Intrusion Detection In Depth: Finding the Needle

Michael R. Wisener
GIAC GCIA Practical (version 3.3)
Submitted: January 28, 2002

Table of Contents

Conventions

This paper uses the following typographical conventions.

12-point arial used for standard text

```
$ 10-point Courier indented is used for anything entered  or
displayed on the command line. A '$' preceding the command means
only user privileges are needed whereas a '#' would represent
the need for root privileges.
```

```
8-point Courier indented is used for all log entries.
```

Assignment #1: Describe the State of Intrusion

Detection

Paketto Keiretsu dissected: A new gen eration of network tools.

Abstract
An attacker is only as good as the tools he can effectively use. Suffice to say, most do not have the know-how or skill to create their own advanced tools in order to leverage the available information of a network they plan on penetrating. This is where tools such as **nmap** and now **Paketto Keiretsu**, come into play.

The goal of this paper is to describe and demonstrate how the Paketto Keiretsu suite of tools, particularly **scanrand**, can be used to gather information on large networks in a shorter period of time than currently available tools.

Introduction
Created by Dan "Effugas" Kaminsky of Doxpara Research, Paketto Keiretsu is a suite of five tools that each perform a specialized task for obtaining or inserting data on a network.

- **Scanrand** Fast network scanner.
- **Paratrace** 'Parasitic' traceroute program.
- **Minewt** Software router.
- **Lc** Layer 2 data insertion tool.
- **Phentrophy** 3-D point plotter.

This paper will mainly concentrate on **scanrand** since this is the particular tool in Paketto Keiretsu that has emphasis on network information gathering.

Obtaining and Installing Paketto Keiretsu
The suite of tools can be obtained from:
http://www.doxpara.com/paketto/paketto-1.10.tar.gz

Paketto Keiretsu relies on the libnet library for sending packets, and libpcap for receiving packets. These libraries deal directly with the datalink layer, bypassing the kernel network stack. In additional to these two libraries, scanrand relies on libtomcrypt for encryption algorithms. All these libraries are included and statically linked to the Paketto distribution but are referenced here for completeness. The number in parenthesis below is the version Paketto Keiretsu 1.0 - 1.10 uses.

Libtomcrypt: http://libtomcrypt.iahu.ca/(0.66)
Libnet: http://www.packetfactory.net/libnet/(1.0.2a)
Libpcap: http://www.tcpdump.org/release/(0.7.1)

Once you have obtained and extracted the paketto-1.10.tar.gz file, the typical

```
$ ./configure
$ make
# make install
```

will compile and install the suite.

Scanrand

Overview

Scanrand is a fast network scanner that can scan single hosts to very large networks efficiently. However, several network mapping utilites boast this same claim. So why is scanrand any different? Scanrand can do what is called *stateless* TCP scanning, which sets it apart from the other network scanners.

TCP - a stateful transport protocol

The TCP/IP stack of each operating system is responsible for keeping track of many variables for each 'connection' to or from the local machine. One of these variables is the state that a particular TCP connection is in.

A typical connection starts off with a client sending a TCP SYN packet to a server or remote machine. When the SYN is sent from the client, the operating system sets the state for this connection to SYN_SENT. When in a SYN_SENT state, the client operating system is expecting a SYN/ACK in return. Note, the client operating system and program are *waiting* on a reply. If the server does not respond within a particular time, the client will retransmit it's SYN since it assumes the packet did not make it to the server. If the server is listening on the port the client sent to, the server receives the SYN and replies with a SYN/ACK. The server then enters SYN_RECD state. If the client were a scanner, it would more than likely have what it wanted: A response from the server that indicated something was listening on the port it sent its SYN to.

Now that it has the information it wants, it doesn't care to talk to the server anymore. However, since TCP is a reliable protocol, if the server does not get a response from the scanner saying it received its SYN/ACK, the server will retransmit. To keep this from happening, the scanner sends a RST packet which tells the TCP connection to terminate regardless of state. You can imagine the quantity of SYN/ACK retransmits a scanner would get if it scanned thousands of machines and did not RST the connections. Another problem the scanner has to worry about is authentication. How does it know the SYN/ACK it received came from the machine it sent the SYN to? Sure the IP address is from the same machine, but what's to say it's not spoofed from a person that notices the scanning and wants to corrupt the gathered data?

TCP relies on sequence numbers to ensure the machine it is talking to is indeed the machine it originally contacted. The *initial* sequence numbers are 32-bit
randomly generated integers that are exchanged in the initial TCP connection by both machines involved in the connection. Refer to *figure 1-1*.

A basic network scanner sends a SYN packet to the remote server. It then stores the connection variables[1], or waits for a reply. When it gets the reply, it makes sure the ACK sequence number is 1 plus the initial sequence number it sent in the SYN. If this is the case, it is a valid response (assuming it came from the IP address it sent the SYN to). The assumption is that an attacker that wanted to insert data into your TCP stream could not synchronize sequence numbers with you because of the randomness. This is actually

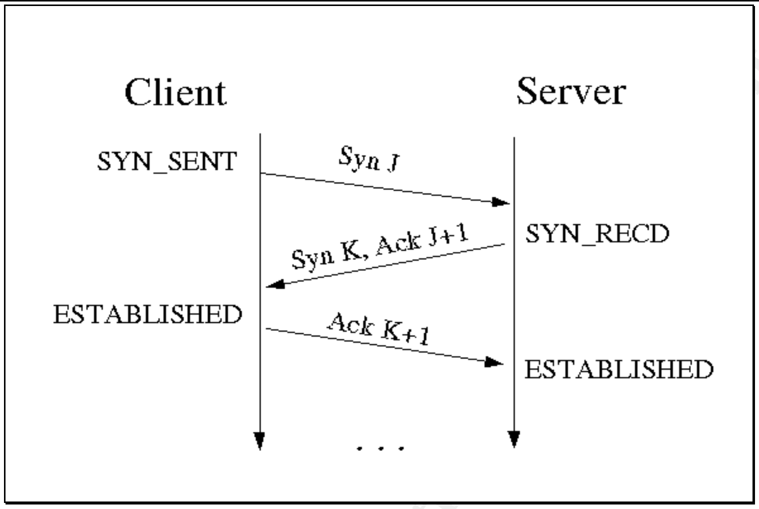

Figure 1-1. *J is the initial sequence number the client sends to the server in a SYN. The server gets the SYN and responds with a SYN/ACK which contains K, it's own initial sequence number, and an ACK which contains the clients sequence number(J) + 1 which tells the client it got its SYN packet. The client then sends an ACK with the servers initial sequence number(K) + 1 to tell the server it received its SYN/ACK. After these three packets, the connection is considered to be in the ESTABLISHED state, and data transfer can occur.[5]*

possible, but often very difficult depending on how random the algorithm for selecting initial sequence numbers and the type of firewall the host is behind.

The point is that a typical scanner either has to store the connection and move on to the next host, or wait for the response, then move on to the next host. Both of these can be costly, either in resources or time. Scanrand implements neither of these methods and takes a more... mathematical approach.

Scanning scanrand style

As mentioned earlier, scanrand takes a little different approach than the typical network scanner. It implements more of a, 'fire and forget' ideology for scanning mixed with a little math.

On startup, scanrand breaks itself up into two processes. One processes is

1 I say 'stores connection variables' because I'm assuming there is a tool out there that can do parallel network scanning. It would send out a SYN packet (or whatever), store the source, destination, ports, etc. and move on to the next host. When it got responses, it would look for the matching data it stored and report based on the response (or lack of). Perhaps SynScan does this.

responsible for doing nothing but sending out SYN packets using libnet. The other process is responsible for receiving the responses the remote computers send back using libpcap. One important thing to note here is that these processes work independently. There is no consulting with the other process on, "Did you send this packet out?" or, "Is this a valid packet?" Scanrand stores no list of IP addresses it is expecting a response from and the sending process does not wait for a response at all. It fires off a SYN, and then moves on to the next computer leaving the receiving process to sort out the flood of responses to come.

So how does the receiving process do this? It doesn't know what the initial sequence number of any of the packets sent are. It only knows it got a response packet. (A SYN/ACK, or a ICMP error message). It could be from anyone. It could be a SYN/ACK from your email client checking your email every five minutes. It could be Windows sending the names of the DVDs you just played to their gigantic database without your knowledge.

The key is what the sending process of scanrand sends out as initial sequence numbers which are not exactly randomly generated. Scanrand takes the source address and port, along with the destination address and port, concatenates them along with key, and runs them through a SHA-1 truncated to 32-bits one-way hash algorithm which becomes the initial sequence number for the outgoing packet. Scanrand calls these 'Inverse SYN cookies.' The reason we use a one-way hash function is because the likely hood of collision is low. This means that given different input data to the hash function, the probability of the hash function generating the same result is very very low.

hash(input$_1$) = output$_1$ hash(input$_2$), hash(input$_3$) ... hash(input$_n$) != output$_1$

Table 1-1: One-way hash algorithm [4]

In *table 1-1*, input$_1$ is the source, source port, destination, and destination port along with a key. Therefore, if we get a response, we subtract 1 from the ACK sequence number in the reply packet (since this should be one greater than the initial sequence number we sent) and hash it. If that hash matches the hash of the source, source port, destination, and destination port (which we also pulled from the reply packet) along with our key, then this is a valid response to our scan.

Using this method, scanrand doesn't have to wait for hosts to respond before moving on to the next host. It can fire off as many SYN packets as the machine or bandwidth can handle. It also doesn't need to keep track of any sequence numbers of any kind. It just fires and forgets.

Traditional scanning across hosts is generally serialized where as the scanrand method is scanning multiple hosts at a time. The traditional scanner will connect to host A, wait for the response, and then move to host B. The problem with this method is that the scanner has to wait for the host to respond. Depending on how long the scanner is set to wait, scanning large networks could be timely. Often there is not always a machine listening or the machine cannot be reached and the timeout for a response will be the maximum. This leads to long scanning times when scanning large networks. Scanrand's method of scanning maximizes scanning time and minimizes waiting time.

Traditional scanning with nmap

The Network Mapper (nmap) is a feature-rich scanning tool which has come to be a very valuable, and well-known in the security industry. It is available from http://www.insecure.org/nmap and can be run under Windows or Unix-based operating systems.

I have been a big fan of nmap for a while. Many a bad guy I have scanned with this feature-rich tool. I figured nmap could parallelize scanning across hosts, keep track of connections and see when a valid response was received. Not exactly the way scanrand does it, but it would give a similar effect if it can do that. Pouring over the nmap man page, I found some features, listed in *table 1-2,* that could possibly give scanrand a run for its money. We will see.

Option	***Description***
-P0	Don't try to ping hosts before scanning them. We don't care if the computer does not exist. If it responds to our stimulus, it's alive, otherwise, we don't really care.
-M	Sets the maximum number of sockets that will be used in parallel for TCP connect() scan, not our preferred method of scanning, but we'll give it a shot.
-sS	Sends only a TCP SYN packet for the scan. This is a basic TCP packet, with no options set.
-n	Never reverse DNS on IP addresses that respond.
-r	Don't randomize ports. I add this for readability, and scanrand does not randomize ports.
-T	Sets canned timeouts for values such as host_timeout, max_rtt_timeout, min_rtt_timeout, initial_rtt_timeout, etc. Valid sets are Paranoid, sneaky, polite, normal, aggressive, and insane.

Table 1-2: Interesting nmap options.

The closest option I see for the effect we are looking for (parallel scanning across hosts) is the -M option. The manual page says this is only used in TCP connect() scans, which is not exactly what we want to use, but if we want to do parallel scanning, we'll have to settle.

We'll put these options to good use to see what nmap can do to represent the traditional scanners in scanning large networks when put up against the new kid on the block, scanrand.

The Showdown... nmap

We'll time and scan around a 16,000 host network with both tools to see how each performs under scanning large networks. Though I don't control a

network with thousands upon thousands of hosts, I do have an internal network made up of a mere four computers. *Figure 1-2* outlines my network which the scanning will occur on. I will be scanning from 192.168.64.2 as it is the machine with the most
power.

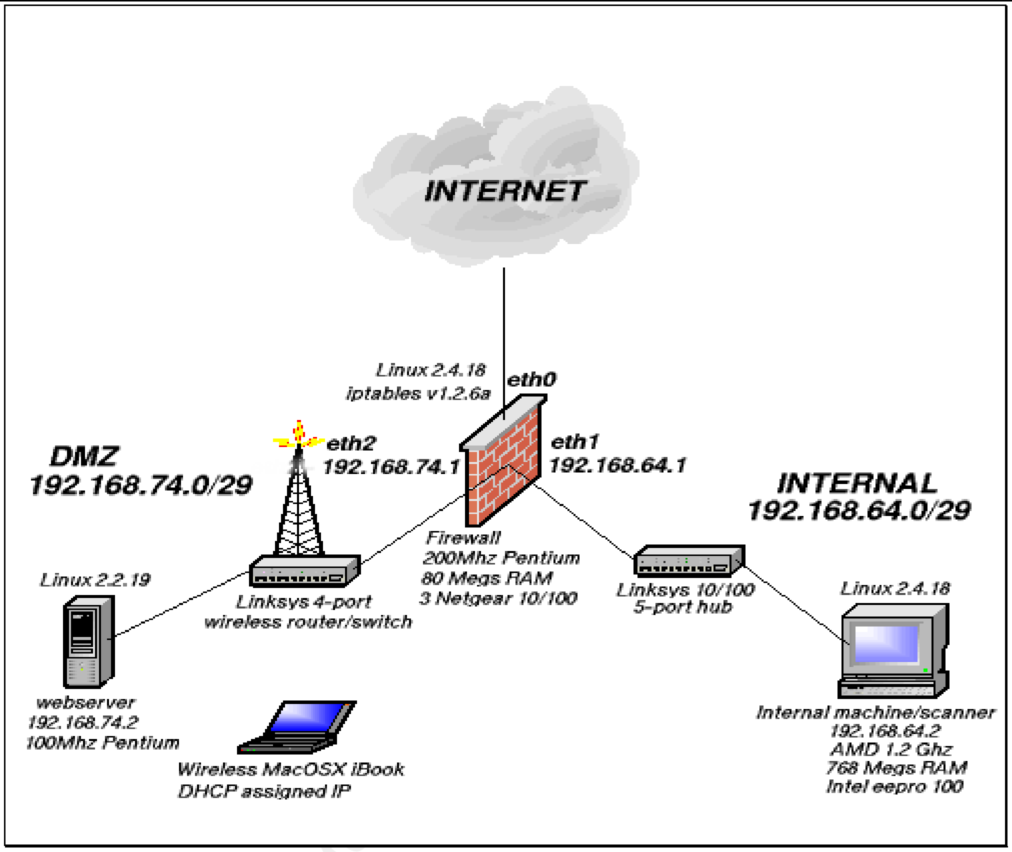

Figure 1-2: My network

The fact that thousands of hosts don't exist on my network does not keep me from scanning thousands of 'hosts'. With a little iptables magic, we can simulate thousands of websites running on the 192.168.0.0/18 network if we wish. We accomplish this by adding a rule to the firewall simliar to the one below for each website we wish to simulate where <host> is the IP we want to act as a webserver.

```
# iptables -A PREROUTING -s 192.168.64.2 -d <host> -j DNAT --to
192.168.74.2
```

This rule will NAT all traffic from the scanning machine to the webserver in the DMZ [3]. This gives us the effect that any 192.168.0.0/18 addresses could be running a webserver. I've created a little perl script to generate rules for the firewall to allow the scanner to connect to port 80 on 192.168.74.2 through this method.

```
    #!/usr/bin/perl -w
```

```
$i=0;
$j=0;

while ($i <= 63) {
  $j=0;
  while ($j <= 254) {
    if (rand() < 0.005) {
      open(OUTPUT, ">>rules.txt") or die "Could not open file:
$!\n";
      print OUTPUT "/sbin/iptables -A FORWARD -s 192.168.64.2
-d 192.168.$i.$j -p tcp --dport 80 -j ACCEPT\n";
    }
    $j++;
  }
$i++;
}
```

The particular run of our perl script generated 72 possible webservers on the 192.168.0.0/18 network. Now that I know how many webservers are on our site, I can accurately judge the data loss in scanning. The traffic to the other IP addresses besides the one generated by our perl script will be silently dropped.

Now that the firewall and network is in place, let the games begin!
The venerable nmap will be the first to bombard my poor network.

```
# nmap -version
nmap V. 3.10ALPHA4
# nmap -v -n -P0 -sS -T 5 -p 80 192.168.0.0/18

Host 192.168.0.1 appears to be up ... good.
Initiating SYN Stealth Scan against 192.168.0.1
The SYN Stealth Scan took 2 seconds to scan 1 ports.
Interesting ports on 192.168.0.1:
Port       State       Service
80/tcp     filtered    http

Host 192.168.0.2 appears to be up ... good.
Initiating SYN Stealth Scan against 192.168.0.2
The SYN Stealth Scan took 2 seconds to scan 1 ports.
Interesting ports on 192.168.0.2:
Port       State       Service
80/tcp     filtered    http

Host 192.168.1.138 appears to be up ... good.
Initiating SYN Stealth Scan against 192.168.1.138
Adding open port 80/tcp
The SYN Stealth Scan took 0 seconds to scan 1 ports.
Interesting ports on 192.168.1.138:
Port       State       Service
80/tcp     open        http
```

Clearly, nmap is much faster when there is a host to respond to its probe as seen in the output above to 192.168.1.138. The tcpdump dump below gives us further insight into nmap's scanning method.

```
02:00:06.232252 192.168.64.2.61623 > 192.168.0.1.80: S 4007069542:4007069542(0)
02:00:06.549103 192.168.64.2.61624 > 192.168.0.1.80: S 45489116:45489116(0)
02:00:06.869128 192.168.64.2.61625 > 192.168.0.1.80: S 2118562022:2118562022(0)
```

```
02:00:07.189156 192.168.64.2.61626 > 192.168.0.1.80: S 4007069542:4007069542(0)
02:00:07.609128 192.168.64.2.61627 > 192.168.0.1.80: S 45489116:45489116(0)
02:00:07.929149 192.168.64.2.61628 > 192.168.0.1.80: S 2118562022:2118562022(0)
02:00:08.253109 192.168.64.2.61623 > 192.168.1.138.80: S
3456156783:3456156783(0)
02:00:08.255327 192.168.1.138.80 > 192.168.64.2.61623: S
2223686605:2223686605(0) ack 3456156784 win 16080 <mss 536> (DF)
02:00:08.255470 192.168.64.2.61623 > 192.168.1.138.80: R
3456156784:3456156784(0) win 0 (DF)
```

As shown, nmap sends up to six SYN packets to the target until it gives up. These packets are sent in time intervals depending on what -T value you use, or by setting the option -max_rtt_timeout manually. This can be very costly(~1.7secs/machine [with -T 5]) in scanning large networks where big chunks of IP addresses don't have a machine listening.

Let's do a little tweaking[2] and not see if we can speed this up just a little. Since nmap tries to scan before knowing if the host is up, we need to remove the -P0 as we only want to scan the hosts that appear to be up. -P0 is very helpful if you know a host is up, but blocking nmap's attempt at determining if it's up.

```
# nmap -v -n -sS -T 5 -p 80 192.168.0.0/18

[ ... ]
Host 192.168.63.254 appears to be down, skipping it.
Host 192.168.63.255 appears to be down, skipping it.
Nmap run completed -- 16384 IP addresses (70 hosts up) scanned
in 194.322 seconds
```

That's more like it. 16,384 hosts in 194 seconds is not bad at all. Of course there were repercussions for such speed, which is loss of data accuracy. On a local network nmap using -T 5 missed 2 possible servers. Backing down on -T a little should give us all servers which it does on our local network. On a fast network that gives us 84.5 hosts/second. The time would, of course, increase as we decreased the -T value, while the accuracy increased. I would not recommend trying -T 5 on anything but a local networks. It will very unreliable over internet connections as it does not way very long (0.3 seconds) for hosts to respond. This is a weakness in the traditional style scanning where hosts are scanned in sequence one by one. Next, we'll see how scanrand fairs in the scanning competitions.

The Showdown... scanrand

Before we begin scanning with scanrand, let's take a look at the output we would expect from scanrand since there is not much documentation on it. I scanned a fairly small network (with permission of course) of a friend's site to get some output with scanrand as shown below.

```
  UP: 10.0.0.3:80   [18]   0.181s
un03: 10.0.0.8:80   [18]   0.274s( 192.168.64.2 -> 10.0.0.8   )
  UP: 10.0.0.22:80  [19]   0.558s
  UP: 10.0.0.10:80  [18]   0.890s
  UP: 10.0.0.30:80  [18]   1.243s
  UP: 10.0.1.3:80   [18]   5.261s
un01: 10.0.0.116:80 [20]   5.429s( 192.168.64.2 -> 10.0.0.116 )
```

[2] I could never get nmap to do parallel scannin g across hosts like I had originally hoped to do though it does do it across ports.

```
255 = 10.104.191.97|80 [17]1673.047s(192.168.64.2 -> 10.1.73.8)
```

The first column is the status of the machine probed. The common statuses can be found in *Table 1-3.*

UP	We received a SYN/ACK from this machine, the specified port is listening.
DOWN	We received an ACK/RST.
Un**XX**	We received an ICMP unreachable packet in regards to this connection. XX represents the type and code of the ICMP message.
X =	We received an ICMP time exceeded message.

Table 1-3: Scanrand statuses

The second column is the machine we received a response from whether it be a ICMP error or a SYN/ACK.

The third column in brackets is the estimated hop count to the target machine based on the time-to-live(TTL) of the IP packet we received as a response. The estimation algorithm, which is essentially listed below[1], takes advantage of the fact that most operating systems use a multiple of 32 as their TTL values. There are some other calculations performed in specific situations. (Particularly with resets).

```
passive_factor = 32;
if(ttl%passive_factor == 0) ttl--;
return(passive_factor - (ttl%passive_factor));
```

The fourth column is the number of seconds elapsed since the beginning of the scan until the host responded. Note, this is not the time elapsed from when the SYN packet was sent out by the scanning host.

The fifth column is only shown when we get a valid ICMP message as a response. RFC 792[2], which defines the ICMP protocol, states that ICMP unreachable messages, as well as time-exceeded messages (scanrand looks for either), must provide the first 8 bytes of the original IP datagram header. Scanrand displays this data in column 5.

Now that we can read the output of scanrand, let's put it to work.

```
# scanrand -c -b0 -t 3000 192.168.0-63.1-254:80

# tcpdump -s 1500 -i eth0 -w scanrand.tcpdump net 192.168.0.0/18
'(tcp and port 80) or icmp
```

The -c tells scanrand to verify that icmp responses are not spoofed. The -b0 option tells scanrand to send packets as fast as possible (default). We could limit the rate of packet output by specifying something like -b1M if we were on a 1 meg link. The -t 3000 option is important. This is how long scanrand will wait for a response (any response) before exiting. I set this high since I knew I was not going to get many responses and I wanted my scan to finish. Scanrand doesn't seem to take the standard CIDR network masks like nmap does which is a little annoying. Networks are specified using ranges and commas. Our scan above is going to cover the 192.168.0.0/18. These are the most useful options of scanrand though more options, as well as some experimental ones, can be found by running scanrand with no options.

Scanrand scanned blazingly fast. A few seconds after issuing the command, my mouse stopped working, and my computer became unresponsive. Another few seconds and it came back alive. A 'quick' top showed scanrand eating all available CPU time as it blasted as many packets as it could towards its several thousand destinations. Even if my 'weak' computer was not enough for the greedy scanrand, it managed to shoot off an impressive amount of packets. Too bad my network could not handle them. The first time with -b0 enabled, scanrand reported only 30 of the 72 webservers. A review of the tcpdump file on the scanning box only showed this many response packets coming back to the scanner so they were being dropped.

A little tweaking around and I found the needed -b value.

```
[ ... ]
UP:   192.168.55.227:80    [01]  28.302s
UP:    192.168.57.39:80    [01]  28.929s
UP:   192.168.59.101:80    [01]  30.045s
UP:   192.168.59.213:80    [01]  30.276s
UP:   192.168.59.226:80    [01]  30.299s
UP:    192.168.60.22:80    [01]  30.402s
UP:   192.168.60.208:80    [01]  30.768s
UP:     192.168.61.5:80    [01]  30.884s
UP:    192.168.62.45:80    [01]  31.494s
UP:   192.168.62.252:80    [01]  31.928s
UP:    192.168.63.81:80    [01]  32.113s
UP:   192.168.63.128:80    [01]  32.241s
```

After only 32.241 seconds using -b85M, scanrand had found all 72 webservers and successfully probed all 16,384 addresses, consistently (although the time did reach all the way up to 68 seconds sometimes though this is more than likely a firewall issue). That averages to 512 hosts/sec. Pretty impressive.

And the winner is...

For scanning an internal network for rogue servers, such as an unauthorized webserver, scanrand has the upper hand for accuracy and speed. On the other hand, scanrand lacks the maturity and features that makes nmap such a great tool.

For bypassing stateless router ACLs looking for servers and other various tricks easily capable with nmap, scanrand just can't compare though it's not supposed to. I can't say for sure what Dan has in mind for scanrand as far as features, but I doubt it was ever intended to be a replacement for nmap.

Scanrand and nmap together make an excellent combination as scanrand can be used to find servers quickly and accurately that you would use nmap to enumerate further on. From my personal experience, nmap seems to do better scanning individual hosts over multiple ports (not to mention the OS fingerprinting and a plethora of other options.).

Detecting Scanrand

Scanrand makes no attempt in covering up what it is. As shown below, both version 1.0 and 1.10 use an IP id of 255 and a TTL of 255 on outgoing SYN packets.

```
16:01:06.450963 192.168.74.2.26922 > 192.168.64.25.80: S [tcp sum ok]
2957819677:2957819677(0) win 4096 (DF) (ttl 255, id 255, len 40)
16:01:06.470964 192.168.74.2.26922 > 192.168.64.26.80: S [tcp sum ok]
3296526466:3296526466(0) win 4096 (DF) (ttl 255, id 255, len 40)
16:01:06.490961 192.168.74.2.26922 > 192.168.64.27.80: S [tcp sum ok]
811733058:811733058(0) win 4096 (DF) (ttl 255, id 255, len 40)
16:01:06.510955 192.168.74.2.26922 > 192.168.64.28.80: S [tcp sum ok]
646406728:646406728(0) win 4096 (DF) (ttl 255, id 255, len 40)
16:01:06.530956 192.168.74.2.26922 > 192.168.64.29.80: S [tcp sum ok]
1922740421:1922740421(0) win 4096 (DF) (ttl 255, id 255, len 40)
16:01:06.550955 192.168.74.2.26922 > 192.168.64.30.80: S [tcp sum ok]
498232123:498232123(0) win 4096 (DF) (ttl 255, id 255, len 40)
```

This is simple enough to detect. If we see many probes from one host with a TTL set to 255 and id set to 255 we know they are using scanrand. Also, we see the same source port being used (across multiple hosts) which is not normal. The source port changes on each innvocation of scanrand. A simple snort signature to detect scanrand being used would be:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"Possible
scanrand Scan"; id:255; ttl:255; flags:S; cla sstype:attempted-
recon; sid:9992; rev:1;)
```

Of course this signature could very easily trigger on valid traffic as an IP id of 255 with a TTL of 255 in a SYN packet is valid traffic. The chances of this combination coming up in an ACK packet is more likely though the chance is there for a SYN. Below we see a SSH session when the IP id rolls over from 65535 to 0 and eventually increase to 255. The signature will never trigger between these Linux boxes as thier TTL values are initially 64. It is more likely to trigger when a Solaris 2.x machine is on the network as it is reported to use an initial TTL of 255. Other operating systems, such as some versions of Cisco IOS, are also known to use 255 as a TTL.

```
22:43:17.842460 192.168.74.2.22 > 192.168.74.1.1027: P 359760:359984(224) ack
4839 win 32120 <nop,nop,timestamp 7557181 31165180> (DF) [tos 0x10]  (ttl 64, id
65535, len 276)
22:43:17.844996 192.168.74.2.22 > 192.168.74.1.1027: P 359984:360208(224) ack
4839 win 32120 <nop,nop,timestamp 7557181 31165180> (DF) [tos 0x10]  (ttl 64, id
0, len 276)
22:43:17.847510 192.168.74.2.22 > 192.168.74.1.1027: P 360208:360416(208) ack
4839 win 32120 <nop,nop,timestamp 7557181 31165180> (DF) [tos 0x10]  (ttl 64, id
1, len 260)
22:43:17.851393 192.168.74.2.22 > 192.168.74.1.1027: P 360416:360800(384) ack
4839 win 32120 <nop,nop,timestamp 7557182 31165182> (DF) [tos 0x10]  (ttl 64, id
2, len 260)
```

Another intersting sidenote about scanrand is that is does not verify ICMP error messages from scanned hosts *by default*. The –c option must be explicitly set to make scanrand verify ICMP error packets are indeed a valid reply. Using hping2, and a home-brewed patch (see Appendix A), we can generate a flood of bad data.

From a host that notices a scanrand scan, we issue our hping command:

```
# hping2 -d 25 --icmptype 3 --icmpcode 1 192.168.74.2 --icmp-
saddr 192.168.64.1 --icmp-daddr 192.168.74.1 --fast

HPING 192.168.74.2 (eth0 192.168.74.2): icmp mode set, 28
headers + 25 data bytes
```

This command sends a barrage of ICMP destination unreachable to the scanning host that floods it with bogus information. The only problem is that valid responses are still sent by the listening hosts as well. This technique *could* be useful depending on who's behind the scanning host.

On the scanrand users console, we get:

```
un01: 192.168.74.1:2222  [01]   1.868s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   1.968s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.068s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.168s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.268s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.368s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.468s( 192.168.64.1 -> 192.168.64.2 )
un01: 192.168.74.1:2222  [01]   2.568s( 192.168.64.1 -> 192.168.64.2 )
```

which was all generated from our hping command. It would be trivial to create a script for hping to send some ICMP unreachable error message for many of our hosts. Moral of the story: use –c.

Conclusion

While the concept of scanrand and the tool itself are very useful and innovative, I can see this technology being abused. Besides the obvious, this type of scanning could greatly increase the ability of a worm to find hosts on the Internet to infect. This could mean faster propagation of worms. Of course, there is always the good and the bad of any technology, and network administrators might use it to find exposed ports on their local networks in a quick and accurate fashion.

Scanrand is a tool that any network administrator should take a look at for auditing internal machines or other various tasks involving network service discovery.

References

[1] Kaminsky, Dan. "Paketto Keiretsu source code" www.doxpara.com. Dec 24, 2002. URL: http://www.doxpara.com/paketto/paketto-1.10.tar.gz

[2] Postel J. "RFC 793 - INTERNET CONTROL MESSAGE PROTOCOL" www.rfc-editor.org Sept, 1981. URL: ftp://ftp.rfc-editor.org/in-notes/rfc792.txt

[3] Russell, Rusty. "Linux 2.4 NAT HOWTO" www.netfilter.org. Jan 14, 2002. URL: http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html

[4] Schneier, Bruce. Applied Cryptography John Wiley and Sons. 1996. 2nd ed.

[5] Stevens, Richard W. UNIX Network Programming. Prentice-Hall. 1998. vol 1. 2nd ed.

[6] Stevens, Richard W. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994.

Assignment #2: Three Network Detects

Detect #1: Opaserv runs rampant

Log Entries

```
[**] [1:533:5] NETBIOS SMB C access [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/24-08:23:43.543993 1.1.2.78:1387 -> 1.1.4.200:139
TCP TTL:126 TOS:0x0 ID:35444 IpLen:20 DgmLen:103 DF
***AP*** Seq: 0x4906A3  Ack: 0x8FFB504D  Win: 0x217C  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS339]

...;.SMBu......................................!\\COMPUTER1\C.A:.

[**] [1:533:5] NETBIOS SMB C access [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/24-08:23:43.704422 1.1.2.78:1388 -> 1.1.4.201:139
TCP TTL:126 TOS:0x0 ID:36468 IpLen:20 DgmLen:106 DF
***AP*** Seq: 0x490757  Ack: 0x6623F589  Win: 0x217C  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS339]

...>.SMBu......................................!\\COMPUTER2\C.A:.

[**] [1:533:5] NETBIOS SMB C access [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/24-08:23:44.360730 1.1.2.78:1389 -> 1.1.4.202:139
TCP TTL:126 TOS:0x0 ID:41588 IpLen:20 DgmLen:103 DF
***AP*** Seq: 0x4909CA  Ack: 0x5D06BA13  Win: 0x217C  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS339]

...;.SMBu......................................!\\COMPUTER3\C.A:.

[**] [1:9995:1] Virus - W32.Opaserv Worm Infection [**]
[Classification: Misc activity] [Priority: 3]
10/24-08:58:53.072830 1.1.2.78:1327 -> 1.1.4.254:139
TCP TTL:126 TOS:0x0 ID:15989 IpLen:20 DgmLen:105 DF
***AP*** Seq: 0x484A71  Ack: 0x12C29F57  Win: 0x2178  TcpLen: 20

...=.SMB.......................................WINDOWS\scrsvr.exe
```

Source of Trace

The alerts above were generated on a customer's network we monitor. This is interesting traffic because it is coming from an "internal" user. The source address is from a user dialed in to the company's modem bank in order to do some work from home. The modem bank is then connected from the extranet to the internal network over a VPN tunnel. See *Figure 2-1*.

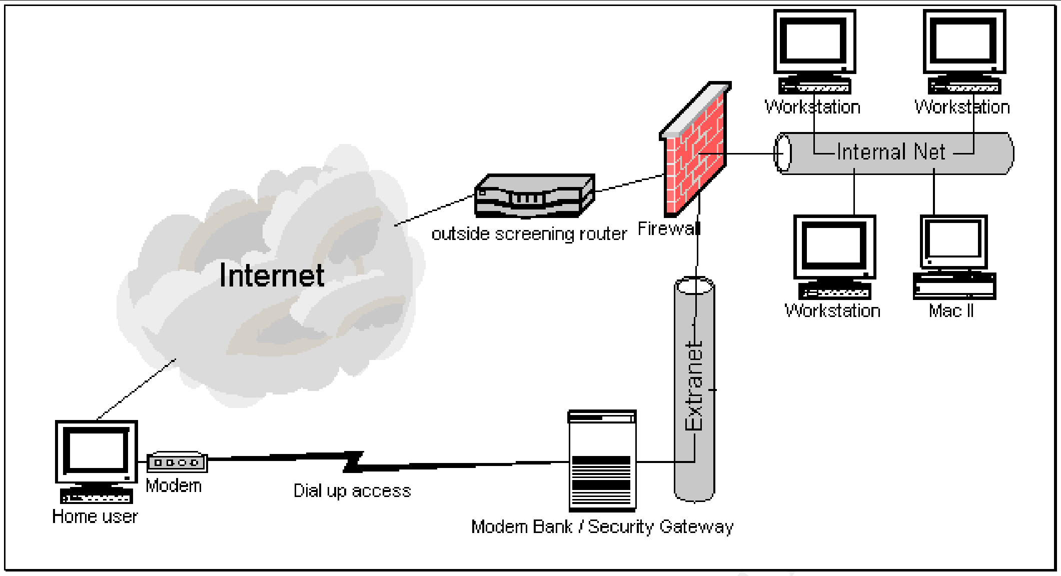

Figure 2-1: Network Topology of detect.

Detect was generated by

The alerts were generated by Snort version 1.8.6 (Build 105). This snort box sees all traffic from clients who are remotely connected through VPN from the Internet or through the modem bank. This sensor sees its fair-share of action since the traffic that comes through them are typically from people connected to the corporate network through their home machine. This is the same machine their kids probably use to check their email, download games and music, and other activities that could put them in harms way if they were not familiar with the Internet. This seems to be an increasing problem as our company see lots of trojan type traffic bang up against the firewall as they attempt to 'phone home.'

The snort signature that generated the first three alerts is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB
C$ access"; flags:A+; content: "|5c|C$|00  41 3a
00|";reference:arachnids,339; classtype:attempted-recon;
sid:533; rev:5;)
```

This signature looks for a packet with the ACK flag, plus any other TCP flag set, and the payload of |5c|C$|00 41 3a 00|. In our alerts, we see this criteria is met. This signature alerts on an *attempt* to access the C share. It does not necessarily mean the attempt was successful. When we saw a multitude of the C share attempts coming from this machine, we assumed it was the W32.Opaserv[3] worm but had no positive proof as we could not get someone on that machine at the time. A colleague of mine, Joe Stewart, wrote the following filter to detect an Opaserv infection.

```
alert tcp any any -> any 139 (msg:"Virus - W32.Opaserv Worm
Infection"; flags:A+; content: "|5c 73 63 72 73 7 6 72 2e 65 78
65|"; sid:9995; classtype:misc-activity; rev:1;)
```

[3] W32.Opaserv (or Opasoft) is the name Symantec has given this worm. Other anti-virus software vendors have supplied their own name although all are simliar to the term 'Opaserv' which is the name I will use to refer to this worm.

To figure out exactly what this signature is looking for, we can use a couple of unix tools I stumbled across on my system in order to manipulate hex.

```
$ echo -n '5c 73 63 72 73 76 72 2e 65 78  65' | xxd -r -p; echo
\scrsvr.exe
```

We see this filter is looking for a file named 'scrsvr.exe' to be transferred to a host over port 139. In case we ever need to, to go the other way, we can use the following command:

```
$ echo -n '\scrsvr.exe' | xxd
0000000: 5c73 6372 7376 722e 6578 65              \scrsvr.exe
```

Shortly after putting this filter in place, we got the 4th alert in the log section indicating that the Opaserv worm had successfully infected a machine. With this signature in place, we could tell the customer exactly which machines were being infected for proper cleaning. This alleviated cleanup of the worm since we could track which machines it spread to and clean them before it could spread further.

Probability the source address was spoofed

The probability the source address was spoofed is extremely low to none. In order to try and access Microsoft Windows network shares, a full TCP three-way handshake must be completed in order to transfer the data needed to authenticate a netbios session. The established connection would also have to be there for Opaserv to replicate itself to the target machine. There is also no known variant of the worm that attempts spoofing of any kind.

Attack mechanism

This attack exploits a vulnerability in the password scheme Microsoft Windows has provided for share-level access. It is possible for the worm to get read and write access to a network share without knowing the password even if the share is password protected. The worm also infects C shares that are not password protected at all. Once the worm has found a share, it creates a C:\tmp.ini, some registry keys to start the worm on boot, and copies over C:\window\scrsvr.exe which is the worm executable. On the next boot, the infected machine begins to scan the network for more network shares to infect.

Correlations

Symantec has a virus description with full details and removal instructions of the Opaserv worm which matches the exact behavior seen in this detect.

Microsoft has released a security bulletin which explains the vulnerability that Opaserv exploits when share-level access of the C drive is enabled.

Many variants of the worm have been found in the wild. McAfee describes many of the variants in their virus description.

Vulnerable versions of Windows include: Windows 95,98,98SE, and Me. Any version of Windows with no password set on a C share is vulnerable as well.

Suprisingly, I could not find another practical that analyzed this traffic.

Evidence of active targeting

The infected machine follows a certain algorithm for picking its victims. McAfee's virus description also explains this algorithm.

Essentially, the worm issues WINS queries to contagious IP addresses on the local subnet to get netbios names of machines. It then tries to infect the C share by connecting to the netbios name of the machine. As you can see from the log files, this particular version of the Opaserv worm also tries to infect the A drive share.

Afterwords, the last octet of the network address is increased by 1 and the process repeats. When all these IP addresses on the local net have been exhausted, a random IP address is chosen, and the process starts from that address.

So yes, the worm actively targets the local network first, then branches off to remote hosts.

Severity

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

Criticality = 2
The machines targeted by this worm are most likely user machines. The customer did not have any critical servers running Windows.

Lethality = 4
When the worm infects a machine, no damage is done. It simply propagates to exist without harming the host (at least this variant). Regardless, the amount of noise the worm generates acts as a 'beacon' for an infected machine. A potential attacker sees this machine scanning his network and connects back to the machine viewing the share that is unprotected. This is a major privacy exposure. The user also has the ability to remove or change files on the system, essentially gaining remote administration. A binary could be loaded, then set to run at startup allowing even more dastardly activity.

System Countermeasures = 1
The machine was vulnerable to the Opaserv worm because of an open network share and because no anti-virus software was installed.

Network Countermeasures = 3
The perimeter firewall does not allow netbios traffic to pass either way. Therefore the virus would be contained to the corporate network (At least while the machine was connected through VPN). The network has two snort machines that could detect and track the propagation of the worm for preventative cleaning. I originally had this value at 4, however the VPN/modem bank security gateway is allowing bidirectional netbios traffic, so

I have decreased this value to 3.

According to the equation, severity is: (6 - 4) = 2.

Defensive recommendations

Educate users on the importance of password protected network shares. Never share C drive completely. Share only directories needed to be shared as this will keep the Windows or WINNT directory from being exposed. All affected versions of Windows needs to install the security update from Microsoft to keep the Opaserv worm out of even password protected C shares. Deny netbios traffic internally where possible but especially from the dial-up, and user VPN subnets. Enforce/Create company policy of having anti-virus software on all machines that connect to the corporate network. Proactive scanning of the network for open C shares and vulnerable systems is also a possibility for finding vulnerable systems first.

Multiple choice test question

Opaserv can spread through which means?

A. C shares with no password.
B. IPC$ share
C. Unpatched Win95/98/Me with C shared
D. A and C
E. A, B, and C

Correct answer is D: A and C.

The Opaserv worm spreads through C shares with no password. Also, if a password is set with C shared, and you have not applied the security patch referenced in the advisory on this bug to your Windows 95/98/Me box, you are also vulnerable.

Detect #2: Apache under attack!

Log Entries

```
[**] [1:1809:1] WEB-MISC Apache Chunked-Encoding worm attempt [**]
[Classification: Web Application Attack] [Priority: 1]
11/22-22:32:24.737836 139.130.70.67:1544 -> 1.1.2.200:80
TCP TTL:48 TOS:0x0 ID:53364 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0xB77687D2  Ack: 0x1E2577E6  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 80054463 2197741280
[Xref => http://www.securityfocus.com/bid/4474]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079]
[Xref => http://www.securityfocus.com/bid/5033]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392]

POST / HTTP/1.1..Host: Unknown..X-CCCCCCC: AAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAhGGGG..1.PPPP..$.SPP1
.1........1.....r....D$..|$. u.1..D$..D$. .d$..D$..D$..D$..T$..T
$...$1..]..1..,$s'1.PPPP..$T..$..$..$..$QP....XXXXX<Ot.XXA.. u..
..1.PQP1..Z...D$..|$..u.1.P..$..4$.hBLE*h*GOB....PS..PP....1.Phn
/shh//bi..PS..PQSP.;.....X-CCCCCCC: AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


[**] [1:1807:1] WEB-MISC Transfer-Encoding: chunked [**]
[Classification: Web Application Attack] [Priority: 1]
11/22-22:32:29.656002 139.130.70.67:1544 -> 1.1.2.200:80
TCP TTL:48 TOS:0x0 ID:53820 IpLen:20 DgmLen:510 DF
***AP*** Seq: 0xB7770442  Ack: 0x1E2577E6  Win: 0x43E0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 80054946 2197741764
[Xref => http://www.securityfocus.com/bid/4474]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0079]
[Xref => http://www.securityfocus.com/bid/5033]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392]

.........................................................X-AAAA
: ..............................................................
X-AAAA: ........................................................
......X-AAAA: ..................................................
............X-AAAA: ............................................
..................X-AAAA: ......................................
........................Transfer-Encoding: chunked....5..BBBBB..
ffffff6e..
```

Source of Trace

The alerts were generated by a snort machine sitting in a DMZ of a fairly standard network configuration. See *figure 2-2.* This segment sees mostly http, https, and smtp traffic (a lot of it at that). The destination machine is a public webserver for the corporation.

Detect was generated by

Snort version 1.8.6 (Build 105) generated the alerts. The signature that snort alerted on was:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-MISC
Apache Chunked-Encoding worm attempt"; flags:A+;
content:"CCCCCCC\: AAAAAAAAAAAAAAAAAAA"; nocase; classtype:web-
```

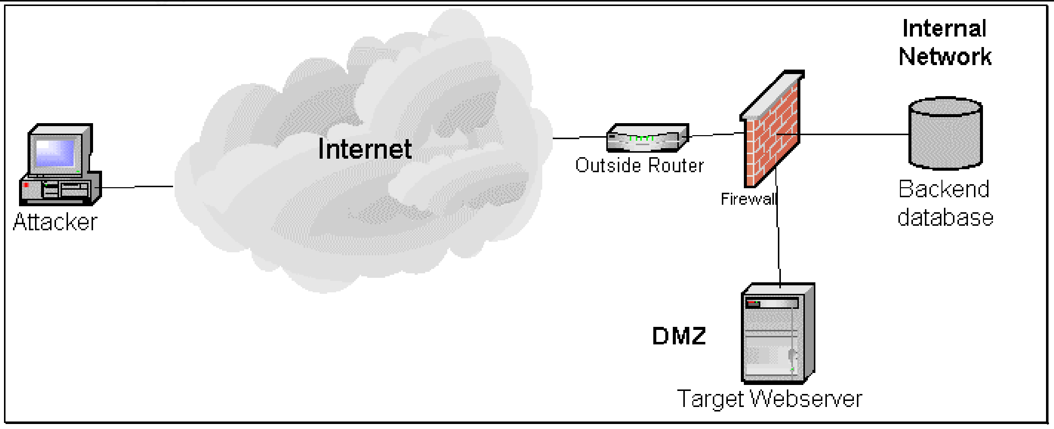

Figure 2-2. Network Topology for detect #2

```
application-attack; reference:bugtraq,4474; reference:cve,CAN-
2002-0079;reference:bugtraq,5033; reference:cve,CAN-2002-0392;
sid:1809; rev:1;)
```

The signature alerts when an 'established' connection is made to port 80 and the specified content is passed to the server. The worm uses all A's to attempt to overflow a buffer in apache which is what this signature is looking for. This signature does not catch any attempts made to ports other than 80. Also, a simple modification to the worm could bypass this filter. For instance, replace the capital A's with B's. This would be particularly easy since the source code is freely available.

Probability the source address was spoofed

It is unlikely that the source address is spoofed. This exploit needs to create an established connection with the apache webserver process in order to exchange the exploit data. If the exploit is successful, remote access of the box is possible which would also require the established connection. Also, there is no attempt of spoofing in the source code of this worm.

Description of attack

The use of the chunked-encoding transfer by a webserver is usually in response to a client's request. This is needed since the client might not know the size of the data to be sent to the server as it's dynamically created. A server can also initiate chunked-encoding transfer method for the same reason. This is often the case with dynamic webpages created with PHP, Coldfusion or other similar applications.

The worm that generated the alerts exploits a particular problem with chunked-encoding transfers in Apache running on TCP port 80. The worm is identified by the 'content:' field in the snort signature as described in 'Detect was generated by' section of this detect. This bug is currently a CVE candidate (CAN-2002-0392).

Attack mechanism

The worm first issues a simple 'GET / HTTP/1.1' to get the headers returned by the webserver. This is not detected by snort, as it is a common method of obtaining the main page of a webserver. If a vulnerable version of Apache is returned in the version string, the worm makes another connection to the websever requesting the chunked encoding transfer method. This ability is enabled by default in Apache and is a normal mode of data transfer for webservers. The worm then sends enough A's to overflow a buffer and runs its malicious code. If the attack is successful, it transfers a .a, and .uaa binary to the /tmp directory on the target host. The .a accepts commands on udp port 2001 that allows it to act as a distributed denial of service agent (DDoS). The .uua binary is a uuencoded version of the .a binary.

Correlations

Although this bug has not been accepted as a CVE entry, it is a CVE candidate (CAN-2002-0392).

CERT describes this vulnerability in their advisory on this issue.

SecurityFocus has a discussion on this topic as well as exploit code for FreeBSD and OpenBSD.

RFC 2616 describes the purpose of the chunked-encoding data transfer method and when it is used for HTTP/1.1.

The most helpful reference on this topic is a link to the actual source code of the worm along with a brief analysis.

The worm is capable of exploiting the vulnerable versions of Apache running on FreeBSD. The vulnerable versions can be found on the SecurityFocus link above. Exploits are also known to exist for OpenBSD.

Martin Walker has written his GCIH practical on this particular worm.

Evidence of active targeting

By examining the source code of the worm, we can tell exactly how the worm targets machines and networks.

```
#define CLIENTS  128
#define SCANPORT 80

unsigned char classes[] = { 3, 4, 6, 8, 9, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 28, 29, 30, 32, 33, 34,
35, 38, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 61, 62, 63, 64, 65, 66, 67, 68, 80, 81, 128, 129, 130,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209,
210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 224, 225,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238,
239 };

[ ... ]

a=classes[rand()%(sizeof classes)];
b=rand();
c=0;
d=0;

[ ... ]

if (myip)
   for (n=CLIENTS,p=0;n<(CLIENTS*2) && p<100;n++)
     if (clients[n].sock == 0) {
       char srv[256];
       if (d == 255) {
         if (c == 255) {
           a=classes[rand()%(sizeof classes)];
```

```
          b=rand();
          c=0;
        }
        else
          c++;
        d=0;
      }
      else
        d++;

      memset(srv,0,256);
      sprintf(srv,"%d.%d.%d.%d",a,b,c,d);
      clients[n].ext=time(NULL);
      atcp_sync_connect(&clients[n],srv,SCANPORT);
      p++;
      [ ... ]
```

The `a`, `b`, `c`, and `d` variables represent the first, second, third and fourth octets in an Ipv4 IP addresses respectively. The first octet will be randomly picked from the `classes` array. The second octet is generated by the function `rand()`. The third and fourth octets start at 0 and increment up to 255. If the third octet reaches 255, then a new first octet is selected, and the second octet is randomized again. The third octet then starts at 0 again.

If there is a successful connect on port `SCANPORT` (80 by default), then the connection socket is saved in the `clients` array. After the scanning is done, (a max of 100 hosts per scan because of the `p` variable), then the exploit is run against each server.

There is no 'active targeting'. Your webserver is attacked by the worm if it happens to get your IP range through the above mentioned algorithm. If the first octet of your network does not appear in the `classes` array, then you should not see this worm ever (unless someone modifies the code and re-releases it).

Severity

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

Criticality = 5
The attacked machine was the companies main public webserver which is normally given a value of 4. However, this machine is also trusted to do back-end database queries to get customer information which makes it that much more critical. Customer information is highly sensitive, therefore I have bumped the criticality of this machine from 4 to 5.

Lethality = 5
If the exploit is successful, it is possible for the attacker to upload and run an arbitrary binary. The default worm uploads a DDoS agent, however a decent coder could modify the source code to upload any kind of binary.

System Countermeasures = 4
The only reason I give this category a 4 is because the victim operating

system is Linux, which is not exploitable by this worm. There have been rumors of a Linux exploit for this vulnerability, but no source code to prove it that I could find (see comments in apache-scalp.c and apache-nosejob.c). Regardless, this worm is unable to exploit Linux systems (In its current state).

```
$ telnet 1.1.2.200 80
Trying 1.1.2.200...
Connected to 1.1.2.200.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 22 Nov 2002 22:34:10
Server: Apache/1.3.24 (Unix)  (Red-Hat/Linux)
Connection: close
Content-Type: text/html

Connection closed by foreign host.
```

Good thing this box is not running on FreeBSD or it would have, more than likely, been exploited.

Network Countermeasures = 4
The segment the webserver sits on is being monitored by a Snort machine that has the ability to detect this event (As seen from this detect). The snort machine is monitored 24x7x365 which means malicious traffic is seen by a human which can take appropriate action based on what he/she sees.

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

Even if there is not highly available exploit code for vulnerable versions of Apache running on Linux, I sure couldn't sleep at night if this were my server. The fact is remote execution of malicious code *could* be possible is enough for me. Also, I've talked to some administrators that claim that it's possible to DOS Apache on Linux with this bug.

Defensive recommendations
As suggested by the Apache group, upgrade to at least 1.3.26 in the 1.3 branch of Apache and upgrade to at least 2.0.39 in the 2.0 branch. Also, block this IP address at a border router or firewall while Apache boxes can be upgraded to a non-vulnerable version.

Multiple choice test question
The worm described in this detect can infect which of the following systems:

A. Apache 1.3.14 on Linux
B. Apache 1.3.26 on FreeBSD
C. Apache 1.3.24 on FreeBSD
D. Apache 1.3.24 on Windows
E. B or C

Correct Answer: C.

This particular worm can infect FreeBSD webservers running a vulnerable version of Apache. Apache 1.3.24 is vulnerable as described in the advisories.

Detect #3: Data in TCP SYN packet strangeness

Log Entries

```
[**] [1:526:4] BAD TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/24-05:28:25.554488 159.54.34.3:2300 -> 138.97.18.88:53
TCP TTL:242 TOS:0x0 ID:60280 IpLen:20 DgmLen:64
******S* Seq: 0x7DDBEE51  Ack: 0x0  Win: 0x800  TcpLen: 20
[Xref => url www.cert.org/incident_notes/IN-99-07.html]

[**] [1:526:4] BAD TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/24-05:28:25.554488 159.54.34.3:2301 -> 138.97.18.88:53
TCP TTL:238 TOS:0x0 ID:18502 IpLen:20 DgmLen:64
******S* Seq: 0x71E07814  Ack: 0x0  Win: 0x800  TcpLen: 20
[Xref => url www.cert.org/incident_notes/IN-99-07.html]

[**] [1:526:4] BAD TRAFFIC data in TCP SYN packet [**]
[Classification: Misc activity] [Priority: 3]
08/24-05:28:25.554488 159.54.34.3:2302 -> 138.97.18.88:53
TCP TTL:242 TOS:0x0 ID:57612 IpLen:20 DgmLen:64
******S* Seq: 0x6DC1B972  Ack: 0x0  Win: 0x800  TcpLen: 20
[Xref => url www.cert.org/incident_notes/IN-99-07.html]
```

Source of Trace

The detect was obtained from a raw log at http://www.incidents.org/logs/Raw/2002-7.24. I chose these particular alerts because I had not run across them before in my daily analysis of customers' networks, and a quick search on Google did not reveal an extensive amount of information on the events.

As far as the network layout is concerned, I can only make presumptions since it is unknown to me. I assume the source is from an external address, passing through an outside screening router or firewall into a DMZ where a snort machine resides along with the target machine.

Detect was generated by

The detect was generated by Snort version 1.9.0 (Build 209). The rule that generated this event is:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC
data in TCP SYN packet"; flags:S; dsize:>6;
reference:url,www.cert.org/incident_notes/IN-99-07.html;
sid:526;  classtype:misc-activity; rev:4;)
```

As described in the snort rule, the traffic that generated this alert was a TCP SYN packet that also had a data payload. In all traffic similar to this, the payload was 24 null bytes as show with tcpdump below.

```
13:14:34.524488 216.33.87.10.2200 > 138.97.18.88.53:  S
```

```
356149851:356149875(24) win 2048 0 [0q] (22)
0x0000   4500 0040 86dd 0000 f206 ffb5 d821 570a
0x0010   8a61 1258 0898 0035 153a 6a5b 0000 0000
0x0020   5002 0800 1343 0000 0000 0000 0000 0000
0x0030   0000 0000 0000 0000 0000 0000 0000 0000
```

The trailing underlined 0's are the payload that snort detected in this TCP SYN packet. We can confirm this by checking the size of the IP and TCP header. The underlined '5's are the IP and TCP header size respectively. Both must be multiplied by four to determine the byte length. In both headers, the size is 20 bytes which is a normal size for both headers with no options set. This leaves 24 null bytes not encompassed by TCP or IP headers which tcpdump indicates by the underlined 24 in parenthesis after the TCP sequence number. The 'dsize' in the snort rule refers to the size of the payload. Since we have a payload of 24 bytes, it fired.

Probability the sourc e address was spoofed

At first glance it seems likely that the source addresses were spoofed. First of all, the packets seemed to have been received at the same time. If packets are logged at the same time, from the same source address, I would expect the TTLs to be the same. Note on one of the packets, the TTL is four less than the others. If that packet had traversed a different route to the destination (since TTLs are different), I would expect the arriving time to be a little different! Also, if these packets were generated on the same machine, at the same time, I would expect the IP ids to be close to each other. The IP ids in these packets are 60280, 57612, and 18520. These are pretty big differences for sending packets out at the 'same' time. We also see the fact that the source port increments by one on each connect. If these are spoofed from different machines, then there must be some collaboration between the hosts creating this traffic.

```
$ tcpdump -vXr 2002.7.24 host 159.54.34.3

05:28:25.554488 159.54.34.3.2300 > 138.97.18.88.domain: S [bad tcp cksum
4040!] 2111565393:2111565417(24) win 2048 0 [0q] (22) (ttl 242, id 60280, len
64, bad cksum ff0c!)
0x0000   4500 0040 eb78 0000 f206 ff0c 9f36 2203        E..@.x.......6".
0x0010   8a61 1258 08fc 0035 7ddb ee51 0000 0000        .a.X...5}..Q....
0x0020   5002 0800 9439 0000 0000 0000 0000 0000        P....9..........
0x0030   0000 0000 0000 0000 0000 0000 0000 0000        ................

05:28:25.554488 159.54.34.3.2302 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1841412466:1841412490(24) win 2048 0 [0q] (22) (ttl 242, id 57612, len 64, bad
cksum ff78!)
0x0000   4500 0040 e10c 0000 f206 ff78 9f36 2203        E..@.......x.6".
0x0010   8a61 1258 08fe 0035 6dc1 b972 0000 0000        .a.X...5m..r....
0x0020   5002 0800 d930 0000 0000 0000 0000 0000        P....0..........
0x0030   0000 0000 0000 0000 0000 0000 0000 0000        ................

05:28:25.554488 159.54.34.3.2301 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1910536212:1910536236(24) win 2048 0 [0q] (22) (ttl 238, id 18502, len 64, bad
cksum ff3e!)
0x0000   4500 0040 4846 0000 ee06 ff3e 9f36 2203        E..@HF.....>.6".
0x0010   8a61 1258 08fd 0035 71e0 7814 0000 0000        .a.X...5q.x.....
0x0020   5002 0800 1671 0000 0000 0000 0000 0000        P....q..........
0x0030   0000 0000 0000 0000 0000 0000 0000 0000        ................
```

On the other hand, IP ids are not necessarily incremental. This is just usual behavior of most operating systems. IP ids can be set to anything they want by software that deals directly with datalink layer (such as hping or the kernel). I believe this is the case, or there is a 'non-standard' TCP/IP stack on whatever device created these requests.

Description of attack

This doesn't appear to be an attack on the DNS server. There is only three packets sent at the same time, so it does not place an unnecessary load on the server itself.

An important question to ask is, "What does an attacker using this method gain?" How should the TCP/IP stack handle a SYN packet like this? We can use the hping utitlity to craft a packet with data in the SYN packet and see how Linux, for example, would handle it.

```
# hping -H 6 -E filename.txt -d 24 -p 80 -S --fast -c 3
192.168.74.2

HPING 192.168.74.2 (eth0 192.168.74.2): S  set, 40 headers + 24
data bytes
len=46 ip=192.168.74.2  flags=SA DF seq=0 ttl=63 id=11514
win=16080 rtt=6.6 ms
len=46 ip=192.168.74.2  flags=SA DF seq=1 ttl=63 id=11528
win=16080 rtt=1.1 ms
len=46 ip=192.168.74.2  flags=SA DF seq=2 ttl=63 id=11542
win=16080 rtt=1.1 ms
```

The `filename.txt` contains all 0's. The `-d 24` tells hping to take 24 bytes out of `filename.txt` and append it as payload to our SYN packet. Hping reports the response as `flags=SA` which means that the operating system gladly responds to our SYN packet with a SYN/ACK. The DNS server targeted by these strange packets is also probably responding to these packets. The responses just don't match any of the snort rules, so we don't see them in these logs.

To see, let's send this packet to a BIND 9 server. Below is the result of the same hping command sent to TCP 53 on the BIND 9 server.

```
23:00:36.336339 192.168.74.2.53 > 192.168.64.2.1721: S 3804884261:3804884261(0)
ack 340883572 win 64320 <mss 1460> (DF)
0x0000   4500 002c f4e0 4000 3b06 ee64 185d 437f        E..,..@.;..d.]C.
0x0010   c0a8 4002 0035 06b9 e2c9 ed25 1451 7874        ..@..5.....%.Qxt
0x0020   6012 fb40 dcab 0000 0204 05b4 ffff             `..@..........
```

We see that no data was sent back in a SYN/ACK. The DNS server probably discarded the data in our SYN packet as well.

Attack mechanism

The 'attack' inserts 24 bytes of null data into a TCP SYN packet and sends it to port 53 on the DNS server. While this activity it not 'normal' TCP traffic, it does not appear to be hostile in nature.

It is not 'normal' TCP traffic because a connection has not been established between the hosts yet so data should not transferred by either of the hosts. All TCP packets transferred until an ESTABLISHED connection is made is TCP overhead (setting up the connection). However, depending on the operating system, it is unknown if the data in the SYN packet is sent to the user-land application after the data connection is created. In the previous section, we see some ways BIND 9 and Linux would respond to this type of traffic.

Correlations

Several other machines have been logged generating the same type of events on this network.

From 2002.7.24 raw file:

```
10:09:02.614488 209.67.29.9.2200 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
337334976:337335000(24) win 2048 0 [0q] (22) (ttl 241, id 27996, len 64, bad
cksum ff15!)
10:09:02.614488 209.67.29.9.2201 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
217547334:217547358(24) win 2048 0 [0q] (22) (ttl 241, id 41664, len 64, bad
cksum ffb1!)
10:09:02.614488 209.67.29.9.2202 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1102405210:1102405234(24) win 2048 0 [0q] (22) (ttl 241, id 13201, len 64, bad
cksum ffe0!)
```

From 2002.7.25 raw file:

```
08:26:11.804488 209.67.29.9.sieve > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1539331346:1539331370(24) win 2048 0 [0q] (22) (ttl 241, id 15970, len 64, bad
cksum ff0f!)
08:26:11.804488 209.67.29.9.2001 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1552732884:1552732908(24) win 2048 0 [0q] (22) (ttl 241, id 40936, len 64, bad
cksum ff89!)
08:26:11.804488 209.67.29.9.2002 > 138.97.18.88.domain: S [bad tcp cksum 4040!]
1219465587:1219465611(24) win 2048 0 [0q] (22) (ttl 241, id 60589, len 64, bad
cksum ffc4!)
```

From 2002.7.24 raw file:

```
15:44:02.764488 216.33.87.8.2400 > 138.97.18.88.53: S 1668743818:1668743842(24)
win 2048 0 [0q] (22)
15:44:02.764488 216.33.87.8.2401 > 138.97.18.88.53: S 747560726:747560750(24)
win 2048 0 [0q] (22)
15:44:02.764488 216.33.87.8.2402 > 138.97.18.88.53: S 1459643400:1459643424(24)
win 2048 0 [0q] (22)
17:07:30.374488 216.33.87.8.2100 > 138.97.18.88.53: S 220455904:220455928(24)
win 2048 0 [0q] (22)
17:07:30.374488 216.33.87.8.2101 > 138.97.18.88.53: S 1758747021:1758747045(24)
win 2048 0 [0q] (22)
17:07:30.374488 216.33.87.8.2102 > 138.97.18.88.53: S 332598127:332598151(24)
win 2048 0 [0q] (22)
```

I also saw this same traffic from 216.33.87.9 and 216.33.87.10. I see these only happening on 7.24.2002. All the traffic like this seems to have the same non-sequential IP id strangeness. The TTL is always the same for these hosts except for that one strange TTL value from 159.54.34.3 which seems to be anomalous.

A very interesting fact lies in that every IP addresses that has shown this type activity is registered to USAToday. They probably have a broken or misconfigured application spitting out these packets.

There was a thread on snort-users that discussed this particular alert. It seems that load-balancing software called 3-DNS by F5 is believed to be responsible. Matt Kettler posted the following dump to the snort-users mailing list which is generating his alerts:

```
20:30:15.070616 172.20.78.202.3000 > dns-server.53: S
1839760761:1839760825(64) win 2048
```

```
aaaa 0300 0000 0800 4500 0068 7985 0000
f406 9cb9 ac14 4eca c0a8 1004 0bb8 0035
6da8 8579 0000 0000 5002 0800 f842 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

It looks similar except it sends 64 bytes of data in the SYN packet whereas I'm getting 24 bytes of 0's. Perhaps it is a different version of whatever software USAToday has.

Also, Dewey Paciaffi posted to the snort-users mailing list back in Jan/2001 that he also had these alerts that were coming from Microsoft servers. Specifically they were coming from two machines with hostnames of sjwu**3dns1**.windowsupdate.com and dcwu**3dns1**.windowsupdate.com.

This seems like the most reasonable explanation for this type of traffic given the data.

Information on F5 3-DNS system which describes some features 3-DNS is capable of. Technical information on 3-DNS is also available from tech.f5.com though it requires a free login.

Evidence of active targeting

The source IP addresses generating the alerts are actively targeting the DNS server. This information was most likely obtained by querying a DNS root server for the authoritative nameserver of the target's domain.

Severity

Criticality = 5
The targeted machine is a DNS server for this site.

Lethality = 1
This does not appear to be an attack or hostile traffic.

System Countermeasures = 3
This is more than likely a public DNS server. If the box has not been compromised yet, then I can assume they have some level of security patches applied and that system software is kept up-to-date to certain level of degree. I see no traffic in any of the raw logs that indicates this box has been compromised.

Network Countermeasures = 3
I'm assuming this organization has some firewall in place as they have their own DNS server to serve their domain name. Again, I can only assume and this is a pervasive network configuration.

Severity = (5 + 1) - (3 + 3) = 0

This seems a reasonable severity as it is not an attack.

Defensive recommendations

Current defensive recommendations are adequate from what I can tell from the network. Capturing the whole packet transaction would be nice to see exactly how this particular DNS server is responding to these packets (On the application level, since it is responding on the transport layer (TCP)).

Multiple choice test question

Which TCP state is data usually transferred?

A. established
B. close_wait
C. recv_ready
D. fin_wait_1
E. none of the above

Answer is A.

Question & Answer

Below are the questions and answers I received and supplied the intrusions mailing list at incidents@intrusions.org. The posting can be found at the archives.

Question #1 from Andrew R. Jones:

```
"You mention it later, but i will reinforce it here: Not all
operating systems use incremental IP IDs. OpenBSD uses random IP IDs,
for instance. I would try fingerprinting the sending machine if You
can (or justify why this cannot be accurately done, which i believe
to be the case) and seeing if the operating system You come up with
sends incremental IP IDs or whether they are randomized.
```

Response #1 from me:

```
I refrained from actively fingerprinting the hosts because minus
curiosity I really had no reason to do it. From looking at
tech.f5.com it looks like 3dns runs BSDi though I cannot be sure.
Their upgrade packages are all labeled something.BS D_OS.im. I only
have access to BSDi 3.1 machines which according to the date on it
was around in 1998. The whole IP ID randomness issue seems to come on
to the scene (at least on bugtraq) in 1999 which is the oldest I
could find. It's possible BSDi has app lied the same patch OpenBSD
has as far as this randomness goes (since the BSDi 3.1 boxes were
using incremental IP Ids). Also, all these options are configurable,
(TTL, IP ID randomness (with a patch), etc) some way or the other
though the chances are they are the default.
```

Question #2 from Andrew R. Jones:

```
"Do You think there are any more packets besides the SYN/ACK from the
server (which will definitely be sent)?"
```

Response #2 from me:

```
I suppose I hint at this in the above paragraph but don't come right
out and say it. If the data is passed to the stack, then BIND will
respond with some type of failure reply packet. If not, it will
establish the connection and wait for the data. An easier and more
sure way (RFC or not) would be to have the whole conversati on logged.
```

Question #3 from Andrew R. Jones:

```
From me:
“A very interesting fact lies in that every IP addresses that has
shown this type activity is registered to USAToday. They probably
have a broken or misconfigured application spitting out these
packets.”

Andrew:
“You say this here, but later You say that it may be load balancing
software. Pick one or the other.”
```

Response #3 from me:

```
The broken application, I believe, is the load balancing software.
```

Assignment #3: Analyze This!

Executive Summary

During the course of analysis for MY.NET, 842,133 alerts, 4,599,575 scans and 5,990 out of spec packets were considered and correlated. Of these millions of events, the biggest problem I noticed was not necessarily the content or meaning of any one or group of events, but the fact that certain events were generated at all. A properly configured rule set for a stateful firewall would go a long way in intrusion prevention and decrease analysis time for the intrusion detection analysts by reducing noise.

Of the events, the most disturbing involved three internal MY.NET machines that seem to be partaking in much data transfer to France and Belgium over suspicious ports. MY.NET.71.230 is likely infected with Nimda and MY.NET.105.204 seems to be leaking some information to Russia. A great deal of users are involved in file sharing applications and gaming. The top eight 'scanners' can be contributed to this type of traffic.

MY.NET could greatly improve their security posture with some well-needed firewall rule changes (or a firewall altogether). Along with a little bit of tuning, this would also increase their visibility with the SnortIDS infrastructure and increase the analysts ability in finding mailicous traffic.

List of Files

The analysis of the university's network traffic is based on five days worth of log files provided by the university. These files were generated by the Snort IDS system with a fairly standard rulebase. The logs were separated into three categories listed below. For the purpose of easily correlating host traffic, each of the three different categories were concatenated into one corresponding file producing just three files to work with.

Alert files

Filename	*Number of Alerts*	*Alerts - portscans*
alert.021230	61,086	21,278
alert.021231	265,381	100,297
alert.030101	275,490	141,731
alert.030102	106,487	48,459
alert.030103	133,689	53,541
alert-all	**842,133**	**365,306**

Table 3-1: Alert files

The alert files contain events Snort deemed interesting. In order for this to happen, the traffic snort watches must match a particular signature in it's signature database. These signatures are maintained by security enthusiasts and professionals around the world. The alert files also contain 'portscan' events. These events were removed from the alerts files as this type of traffic is contained in the 'scan files' explained below.

Scans files

Filename	Number of Alerts	Unique source IPs
scans.021230	302,043	73
scans.021231	168,3911	225
scans.030101	1,173,862	200
scans.030102	614,464	162
scans.030103	825,295	151
scans-all	**4,599,575**	**596**[3]

Table 3-2: Scans files

The scan files contain alerts of machines which have connected or attempted to connect to several different machines in a specific amount of time. This type of activity is monitored because this often indicates a host scanning for a particular vulnerability or service. It is also indicative of a host that is infected by a virus or compromised.

Out of Spec Files

Filename	Number of Alerts
OOS_Report_2002_12_30_5440	708
OOS_Report_2002_12_31_10852	2,148
OOS_Report_2003_01_01_19650	1,176
OOS_Report_2003_01_02_2030	725
OOS_Report_2003_01_03_17638	1,233
oos-alerts	**5,990**

Table 3-3: Out of Spec files

The Out of Spec(OOS) files contain traffic detected by Snort that is not normal in standard network communications. This type of traffic is monitored for just that reason: it's not normal. It can often indicate communication of trojans or denial of service(DOS) attacks.

Alerts Reported More than 2,000 times

Event Summary	Occ.	Severity
Russia Dynamo - SANS Flash 28-jul-00	156,445	Medium
spp http decode: IIS Unicode attack detected	52,014	Noise
SMB Name Wildcard	49,289	High
High port 65535 tcp - possible Red Worm - traffic	48,866	High
TFTP-External UDP connection to internal tftp server	27,022	Noise
NIMDA-Attempt to execute cmd from campus host	12,461	Medium
Watchlist 000220 IL-ISDNNET-990517	10,531	Low
High port 65535 udp - possible Red Worm - traffic	2,647	High
spp_http_decode: CGI Null Byte attack detected	2,034	Noise

Table 3-4: Alerts synopsis

Russia Dynamo - SANS Flash
Severity: Med. Occurrences: 156,445 Internal Machines Involved: 1
Snort Signature ID: not available

3 This column does not add up because the same source address appears in multiple or all of the separate scans files. The scans-all represents the real number of unique addresses.

Although this event is the most generated alert at the university, only one internal machine is generating the traffic.

```
12/31-20:57:47.063180  [**] Russia Dynamo - SANS Flash 28-jul-00 [**] MY.NET.105
.204:3514 -> 194.87.6.75:1037
12/31-20:57:47.063432  [**] Russia Dynamo - SANS Flash 28-jul-00 [**] MY.NET.105
.204:3514 -> 194.87.6.75:1037
12/31-20:57:47.568144  [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 194.87.6.7
5:1037 -> MY.NET.105.204:3514
```

This host more than likely has a trojan that is leaking information to the Russian address. We see the 'initial' connection was requested by the universities' host. This indicates an infection through some means snort is not aware of. This could have been through an email attachment or directly connecting to the internal machine. The attacker would more than likely do a scan sweep for the latter method in order to locate its victim and thus ending up in a scans file though it does not appear in mine. I presume this host was compromised before I obtained the log files and there is indeed evidence of scanning for vulnerabilities.

Although this is a very high concern, it does not appear the host has been compromised fully. There is no evidence of malicious traffic from MY.NET.105.204 in the logs to indicate it is attacking or scanning any other computers.

Correlations: SANS had some further information on these Russian hosts. The release date of this information (7/29/00) leads me to further believe the host was compromised some time ago. Miika Turkia also analyzed this type of traffic in Jan/2001 although he does not come to any hard conclusions on such limited information.

Recommendation: As suggested by SANS, it would be wise to block incoming and outgoing traffic to the Russian class C address space. Also, this box should be taken offline, scanned with anti-virus software, and brought up-to-date with patches.

spp http decode: IIS Unicode attack detected
Severity: Noise   Occurrences: 52,014   Internal Machines Involved: 130
Snort Signature ID: spp_http_decode

This indicates that that unicode characters were detected being transferred over a well-known http port. Many worms such as sadmind, Code Red, Code Red II, and Nimda are dependent on a webserver mishandling unicode encoded URLs in order to trick the webserver to give it access to a directory (such as C:\windows\system32).

At first glance it would appear that many internal hosts have been infected with some of the worms mentioned in the previous paragraph. However, when the source addresses that generated these alerts were cross-referenced with all the other alerts for other signatures that would indicate infection, only MY.NET.71.230 stood out as being infected. It shows up as attempting to execute cmd.exe and root.exe to thousands of sequential hosts which is an indication of infection. Minus the one infected host, It would seem these are false positives.

Correlations: Bradley Urwiller has also analyzed this type of traffic in his GCIA practical and come to the same conclusion that these are noise. Joe Stewart has a post on the snort-users mailing list regarding these alerts. He determined that many of these false positives are URL-encoded binary data put into a websites cookie which loaded everytime the page is visited.

Recommendation: From the Snort FAQ:

> Q: I am getting too many "IIS Unicode attack detected" and/or "CGI Null Byte attack detected" false positives. How can I turn this detection off?
>
> A: These messages are produced by the http_decode preprocessor. If you wish to turn these checks off, add -unicode or -cginull to your http_decode preprocessor line respectively.
>
> preprocessor http_decode: 80 8080 -unicode -cginull
>
> Your own internal users normal surfing can trigger these alerts in the preprocessor. Netscape in particular has been known to trigger them. Instead of disabling them, try a BPF filter to ignore your outbound http traffic such as: snort -d -A fast -c snort.conf not (src net xxx.xxx and dst port 80) This has worked very well for us over a period of 5-6 months and Snort is still very able to decode actual and dangerous cgi null and unicode attacks on our public web servers.

We see that the one machine infected with Nimda could have easily been seen using the Nimda signatures. This machine should be taken offline, cleaned, patched, and put back into service.

SMB Name Wildcard
Severity: High Occurrences: 48,866 Internal Machines Involved: 912
Snort Signature ID: note available

This event indicates normal NETBIOS query traffic. All but five of the 48,866 alerts of this signature were inbound from the Internet. The five alerts that were not within the MY.NET range were internal addresses from RFC 1918. These probably indicate misconfigured clients and not hostile in nature, though there is a possibility of spoofing. This doesn't seem necessary however since it seems port 137 is wide-open to the Internet which is why this is classified as 'high' severity. Seeing these events internally is usually normal activity; seeing them from the Internet suggests a needed firewall change.

Correlations: The events being alerted only when the source address is not the MY.NET seems to suggest the snort machine was configured to only generate this alert on inbound traffic only as Max Vision suggests on the snort-users mailing list on January 17th 2000. The 912 internal machines are more than likely communicating outbound to the hosts that initiated the connection.

To stress the severity of protecting access to this port, incidents.org is reporting this port (137) as the most attacked port on the Internet (as of Jan 6th 2003).

Mike Bell mentions this event in Jan/2001 as a way to enumerate target hosts by getting all NETBIOS names they are aware of.

The Snort FAQ also addresses this issue suggesting to only generate this alert when it's not on the local network. It also directs us to IDS177 of WhiteHats network security which says this is a way to "extract useful information such as workstation name, domain, and users who are currently logged in."

Recommendations: NETBIOS traffic should not be allowed to pass through the university's firewalls or outside screening routers. Blocking this type of traffic gives the university's students and faculty added protection from many types of automated worms/viruses that attack these services.

High port 65535 tcp or udp - possible Red Worm - traffic[4]
Severity: High Occurrences: 51,513 Internal Machines Involved: 37
Snort Signature ID: not available

These alerts indicate that traffic was detected on port 65535. This is the highest port available for UDP and TCP transfers. The Red Worm (better known as AdoreWorm) uses port 65535 as a backdoor entry. While port 65535 is used as a valid source port, the majority of the 'destination' ports are not well-known server ports. Many of these alerts are from university computers connecting to port 65535 of a remote host. These could indicate compromised hosts and deserve further analysis.

The Adoreworm (or RedWorm) scans the Internet looking for hosts vulnerable to one of the four vulnerabilities it tries to exploit. The services it searches for are LPRng, rpc-statd, wu-ftpd and BIND though no scanning of these four services was detected over the five day monitoring period.

Correlations: SANS, and F-Secure both have an analysis of the worm along with how to detect and remove the worm. Dartmouth's ITSU has also released a tool called Adorefind which will detect the presence of the worm on Linux machines.

Michael Reiter analyzed the Adoreworm and describes Linux kernel module trojans for his GCIH in Feb/2001.

Recommendation: Block inbound traffic trying to *establish* connections on port 65535. Also see the 'Backdoor/Trojan activity' section for further analysis and recommendations.

4 This section covers both TCP and UDP versions of this alert.

TFTP-External UDP connection to internal tftp server
Severity: Noise Occurrences: 27,022 Internal Machines Involved: 7
Snort Signature ID: not available

The alert is generated anytime a machine not defined in Snort's HOME_NET variable communicates to an internal machine on port 69.

Of 27,022 alerts, all but one is from 192.168.0.253 which is a RFC 1918 address used for internal purposes. It's possible this is some kind of router or similar device that uses TFTP to backup or update it's configuration files regularly. It appears to be transferring, or retrieving data from MY.NET.111.219, MY.NET.111.230-232, and MY.NET.111.235 in an automated fashion every ten minutes.

The other connection is inbound from a Yahoo Broadcast address.

```
01/03-14:34:37.894242  [**] TFTP - External UDP connection to internal tftp
server [**] 63.250.205.26:6971 -> MY.NET.177.61:69

$ whois -h whois.arin.net 63.250.205.26

OrgName:    Yahoo! Broadcast Services, Inc.
OrgID:      YAHO

NetRange:   63.250.192.0 - 63.250.223.255
CIDR:       63.250.192.0/19
NetName:    NETBLK2-YAHOOBS
NetHandle:  NET-63-250-192-0-1
Parent:     NET-63-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.YAHOO.COM
NameServer: NS2.YAHOO.COM
NameServer: NS3.YAHOO.COM
NameServer: NS4.YAHOO.COM
NameServer: NS5.YAHOO.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-11-24
Updated:    2002-03-27
```

While I believe this is not hostile and somehow related to MY.NET.177.61 attempting to use Yahoo Broadcast, I cannot say for sure due to lack of data. I can say that this type of traffic should not be allowed inbound unless explicitly allowed on the firewall.

Correlations: Cisco has published a support page that explains how to enable TFTP for for use by cisco products (routers, firewalls etc). They suggest entering the following line in /etc/inetd.conf:

```
tftp  dgram  udp  wait  root  /usr/sbin/tftpd tftpd  -d /tftpboot
```

Notice the use of the UDP protocol for this in the third column. While RFC for TFTP (RFC 1350) builds TFTP upon UDP, they encourage the TFTP protocol to be implemented on top of other transport protocols such as TCP.

Recommendations: Allow TFTP through the firewall only for authorized hosts.

All other hosts should be denied this service. If 192.168.0.253 is authorized for this type of activity, add the address to the HOME_NET variable to bring the noise level down.

NIMDA-Attempt to execute cmd from campus host
Severity: Med.   Occurrences: 12,461   Internal Machines Involved: 3
Snort Signature ID: 1002 (with direction reversed)

Will it ever die! A classic Nimda infection. As shown, there is a total of only three internal machines that generated these events. Of the three, one machine (MY.NET.71.230) is responsible for 12,457 occurrences of these outbound connections.

I do not believe the other two machines are infected with Nimda. There is no mass scanning for HTTP servers from these two machines which is indicative of Nimda.

Correlations: CERT has an advisory about the Nimda worm. SANS, F-Secure, Symantec, McAfee, and countless other security professionals have detailed information on this worm along with removal instructions. This particular worm as been analyzed by countless people.

Recommendations: Take the infected machine offline. Scan for Nimda and remove if necessary. Apply valid patches. Allow incoming HTTP traffic to only authorized web servers where knowledgeable personnel can keep them up-to-date.

Watchlist 000220 IL-ISDNNET-990517
Severity: Low   Occurrences: 10,531   Internal Machines Involved: 34
Snort Signature ID: not available

'Watchlist' signatures alert when a host in a certain IP range connects to the university's network. This particular watchlist signature seems to alert on the 212.179.64.0/18 net which is assigned to ISDN Net Ltd of Israel.

- 5,426 of these events can be almost positively contributed to file-sharing applications such as KaZaA, Morpheus, and Gnutella. An additional 2,264 of these can be attributed to these same programs with a fair degree of certainty.
- 2,774 of the events can be contributed to web traffic.
- 8 can be attributed to SMTP connections.

Correlations: The remaining 356 events are using unknown port combinations. None of the ports however, are listed as trojan on the neohapsis port list and can probably be attributed to file-sharing as well.

Recommendation: If the university is interested in such activity, snort signatures could be added to alert on the use of file-sharing applications. If not, attempting to block access for an increase in bandwidth could be implemented.

spp_http_decode: CGI Null Byte attack detected
Severity: Noise   Occurrences: 10,531   Internal Machines Involved: 34
Snort Signature ID: http_decode

This particular event indicates that a %00 was detected in the content of a packet. This alert was designed to catch attackers trying to exploit the fact that Perl does not recognize %00(NULL) has a delimiter. This has security implications when Perl must pass data to the underlying operating system or any other C program which does look at null has a delimiter.

Correlations: This alert is known to have a very high amount of false positives since it often alerts on URL-encoded cookies in URLs and SSL traffic as described by Joe Stewart on the snort-users mailing list. Also there is not real way to tell if these are real attacks without have the packet dump as these are content-based alerts.

This type of attack was described by Rain Forest Puppy in Phrack #55.

Recommendations: See recommendations for 'spp http decode: IIS Unicode attack detected'

Scans Reported More than 10,000 times

Pos.	*IP Address*	*Scan Occurrence*	*Synopsis*
1	MY.NET.83.146	1,306,279	Hog
2	MY.NET.70.176	746,709	Hog
3	MY.NET.84.244	418,341	Hog
4	MY.NET.168.175	292,461	Hog
5	MY.NET.150.213	289,102	Hog
6	MY.NET.91.252	243,962	Hog
7	MY.NET.87.50	217,967	Hog
8	MY.NET.82.2	191,847	Hog
9	MY.NET.132.20	188,005	Hog
10	MY.NET.70.207	134,694	Hog
11	MY.NET.190.90	85,623	Infected with worm
12	80.14.115.177	79,593	Information Gathering
13	MY.NET.84.193	71,261	Hog
14	80.200.151.134	24,861	Information Gathering
15	81.50.52.235	24,558	Information Gathering
16	150.187.177.12	21,323	Information Gathering
17	MY.NET.150.220	11,781	Hog
18	MY.NET.88.225	11,743	Hog
19	202.181.214.4	11,517	Information Gathering
20	MY.NET.88.69	11,102	Hog
21	194.248.237.10	10,707	Information Gathering

Table 3-5: Scans synopsis

MY.NET.83.146(1st) - The Alpha Hog
Severity: Hog Occurrences: 1,306,279
Snort Signature ID: spp_portscan

Nothing to see here, except your bandwidth drop. 75% of the 1.3 million alerts were generated from or to port 6257. The neohapsis port list has only WinMX registered to this port. According to their website, WinMX "... allows you to connect, download, and share files with MILLIONS of other users through the decentralized WinMX Peer Network" which seems consistent with the amount of events we get for this port. I did notice outbound connections to 65535 but alas, it was only an outbound WinMX transfer, which seems to be this persons preferred method of bandwidth hogging. They will use KaZaA now and then though which seems to be the majority of the rest of the traffic.

One plus here is it looks like WinMX is dependent upon the 6257 and 6699 port unlike KaZaA which will even use port 80 for inbound connections now.

```
Dec 30 20:50:22 MY.NET.83.146:6257 -> 68.6.238.203:6257 UDP
Dec 30 20:50:22 MY.NET.83.146:6257 -> 12.249.4.162:6257 UDP
Dec 30 20:50:24 MY.NET.83.146:6257 -> 80.117.211.80:6257 UDP
Dec 30 20:50:24 MY.NET.83.146:6257 -> 80.193.36.144:6257 UDP
Dec 30 20:50:24 MY.NET.83.146:6257 -> 80.13.145.140:6257 UDP

Jan  2 22:52:40 MY.NET.70.176:1268 -> 24.83.255.172:6699 SYN ******S*
Jan  3 05:31:43 MY.NET.70.176:1270 -> 80.129.145.201:6699 SYN ******S*
```

The user is using an older version of KaZaA as well, which used port 1214 for many of its connections, so it would not be to hard to block that either until they downloaded the newest version.

Peer-to-Peer(p2p) network clients not only eat up lots of available bandwidth that could be used for legitimate purposes, but could also very easily introduce viruses into the campus networks.

Correlations: MY.NET.84.176(2nd) only makes it second on this list because of WinMX as well as it shows up almost exclusively in its scan logs.

Paul Farley also talks about the dangers of P2P networks clients in his GCIA practical in May/2002.

Recommendations: If the university does not allow this type of network activity, it is probable that WinMX could be blocked. The machines in question should also be investigated and a policy should be created/enforced that such activity is not allowed.

MY.NET.82.2(8th) - Mr. Gamer
Severity: Hog Occurrences: 191,847
Snort Signature ID: spp_portscan

Second eater of bandwidth on our list of scanners next to the file-sharing group is Mr. Gamer. This person has a "The Lost Battalion" game server setup on port 12203 and port 12300. Port 12203 was responsible for 83,765 of the events while port 12300 takes the rest. The Lost Battalion website has a help page that describes a router/firewall setup in order to allow the game to function properly.

MOH:AA Firewall and Router Setup:
Open UDP ingoing and outgoing ports (these are for the Multiplayer Demo and Full version):
12201, 12202, 12203, 12210, 12300.

```
Dec 30 23:40:55 MY.NET.82.2:12203 -> 4.33.6.75:64489 UDP
Dec 30 23:40:55 MY.NET.82.2:12203 -> 24.102.206.102:24641 UDP
Dec 30 23:40:54 MY.NET.82.2:12300 -> 24.127.105.12:1723 UDP
Dec 30 23:40:54 MY.NET.82.2:12300 -> 66.58.176.54:1530 UDP
Dec 30 23:40:55 MY.NET.82.2:12300 -> 65.40.74.157:21000 UDP
Dec 30 23:40:55 MY.NET.82.2:12300 -> 211.193.91.252:1045 UDP
Dec 30 23:40:57 MY.NET.82.2:12203 -> 24.93.244.163:50311 UDP
```

Notice the mentioning of opening the ports our MY.NET address uses exclusively.

Correlations: MY.NET.82.207(10th) exhibits exact same traffic pattern which indicates it is also a "Lost Battalion" game server.

Recommendations: If this type of network activity is not allowed by the university, the machine’s owner should be informed and the server should be taken down.

MY.NET.84.244(3rd) - KaZaA
Severity: Hog   Occurrences: 418,341
Snort Signature ID: spp_portscan

This user enjoys banking at Chevy Chase while downloading and distributing much data through the KaZaA file-sharing system. As mentioned earlier, it's very hard to block the use of KaZaA or even tell if something is using it without looking at the packet data. Shown below is internal MY.NET host connecting to another KaZaA user for file-sharing.

```
Jan  3 21:13:10 MY.NET.84.244:3355 -> 24.44.196.183:1797 SYN ******S*
Jan  3 21:36:01 MY.NET.84.244:3708 -> 24.44.196.183:1797 SYN ******S*
Jan  3 21:36:02 MY.NET.84.244:3708 -> 24.44.196.183:1797 SYN ******S*
Jan  3 22:44:42 MY.NET.84.244:2320 -> 24.44.196.183:1797 UDP
```

The TCP connections of KaZaA never seem to use the same port. The 2320 UDP packet appears to be the beginning of the file transfer as this is the port that shows up most in the log files (416,762 times). Most of the rest are port 80 or the not-so-obvious KaZaA connections. One way to tell if this is KaZaA or not is to try to connect on the same port our internal user is using. With a little luck, the KaZaA user will still be connected.

```
$ telnet 24.44.196.183 1797
Trying 24.44.196.183...
Connected to 24.44.196.183.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 404 Not Found
X-Kazaa-Username: DMAG
X-Kazaa-Network: KaZaA
X-Kazaa-IP: 24.44.196.183:1797
```

```
Connection closed by foreign host.
```

This is a fairly good way to determine if KaZaA is being used when strange ports seem to appear. The only problem is, as time passes, the more likely that user will go offline.

Correlations: Other KaZaA hogs on this list: MY.NET.91.252(6th), MY.NET.132.20(9th), MY.NET.84.193(13th), MY.NET.150.220(17th), MY.NET.88.69(20th).

Recommendations: See recommendation for the 'Alpha Hog' above.

MY.NET.168.175(4th) - I should have taken Korean
Severity: Hog   Occurrences: 292,461
Snort Signature ID: spp_portscan

The events from this host are mostly to a site called http://www.songn.com/. From what I gather, this is like a Korean amazon.com of sorts. By clicking randomly around on links I found you can play clips of musical artists, buy DVDs, electrical equipment and a slew of other things (at least it seemed from the pictures). 114,142 of the events were to port 22321 UDP while 177,833 where to port 7674 UDP. The connection types are identical with the remote side have the same port number as the local side. It seems these two transfers happened concurrently with many hosts.

```
Dec 31 18:57:56 MY.NET.168.175:7674 -> 211.225.92.38:7674 UDP
Dec 31 18:57:56 MY.NET.168.175:7674 -> 61.77.17.203:7674 UDP
Dec 31 18:57:56 MY.NET.168.175:7674 -> 211.59.45.198:7674 UDP
Dec 31 18:57:56 MY.NET.168.175:7674 -> 211.231.67.60:7674 UDP
Dec 31 19:01:59 MY.NET.168.175:22321 -> 61.84.229.106:22321 UDP
Dec 31 19:01:59 MY.NET.168.175:22321 -> 211.109.177.62:22321 UDP
Dec 31 19:01:59 MY.NET.168.175:22321 -> 211.210.224.98:22321 UDP
Dec 31 19:01:59 MY.NET.168.175:22321 -> 203.232.89.36:22321 UDP
Dec 31 19:02:00 MY.NET.168.175:22321 -> 61.77.223.243:22321 UDP
```

The hosts are generally all located in Europe or the Asia-Pacific.

It seems www.song.com uses the Linux Virtual Server (LVS) for a music media portal according to the LVS deployment page.

> " [songn.com is a] Music portal Site with soribada(famous mp3 p2p in Korea)."

The type of traffic by MY.NET.168.175 does indeed seem to resemble a file-sharing program which is more than likely soribada.

Correlations: MY.NET.88.225(18th) has visited the same website (www.songn.com) and exhibited the exact same traffic pattern.

Recommendations: If this type of activity is not allowed by the university, it seems it might be possible to block this application by dropping ports 7674 and 22321. Soribada is unknown to me (and I can't read Korean) so it might take corrective measures and pick different ports like KaZaA.

MY.NET.87.50 - Half-Life boot_camp
Severity: Hog   Occurrences: 217,967
Snort Signature ID: spp_portscan

Of the 217,967 alerts from this host, only 2 were not from source port 888 or source port 999. Data like this usually means that 888 and 999 are not source ports, but destination ports from the other hosts and MY.NET.87.50 is a server of some sort. Examining the external host ports we see 135,611 are from port 27005 which is the default client port for Half-Life, a popular first-person shooter.

Correlations: Michael McDonnell analyzed this same exact traffic fro the same IP address Nov/2001 in his GCIA and determined it was a Half-Life server as well.

Recommendations: If this type of activity is not allowed by the university then this machine's administrator should be informed and the server should be taken down.

MY.NET.190.90 - Opaserv again
Severity: Infected w/ worm   Occurrences: 85,623
Snort Signature ID: spp_portscan

Looks like this host is either infected with the Opaserv worm, or this is a malicious user scanning for open shares. My guess is the host is infected with Opaserv.

```
Dec 31 11:33:19 MY.NET.190.90:1031 -> 61.35.180.230:137 UDP
Dec 31 11:33:19 MY.NET.190.90:1031 -> 61.35.180.231:137 UDP
Dec 31 11:33:19 MY.NET.190.90:1031 -> 61.35.180.233:137 UDP
Dec 31 11:33:19 MY.NET.190.90:1031 -> 61.35.180.234:137 UDP
Dec 31 11:33:20 MY.NET.190.90:1031 -> 61.35.180.235:137 UDP
Dec 31 11:33:20 MY.NET.190.90:3727 -> 61.35.180.235:139 SYN ******S*
Dec 31 11:33:20 MY.NET.190.90:1031 -> 61.35.180.236:137 UDP
Dec 31 11:33:20 MY.NET.190.90:3728 -> 61.35.180.236:139 SYN ******S*
Dec 31 11:33:20 MY.NET.190.90:1031 -> 61.35.180.237:137 UDP
Dec 31 11:33:20 MY.NET.190.90:1031 -> 61.35.180.238:137 UDP
Dec 31 11:33:20 MY.NET.190.90:1031 -> 61.35.180.239:137 UDP
```

In Detect #1, we do not see the 'scanning' of the hosts. All we see is the share access attempts from the worm. We do see the TCP source port increment by one on each successive connection though. The worm in this case is using UDP to request shares on the victim machine. If the worm gets a reply, it does a TCP connection to the same host in an attempt at infection. These attempts are displayed as the TCP port 139 connections in the logs.

If this were scanning, it probably would not be interleaved like this. We would see a block of addresses be scanned by our attacker, and then after, connection attempts. This is in the general sense, scripting such an attack as above would not be extremely difficult.

Correlations: This machine appears in dshield.org 44 times. Though dshield does not specify the ports scanned for some reason (database problem?), I'm going to go out on a limb here and say it was port 137.

Recommendations: See Detect #2, 'Defensive Recommendations' section.

80.14.115.177, 80.200.151.134, 81.50.52.235 - Typical Scanning
Severity: Information Gathering   Occurrences: 79,593 - 24,861 - 24,558
Snort Signature ID: spp_portscan

Attacker	***Port***	***Internal Hosts***
80.14.115.177	80	195
	135	1,306
	445	195
80.200.151.134	21	10
	80	33
	135	2,914
	137	6
	139	7
	445	190
81.50.52.235	80	2,119
	137	859
	139	882
	445	2,223

Table 3-6: Summary of scanners

Nothing terribly interesting about the scan done by 80.14.115.177. Looks like a scan for port 135 across multiple subnets with a scan of port 80 and 445 across the same hosts.

```
Jan  1 12:16:46 80.14.115.177:4314 -> MY.NET.163.131:135 SYN ******S*
Jan  1 12:16:46 80.14.115.177:4315 -> MY.NET.163.132:135 SYN ******S*
Jan  1 12:16:45 80.14.115.177:4316 -> MY.NET.163.133:135 SYN ******S*
Jan  1 12:16:46 80.14.115.177:4317 -> MY.NET.163.134:135 SYN ******S*
Jan  1 12:16:47 80.14.115.177:4315 -> MY.NET.163.132:135 SYN ******S*
Jan  1 12:16:50 80.14.115.177:4316 -> MY.NET.163.133:135 SYN ******S*
Jan  1 12:16:55 80.14.115.177:4494 -> MY.NET.163.141:135 SYN ******S*
```

This attacker also found a couple of FTP servers while scanning the university's network. From the alerts file, we see:

```
FTP_passwd_attempt:12/31-08:42:12.992026  [**] FTP passwd attempt [**]
81.50.52.235:4621 -> MY.NET.111.21:21
FTP_passwd_attempt:12/31-08:54:32.440413  [**] FTP passwd attempt [**]
81.50.52.235:4381 -> MY.NET.113.208:21
FTP_passwd_attempt:12/31-10:33:17.664342  [**] FTP passwd attempt [**]
81.50.52.235:4403 -> MY.NET.130.27:21
FTP_passwd_attempt:12/31-10:37:29.528948  [**] FTP passwd attempt [**]
81.50.52.235:3518 -> MY.NET.130.123:21
FTP_passwd_attempt:12/31-10:37:43.798113  [**] FTP passwd attempt [**]
81.50.52.235:3598 -> MY.NET.130.40:21
FTP_passwd_attempt:12/31-10:38:24.880950  [**] FTP passwd attempt [**]
81.50.52.235:3805 -> MY.NET.130.14:21
FTP_passwd_attempt:12/31-10:41:45.256369  [**] FTP passwd attempt [**]
81.50.52.235:4633 -> MY.NET.130.187:21
FTP_passwd_attempt:12/31-10:45:20.352710  [**] FTP passwd attempt [**]
81.50.52.235:3573 -> MY.NET.130.86:21
```

Sometimes an administrator of an FTP server will not lock the FTP permissions down and leave the /etc/passwd readable to anonymous users. When an attacker has this file, he is halfway to having a username/password couple in order to log on as the passwd file contains valid users on the machine.

This snort signature detects when a user attempts do download the /etc/passwd file over FTP.

Correlations: This attacker appears in dshield.org two times.

Recommendations: Update dshield.org database since it does not accurately reflect the number of scanned hosts by the attacker. Block this IP at outside screening routers or perimeter firewalls. Allow access to scanned ports (or any port for that matter) to authorized nets or servers. Send an abuse letter to network administrator of that particular netblock. The whois database gives a very good start in to tracking down the contact to send this information to.

```
$ whois -h whois.ripe.net 80.14.115.177
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.14.115.0 - 80.14.115.255
netname:      IP2000-ADSL-BAS
descr:        BSLIL208 Lille Bloc1
country:      FR
admin-c:      WITR1-RIPE
tech-c:       WITR1-RIPE
status:       ASSIGNED PA
remarks:      for hacking, spamming or security problems send
mail to
remarks:      postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:      for ANY problem send mail to
gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20020227
changed:      gestionip.ft@francetelecom.com 20020709
source:       RIPE

route:        80.14.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      -------------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to      abuse@francetelecom.net
remarks:      -------------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-by:       FT-BRX
changed:      karim@rain.fr 20011221
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
```

```
e-mail:       postmaster@wanadoo.fr
admin-c:      FTI-RIPE
tech-c:       TEFS1-RIPE
nic-hdl:      WITR1-RIPE
notify:       gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010 504
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.com 20011204
source:       RIPE
```

WOW, it's our lucky day. This nice ISP has included its contact information for abuse in the whois registry. As we will find out later, it is not always this easy. Send an abuse letter to abuse@wanadoo.fr with details of the scanning and hope they speak English.

We see similar scanning from 80.200.151.134. A quick search on dshield.org shows nothing on this IP address. Well, that will have to change. This could be an indication that this is a directed scan at the university.

Correlations: Google does reveal one interesting piece of information about this IP address in some IRC logs.

```
[19:20] fr0lux (bsd@226.153-136-217.adsl.skynet.be) joined #eci.
[19:21] <fr0lux> jeanseb: c'est toi qui me disait No comment , newbie power !
???
[19:21] <jeanseb> vi pkoi ?
[19:22] <fr0lux> jeanseb: tu va m'aider alors.. je l'ai remis cette debian , et
j'ai configuré X tout va à part la souris qui déconne au click elle click tout
le temps
[19:22] <fr0lux> elle a la clickophobie
[19:22] <fr0lux> :)
[19:22] <jeanseb> ben t'a pas configurer le bon protocol
[19:23] <fr0lux> si PS/2
[19:23] <jeanseb> kel version de X ?
[19:23] <fr0lux> et c'est une PS/2
[19:23] <fr0lux> Xfree 4.0 je pense
[19:23] <fr0lux> arf
[19:23] <fr0lux> faut que je vérifie
[19:23] <fr0lux> c'est celui avec debian woody
[19:23] <jeanseb> dpkg -l x* \ grep ^i | grep 86
[19:23] <jeanseb> dpkg -l x* | grep ^i | grep 86
[19:23] <fr0lux> et ca me fait quoi ?
[19:24] <fr0lux> dpkg = ?
[19:24] <fr0lux> grep = ?
[19:24] <jeanseb> tape moi ca ds un shell
[19:24] <fr0lux> ben oui
[19:24] <fr0lux> faut que j'y aille
[19:24] fr0lux (bsd@226.153-136-217.adsl.skynet.be) left irc: Client Quit
[19:25] fr0lux (bsd@80.200.151.134) joined #eci.
[19:25] <fr0lux> quel bête !
[19:25] <fr0lux> j'ai pas noté la commande :/ (no comment)
[19:26] <jeanseb> dpkg -l x* | grep ^i | grep 86
[19:26] <fr0lux> merci
[19:26] <fr0lux> je note lol
[19:27] fr0lux (bsd@80.200.151.134) left irc: Client Quit
[19:27] webdaemon (~webdaemon@Mix-Marseille-107-4-158.abo.wanadoo.fr) joined
#eci.
[19:27] #eci: mode change '+o webdaemon' by ChanServ!ChanServ@services.
[19:36] |DaRk| (~funny33@80.14.77.45) joined #eci.
[19:36] fr0lux (bsd@80.200.159.209) joined #eci.
```

It seems 80.200.151.134 could be fr0lux's shell account, though my French is not so good. We see fr0lux using several IP addresses: 80.200.151.134, 226.153-136-217.adsl.skynet.be(217.136.153.226), 80.200.159.209, and a little later, 217.136.155.205.

Of the three addresses, 80.200.159.209 appears in dshield.org 1 time for scanning port 137 though it is possible that this ISP does not normally give out the same IP to their clients.

Recommendations: See recommendations for 80.14.115.177.

```
$ whois -h whois.ripe.net 80.200.151.134
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      80.200.0.0 - 80.200.255.255
netname:      BE-SKYNET-20011108
descr:        ADSL Customers
descr:        Skynet Belgium
country:      BE
admin-c:      JFS1-RIPE
tech-c:       PDH16-RIPE
status:       ASSIGNED PA
mnt-by:       SKYNETBE-MNT
changed:      ripe@skynet.be 20011212
source:       RIPE

route:        80.200.0.0/15
descr:        SKYNETBE-CUSTOMERS
origin:       AS5432
notify:       noc@skynet.be
mnt-by:       SKYNETBE-MNT
changed:      noc@skynet.be 20011116
source:       RIPE

person:       Jean-Francois Stenuit
address:      Belgacom Skynet NV/SA
address:      Rue Carli 2
address:      B-1140 Bruxelles
address:      Belgium
phone:        +32 2 706-1311
fax-no:       +32 2 706-1150
e-mail:       jfs@skynet.be
nic-hdl:      JFS1-RIPE
remarks:      ----------------------------------------
remarks:      Network problems to: noc@skynet.be
remarks:      Peering requests to: peering@skynet.be
remarks:      Abuse notifications to: abuse@skynet.be
remarks:      ----------------------------------------
mnt-by:       SKYNETBE-MNT
changed:      jfs@skynet.be 19970707
changed:      ripe@skynet.be 20021125
source:       RIPE

person:       Pieterjan d'Hertog
address:      Belgacom Skynet sa/nv
address:      2 Rue Carli
address:      B-1140 Brussels
address:      Belgium
```

```
phone:          +32 2 706 13 11
fax-no:         +32 2 706 13 12
e-mail:         piet@skynet.be
nic-hdl:        PDH16-RIPE
remarks:        ----------------------------------------
remarks:        Network problems to: noc@skynet.be
remarks:        Peering requests to: peering@ skynet.be
remarks:        Abuse notifications to: abuse@skynet.be
remarks:        ----------------------------------------
mnt-by:         SKYNETBE-MNT
changed:        jfs@skynet.be 19990415
changed:        piet@skynet.be 19991210
changed:        piet@skynet.be 20000302
changed:        piet@skynet.be 20020329
source:         RIPE
```

What a beautiful sight. Another abuse address listed in the registry information. I think this should be a requirement! Abuse information should be sent to abuse@skynet.be.

Microsoft Windows services comes under attack once again as 81.50.52.235 looks for something to break.

```
Dec 31 13:03:51 81.50.52.235:3874 -> MY.NET.135.24:139 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3875 -> MY.NET.134.201:80 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3876 -> MY.NET.134.200:80 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3877 -> MY.NET.135.25:445 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3879 -> MY.NET.135.25:139 SYN ******S*
Dec 31 13:03:52 81.50.52.235:137 -> MY.NET.135.25:137 UDP
Dec 31 13:03:52 81.50.52.235:137 -> MY.NET.135.27:137 UDP
Dec 31 13:03:52 81.50.52.235:137 -> MY.NET.135.28:137 UDP
Dec 31 13:03:52 81.50.52.235:3880 -> MY.NET.134.202:80 SYN ******S*
Dec 31 13:03:52 81.50.52.235:2765 -> MY.NET.135.21:139 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3859 -> MY.NET.135.21:445 SYN ******S*
Dec 31 13:03:52 81.50.52.235:3881 -> MY.NET.135.26:445 SYN ******S*
```

Correlations: This host appears in dshield.org 40 times for port 137 scanning.

Recommendations: Submit this host to dshield.org to update the database. See recommendation section for previous two attackers.

```
$ whois -h whois.ripe.net 81.50.52.235
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:        81.50.52.0 - 81.50.52.255
netname:        IP2000-ADSL-BAS
descr:          BSLIL208 Lille Bloc1
country:        FR
admin-c:        WITR1-RIPE
tech-c:         WITR1-RIPE
status:         ASSIGNED PA
remarks:        for hacking, spamming or security problems send
mail to
remarks:        postmaster@wanadoo.fr AND abuse@wanadoo.fr
remarks:        for ANY problem send mail to
gestionip.ft@francetelecom.com
mnt-by:         FT-BRX
```

```
changed:      gestionip.ft@francetelecom.com 20021120
source:       RIPE

route:        81.50.0.0/16
descr:        France Telecom
descr:        Wanadoo Interactive
remarks:      -------------------------------------------
remarks:      For Hacking, Spamming or Security problems
remarks:      send mail to   abuse@wanadoo.f r
remarks:      -------------------------------------------
origin:       AS3215
mnt-by:       RAIN-TRANSPAC
mnt-routes:   RAIN-TRANSPAC
changed:      tom@rain.fr 20021030
source:       RIPE

role:         Wanadoo Interactive Technical Role
address:      WANADOO INTERACTIVE
address:      48 rue Camille Desmoulins
address:      92791 ISSY LES MOULINEAUX CEDEX 9
address:      FR
phone:        +33 1 58 88 50 00
e-mail:       abuse@wanadoo.fr
e-mail:       postmaster@wanadoo.fr
admin-c:      FTI-RIPE
tech-c:       TEFS1-RIPE
nic-hdl:      WITR1-RIPE
notify:       gestionip.ft@francetelecom.com
mnt-by:       FT-BRX
changed:      gestionip.ft@francetelecom.com 20010504
changed:      gestionip.ft@francetelecom.com 20010912
changed:      gestionip.ft@francetelecom.c om 20011204
source:       RIPE
```

Like previously, abuse info should be sent to abuse@wanadoo.fr

194.248.237.100, 202.181.214.4, 150.187.177.12 - More Scanning
Severity: Information Gathering Occurrences: 418,341 - 11,517 - 10,707
Snort Signature ID: spp_portscan

Attacker	***Port***	***Internal Hosts***
194.248.237.100	80	6,047
202.181.214.4	80	73
	443	6,377
150.187.177.12	135	7,177

Here we see 194.248.237.100 do a typical SYN scan of port 80(HTTP) across multiple subnets.

```
Jan  1 15:34:04 194.248.237.100:4717 -> MY.NET.199.241:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4701 -> MY.NET.199.225:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4721 -> MY.NET.199.245:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4728 -> MY.NET.199.252:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4725 -> MY.NET.199.249:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4722 -> MY.NET.199.246:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4729 -> MY.NET.199.253:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4726 -> MY.NET.199.250:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4720 -> MY.NET.199.244:80 SYN ******S*
Jan  1 15:34:04 194.248.237.100:4723 -> MY.NET.199.247:80 SYN ******S*
```

Regardless of the output of this log, we still see this is a sequential scan. Pick a source port, and map that to the last octet of the destination address. Increase the source port by one, and you should get the last octet of the destination address+1.

Correlations: This IP address appears on dshield.org 189,502 times for port 80 scanning (which is what we see here also).

Recommendations: Block this IP address permanently at outside screening routers or the perimeter firewalls. Allow only port 80 connection attempts to authorized servers internally. Report abuse to the network administrator of this net block.

```
$ whois -h whois.ripe.net 194.248.237.100
% This is the RIPE Whois secondary server.
% The objects are in RPSL format.
% Please visit http://www.ripe.net/rpsl for more information.

inetnum:      194.248.237.96 - 194.248.237.127
netname:      NO-DRILLING-SUPPORT-SYSTEMS-AS-NET
descr:        Drilling Support Systems AS
country:      NO
admin-c:      GO18-RIPE
tech-c:       GO18-RIPE
status:       ASSIGNED PA
mnt-by:       TNXHM-MNT
changed:      hansen@nextel.no 20011004
source:       RIPE

route:        194.248.0.0/16
descr:        Nextra, Postboks 393 - Skoyen, N-0212 Oslo, Norway
origin:       AS2119
mnt-by:       AS8210-MNT
changed:      tna@nextel.no 19990618
source:       RIPE

person:       Gisle Odemotland
address:      Drilling Support Systems AS
address:      Skvadronv. 25
address:      N-4050 Sola
address:      Norway
phone:        +47 51 64 49 00
fax-no:       +47 51 64 49 20
e-mail:       gisle@dss.as
nic-hdl:      GO18-RIPE
mnt-by:       TNXHM-MNT
changed:      hansen@nextel.no 20011004
source:       RIPE
```

The abuse letter could be sent to gisle@dss.as. There is also a list of contacts for the whole company on their website (www.dss.as) under "Requests" section in case gisle is unresponsive. If none of the company responds, escalating the the ISP is advisable, which seems to be nextel.no or telenor.com.

We see a similar scan from 202.181.214.4 except for port 443(SSL) across much of the same hosts as 194.248.237.100.

```
Dec 31 00:33:21 202.181.214.4:2039 -> MY.NET.199.226:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2038 -> MY.NET.199.225:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2037 -> MY.NET.199.224:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2036 -> MY.NET.199.223:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2035 -> MY.NET.199.222:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2034 -> MY.NET.199.221:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2033 -> MY.NET.199.220:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2032 -> MY.NET.199.219:443 SYN ******S*
Dec 31 00:33:21 202.181.214.4:2031 -> MY.NET.199.218:443 SYN ******S*
```

Correlations: 202.181.214.4 also appears in dshield.org 18,891 times for scanning port 443.

Recommendations: See recommendations for 194.248.237.100. Block the IP, send an abuse letter, allow only 443 connection requests to authorized internal machines. Again, whois gives us our preliminary information for finding the abuse contact.

```
$ whois -h whois.apnic.net 202.181.214.4
% [whois.apnic.net node-1]
% How to use this server        http://www.apnic.net/db/
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:      202.181.192.0 - 202.181.223.255
netname:      HKCIX
descr:        - HKCIX -
descr:        HongKong Commercial Internet Exchange
country:      HK
admin-c:      CW57-AP
tech-c:       KY28-AP
mnt-by:       APNIC-HM
mnt-lower:    MAINT-HKCIX-AP
changed:      hostmaster@apnic.net 19991206
status:       ALLOCATED PORTABLE
source:       APNIC

person:       CM Wu
address:      IXTech Limited
address:      7/F Ever Gain Plaza, Tower 2,
address:      88 Container Port Road,
address:      Kwai Chung, N.T.
country:      HK
phone:        +852-2603-7955
fax-no:       +852-2603-7952
e-mail:       cmwu@hkcix.com
nic-hdl:      CW57-AP
mnt-by:       MAINT-HKCIX-AP
changed:      kyeung@hkcix.com 20000313
source:       APNIC

person:       Katson Yeung
address:      IXTech Limited
address:      7/F Ever Gain Plaza, Tower 2,
address:      88 Container Port Road,
address:      Kwai Chung, N.T.
country:      HK
phone:        +852-2603-7955
fax-no:       +852-2603-7952
```

```
e-mail:       kyeung@hkcix.com
nic-hdl:      KY28-AP
mnt-by:       MAINT-HKCIX-AP
changed:      kyeung@hkcix.com 20000313
source:       APNIC
```

Abuse email can be sent to cmwu@hkcix.com. If unresponsive, try hyeung@hkcix.com. Last resort would be the contact page on IXtech's website.

150.187.177.12 scanned many MY.NET networks searching for open 135 ports. This is usually associated with Microsoft Windows RPC services and has been a common target due to many security vulnerabilities with this service.

```
Jan  2 07:20:46 150.187.177.12:4935 -> MY.NET.113.222:135 SYN ******S*
Jan  2 07:20:46 150.187.177.12:2155 -> MY.NET.113.223:135 SYN ******S*
Jan  2 07:20:47 150.187.177.12:3847 -> MY.NET.113.225:135 SYN ******S*
Jan  2 07:20:47 150.187.177.12:4121 -> MY.NET.113.226:135 SYN ******S*
Jan  2 07:20:47 150.187.177.12:4291 -> MY.NET.113.227:135 SYN ******S*
Jan  2 07:20:47 150.187.177.12:1544 -> MY.NET.113.228:135 SYN ******S*
Jan  2 07:20:47 150.187.177.12:3339 -> MY.NET.113.206:135 SYN ******S*
```

It also seems that the source ports being used are not sequential as one would think, but 'random' (Sort of).

Correlations: This IP appears in dshield.org 4 times for scanning port 135, and 1 times for scanning 1433.

Recommendations: Report to dshield.org as the number of machines scanned is not accurately reflected in dshield. Block this IP at outside screening routers or perimeter firewalls. Deny port 135 inbound except to authorized machines. Report abuse to source addresses' network administrator.

```
$ whois -h whois.arin.net 150.187.177.12

OrgName:    Consejo Nacional de Investigaciones
OrgID:      CNDI

NetRange:   150.187.0.0 - 150.187.255.255
CIDR:       150.187.0.0/16
NetName:    VZ-NET3
NetHandle:  NET-150-187-0-0-1
Parent:     NET-150-0-0-0-0
NetType:    Direct Assignment
NameServer: DNS.REACCIUN.VE
NameServer: DNS2.REACCIUN.VE
Comment:
RegDate:    1991-05-30
Updated:    1999-10-01

TechHandle: OEA3-ARIN
TechName:   Aguirre M., Oswaldo
TechPhone:  (+58-2) 794-0695/0850
TechEmail:  oaguirre@reacciun.ve

# ARIN Whois database, last updated 2003-01-11 20:00
# Enter ? for additional hints on searching ARIN's Whois
database.
```

Abuse information could be sent to oaguiree@reacciun.ve. This information was last updated in 1999, so there is a chance it is not valid anymore. A quick visit to their website left me confused as there is no English section. I did manage to find the email reacciun@reacciun.ve off the 'Contactenos' section of the website.

Interesting Alerts Related to Backdoor/Trojan activity

Further analysis of 'High port 65535 tcp or udp - possible Red Worm - traffic'

Though the use of port 65535 can very easily be normal traffic, especially on a machine using the network extensively, it gets my attention because it is a port that has *only* trojans associated with it according to neohapsis.

Since there are many hosts involved in this traffic, I'll need to create a link graph to better visualize the data. However, since there is so much data, I will first need to eliminate some of the alerts that can be attributed to something else.

The following list are all MY.NET hosts associated with port 65535 in the alerts file. Any IP with an explanation left blank will be further researched. The rest of the machines have an 'Explanation of False Positive.' This is the program or service responsible for the 65535 port usage. Since this is a valid ephemeral port, machines that are often using the network extensively will use this port eventually as the chances of generating it are greater. Also, many operating systems start at ephemeral port 1024, then increase the source port as more connections are made until they reach 65535 and then roll over which would generate this alert. Notice most hosts that show up in this list are using the network extensively either through file-sharing or games.

Internal Address	***Explanation of False Positive***
MY.NET.110.70	ad.web.aol.com
MY.NET.116.44	SLP, RFC 2165
MY.NET.132.50	Bearshare
MY.NET.140.136	Edonkey
MY.NET.140.9	All to universities, some kind of research.
MY.NET.150.213	Valid TFTP connections
MY.NET.162.111	IRC ident
MY.NET.162.67	FTP server
MY.NET.163.233	store.yahoo.com
MY.NET.168.142	Porn surfing
MY.NET.198.220	?
MY.NET.6.40	University mail server
MY.NET.70.176	WinMX
MY.NET.70.207	Lost Battalion game
MY.NET.82.117	KaZaA
MY.NET.82.2	Lost Battalion game
MY.NET.83.146	WinMX

Internal Address	Explanation of False Positive
MY.NET.84.151	?
MY.NET.84.193	KaZaA
MY.NET.84.244	KaZaA
MY.NET.86.48	Microsoft 445 access (Still BAD)
MY.NET.88.193	?
MY.NET.91.104	KaZaA
MY.NET.91.252	KaZaA

Table 3-7: Reduction of false posit ives over port 65535

Now that we have successfully cut down much of our data due to false positives, we can create a link graph that is somewhat manageable.

The following machines left to analyze can be found in table 3-8 below.

Internal Address	External Addresses
MY.NET.88.193	80.200.117.120 80.200.151.134 80.200.158.95 217.136.65.154
MY.NET.198.220	80.200.151.134 80.200.158.95
MY.NET.84.151	Many!

Table 3-8: Mapping of MY.NET hosts to external addresses using port 65535

The link graph and correlations are particularly disturbing for MY.NET.88.193 and MY.NET.198.220 as well as MY.NET.84.151.

```
01/02-15:05:28.116064  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.117.120:1646 -> MY.NET.88.193:65535
01/02-10:27:15.953572  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1825 -> MY.NET.88.193:65535
01/02-10:27:18.598692  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1825 -> MY.NET.88.193:65535
01/02-10:27:18.599250  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.88.193:65535 -> 80.200.151.134:1825
01/02-11:00:15.853600  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.88.193:65535 -> 80.200.151.134:1964
01/02-11:00:16.267739  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1964 -> MY.NET.88.193:65535
01/02-10:25:15.340856  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.117.120:2936 -> MY.NET.88.193:65535
01/02-10:25:15.341320  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.88.193:65535 -> 80.200.117.120:2936
01/02-10:25:15.709271  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1825 -> MY.NET.88.193:65535
01/02-10:25:17.824907  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.88.193:65535 -> 80.200.117.120:2936
01/02-10:25:18.167714  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1825 -> MY.NET.88.193:65535
01/02-10:25:21.667551  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.117.120:2936 -> MY.NET.88.193:65535
01/03-15:07:32.694288  [**] High port 65535 tcp - possible Red Worm - traffic

[**] 80.200.151.134:1373 -> MY.NET.198.220:65535
01/03-15:07:32.721335  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.198.220:65535 -> 80.200.151.134:1027
01/03-15:07:33.091772  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1027 -> MY.NET.198.220:65535
01/03-15:16:15.200349  [**] High port 65535 tcp - possible Red Worm - traffic
[**] 80.200.151.134:1373 -> MY.NET.198.220:65535
01/03-15:16:15.424130  [**] High port 65535 tcp - possible Red Worm - traffic
```

```
[**] 80.200.151.134:1373 -> MY.NET.198.220:65535
```

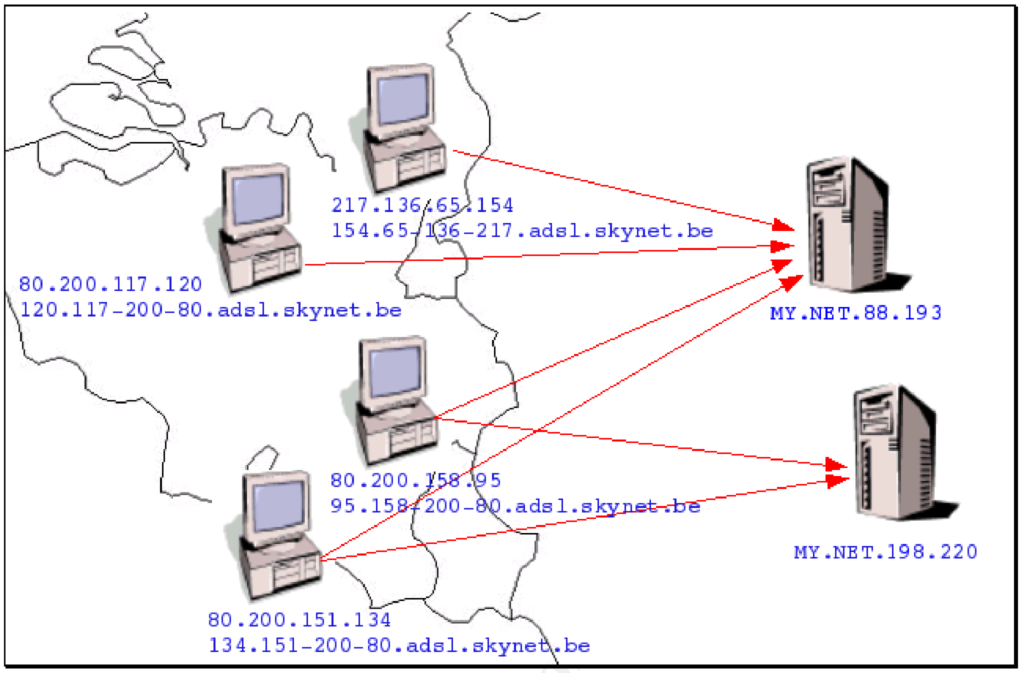
217.136.65.154
154.65-136-217.adsl.skynet.be
80.200.117.120
120.117-200-80.adsl.skynet.be
MY.NET.88.193
80.200.158.95
95.158-200-80.adsl.skynet.be
MY.NET.198.220
80.200.151.134
134.151-200-80.adsl.skynet.be

Figure 3-1 Link graph of potential bad traffic

Remember our friend fr0lux from the previous section? He logged on to the #eci channel using at least four different IP addresses within a few minutes. One of these addresses was 80.200.151.134, an address that scanned the MY.NET networks for several Microsoft networking ports across thousands of the university's computers. As shown in the link graph, 80.200.151.134 is also a machine used to communicate with the university's machines over port 65535. This IP has also attempted to retrieve the /etc/passwd file from several FTP servers on the university's network. All the external machines , and all machines fr0lux used to connect to IRC originate from a skynet.be. MY.NET.84.151 has many (48) connections to port 65535 from the outside. 7 of these addresses are skynet.be addresses with another 29 coming from the abo.wanadoo.fr ISP.

The data seems to suggest that there is some kind of server running on port 65535 on these three MY.NET hosts. I presume these University hosts have been compromised and are now part of some sort of warez, music, or movie site with MY.NET.84.151 being the main repository. (Perhaps it has been determine it has more resources). Either that, or all three MY.NET computers involved are run by French (or Belgium) students who run servers on port 65535 and transfer lots of data with their friends over the ocean (haha).

MY.NET.88.193, MY.NET.88.220, and MY.NET.85.151 need to be checked for Backdoor/Trojan processes running and scanned with anti-virus software as soon as possible as they are actively communicating with these hosts over port 65535 as shown in the log data. Blocking these Belgium addresses permanently at the outside screening router or perimeter firewalls is recommended as well depending on what is found in researching the hosts.

Some Trojans listed on the neohapsis ports list that communicate over TCP port 65535 are:

- `Adore worm (Red Worm)`
- `RC1 trojan`
- `Sins`

I'm guessing, however, that these hosts were compromised through other means and there is a file sharing server running on these ports (Such as FTP) judging from the amount of data and connection patterns.

Out of Spec Packets

Packets that are 'out of spec' are packets that have particular values which should not show up in normal network traffic. A TCP packet, for example, with the SYN flag set and the FIN flag set. This should not happen because these flags are mutually exclusive and should not be set at the same time. It does not make sense to try and initiate a connection while at the same time tearing that connection down.

The total amount of out of spec packets came to 5,985. Table 3-9 outlines the majority of occurrences of out of spec traffic within the five day monitoring period.

TCP Flags	***Occ.***	***TCP Flags***	***Occ.***
12****S*	4,591	12UAPR**	3
********	1,220	12UA*R*F	3
****P***	82	12*A*RS*	3
12***R**	10	****PRSF	3
1**AP*SF	5	12UAP*SF	2

Table 3-9: Top 10 Out of Spec flag combinations

77% of the out of spec packets were caught by snort because the two reserved flag bits were set in a SYN packet (12****S*).

This type of traffic is listed below.

```
12/29-21:20:58.645148 209.47.251.14:56219 -> MY.NET.6.40:25
TCP TTL:48 TOS:0x0 ID:11370 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xE723123E  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1380 SackOK TS: 407528058 0 NOP WS: 0

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

12/29-21:29:52.642443 209.116.70.75:41665 -> MY.NET.139.230:25
TCP TTL:50 TOS:0x0 ID:17917 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x388E8492  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1602077039 0 NOP WS: 0

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

12/29-21:40:09.920530 209.116.70.75:44035 -> MY.NET.139.230:25
TCP TTL:50 TOS:0x0 ID:53195 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x5E987D46  Ack: 0x0  Win: 0x16D0  TcpLen: 40
```

```
TCP Options (5) => MSS: 1460 SackOK TS: 1602138765 0 NOP WS: 0

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

12/29-21:56:12.262382 198.137.194.222:52866 -> MY.NET.6.40:25
TCP TTL:50 TOS:0x0 ID:17088 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xBA08AA17  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 570288343 0 NOP WS: 0
```

This type of traffic can be attributed to ECN (Explicit Congestion Notification) as described in RFC 2481.

"This proposal specifies two new flags in the Reserved field of the TCP header. The TCP mechanism for negotiating ECN-Capability uses the ECN-Echo flag in the TCP header. (This was called the ECN Notify flag in some earlier documents.) Bit 9 in the Reserved field of the TCP header is designated as the ECN-Echo flag. The location of the 6-bit Reserved field in the TCP header is shown in Figure 3 of RFC 793 [RFC793]."

This traffic is legitimate and should not be considered 'out of spec' anymore (as it is clearly in spec).

The second most seen out of spec packet on the university's network are NULL TCP packets. These are TCP packets which have no TCP flags set as shown below.

```
01/02-23:04:29.449500 MY.NET.70.183:53092 -> MY.NET.1.4:37
TCP TTL:64 TOS:0x0 ID:346 IpLen:20 DgmLen:40
******** Seq: 0x46000000  Ack: 0x0  Win: 0x5AC  TcpLen: 20

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

01/02-23:04:31.446226 MY.NET.70.183:53092 -> MY.NET.1.4:37
TCP TTL:64 TOS:0x0 ID:347 IpLen:20 DgmLen:40
******** Seq: 0x46000000  Ack: 0x0  Win: 0x5AC  TcpLen: 20

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

01/02-23:04:33.443080 MY.NET.70.183:53092 -> MY.NET.1.4:37
TCP TTL:64 TOS:0x0 ID:348 IpLen:20 DgmLen:40
******** Seq: 0x46000000  Ack: 0x0  Win: 0x5AC  TcpLen: 20
```

The majority of these NULL packets were to MY.NET.1.4 which seems to be running a time server on port 37 (not to be confused with ntpd on port 123). All time requests are coming from MY.NET.53.10, MY.NET.53.84, and MY.NET.70.183 which are all probably running similar software with a TCP/IP stack implementation problem. This does not appear to be malicious traffic though it should not happen within the TCP protocol. These types of packets and other anomalous TCP packets are commonly used for OS fingerprinting though this does not appear to be the case here.

Another reason out of spec packets occur is simply because somewhere along the way, a packet gets mangled, corrupted, or altered into a invalid state which is most likely what happened in most of our lower occurring out of spec packets. Below is an example of such corruption.

```
01/02-18:27:24.786335 200.167.121.16:4251 -> MY.NET.150.220:1214
TCP TTL:108 TOS:0x0 ID:27019 IpLen:20 DgmLen:52 DF
****P*** Seq: 0xAF7740A  Ack: 0x0  Win: 0x2000  TcpLen: 20
```

```
B4 36 9F 18 8A 3B 0F 08 0A 50 8A 19              .6...;...P..

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

01/02-18:28:28.790895 200.167.121.16:4251 -> MY.NET.150.220:1214
TCP TTL:108 TOS:0x0 ID:6185 IpLen:20 DgmLen:52 DF
****P*** Seq: 0xAF7740A  Ack: 0x0  Win: 0x2000  TcpLen: 20
B4 36 9F 18 8A 3B 0F 08 0A 50 8A 19              .6...;...P..
```

And from our scan logs we have:

```
Jan  2 18:13:21 200.167.121.16:3937 -> MY.NET.150.220:1214 VECNA ****P***
Jan  2 18:22:04 200.167.121.16:4251 -> MY.NET.150.220:1214 VECNA ****P***
Jan  2 18:23:08 200.167.121.16:4251 -> MY.NET.150.220:1214 VECNA ****P***
Jan  2 18:24:12 200.167.121.16:4251 -> MY.NET.150.220:1214 VECNA ****P***
Jan  2 18:25:16 200.167.121.16:4251 -> MY.NET.150.220:1214 VECNA ****P***
Jan  2 18:27:24 200.167.121.16:4251 -> MY.NET.150.220:1214 VECNA ****P***
```

It seems that every once in a while the packets from 200.167.121.16 (a KaZaA client) will come through with just the PUSH flag set. The PUSH flag is most often paired with the ACK flag during data transfer to tell the stack that the data should be made available to the upper layers as soon as possible.

Top Talkers

Table 3-10 shows the top 10 talkers from the alert files. The top talkers of the scan files were already analyzed in scans section. Many of these IP addresses will be directly related to the alerts previously analyzed in the 'Alerts of 2,000' section.

Address	***Occurrences***
MY.NET.105.204	91,160
194.87.6.75	65,285
MY.NET.112.204	26,800
MY.NET.71.230	20,561
MY.NET.84.151	11,499
MY.NET.88.193	9,040
81.50.52.235	8,470
217.136.65.154	8,082
MY.NET.111.235	5,418
MY.NET.111.232	5,414

Table 3-10: Top 10 talkers

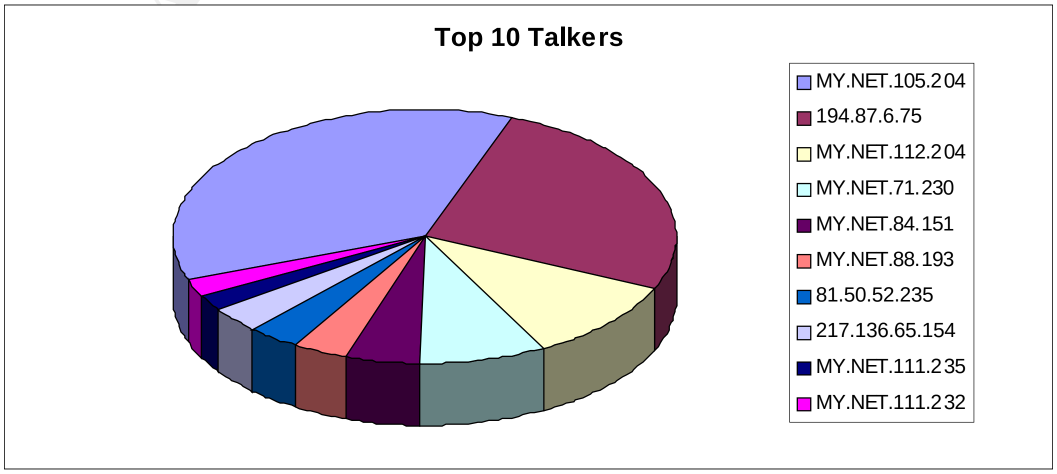

Figure 3-2: Comparison of Top 10 Tal kers.

MY.NET.105.204 and 194.87.6.75
These addresses were responsible for the 'Russia Dynamo - SANS Flash 28-jul-00' alerts.

MY.NET.112.204
Responsible for many of the 'IIS Unicode attack detected' alerts.

MY.NET.71.230
This machine appears to be infected with Nimda as described in the 'Alerts generated more than 2,000 times' section.

MY.NET.84.151, MY.NET.88.193 , 81.50.52.235 and 217.136.65.154
These hosts make the top 10 talker list because of the amount of traffic they push over port 65535. These hosts are described in detail in the 'Trojan/Backdoor activity' section.

MY.NET.111.235 and MY.NET.111.232
These hosts generate thousands of 'TFTP - External UDP connection to internal tftp server' alerts as described in the 'Alerts generated more than 2,000 times' section.

Defensive Recommendation/Summary

If there is not already some sort of stateful firewall device controlling access to MY.NET, one should be installed immediately. Almost all modern firewalls are stateful and there are several viable solutions. A proxy-based firewall would also be a viable solution depending on the user-base and deployment. In general, proxy-based firewalls are more resource intensive on the firewall hardware than a purely stateful firewall and would require higher-end hardware to support the same amount of users a non-proxy-based stateful firewall could sustain. Several, though not near all, options are listed in *Table 3-11*. Proxy-based firewalls provide similar security to stateful firewalls (and can be considered stateful themselves).

OS	Software	Type
Linux	Iptables	stateful
PixOS	PIX	stateful
Linux SunOS	Checkpoint	stateful
BSDi SunOS	Gauntlet	proxy

Table 3-11: Available firewall solutions

The firewall ruleset, or router ACLs need to be adjusted as suggested in previous 'Recommendations' sections of traffic analyzed. It appears almost no access control is being done at present which needs to be rectified as soon as possible.

The hosts suspect of Trojan activity or worm infection should be taken offline as soon as possible and cleaned if needed.

If the university does not have policies and procedures stating acceptable use of the school networks and network bandwidth, one should be created and adhered to.

Tools used for Analyze This
Debian GNU/Linux (http://www.debian.org)
Openoffice 1.0.1
Snort 1.9.0 (www.snort.org)
Google (www.google.com)
Snort signature/port database (http://www.snort.org/snort-db/)
SANS Institute (www.sans.org)
Distributed Intrusion Detection System (www.dshield.org)
Neohapsis port list (http://www.treachery.net/security_tools/ports/)
Managed SherlockESM (www.lurhq.com/msesm.htm)

I used the following Perl script for sorting alerts which I hijacked from Craig Baltes' GCIA practical.

```
#!/usr/bin/perl
# Copyright (c) Joe Stewart


#09/18-10:03:26.121357  [**] Watchlist 000220 IL-ISDNNET-990517
[**] 212.179.35.118:80 -> MY.NET.153.150:1870
# The above is for reference on how the alerts are formatted

my $count = 0;
while (<>) {
      if (/.*\[\*\*\] (.*) \[\*\*\] (.*) -> (.*)/) {
                  $count++;
                  my $msg = $1; my $src = $2; my $dst = $3;
                  $msg =~ s/[^A-Za-z0-9 -_]//g;
                  $msg =~ s/ /_/g;
                  open(OUT, ">>msg/$msg");
                  print OUT;
                  close OUT;
                  $src =~ s/:.*//g;
                  open(OUT, ">>src/$src");
                  print OUT;
                  close OUT;
                  print "Processed $count lines\n";
      } else {
                  print "Skipped $_";
                  $skipped .= $_;
      }
}
print "Skipped the following lines:\n$skipped\n";
```

I also used a plethora of perl 'one liners' (though they usually wrapped a couple of lines) which I tweaked through-out analyzing the data.

I mainly used cat, awk, grep, uniq, and sort for fine-tuning data and counting occurrences.

Other standard network commands like dig, whois, etc were also utilized.

Appendix A: hping2 patch

Disclaimer: I make no guarantees this patch is portable or doesn't break anything. I only needed this to work on my box for testing and made a quick hack.

```
--- hping2/globals.h2001-08-10 11:57:44.000000000 -0400
+++ hping2-3/globals.h           2002-12-07 11:10:53.000000000 -0500
@@ -105,6 +105,8 @@
                    sign[1024],
                    rsign[1024],
                    ip_opt[40],
+                   icmp_saddr[1024],
+                   icmp_daddr[1024],
                    ip_optlen;

 extern struct sockaddr_in local, remote;
--- hping2/main.c   2001-08-13 20:07:33.000000000 -0400
+++ hping2-3/main.c 2002-12-07 11:52:07.000000000 -0500
@@ -23,6 +23,7 @@
 #include <signal.h>
 #include <sys/time.h>
 #include <sys/types.h>
+#include <time.h>

 #include "hping2.h"

@@ -123,6 +124,8 @@
       ifname                   [1024] = {'\0'},
       ifstraddr    [1024],
       spoofaddr    [1024],
+      icmp_saddr [1024],
+      icmp_daddr [1024],
       sign                     [1024],
       rsign                    [1024], /* reverse sign (hping -> gniph) */
       ip_opt                   [40],
@@ -225,7 +228,7 @@
                    resolve((struct sockaddr*)&local, ifstraddr);
       else
                    resolve((struct sockaddr*)&local, spoofaddr);
-
+
       srand(time(NULL));

       /* set initial source port */
--- hping2/parseoptions.c        2001-08-14 07:02:52.000000000 -0400
+++ hping2-3/parseoptions.c      2002-12-07 11:38:16.000000000 -0500
@@ -102,6 +102,8 @@
                    ":-icmp-cksum",
                    "=-icmp-ts",
                    "=-icmp-addr",
+                   ":-icmp-saddr",
+                   ":-icmp-daddr",
                    "=-tcpexitcode",
                    "=-fast",
                    "=-tr-keep-ttl",
@@ -432,6 +434,16 @@
                                usec_delay.it_value.tv_usec =
                                usec_delay.it_interval.tv_usec = 100000;
                    ESAC
+                   ONLYGNUCASE("-icmp-saddr")
+                               SUIDLIMIT;
+                               CHECKARG;
+                               strncpy (icmp_saddr, hoptarg, 1024);
+                   ESAC
+                   ONLYGNUCASE("-icmp-daddr")
+                               SUIDLIMIT;
+                               CHECKARG;
+                               strncpy (icmp_daddr, hoptarg, 1024);
+                    ESAC
                     ONLYGNUCASE("-tr-keep-ttl")
                                  opt_tr_keep_ttl = TRUE;
                     ESAC
--- hping2/sendicmp.c             2001-08-13 19:27:52.000000000 -0400
+++ hping2-3/sendicmp.c           2002-12-07 12:27:38.000000000 -0500
@@ -16,6 +16,10 @@
 #include <string.h>
 #include <signal.h>
 #include <errno.h>
+#include <netinet/in.h>
+#include <arpa/inet.h>
+
+#include <time.h>

 #include "hping2.h"
 #include "globals.h"
@@ -264,9 +268,12 @@
        struct myicmphdr *icmp;
        struct myiphdr icmp_ip;
        struct myudphdr icmp_udp;
+       struct in_addr temp;
        int errno_save = errno;
        int left_space = data_size;

+       memset(&temp, 0, sizeof(struct in_addr));
+
        packet = malloc(ICMPHDR_SIZE + data_size);
        if (packet == NULL) {
                     perror("[send_icmp] malloc");
@@ -299,9 +306,27 @@
        icmp_ip.ttl      = 64;                                     /* 64 */
        icmp_ip.protocol = icmp_ip_protocol;                /* 6 (TCP) */
        icmp_ip.check = 255;                                       /*
FIXME: compute */
-       memcpy(&icmp_ip.saddr, "AAAA", 4);
-       memcpy(&icmp_ip.daddr, "BBBB", 4);

+       if (icmp_saddr[0] == '\0') {
+                    memcpy(&icmp_ip.saddr, "AAAA", 4);
+       }
+       else {
+                    if (inet_aton(icmp_saddr, &temp) == 0) {
+                                 goto no_space_left;
+                    }
+                    memcpy(&icmp_ip.saddr, &temp.s_addr, 4);
+       }
+
+       if (icmp_daddr[0] == '\0') {
+                    memcpy(&icmp_ip.daddr, "BBBB", 4);
+       }
+       else {
+                    if (inet_aton(icmp_daddr, &temp) == 0) {
+                                 goto no_space_left;
+                    }
+                    memcpy(&icmp_ip.daddr, &temp.s_addr, 4);
+       }
+
       /* UDP header */
       icmp_udp.uh_sport = htons(1111);
       icmp_udp.uh_dport = htons(2222);
--- hping2/sendtcp.c2001-08-13 19:27:52.000000000 -0400
+++ hping2-3/sendtcp.c            2003-01-26 23:20:23.000000000 -0500
@@ -15,6 +15,7 @@
 #include <unistd.h>
 #include <signal.h>
 #include <errno.h>
+#include <time.h>

 #include "hping2.h"
 #include "globals.h"
--- hping2/sendudp.c2001-08-13 19:27:52.000000000 -0400
+++ hping2-3/sendudp.c            2003-01-26 23:20:38.000000000 -0500
@@ -15,6 +15,7 @@
 #include <unistd.h>
 #include <signal.h>
 #include <errno.h>
+#include <time.h>

 #include "hping2.h"
 #include "globals.h"
--- hping2/usage.c  2001-08-10 11:57:44.000000000 -0400
+++ hping2-3/usage.c2002-12-07 11:34:18.000000000 -0500
@@ -115,6 +115,8 @@
 "  --icmp-ipid      set ip id                    ( default random )\n"
 "  --icmp-ipproto   set ip protocol              ( default IPPROTO_TCP )\n"
 "  --icmp-cksum     set icmp checksum            ( default the right cksum)\n"
+"  --icmp-saddr     set saddr in icmp data       ( default AAAA )\n"
+"  --icmp-daddr     set daddr in icmp data       ( default BBBB )\n"
        );
        exit(0);
 }
```

References

[1] Anonymous. “apache-worm.c” http://dammit.lt. URL: http://dammit.lt/apache-worm/apache-worm.c

[2] Anonymous. “[Snort Signature] WEB-IIS cmd.exe access” www.snort.org. URL: http://www.snort.org/snort-db/sid.html?sid=1002

[3] Bell, Mike. “GCIA Practical” www.giac.org. Jan, 2001. URL: http://www.giac.org/practical/Mike_Bell_GCIA.doc

[4] CERT. “CERT® Advisory CA-2002-17 Apache Web Server Chunk Handling Vulnerability” www.cert.org. Sept 25, 2002. URL: http://www.cert.org/advisories/ CA-2002-17.html

[5] CERT. “CERT® Advisory CA-2001-26 Nimda Worm” www.cert.org. Sept 25, 2002. URL: http://www.cert.org/advisories/CA-2001-26.html

[6] Cisco Systems. “Preparing To Use Essentials Applications” www.cisco.com. Sept 30, 1999. URL: http://www.cisco.com/univercd/cc/td/doc/product/rtrmgm t/cw2000/camp_mgr/cwsi_2x/cwsi_2_2/ig_aix/solprep.htm

[7] CVE. “CAN-2002-0392 (under review)” http://cve.mitre.org. Aug 8, 2002. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0392

[8] F5 Networks. “3-DNS controller” www.f5.com. URL: http://www.f5.com/f5prod ucts/3dns/index.html

[9] Farley, Paul. “GCIA Practical Assignment” www.giac.org. May 10, 2002. URL: http://www.giac.org/practical/Paul_Farley_GCIA.doc

[10] Fielding R., Gettys J., Mogul J, Frystyk H. “Hypertext Transfer Protocol -- HTTP/1.1” www.rfc-editor.org. June 1999. URL: ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt

[11] F-Secure. “F-Secure Virus Descriptions – Adore” www.fsecure.com. Apr, 2001. URL: http://www.f-secure.com/v-descs/adore.shtml

[12] GIAC. “GIAC Certified Intrusion Analyst (GCIA) Practical Assignemnt”

www.giac.org. Aug 15 2002. URL: http://www.giac.org/GCIA_assignment.php

[13] Incidents.org. “Top 10 attacked ports” www.incidents.org. Jan 24, 2003. URL: http://isc.incidents.org/top10.html

[14] Institute for Security Technology Studies. “Adore Worm Detection and Removal Tool” www.ists.dartmouth.edu. Jan 7, 2003. URL: http://www.ists.dartmouth.e du/IRIA/knowledge_base/tools/adorefind.htm

[15] Linux Virtual Server. “LVS Deployment” www.linuxvirtualserver.org. URL: http://www.linuxvirtualserver.org/deployment.html

[16] McAfee. “Virus Profile W32/Opaserv.worm” www.mcafee.com. Jan 16, 2003. URL: http://vil.mcafee.com/dispVirus.asp?virus_k=99729

[17] McDonnel, Michael. “GCIA Practical Assignment” www.giac.org. Novr 27, 2001. URL: http://www.giac.org/practical/Michael_McDonnell_GCIA.doc

[18] Mituzas, Domas. “Brief analysis of apache-worm.c” http://dammit.lt. URL: http://dammit.lt/apache-worm/apache-worm.c

[19] Microsoft Corporation. “Microsoft Security Bulletin (MS00-072).” www.microsoft.com. Feb 16, 2001. URL: http://www.microsoft.com/technet /treeview/default.asp?url=/technet/security/bulletin/MS00-072.asp

[20] Paciaffi, Dewey. “Re: [Snort-users] BAD TRAFFIC data in TCP SYN packet” archives.neohapsis.com. Jan 14, 2002. URL: http://archives.neohapsis.com/ar chives/snort/2002-01/0312.html

[21] Puppy, Rain Forest. “Poison NULL byte” www.phrack-dont-give-a-shit-about-dmca.org Sept 9, 1999. URL: http://www.phrack-dont-give-a-shit-about-dmca.org/phrack/55/P55-07

[22] Reiter, Michael. “GCIH Kernel Subversion” www.giac.org. URL: http://www.giac.org/practical/Michael_Reiter_GCIH.zip

[23] Rekhter, Y., Moskowitz B. “RFC 1918 Address Allocation for Private Internets” www.rfc-editor.org. Feb, 1996. URL: ftp://ftp.rfc-editor.org/in-notes/rfc1918.txt

[24] Ruiu, Dragos and others. “Snort FAQ” www.snort.org. Mar 25, 2002. URL: http://www.snort.org/docs/faq.html

[25] SANS Institute. “Global Incident Analysis Center - Detects Analyzed” www.sans.org. July 29, 2000. URL: http://www.sans.org/y2k/072818.htm

[26] SecurityFocus. “Apache Chunked-Encoding Memory Corruption Vulnerability” www.securityfocus.com. URL: http://online.securityfocus.com/bid/5033

[27] Sollins, K. “RFC 783 - THE TFTP PROTOCOL (REVISION 2)” www.rfc-editor.org. July 1992. URL: ftp://ftp.rfc-editor.org/in-notes/rfc1350.txt

[28] Stewart, Joe. “Re: [Snort-users] CGI Null Byte Attack” archives.neohapsis.org. Nov 20, 2000. URL: http://archives.neohapsis.com/archives/snort/2000-11/0244.html

[29] Stewart, Joe. “Re: [Snort-users] What defines a IIS Unicode Attack?”. Dec 12, 2000. URL: http://archives.neohapsis.com/archives/snort/2000-12/0180.html

[30] Symantec Corporation. “W32.Opaserv Worm.” www.symantec.com. Dec 5, 2002. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.opaserv.worm.html

[31] The Lost Battalion. “Commands”. www.thelostbattalion.net. URL: http://www.thelostbattalion.net/ccomands.html

[32] Turkia, Miika. “GCIA Practical Assignment” www.giac.org. Jan 28, 2001. URL: http://www.giac.org/practical/Miika_Turkia_GCIA.html

[33] Urwiller, Bradley. “GCIA Practical Assignement” www.giac.org. Apr 23, 2002. URL: www.giac.org/practical/Bradley_Urwiller_GCIA.pdf

[34] Walker, Martin. “Apache Web Server Chunk Handling Vulnerability: An Exploit In Action” GCIH. Oct 7, 2002. URL: http://www.giac.org/practical/Walker_Martin_GCIH.doc

[35] Whitehats. “IDS177 ‘NETBIOS-NAME-QUERY’” www.whitehats.com. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids177&view=event

[36] Vision, Max. "Re: [snort] 'SMB Name Wildcard.'" archives.neohapsis.com. Jan 17, 2000. URL: http://archives.neohapsis.com/archives/snort/2000-01/0220.html