_port	destips	sports	src_ip	protocol	hits
6129	10180	1990	80.57.227.167	SYN	12923
6129	11317	1	24.195.127.101	SYN	12153
6129	9582	3725	130.160.104.196	SYN	12035
6129	9273	1	209.195.174.212	SYN	9273
6129	6568	3263	151.8.45.167	SYN	7757
6129	6476	3137	199.103.192.38	SYN	7742
6129	6683	3453	66.76.215.81	SYN	7731
6129	6229	6229	213.104.76.54	SYN	7217
6129	6510	2	67.71.58.188	SYN	6510
6129	5620	2938	65.105.133.82	SYN	6421

As all of the source IP's listed are external to MY.NET, there is not much we can do to stop the scanning at the source, however, a firewall or router access control list (ACL) at the border of MY.NET could be configured to drop the packets, thus protecting the hosts in MY.NET from attack via this vector. Contacting the abuse contact for the source addresses may be of some help, but it can be time consuming. Also, a secondary check of the database shows that there are no scanners with a source address on the MY.NET network. This is a good sign, as it implies that there are no infected machines inside MY.NET, as they would likely start scanning once infected.

```
select distinct dest_port, count(distinct(dest_ip)) as destips, count(distinct(src_port)) as
sports, src_ip, protocol, count(src_ip) as hits from scans2 WHERE protocol='SYN' and
dest_port=6129 AND src_ip LIKE 'MY.NET%' group by src_ip order by hits desc limit 10;
Empty set (1 min 2.03 sec)
```

Top 10 Source Ports

Yet again, we turn around and look at things from the other direction. The table below shows the top 10 source port / protocol pairs, listed by frequency.

```
select distinct src_port, count(distinct(src_ip)) as srcips, count(distinct(dest_ip)) as
destips,count(distinct(dest_port)) as dports, protocol, count(src_ip) as hits from scans2 group
by src_port,protocol order by hits desc limit 10
```

src_port	srcips	destips	dports	protocol	hits
32783	3	89692	1	UDP	2192259
14052	4	31229	16312	UDP	294677
32788	2	25431	1	UDP	235146
8767	1	45	138	UDP	153959
12203	2	136	192	UDP	73921
1690	1	20059	3410	UDP	62997
220	14	15584	2	SYN	53095
8888	1	241	388	UDP	41029
7674	2	16532	1	UDP	23981
1710	1	6633	2707	UDP	20338

The first two highlighted lines show 1 destination port each and account for over 2.4 million hits. Investigating these 2 ports more fully, we find that the 2 likely DNS servers (MY.NET.1.3 and MY.NET.1.4) account for all but 5 of the packets. If these DNS servers were configured to use a single port each as the source of any external queries, this would be the result.

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips,dest_port, protocol, count
(src_ip) as hits from scans2 WHERE src_port=32783 AND protocol='UDP' group by src_port,src_ip
order by hits desc limit 10
```

src_port	src_ip	destips	dest_port	protocol	hits
32783	MY.NET.1.3	89691	53	UDP	2192255
32783	12.129.73.236	1	53	UDP	2
32783	63.241.203.110	1	53	UDP	2

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips,dest_port, protocol, count
(src_ip) as hits from scans2 WHERE src_port=32788 AND protocol='UDP' group by src_port,src_ip
order by hits desc limit 10
```

src_port	src_ip	destips	dest_port	protocol	hits
32788	MY.NET.1.4	25430	53	UDP	235145
32788	12.129.73.236	1	53	UDP	1

As we've seen before, port 8767 is commonly used for TeamSpeak, and this specific host appears to have been running TeamSpeak.

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips,count(distinct(dest_port))
as dports, protocol, count(src_ip) as hits from scans2 WHERE src_port=8767 AND protocol='UDP'
group by src_port,src_ip order by hits desc limit 10
```

src_port	src_ip	destips	dports	protocol	hits
8767	MY.NET.110.72	45	138	UDP	153959

Detect 7

Also, we've previously seen port 12203 associated with MY.NET.110.72, and determined that it is required for playing various online games. The host MY.NET.70.207 may also be violating policy in a manner similar to MY.NET.110.72.

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips,count(distinct(dest_port))
as dports, protocol, count(src_ip) as hits from scans2 WHERE src_port=12203 AND protocol='UDP'
group by src_port,src_ip order by hits desc limit 10
```

src_port	src_ip	destips	dports	protocol	hits
12203	MY.NET.110.72	101	140	UDP	73593
12203	MY.NET.70.207	37	53	UDP	328

Detect 8

The host MY.NET.112.216 is connecting to a large number of hosts, using a source port of 1690. Searching portsdb.org and google for information on UDP port 1690 turned up only a registered service name of “ng-umds” (Payne).

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips, count(distinct(dest_port))
as dports, protocol, count(src_ip) as hits from scans2 WHERE src_port=1690 AND protocol='UDP'
group by src_port,src_ip order by hits desc limit 10
```

src_port	src_ip	destips	dports	protocol	hits
1690	MY.NET.112.216	20059	3410	UDP	62997

Because of the large number of destination IP's, some more searching is in order. The table below lists the most frequently accessed destination ports. As port UDP 1214 is commonly associated with Kazaa, this host may be violating copyright policy, and should be visited.

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as dests ,dest_port, protocol, count
(src_ip) as hits from scans2 WHERE src_port=1690 AND protocol='UDP' AND src_ip='MY.NET.112.216'
group by dest_port  order by hits desc limit 5;
```

src_port	src_ip	dests	dest_port	protocol	hits
1690	MY.NET.112.216	491	1214	UDP	958
1690	MY.NET.112.216	9	3312	UDP	106
1690	MY.NET.112.216	46	32656	UDP	102
1690	MY.NET.112.216	11	2919	UDP	102
1690	MY.NET.112.216	9	2292	UDP	98

Detect 9

After some web searching, Experts Exchange had a tidbit of information on what might be using port 7674 UDP: a peer-to-peer file sharing program called ”soribada” (Sailor). According to USA Today, the Korean authors of soribada were attemting to create a “Napster of the East” (Associated Press).

```
select distinct src_port, src_ip, count(distinct(dest_ip)) as destips,dest_port, protocol, count
(src_ip) as hits from scans2 WHERE src_port=7674 AND protocol='UDP' group by src_port,src_ip
order by hits desc limit 10
```

src_port	src_ip	destips	dest_port	protocol	hits
7674	MY.NET.97.70	12096	7674	UDP	16819
7674	MY.NET.98.11	4549	7674	UDP	7162

In order to get a better idea of where this traffic was headed, each of the top 10 IP's was looked up using DNS and whois. DNS had no records for any of the top 10, and whois listed the country for all of them as KR, or South Korea (ICANN). Based on these factors, I recommend that these 2 machines be

checked for violations of copyright laws and of the acceptable use policy.

```
select distinct src_port, src_ip, dest_ip ,dest_port, protocol, count(src_ip) as hits from scans2
WHERE src_port=7674 AND protocol='UDP' group by src_port,src_ip,dest_ip order by hits desc limit
10
```

src_port	src_ip	dest_ip	dest_port	protocol	hits
7674	MY.NET.98.11	211.38.111.37	7674	UDP	7
7674	MY.NET.98.11	221.164.183.156	7674	UDP	6
7674	MY.NET.97.70	220.94.85.134	7674	UDP	6
7674	MY.NET.98.11	211.105.138.28	7674	UDP	6
7674	MY.NET.98.11	211.222.179.95	7674	UDP	6
7674	MY.NET.97.70	221.148.239.18	7674	UDP	6
7674	MY.NET.97.70	218.146.121.82	7674	UDP	6
7674	MY.NET.97.70	211.219.122.234	7674	UDP	6
7674	MY.NET.98.11	220.71.83.141	7674	UDP	6
7674	MY.NET.98.11	61.72.144.239	7674	UDP	6

Preprocessor Alerts

The data from the "alerts" data set was split into rule-based alerts and alerts generated by snort's preprocessors. Only one of snort's preprocessor created any alerts – the portscan preprocessor. As the "scan" logs were also generated by the same mechanism, no further analysis is necessary.

Rule – Based Alerts

After filtering out the preprocessor generated alerts, we are left with the alerts generated by snort's rules for inspecting individual packets and streams.

Top 10 Source Addresses

This is a list of the top hosts triggering snort rules, sorted by the number of entries in the database for the IP. Also shown is the number of different rules, source ports, destination addresses, and destination ports touched by each source address.

```
select distinct src_ip, count(distinct(message)) as rules, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as destips, count(distinct(dest_port)) as dports, count(dest_ip) as hits
from rule group by src_ip order by hits desc limit 10
```

src_ip	rules	sports	destips	dports	hits
MY.NET.27.103	2	2181	2	5	45338
68.50.102.64	1	831	1	2	8709
68.55.191.197	1	229	1	2	1710
68.34.27.67	1	290	1	1	1518
MY.NET.190.97	1	1	6	1	1415
MY.NET.70.37	1	1	1299	1	1299
68.55.250.229	2	125	2	1	1256
MY.NET.11.7	1	1	1	1	1151
63.159.88.57	1	230	1	2	963
209.126.201.99	1	4	2	844	936

Note that the top 3 hosts span nearly 2 orders of magnitude and that most of the top 10 touch fewer than 10 destination ports or addresses. This may be an indication that the rules being triggered most often are host or port based. Rules which are written to trigger based only on the host or port information in the packet headers can generate a large number of false positives.

Four hosts on MY.NET are listed in the table among the top 10 sources of alerts. These hosts, and the snort rules that triggered the relevant alerts will need to be examined to reduce the noise they are causing. If any of these hosts are compromised, they must be taken off the network as soon as possible. We will examine these hosts, and the rules they triggered shortly.

Also, notice that the top 4 external sources are all in the 68.32.0.0/11 network block operated by Comcast Cable Communications, Inc, according to whois. As many cable modem users are not technically savvy, their computers often provide easy targets to malicious crackers. However, as the table below shows, they appear to be triggering what appears to be a host-based rule.

```
select src_ip, message, count(distinct(src_port)) as sports, count(distinct(dest_ip)) as destips,
count(distinct(dest_port)) as dports, count(dest_ip) as hits from rule WHERE src_ip NOT LIKE
'MY.NET%'group by src_ip,message order by hits desc limit 5
```

src_ip	message	sports	destips	dports	hits
68.50.102.64	MY.NET.30.4 activity	831	1	2	8709
68.55.191.197	MY.NET.30.4 activity	229	1	2	1710
68.34.27.67	MY.NET.30.3 activity	290	1	1	1518
68.55.250.229	MY.NET.30.3 activity	121	1	1	1207
63.159.88.57	MY.NET.30.4 activity	230	1	2	963

Top 10 Destination Addresses

The following shows the top 10 most frequently listed destination addresses.

Again, the frequency drops rapidly among the top hosts, and only a small number of rules are triggered by any given host.

Also, the inclusion in the list of hosts outside of MY.NET indicates that outbound attacks are occurring, and at a non-trivial rate.

```
select distinct dest_ip, count(distinct(message)) as rules, count(distinct(dest_port)) as dports,
count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports, count(src_ip) as hits
from rule group by dest_ip order by hits desc limit 10
```

dest_ip	rules	dports	srcips	sports	hits
209.126.201.99	1	4	2	2181	45329
MY.NET.30.4	1	25	293	2108	17536
MY.NET.30.3	1	20	142	784	6312
62.166.61.120	1	1	1	1	1405
169.254.0.0	1	1	3	1	1248
MY.NET.27.103	2	844	3	6	932
64.246.65.158	1	1	2	1	690
MY.NET.42.5	5	21	14	166	630
169.254.45.176	1	1	118	1	596
MY.NET.70.247	1	1	3	2	570

Detect 10

Two of the three lines highlighted in the table above show a network address (169.254.0.0) and a host on that network(169.254.45.176). According to RFC 3330, this network is reserved for “link-local” use(RFC 3330). That is, this network is reserved for hosts within a broadcast domain to use when they are unable to get their address any other way. These addresses should not be seen on the internet. These hosts should be tracked down, and investigated. It is possible that the addresses are being spoofed, in which case, the router's access control lists should be updated to disable routing for packets to or from this network.

Based on the makeup of the top 5 hosts, MY.NET.11.7 and MY.NET.11.6 seem to be sending broadcasts to the link-local subnet. These hosts should be examined to ensure they are properly configured. If these hosts are intended to be sending this traffic, adjusting the IDS rules to ignore it would be beneficial.

```
select distinct src_ip,dest_ip, count(src_ip) as hits from rule WHERE dest_ip LIKE '169.254%'
GROUP BY src_ip ORDER BY hits DESC LIMIT 5
```

src_ip	dest_ip	hits
MY.NET.11.7	169.254.0.0	1151
MY.NET.11.6	169.254.0.0	96
MY.NET.112.229	169.254.45.176	15
MY.NET.75.114	169.254.45.176	12
MY.NET.66.61	169.254.45.176	12

```
select distinct message,dest_ip from rule WHERE dest_ip LIKE '169.254%'
```

message	dest_ip
SMB Name Wildcard	169.254.0.0
SMB Name Wildcard	169.254.45.176

Detect 11

The other host, as we have not seen it previously also bears looking into.

```
select distinct message, src_ip, src_port, dest_ip, dest_port from rule where
dest_ip='62.166.61.120'
```

message	src_ip	src_port	dest_ip	dest_port
SMB Name Wildcard	MY.NET.190.97	137	62.166.61.120	137

Looking up a couple of quick pieces of information about 62.166.61.120, we find that its hostname is dslam120-61-166-62.adsl.zonnet.nl, and that whois states:

```
whois 62.166.61.120
[snip]
inetnum:      62.166.0.0 - 62.166.63.255
netname:      VERSATEL-CUST-VERSNET-ADSL-1
descr:        Zon internet is one of the largest free ISP in the Netherlands
country:      NL
[snip]
```

That the host MY.NET.190.97 is repeatedly attempting to connect to a host in the Netherlands to do Windows file sharing is highly suspect. This host should be examined for the presence of a virus or worm.

We will look at the SMB Name Wildcard alerts later. For the moment, it is enough to note that based on the IP address and the alert, the hosts which need to be reconfigured to prevent these alerts from occurring in the future are likely running a version of Windows (Forster).

Top 10 Rules Triggered - Overall

Listed below are the top 10 rules triggered by the snort system. Notice that at least 4 of these rules appear to be custom rules, based on the presence of “UMBC NIDS” and “MY.NET” in the message. Also notice the stratification that matches with the other data from the rule-based alerts.

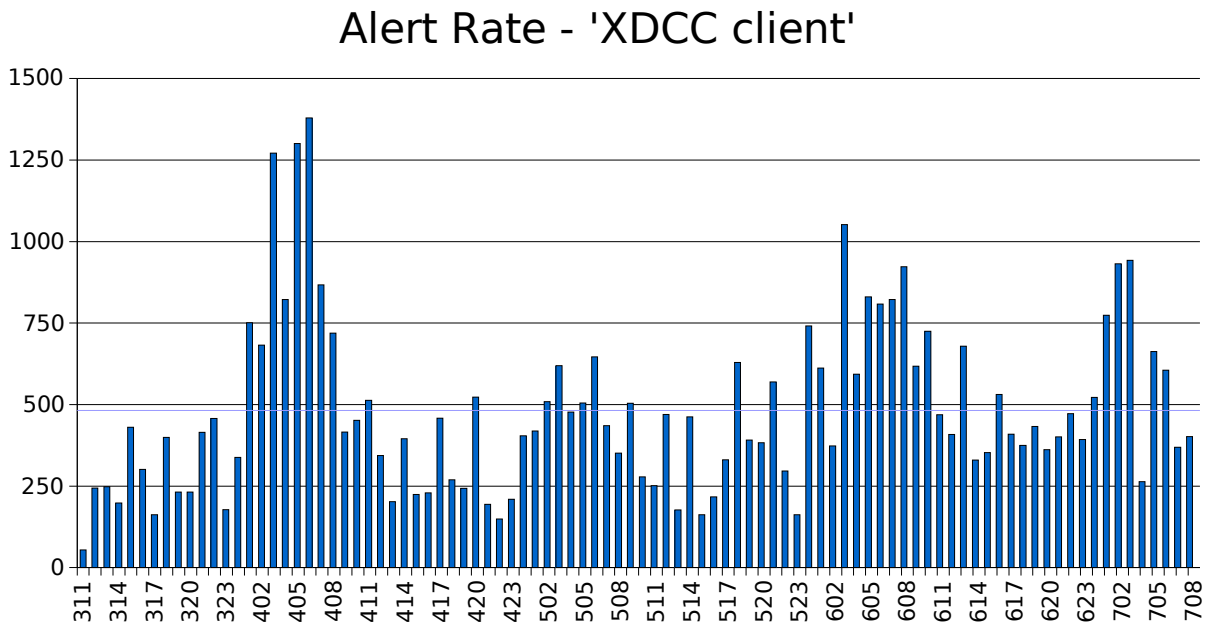
```
select distinct message, count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as destips, count(distinct(dest_port)) as dports, count(message) as hits
from rule group by message order by hits desc limit 10
```

message	srcips	sports	destips	dports	hits
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	5	2184	4	4	45333
MY.NET.30.4 activity	293	2108	1	25	17536
SMB Name Wildcard	161	11	1630	1	6478
MY.NET.30.3 activity	142	784	1	20	6312
EXPLOIT x86 NOOP	157	430	46	39	1181
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	45	9	27	979	1140
SUNRPC highport access!	31	12	29	1	1049
High port 65535 tcp - possible Red Worm - traffic	85	29	110	32	753
NMAP TCP ping!	140	14	49	40	625
Null scan!	82	126	49	45	353

Detect 12

Looking at the vast quantity of the XDCC client alerts, we want to know wether they spanned the entire time range we're examining. We might be

able to determine when the machine was compromised from this information.



The “IRC Alert” hosts represent a small fraction of the source and destination population and are responsible for a large portion of the alerts, thus, we want to find out which hosts they are.

```
select message, src_ip, count(distinct(src_port)) as sps, dest_ip, dest_port as d_p, count
(message) as hits from rule WHERE message LIKE '[UMBC NIDS IRC Alert] XDCC%' group by
src_ip,dest_ip,dest_port order by hits desc
```

message	src_ip	sps	dest_ip	d_p	hits
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.27.103	738	209.126.201.99	6667	11583
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.27.103	718	209.126.201.99	7000	11543
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.27.103	742	209.126.201.99	6668	11494
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.27.103	675	209.126.201.99	6669	10708
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.112.199	2	67.130.99.99	6667	2
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.15.198	1	64.157.246.22	6667	1
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.80.5	1	209.126.201.99	6667	1
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	MY.NET.97.86	1	216.152.64.62	7000	1

As we can plainly see in the table above, one source and one destination are

responsible for the overwhelming majority of the XDCC events generated by this snort instance. Also, we can see that there is a second host, MY.NET.80.5, communicating with the noisiest destination. Searching the DNS for more information about the destination host, uncovers that the address's reverse record is desire.of.hotgirlz.org but that there is no forward record for the reverse.

```
% host 209.126.201.99
99.201.126.209.IN-ADDR.ARPA domain name pointer desire.of.hotgirlz.org

% host desire.of.hotgirlz.org
Host not found, try again.
```

Consulting the whois database shows the following registration for the domain hotgirlz.org:

```
% whois hotgirlz.org
Domain ID:D96720295-LROR
Domain Name:HOTGIRLZ.ORG
Created On:11-Apr-2003 14:37:25 UTC
Last Updated On:14-Nov-2003 03:45:13 UTC
Expiration Date:11-Apr-2004 14:37:25 UTC
Sponsoring Registrar:R39-LROR
Status:OK
Registrant ID:F2825ED08233195D
Registrant Name:roy elisa
Registrant Organization:Shell.web.id Internet Services
Registrant Street1:Jl. Anoa I No. 27
Registrant City:Palu
Registrant Postal Code:94113
Registrant Country:ID
Registrant Email:webmaster@shell.web.id
Admin ID:F2825ED08233195D
Admin Name:roy elisa
Admin Organization:Shell.web.id Internet Services
Admin Street1:Jl. Anoa I No. 27
Admin City:Palu
Admin Postal Code:94113
Admin Country:ID
Admin Email:webmaster@shell.web.id
Tech ID:F2825ED08233195D
Tech Name:roy elisa
Tech Organization:Shell.web.id Internet Services
Tech Street1:Jl. Anoa I No. 27
Tech City:Palu
Tech Postal Code:94113
Tech Country:ID
Tech Email:webmaster@shell.web.id
Name Server:NS1.EGGDROP.WEB.ID
Name Server:NS2.EGGDROP.WEB.ID
```

Further exploration of the whois database, reveals the owner of the IP address block:

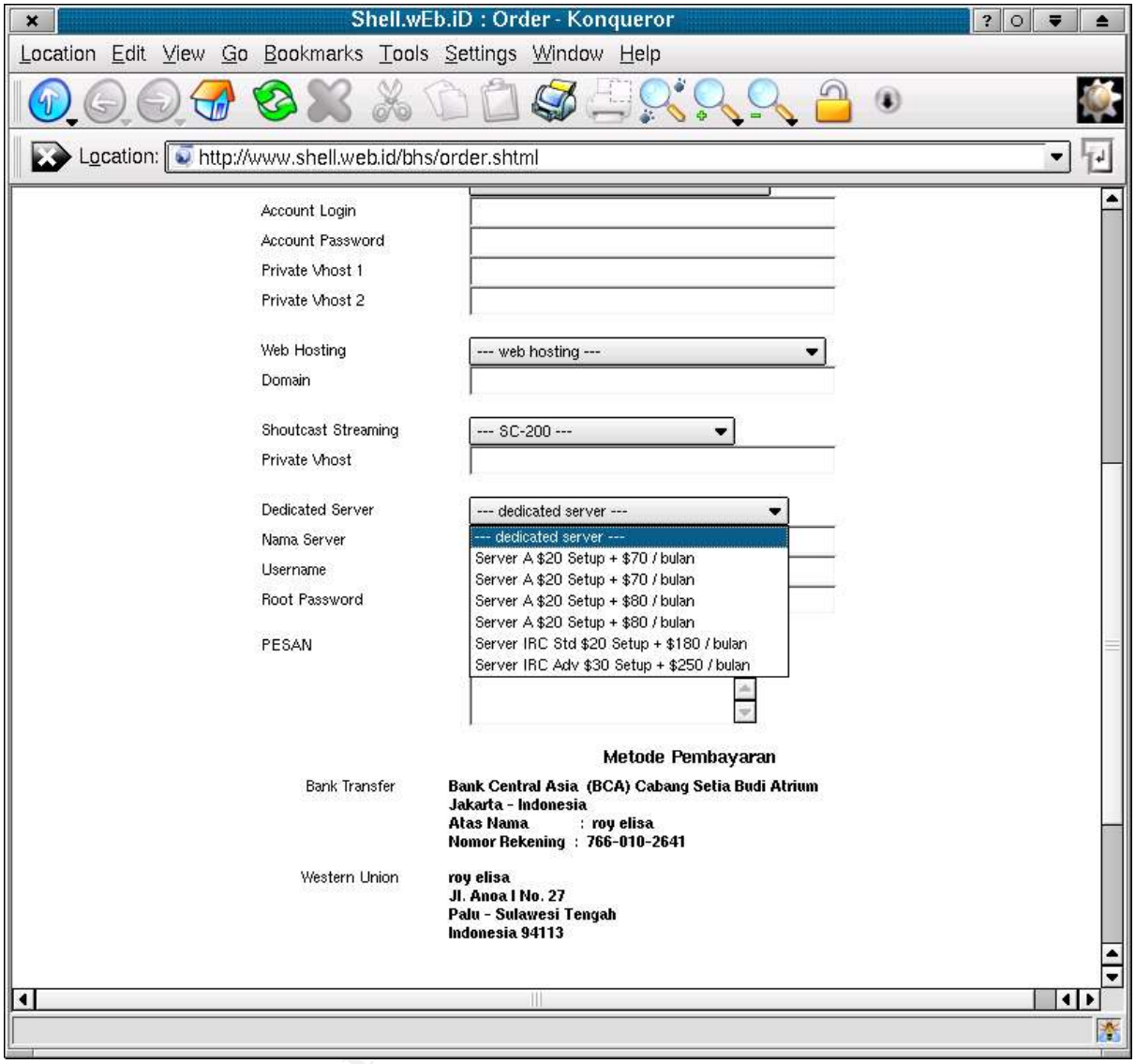
```
% whois 209.126.201.99

OrgName:    California Regional Internet, Inc.
```

```
OrgID:      CALI
Address:    8929A COMPLEX DRIVE
City:       SAN DIEGO
StateProv:  CA
PostalCode: 92123
Country:    US
NetRange:   209.126.128.0 - 209.126.255.255
CIDR:       209.126.128.0/17
NetName:    CARI
NetHandle:  NET-209-126-128-0-1
Parent:     NET-209-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.ASPADMIN.COM
NameServer: NS2.ASPADMIN.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-03-12
Updated:    2003-07-01
[snip]
```

In addition to being listed in whois as the registrant and tech for hotgirlz.org, Roy Elisa is listed on the ISP's Order page (http://www.shell.web.id/bhs/order.shtml) as the owner of the bank account which customers of "shell.web.id" must send money to to get service. It appears that part of the service customers get is anonymity behind the guise of "roy elisa." Not knowing anything about Indonesia's computer crime laws, this seems likely to be an attractive service provider for many an evildoer. The whois registration information for the IP address lists California Regional Internet as the owner of the network block.

The screenshot below shows the services offered for dedicated servers, including both "Server IRC Std" and "Server IRC Adv" options. It would appear that this hosting company takes pride in its abilities to provide IRC servers for its customers.



As one of the other custom-looking rule messages in the top 10 also appears to be IRC related, let's find out what hosts are generating the most alerts for that rule.

```
SELECT message, src_ip,dest_ip, count(message) as hits FROM rule WHERE message LIKE '%IRC user / kill%' GROUP BY src_ip,dest_ip ORDER BY hits desc limit 10
```

message	src_ip	dest_ip	hits
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	209.126.201.99	MY.NET.27.103	930
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	65.248.51.47	**MY.NET.42.3**	90
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	69.50.189.88	**MY.NET.42.6**	9
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	209.126.206.54	**MY.NET.42.5**	8
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	209.126.201.99	MY.NET.80.5	6
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	69.50.189.88	**MY.NET.42.7**	5
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	203.56.139.100	**MY.NET.42.4**	5
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	69.56.199.206	**MY.NET.42.2**	5
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	69.28.250.108	**MY.NET.42.4**	4
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	64.237.52.219	MY.NET.42.10	4

Again we see that 209.126.201.99 is the most frequently accessed IRC server triggering alerts. Also, there is another server in the 209.126.128.0/17 network block, owned by California Regional Intranet, Inc., that is triggering IRC alerts on MY.NET.

The block of bold addresses in the table above shows another trend in the IRC / trojan messages: a sequential address block. It has been my experience that sequential blocks of hosts are often managed similarly, or at least run similar operating systems. As IRC regularly operates using TCP, and the snort rule's message implies that packet content is being analyzed, we can infer that these hosts are establishing a connection with the external host. All of the MY.NET machines listed in the table above should be visited and checked for trojans or other signs of a compromise.

Top 10 Rules Triggered – Internal Source

Next, we look at the most frequently triggered alerts with an internal source address. This should help us find misconfigured and cracked machines on MY.NET.

```
select distinct message, count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as destips, count(distinct(dest_port)) as dports, count(message) as hits
from rule where src_ip LIKE 'MY.NET%' group by message order by hits desc limit 10
```

message	srcips	sports	destips	dports	hits
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	5	2184	4	4	45333
SMB Name Wildcard	161	11	1630	1	6478
High port 65535 tcp - possible Red Worm - traffic	30	25	79	6	347
IRC evil - running XDCC	7	20	7	2	137
High port 65535 udp - possible Red Worm - traffic	8	5	13	1	65
Possible trojan server activity	9	4	16	4	37
connect to 515 from inside	2	20	2	1	28
TFTP - Internal TCP connection to external tftp server	1	7	1	1	17
DDOS mstream handler to client	1	1	1	1	9
HelpDesk MY.NET.70.49 to External FTP	1	2	1	1	2

Here again, we see the top rule corresponds to the XDCC traffic triggered by MY.NET.27.103 as the most frequent alert message.

Detect 13

The next alert referring to “SMB Name Wildcard” appears not to be a rule from snort.org, but the rule message appears in several places with a simple web search, including the SANS website (Green) and the snort-announce list archive (Forster). Based on Forster's message to the list, the SMB Name Wildcard query can be used to “to extract useful information such as workstation name, domain, and users currently logged in.” Attackers can also enumerate administrator accounts on the target machine. The SMB Name Wildcard query is also used by Windows as a normal part of the file sharing protocol.

These types of queries should not be passed outside of MY.NET without a specific need. The internal hosts responsible for these alerts should be investigated. If possible, Windows file sharing to computers outside MY.NET should be prohibited at the border either by router access control lists or firewall rules.

A quick query of the database reveals that a couple of hosts generate a large number of these alerts, and that one highlighted host appears to be scanning for this service (MY.NET.70.37). This host should be examined for trojans and other signs of compromise.

```
select distinct message, src_ip, count(distinct(src_port)) as sports, count(distinct(dest_ip)) as
destips, dest_port, count(message) as hits from rule where src_ip LIKE 'MY.NET%' AND message LIKE
'SMB Name Wildcard' group by src_ip order by hits desc limit 10
```

message	src_ip	sports	destips	dest_port	hits
SMB Name Wildcard	MY.NET.190.97	1	6	137	1415
SMB Name Wildcard	MY.NET.70.37	1	1299	137	1299
SMB Name Wildcard	MY.NET.11.7	1	1	137	1151
SMB Name Wildcard	MY.NET.190.93	1	9	137	484
SMB Name Wildcard	MY.NET.75.13	1	131	137	358
SMB Name Wildcard	MY.NET.190.92	1	5	137	343
SMB Name Wildcard	MY.NET.150.198	6	113	137	267
SMB Name Wildcard	MY.NET.150.44	6	68	137	170
SMB Name Wildcard	MY.NET.11.6	1	1	137	96
SMB Name Wildcard	MY.NET.29.30	1	1	137	71

Top 10 Rules Triggered – External Source

In the table below, the top 10 rules triggered by external source addresses are listed by frequency. The 2 rules generating the most hits appear to be custom written rules for MY.NET, and both focus on a single host each.

```
select distinct message, count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as destips, count(distinct(dest_port)) as dports, count(message) as hits
from rule where src_ip NOT LIKE 'MY.NET%' group by message order by hits desc limit 10
```

message	srcips	sports	destips	dports	hits
MY.NET.30.4 activity	293	2108	1	25	17536
MY.NET.30.3 activity	142	784	1	20	6312
EXPLOIT x86 NOOP	157	430	46	39	1181
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	45	9	27	979	1140
SUNRPC highport access!	31	12	29	1	1049
NMAP TCP ping!	140	14	49	40	625
High port 65535 tcp - possible Red Worm - traffic	55	6	31	29	406
Null scan!	82	126	49	45	353
High port 65535 udp - possible Red Worm - traffic	31	8	21	15	117
Incomplete Packet Fragments Discarded	49	1	42	1	97

Detect 14

As each of the 2 hosts referenced in the top 2 alerts lists a limited number of destination ports, it is worth finding out which occur most frequentlly. Also, based on the number of destination ports listed, it would seem that these 2 rules are triggering on any communication with the host specified in the rule message.

```
select distinct message, count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as dests, dest_port, count(message) as hits from rule where src_ip NOT
LIKE 'MY.NET%' AND dest_ip REGEXP 'MY.NET.30.[34]' group by dest_port order by hits desc limit 10
```

message	srcips	sports	dests	dest_port	hits
MY.NET.30.4 activity	14	1387	1	51443	14331
MY.NET.30.3 activity	33	1038	2	524	7965
MY.NET.30.4 activity	276	487	2	80	1364
MY.NET.30.3 activity	17	19	2	6129	28
MY.NET.30.3 activity	9	15	2	4899	24
MY.NET.30.4 activity	8	13	2	20168	19
MY.NET.30.4 activity	9	14	2	1080	18
MY.NET.30.4 activity	5	10	2	443	16
MY.NET.30.3 activity	7	11	2	3128	14
MY.NET.30.3 activity	1	4	2	81	13

As the first three rows in the table show, ports 80, 524, and 51443 are responsible for the majority of the alerts. According to portsdb.org and Novell, these ports are commonly associated with HTTP, NCP or NetWare Core Protocol, and Novell NetStorage, respectively. Based on Novell's website, NetStorage provides browser based access to users' files stored on a NetWare server.

Both of these hosts also appear to be being probed on ports commonly used for web proxies (1080,3128,8000,8080,8081), http (80), smtp (25), ftp (21), and DameWare (6129) among others (Payne). Depending on how the rule triggering these alerts is written, these could indicate probe packets only, or full connections to the server.

If these really are publicly accessible servers for providing remote access to users' files, then logging all connections from outside to the these servers will provide an audit trail mechanism, and should be continued. Based on the existence of the rules specifically written to trigger on traffic to these hosts, it would seem that this is the case.

Source & Destination Both External

Detect 15

When snort's HOME_NET and EXTERNAL_NET variables are set to be compliments of each other, the source and destination addresses can never both be external. This can be accomplished by defining HOME_NET to list the networks snort is to consider internal, and setting EXTERNAL_NET to be ! HOME_NET. As the alerts being analyzed contain alerts where the source and destination are both outside the HOME_NET, snort is obviously configured another way.

```
select distinct message, count(distinct(src_ip)) as srcips, count(distinct(src_port)) as sports,
count(distinct(dest_ip)) as destips, count(distinct(dest_port)) as dports, count(message) as hits
from rule where dest_ip NOT LIKE 'MY.NET%' and src_ip NOT LIKE 'MY.NET%' group by message order
by hits desc limit 10
```

message	srcips	sports	destips	dports	hits
TCP SRC and DST outside network	27	35	37	12	83

The table below lists the unique source addresses of the alerts that have a source and destination outside the network. These source addresses break down into two ranges: (1) 192.168.0.0/16, a private network per RFC-1918, (2) the 172.128.0.0-172.211.255.255 block owned by America Online (whois).

```
select distinct(src_ip), count(distinct(src_port)) as sports, count(distinct(dest_ip)) as
destips, count(distinct(dest_port)) as dports, count(src_ip) as hits from rule where dest_ip NOT
LIKE 'MY.NET%' and src_ip NOT LIKE 'MY.NET%' group by src_ip order by src_ip desc
```

src_ip	sports	destips	dports	hits
192.168.123.195	4	3	1	7
192.168.1.47	1	1	1	1
192.168.1.46	3	3	1	3
192.168.1.41	2	1	1	2
192.168.1.102	1	1	1	1
172.210.36.171	1	1	1	2
172.208.104.168	1	1	1	1
172.168.196.51	1	1	1	2
172.165.205.164	2	2	2	4
172.164.150.81	1	1	1	5
172.162.181.215	1	1	1	2
172.161.176.241	1	1	1	1
172.161.147.103	1	1	1	1
172.160.220.65	1	1	1	2
172.158.184.23	1	1	1	1
172.155.49.4	2	2	1	9
172.152.188.157	1	1	1	1
172.147.40.169	1	1	1	6
172.145.229.3	1	1	1	2
172.145.22.36	1	1	1	2
172.142.196.231	3	3	1	5
172.141.55.62	2	2	2	10
172.139.130.38	1	2	2	2
172.136.225.106	1	1	1	5
172.134.167.15	1	1	1	4
172.133.94.69	1	1	1	1
172.129.197.197	1	1	1	1

This could indicate a peering type arrangement between America Online and MY.NET, as the routers could be using the 192.168 network for private communications between themselves to arrange routes for the hosts in the 172.128 network block. This would imply that the snort sensor is able to see all traffic from the MY.NET and from the AOL block heading toward the internet at large, as shown in the diagram below. Based on the small number of alerts, all of AOL's traffic is obviously not being sent through this router link. However, it is possible that AOL is providing modem access for the campus, or is providing broadband for residence halls. In either scenario, the configuration of the snort sensor should be adjusted to better

reflect the situation.

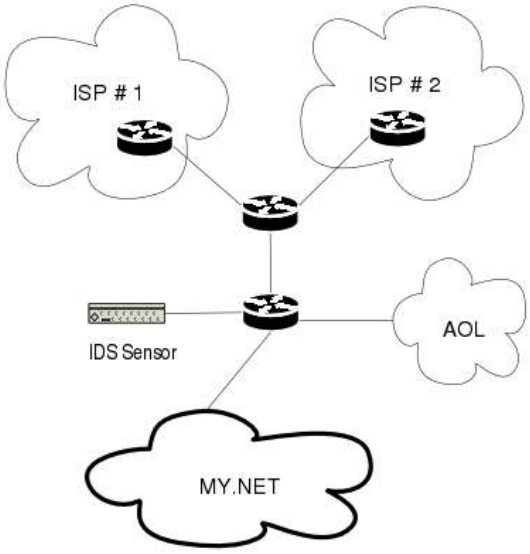


Illustration 4 Possible Network Topology

Out Of Specification

These packets are logged by snort's packet decoder as not following the standard protocol specification. However, as protocols are extended and updated, snort's decoder also needs to be adjusted to follow the changes.

Top Source Addresses

Each of the source IP addreses listed below lists the number of times the IP occurs in the database (hits), the number of unique source ports, desintaion IP's, destination ports, T.O.S classifications, and flag settings. Note that only 24.136.69.34 appears to be mixing flags in the packets it is sending to a single destination.

The number of hits in the top 20 hosts seen spans nearly 2 orders of magnitude.

Also, the close alignment between the number of hits and source ports may indicate than snort is configured to alert on protocol options which are valid for experimental purposes, but determined to be out of specification by the older version of snort on the sensor.

```
select distinct srcip, count(srcip) as hits,  count(distinct(srcport)) as sports, count(distinct
(dstip)) as destips, count(distinct(dstport)) as dports, count(distinct(tos)) as svc_types, count
(distinct(flags)) as flag_types from oos group by srcip order by hits desc limit 20
```

srcip	hits	sports	destips	dports	svc_types	flag_types
68.54.84.49	881	881	1	1	1	1
217.125.5.139	145	102	1	1	1	1
MY.NET.199.158	113	113	5	2	1	1
MY.NET.199.138	110	93	5	4	1	1
67.114.19.186	86	82	1	1	1	1
66.225.198.20	79	78	1	1	1	1
68.122.128.1	48	48	1	1	1	1
35.8.2.252	48	48	1	1	1	1
80.126.206.180	27	27	1	1	1	1
65.118.185.48	19	5	1	1	1	1
63.71.152.2	18	6	1	1	1	1
35.8.2.251	17	17	1	1	1	1
66.180.237.99	16	14	1	1	1	1
62.109.103.173	15	15	1	1	1	1
218.58.63.34	12	7	2	1	1	1
213.216.249.20	11	11	1	10	1	1
24.136.69.34	11	8	1	8	1	10
66.180.236.228	11	10	1	1	1	1
81.208.58.2	11	11	2	1	1	1
212.241.49.98	10	10	1	1	1	1

Detect 16

The host above appears to be attempting an OS fingerprint of a single host on MY.NET by altering the flags in packets sent to that host. A quick check of the database shows that the target host is MY.NET.24.47.

```
select distinct srcip,destip from oos WHERE srcip='24.136.69.34'
```

src_ip	dest_ip
24.136.69.34	MY.NET.24.47

To find out a bit more about the source, we consult the DNS system and whois.

```
% host 24.136.69.34
34.69.136.24.IN-ADDR.ARPA domain name pointer user-0c8gh92.cable.mindspring.com

% whois 24.136.69.34
EarthLink, Inc. ERLK-CBL-TW-WEST (NET-24-136-64-0-1)
                                  24.136.64.0 - 24.136.95.255
EARTHLINK, INC. ERLK-TW-LOSANGELES25 (NET-24-136-68-0-1)
                                  24.136.68.0 - 24.136.71.255

# ARIN WHOIS database, last updated 2004-04-27 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

From the data provided, it appears that this host belongs to one of EarthLink's cable modem customers in the LosAngles area.

Top Destination Addresses

Below are the most frequently seen destination addresses in the out-of-specification data. The frequency of hits among the top 20 destination hosts spans nearly 3 orders of magnitude.

Again, we see a strong correlation between the number of hits, and the number of unique source ports. As mentioned previously, this could be caused by external hosts using protocol options which did not exist at the time the snort sensor was installed.

The highlighted lines in the table below indicate hosts which are frequently receiving many different types of flags.

```
select distinct dstip, count(dstip) as hits, count(distinct(dstport)) as dports, count(distinct
(srcip)) as srcips, count(distinct(srcport)) as sports, count(distinct(tos)) as svc_types, count
(distinct(flags)) as flag_types from oos group by dstip order by hits desc limit 20
```

dstip	hits	dports	srcips	sports	svc_types	flag_types
MY.NET.6.7	919	2	12	917	2	1
MY.NET.12.6	587	1	235	507	2	7
MY.NET.24.44	224	1	35	219	1	3
MY.NET.6.47	210	1	41	60	1	1
MY.NET.153.79	160	1	2	117	1	1
MY.NET.12.7	112	2	2	99	1	1
MY.NET.24.34	80	1	6	77	1	2
MY.NET.12.4	51	2	2	51	1	2
MY.NET.34.11	43	1	15	42	3	1
MY.NET.34.14	30	2	8	15	1	1
MY.NET.29.3	30	1	1	30	1	1
MY.NET.24.47	26	19	3	18	1	11
MY.NET.42.6	22	4	5	5	1	10
MY.NET.42.5	18	3	7	7	1	12
MY.NET.60.14	12	1	7	10	1	1
MY.NET.42.8	12	11	2	12	1	1
MY.NET.60.17	9	1	4	4	1	1
MY.NET.42.3	8	4	4	4	1	6
MY.NET.153.92	7	1	6	6	1	1
MY.NET.42.10	6	3	3	3	1	6

Top Flag Types

Looking at what flags are set most frequently in the OOS data gives some insight into what is going on. The top flag set has nearly 35 times as many entries as the second, and over 500 times as many as the third. Also, the top flag set is the only one which shows any variation in the type of service bits. Since there are only a small number of different types of service, we'll break down the top flags by type of service later.

```
select distinct flags, count(flags) as hits, count(distinct(srcip)) as srcips, count(distinct
(dstip)) as destips, count(distinct(tos)) as svc_types from oos group by flags order by hits desc
limit 10
```

flags	hits	srcips	destips	svc_types
12****S*	2512	382	45	4
********	72	19	13	1
12***R**	5	2	5	1
U***	4	4	3	1
*2UA*RSF	3	3	3	1
URSF	3	3	3	1
12U**RS*	3	3	3	1
12UAPRSF	2	2	2	1
*2U*PRSF	2	2	2	1
12*A****	2	2	2	1

Top Types of Service

This table lists all of the types of service seen in the OOS data set, sorted by the number of times each TOS was seen. As we can see, the only set of flags which touches all 4 types of service is "12****S*" from the table above.

```
select distinct tos, count(tos) as hits, count(distinct(srcip)) as srcips, count(distinct(dstip))
as destips from oos group by tos  order by hits desc limit 20
```

tos	hits	srcips	destips
0x0	2642	424	64
0x10	5	2	2
0x80	5	1	2
0x40	1	1	1

Comer's book describes the type of service bits in the IP datagram as "a hint to the routing algorithm" which is responsible for choosing the packet's next hop. Based on the definitions for each of the bits he provides, the type of service codes would translate into the following:

0x00 = Normal Packet

0x10 = Packet from an application requesting low delay

0x40 = Packet stating its precedence as 2 of a maximum 7

0x80 = Packet stating its precedence as 4 of 7.

Alternatively, RFC 3168 defines the use of some bits in the type of service field in the IP header and some bits in the flags field of the TCP header for the use of explicit congestion notification. Using an RFC 3168 interpretation would define all four types of service listed above as not using an ECN-capable transport, yet having various differentiated services codepoints.

Top Flags, with Type of Service

Breaking out the type of service with the flags and again sorting by hits, we see that "12****S*" is the only set of flags for which TOS is not 0x0.

```
select distinct flags, tos, count(flags) as hits, count(distinct(srcip)) as srcips, count
(distinct(dstip)) as destips from oos group by flags,tos order by hits desc limit 10
```

flags	tos	hits	srcips	destips
12****S*	0x0	2501	378	45
********	0x0	72	19	13
12****S*	0x10	5	2	2
12****S*	0x80	5	1	2
12***R**	0x0	5	2	5
U***	0x0	4	4	3
*2UA*RSF	0x0	3	3	3
URSF	0x0	3	3	3
12U**RS*	0x0	3	3	3
*2U*PRSF	0x0	2	2	2

Per the definition in RFC 3168, the flag set “12****S*” is considered an “ECN-setup SYN packet” (15). Thus, it would appear that these packets correspond to attempts by ECN-aware hosts to establish an ECN capable TCP session with the opposite host. Also, as these are SYN packets, the sending host must set the IP header bits used for ECN to 0, which matches the type of service flags seen previously (RFC 3168, 16).

Based on the above assessment, it appears that the snort sensor should be upgraded so that proper decoding of the ECN bits is performed.

Correlations

Visualization of large datasets can help analysts find more obscure events which don't fall into the typical “top 10” ranges that basic database manipulation allows. In his GCIA practical, Alex Wood used graphs of destination port against time to find interesting events. Below, I've taken a simpler approach using the open-source ploticus software to analyze source & destination addresses as a function of the time of day (Grubb).

Time and Address Correlation

Detect 17

The following graphs, show the frequency of addresses in the scans data plotted against the first octet of the address (horizontal axis) and the hour of the day (vertical axis). In the scans destination graph, a strong horizontal line shows up, indicating that at that time of day, some number of hosts on MY.NET were attempting to scan addresses covering the range from 0.0.0.0 to roughly 224.0.0.0 over the course of 2 hours.

More information on how these plots were generated is available in the appendix.

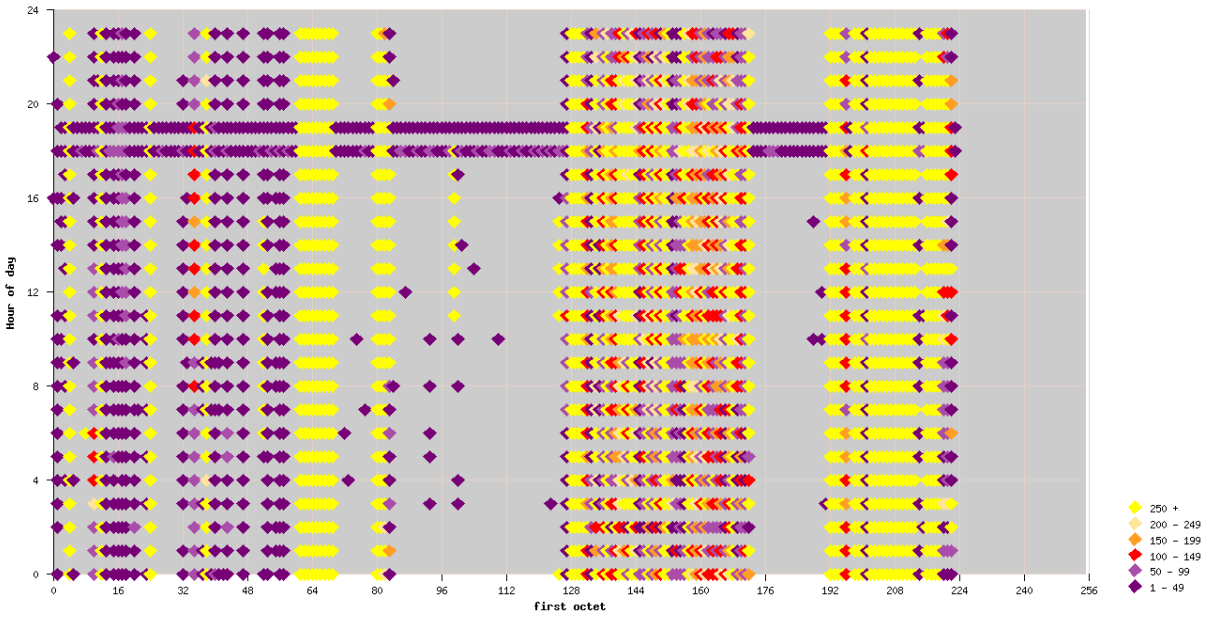


Illustration 5 Destination Address Time Distribution – Scan-based Alerts

Notice the vertical patterns spanning the full duration of the day. These give an indication of the host networks to which MY.NET sends most of its traffic. Based on the lack of entries for a destination address in the netblock reserved for multicast communications by RFC 3171 (224.0.0.0/4), we can infer that this site does not use multicast.

Looking back at the database, we can determine which host or hosts are responsible for the horizontal line in the graph above. In the table below, the "topocs" column lists the number of unique destination address top octets touched by the listed source IP. The two highlighted hosts have not been listed previously, and thus require further investigation.

```
select src_ip, count(distinct(src_port)) as sports, dest_port, count(distinct(dest_ip)) as dests,
count(distinct(LEFT(dest_ip, LOCATE('.', dest_ip) -1))) as topocs, count(src_ip) as hits FROM
scans2 WHERE dest_ip NOT LIKE 'MY.NET%' AND (LEFT(RIGHT(stamp,6),2)=18 OR LEFT(RIGHT(stamp,6),2)
=19) GROUP BY src_ip,dest_port ORDER BY hits DESC limit 10
```

src_ip	sports	dest_port	dests	topocs	hits
MY.NET.1.3	6	53	24833	105	183379
MY.NET.97.147	3827	80	11605	220	16198
MY.NET.80.224	3814	135	12645	2	12645
MY.NET.53.169	669	6346	2349	73	11464
MY.NET.1.4	1	53	2796	91	8587
MY.NET.34.14	6569	25	231	55	8539
MY.NET.97.70	1	7674	3867	54	4434
MY.NET.97.70	1	22321	3076	61	3357
MY.NET.110.72	1	33309	1	1	2179
MY.NET.153.97	1528	4662	1249	60	1927

Digging a bit deeper into the database, we can see that MY.NET.97.147 is largely occupied with scanning destination port 80, but that MY.NET.53.169 is doing other things as well.

Nearly all of MY.NET.97.147's traffic is used for scanning port 80.

```
select count(src_ip) from scans2 WHERE src_ip='MY.NET.97.147'
16213
select protocol, count(src_ip) from scans2 WHERE src_ip='MY.NET.97.147' AND dest_port=80 group by
protocol
SYN,16198
```

However, MY.NET.53.169 is doing other things as well, including scanning for port 6346 during other times of the day.

```
select count(src_ip) from scans2 WHERE src_ip='MY.NET.53.169'
236868
select count(src_ip) from scans2 WHERE src_ip='MY.NET.53.169' and dest_port=6346
35250
```

Also, we see that MY.NET.153.97 is sending traffic to a large number of hosts on the port commonly associated with the eDonkey peer-to-peer file sharing program. This could be another source of a policy violation.

Generating the time versus destination address plot again with the source address limited to MY.NET.97.147 reveals that one of the responsible hosts has been found. Note that although the color scaling is different between the 2 plots, the corresponding numeric ranges match properly.

The plot for MY.NET.53.169 scanning destination port 6346 is not included, as it shows the vertical banding that matches with the destination addresses that are being scanned throughout the day. However, as port 6346 is commonly associated with Gnutella, this host should be examined for possible copyright or policy infringements.

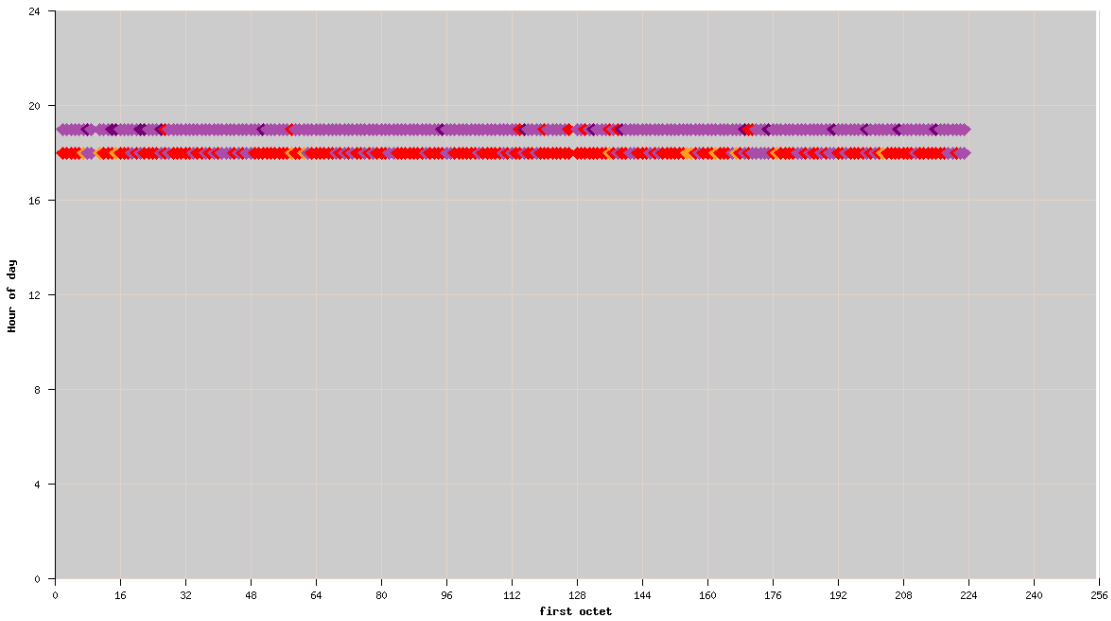


Illustration 6 Destination Address Time Distribution - Source IP = MY.NET.97.147

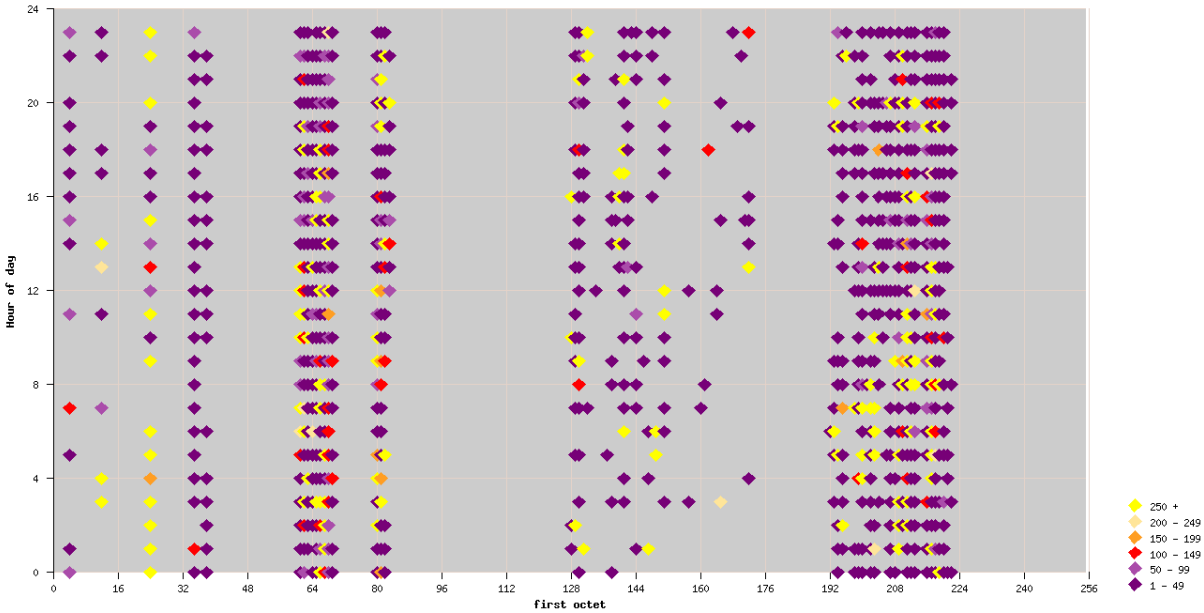


Illustration 7 Source Address Time Distribution – Scan-based Alerts

The graph above, showing the distribution of source addresses, lacks the strong horizontal line found in the destination plot, thus indicating that inbound scanning seems to occur continuously. The source distribution

graph contains vertical bands which correspond to the strongest vertical bands in the destination graph. This seems reasonable, as most communications are bidirectional, and the scan data is generated without any inspection of the packet's payload.

Below, similar graphs for the rule-based alerts show some correlation between the scan and rule based datasets. The primary correlation shows that the 62.x.x.x to 65.x.x.x address ranges are the destination for large amounts of scanning and for frequent rule based alerts. Also, the 192.x.x.x to 223.x.x.x ranges show a strong correlation. However, the destination graphs are less similar over the 0.x.x.x to 60.x.x.x and 128.x.x.x to 176.x.x.x ranges.

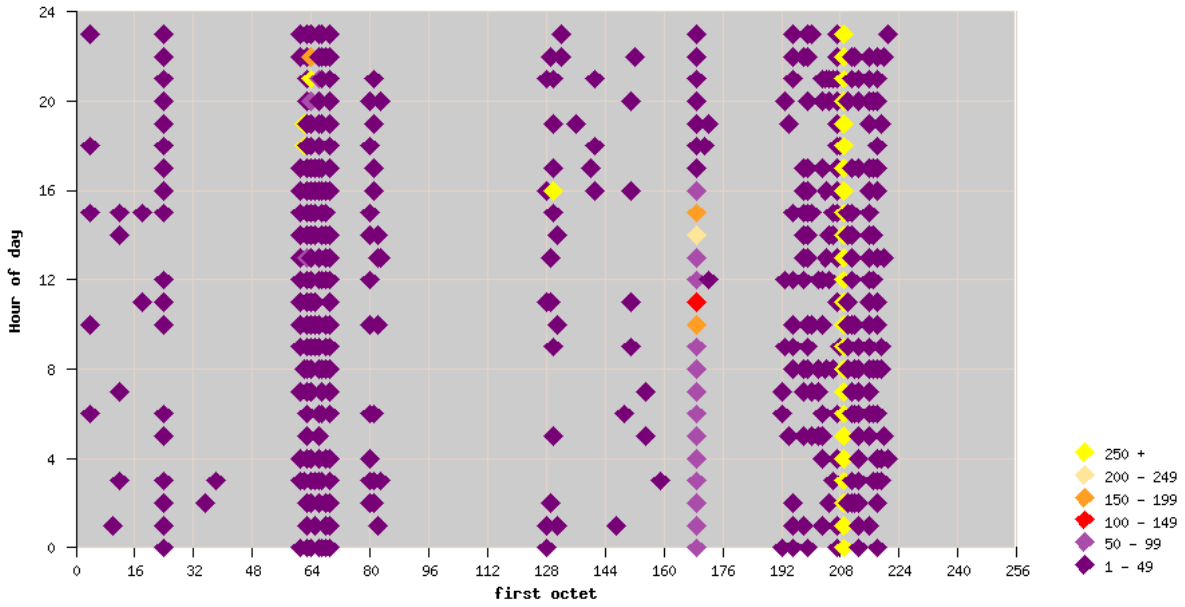


Illustration 8 Destination Address Time Distribution - Rule-based Alerts

The source address bands also match well between the two datasets, across all address ranges, as can be seen in the following graph.

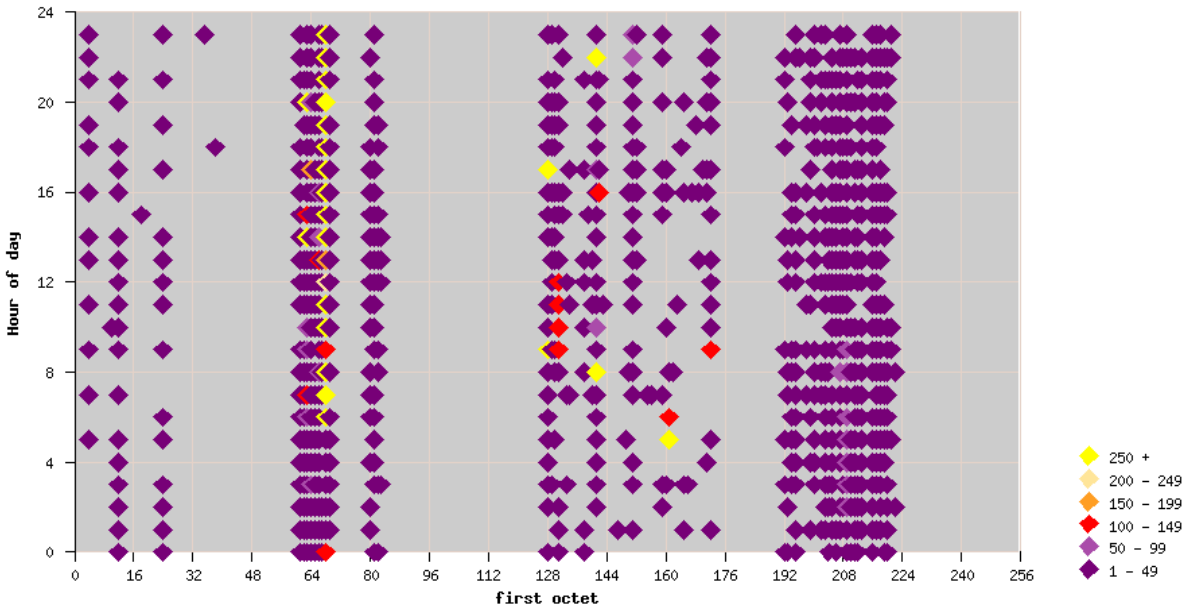


Illustration 9 Source Address Time Distribution - Rule-based Alerts

External Hosts of Interest

The table below summarizes the external hosts which have been listed previously, with a brief summary of their information.

Host	Reason
209.126.201.99	XDCC server in Indonesia or California
24.136.69.34	Flag Scanning Cable Modem in Los Angeles
172.128.0.0/10	Network block owned by America Online, possibly co-located with MY.NET
62.166.61.120	Host in the Netherlands to which MY.NET.190.97 is sending traffic
213.202.254.116	TeamSpeak.org host.
207.38.8.34	Master.gamespy.com – likely gaming server.

Internal Hosts of Interest

The table below lists machines in MY.NET which need to be examined further for various reasons. The “code” column indicates the primary reason the host is listed using the following codes:

POL Policy Violation

INF Infection / Virus / Trojan

TUN Sensor Tuning Opportunity

Host	Code	Description
MY.NET.110.72	POL	Gaming Server; TeamSpeak
MY.NET.34.14	TUN	Likely SMTP server – Adjust IDS rules
MY.NET.80.224	INF	Scanning destination port 135. Likely worm.
MY.NET.25.70	TUN	Likely SMTP server participating in SORBS – Adjust IDS Rules
MY.NET.81.39	INF	Scanning destination port 135. Likely worm.
MY.NET.70.207	POL	Gaming Server
MY.NET.112.216	POL	Peer to Peer: Kazaa
MY.NET.97.70	POL	Peer to Peer: Soribada
MY.NET.98.11	POL	Peer to Peer: Soribada
MY.NET.1.3 MY.NET.1.4	TUN	DNS Server – Adjust IDS rules to reduce false positives.
169.254.0.0/16	TUN	"link-local" addresses should not be traversing the campus border
MY.NET.11.6 MY.NET.11.7	TUN	These hosts were responsible for sending broadcasts to the link-local network.
MY.NET.27.103	POL/INF	40000+ XDCC IRC alerts to 209.126.201.99. Could be an infection.
MY.NET.42.{2-7}	INF	This block of hosts all triggered the IRC /kill custom rule.
MY.NET.190.97	POL/INF	SMB Name Wildcard alerts – these lookups should not cross the border.
MY.NET.70.37	POL/INF	SMB Name Wildcard alerts – these lookups should not cross the border.
MY.NET.30.3 MY.NET.30.4	TUN	Novell NetStorage Server – adjust IDS rules to log "normal" traffic and alert on the remainder.
MY.NET.97.147	INF	This host attempted to scan hosts with 220 different first octets. It is likely infected.
MY.NET.53.169	POL	Peer to Peer: Gnutella
MY.NET.153.97	POL	Peer to Peer: eDonkey

Defensive Recommendations

In order to reduce the volume of scan alerts, the snort sensor rules need to be adjusted for the DNS servers and known SMTP servers. Alerts for these hosts should not be discarded entirely, as having an audit trail for them is important. According to the Snort User's Manual, the "portscan-ignorehosts" preprocessor is designed to deal with hosts which trigger the portscan detector frequently. DNS servers are specifically mentioned.

Adjust network or sensor rules with respect to "link local" addresses. As these address are intended for use on a local network segment only, and should not be routed, they should be filtered at the border, inbound and outbound. Also, examine MY.NET.11.6 and MY.NET.11.7 to verify that they are properly configured.

The AOL address block which seems to be neither internal nor external, 172. 128.0.0 to 128.211.255.255, ought to be handled differently – either made part of snort's HOME_NET, or routed differently so as not to generate unnecessary alerts.

Place a firewall in front of 30.3 & 30.4, with IDS behind, and a tcpdump audit trail or flow captures done on firewall. The firewall protecting these hosts should also control access from MY.NET hosts, with different policies if needed. These hosts deserve extra protection and attention as they appear to be intended to enable access of internal users' data from outside the network. As these would likely be considered mission critical systems, redundant and load-balancing firewalls are likely in order.

The infected machines listed in the table above need to be cleaned or rebuilt quickly to prevent them from infecting other machines inside MY.NET. Additionally, for network segments where hosts change frequently (residence hall networks, wireless LANs, publicly available jacks), an authenticating firewall can help reduce the time it takes to find the owner of an infected host. There are a number of open source and commercial implementations available, however, it can be a challenge to find a solution which integrates with an existing network infrastructure.

Finally, there are a number of hosts detected using peer to peer file sharing systems. These may not be a direct threat to the network or other systems on the network, however the legal risk involved with copyright infringements is high, and can be mitigated by enforcing a policy forbidding peer to peer applications across the campus border. This is often more difficult than blocking a set of port and protocol pairs, as the authors of peer to peer applications give their users ready access to methods to bypass simple port blocking.

Method

The vast majority of the analysis was based on querying a database which contained the event data. Getting the data into the database proved to be a challenge because the original data sets contained some errant lines. Small sections of the scans and alert data showed signs of 2 processes attempting to write to the file simultaneously. These were discovered by some preliminary queries to the database which revealed misaligned fields. These were then cleaned up by hand with a text editor and re-imported into the database.

Importing data into the database was primarily done with heavy use of sed & awk to create delimited text files which the database could read directly. For the out of specification data, Gary Morris' parseOOS.pl script was modified to work with the schema I used.

The time / address distribution plots were generated with the open source package “ploticus” by Stephen Grubb using data selected from the database. An example is provided in the appendix.

References

Associated Press. “Brothers face prosecution for 'Napster of the East'”. <http://www.usatoday.com/tech/techreviews/2001-08-15-korea-napster.htm>. 2001. The Associated Press. Visited 22 April 2004.

Canavan, John. “W32.Mockbot.A.Worm”. <http://securityresponse.symantec.com/avcenter/venc/data/w32.mockbot.a.worm.html >. Symantec Corporation. 25 February 2004. Visited 19 April 2004.

Castanet Communications. “Castanet Communications Online”. <http://www.castanet.ca/tech_guides_details.php?a_id=2>. 1997-2003. Visited 18 April 2004. Victoria, BC, Canada.

CERT. “Vulnerability Note Search Results”. <http://www.kb.cert.org/vuls/byid?searchview&query=Microsoft+and+IIS>. 2004 Carnegie Mellon University. Visited 28 April 2004.

Comer, Douglas E. Internetworking with TCP/IP. 2nd ed. Vol 1. New Jersey: Prentice-Hall, 1991. 93-94.

FileNet Corporation. “FileNet - Enterprise Content Management Solutions” <http://www.filenet.com/index.asp>. 2003. Visited 18 April 2004.

Forster , Jim. “Re: [snort] 'SMB Name Wildcard'”. <http://archives.neohapsis.com/archives/snort/2000-01/0222.html>. 17 January 2000. Visited 23 April 2004.

Green, John. “Global Incident Analysis Center - Detects Analyzed 4/24/00 -”. <http://www.sans.org/y2k/042400.htm>. 24 April 2000. 1999 - 2000 SANS Institute. Visited 23 April 2004.

Grubb, Stephen C. “ploticus: welcome”. <http://ploticus.sourceforge.net/>. Copyright 1998-2004. Visited 28 April 2004.

ICANN. “Root-Zone Whois Information”. <http://www.iana.org/cctld/cctld-whois.htm >. 2004 The Internet Corporation for Assigned Names and Numbers. 26 November 2001. Visited 22 April 2004.

Kevill, Scott. “Help: Ports for Hosting”. <http://www.gameranger.com/help/ports/>. 1998-2003. Visited 18 April 2004.

Microsoft. “Microsoft Security Bulletin MS03-026”. <http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx>. 16 July 2003. Visited 21 April 2004.

Microsoft. “Required Network Ports for a Multiplayer Rise of Nations Game”. <http://support.microsoft.com/?kbid=820877>. 9 June 2003. Visited 16 April 2004.

Morris, Gary. “Contemporary Intrusion Detection and Analysis”. <http://www.giac.org/practical/Gary_Morris_GCIA.doc>. October 17, 2002

Novak, Judy H. Track 3 – Intrusion Detection In-Depth. Vol 2. N.p.: The SANS Institute, 2003. p 4-11, 5-19 to 5-22.

Novell. “SUPPORT PACK 1 README”. <http://www.novell.com/documentation/lg/

nw6p/pdfdoc/readmesp1/readmesp1.pdf>. February 2002. Visited 23 April 2004.

Payne, Rick, et.al. “The Internet Ports Database”. <http://www.portsdb.org/bin/portsdb.cgi?portnumber=32773>. Visited 18 April 2004.

Rafail, Jason A. “Vulnerability Note VU#909678”. <http://www.kb.cert.org/vuls/id/909678>. 14 December 2003. Visited 19 April 2004.

Ramakrishnan, K., et. al. “The Addition of Explicit Congestion Notification (ECN) to IP”. <http://www.ietf.org/rfc/rfc3168.txt>. September 2001. The Internet Society.

Roesch, Martin. “Snort™ Users Manual v2.1.1”. <http://www.snort.org/docs/snort_manual.pdf> The Snort Project. 25 February 2004.

Roesch, Marty. “snort-lib”. <http://packetstormsecurity.nl/sniffers/snort/snort-1.0-lib>. 17 August 1999. Visited 23 April 2004.

Sailor. “how can soribada client create room behind NAPT router”. <http://oldlook.experts-exchange.com:8080/Networking/Broadband/ISPs/Q_20816630.html>. 4 December 2003. Visited 22 April 2004.

Seifried, Kurt. “TCP-IP, UDP-IP, TCP, UDP Ports – 8,547 ports”. <http://www.seifried.org/security/ports/>. 2001. 13 January 2003. Visited 18 April 2004.

Shannon, Heather. “W32.HLLW.Gaobot.gen”. <http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.gaobot.gen.html>. 21 November 2003. Visited 21 April 2004.

Sniperstein. “Gaming Forums - Spearhead [comp] ports”. <http://www.gamingforums.com/showthread.php?p=581130#post581130> . 2000-2004. Visited 18 April 2004.

Sullivan, Matthew. “Spam and Open Relay Blocking System”. <http://www.us.sorbs.net/>. Visited 15 April 2004.

TeamSpeak. “Which ports does the TS2 Server use?”. <http://teamspeak.org/forums/showthread.php?threadid=866>. 2000,2001. Visited 18 April 2004.

TeamSpeak. “who is 213.202.250.101?”. <http://teamspeak.org/forums/showthread.php?threadid=4708>. 2000,2001. Visited 18 April 2004.

Wood, Alex. “Intrusion Detection: Visualizing Attacks in IDS Data”. <http://www.giac.org/practical/GCIA/Alex_Wood_GCIA.pdf>. 2 February 2003.

Appendix

Ploticus Heatmap

The time vs. address plots were generated by creating a file of data from the database, and plotting it using the “heatmap” prefab from ploticus (Grubb).

The database query:

```
select LEFT(dest_ip, LOCATE('.', dest_ip) -1) as topoc,
LEFT(RIGHT(stamp,6),2) as hr,
count(dest_ip) as hits
INTO OUTFILE '/tmp/dests21'
FIELDS TERMINATED BY ' ' LINES TERMINATED BY '\n'
FROM scans2
WHERE dest_ip NOT LIKE 'MY.NET%' GROUP BY topoc,hr ORDER BY topoc,hr ASC;
```

The invocation of ploticus (one long line):

```
% pl -prefab heatmap data=/tmp/dests21 x=1 y=2 contentfield=3 xrange="0 255" xbinsize=1
ybinsize=1 yrange="0 24" ylbl="Hour of day" xlbl="first octet" zerocolor="gray(0.8)" yinc=4
xinc=16 xgrid=yes ygrid=yes title="Attack Rate" rectangle="1 1 12 7" -pagesize 14,8
symbol="shape=diamond radius=0.07" -png -o dests21-01.png
```

For more information on the “prefab” templates included in ploticus visit http://ploticus.sourceforge.net/doc/prefabs.html .

Database Schema

The MySQL database schema used for the data analysis is given below.

```
CREATE TABLE `oos` ( `stamp` timestamp(14) NOT NULL,  `src_ip` varchar(25) NOT NULL default '',
`src_port` mediumint(9) unsigned NOT NULL default '0',`dest_ip` varchar(25) NOT NULL default '',
`dest_port` mediumint(9) NOT NULL default '0', `tos` varchar(10) NOT NULL default '',
`flags` varchar(25) NOT NULL default '', `ttl` int(11) NOT NULL default '0',
`id` bigint(20) unsigned NOT NULL auto_increment, PRIMARY KEY  (`id`), KEY `dest_ip` (`dest_ip`),
KEY `src_ip` (`src_ip`)  ) TYPE=MyISAM;

CREATE TABLE `rule` (     `stamp` timestamp(14) NOT NULL,   `src_ip` varchar(15) default NULL,
`src_port` mediumint(8) unsigned default NULL,   `dest_ip` varchar(15) default NULL,
`dest_port` mediumint(8) unsigned default NULL,   `message` varchar(255) NOT NULL default '',
`id` mediumint(8) unsigned NOT NULL auto_increment, PRIMARY KEY  (`id`), KEY `src_ip` (`src_ip`),
KEY `dest_ip` (`dest_ip`), KEY `src_port` (`src_port`), KEY `dest_port` (`dest_port`),
KEY `message` (`message`)  ) TYPE=MyISAM;

CREATE TABLE `scans2` (  `stamp` timestamp(14) NOT NULL,`src_ip` varchar(15) NOT NULL default '',
`src_port` mediumint(9) NOT NULL default '0', `dest_ip` varchar(15) NOT NULL default '',
`dest_port` mediumint(9) NOT NULL default '0', `protocol` varchar(8) NOT NULL default '',
`id` mediumint(8) unsigned NOT NULL auto_increment,  PRIMARY KEY (`id`), KEY `src_ip` (`src_ip`),
KEY `src_port` (`src_port`), KEY `dest_ip` (`dest_ip`), KEY `stamp` (`stamp`) ) TYPE=MyISAM;

CREATE TABLE `spp` ( `stamp` timestamp(14) NOT NULL, `facility` varchar(24) NOT NULL default '',
`message` varchar(255) NOT NULL default '', `info` varchar(255) default '',
`id` mediumint(8) unsigned NOT NULL auto_increment, PRIMARY KEY  (`id`)  ) TYPE=MyISAM;
```