Jared McLaren
GCIA Practical Assignment
Version 3.3

Submitted March 12, 2003

Table of Contents

Abstract

This paper is comprised of three sections. The first section takes a look at how Idlescanning works. The second section analyzes network packet captures for anomalous or malicious intent. The third part studies five consecutive days worth of data on a university network.

Part 1 – Reconnaissance Technique

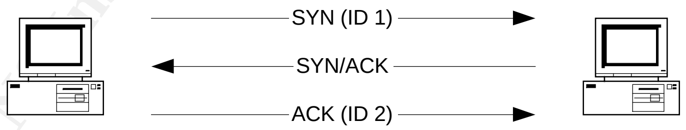
Nmap Idlescan

Network Mapper (Nmap) is the most popular port scanning utility available on the Internet. This free scanner was made to rapidly scan through networks for open ports and services. It has the ability to change how it scans for remote hosts. Examples include a normal connect scan, a SYN scan, and a FIN scan among many others. Nmap also provides valuable services like remote operating system identification, remote protocol identification, and built-in spoofing capabilities. The program has more advanced options than many people are likely to ever use. One advanced option that I will discuss in more detail is the "idlescan" option. This scan offers a blind port scan of a remote host so that they do not ever see your true source IP. This paper will analyze SYN scans and spoofing in relation to a working idlescan.

Nmap is a product of Insecure.org. At the time of this report, Nmap version 3.10 ALPHA9 was available for download. All analysis will be done using Nmap version 3.10 ALPHA9 on a Linux system running kernel version 2.4.18-19.7.x.

Download Nmap:
http://www.insecure.org/nmap/nmap_download.html

How is this possible? How can you scan a remote host with reliable results and never show your IP address? The answer lies in the IP Identification field. The IP ID is incremental on many operating systems. This means that every time an IP packet leaves your machine, the IP ID field is incremented by 1.



We need to figure out how to get that incremental ID number working for us. If we know that the IP ID increments by 1 every time a packet is sent, we can figure out how many packets have been sent by a host. The challenge is getting that to work to our advantage. Eventually, we'll learn how a SYN scan when combined with spoofing will give us positive results.

SYN Scanning

Before we get into details about idlescanning, it's important to understand how a SYN scan works. SYN scans are popular for their speed and high reliability. They do not create a full connection to a remote host. This makes them stealthier to application-level logging. Keep in mind that they can trigger alerts on firewalls and intrusion detection systems.

It's pretty easy to know when you've found an open port in a SYN scan. Let's say that a SYN packet is sent to a web server on port 80. The TCP/IP 3-way-handshake connection procedure will result in the web server sending back a SYN/ACK packet in response to the client's SYN. This means that when we are SYN scanning, a responding SYN/ACK means that the port is open. If a SYN/ACK is received, we will respond with a reset (RST) packet and move on to the next port. The RST packet is used to quickly tear down a connection with no further communications required from either host.

(tcpdump captures of Nmap are used for the below examples)

<u>SYN Packet from client</u>

```
14:08:44.350876 192.168.0.7.57299 > 192.168.0.9.80: S 2292115001:2292115001(0) win 3072
```

<u>SYN/ACK Response from server</u>

```
14:08:44.355547 192.168.0.9.80 > 192.168.0.7.57299: S 900258327:900258327(0) ack
2292115002 win 16616 <mss 1460> (DF)
```

<u>RST sent back from client to server</u>

```
14:08:44.355560 192.168.0.7.57299 > 192.168.0.9.80: R 2292115002:2292115002(0) win 0 (DF)
```

A closed port is even easier to detect. Let's say that you send a SYN packet to a web server on port 81. That port is not open since normal http services run over port 80, so the server will send back a reset (RST) packet in response to our initial SYN packet. This is a sure sign to the client that the server is not listening on that port. The SYN scanner then moves on to the next port without any further action.

(tcpdump captures of Nmap are used for the below examples)

<u>SYN Packet from client</u>

```
14:08:44.350482 192.168.0.7.57299 > 192.168.0.9.81: S 2292115001:2292115001(0) win 3072
```

<u>RST Packet from server</u>

```
14:08:44.353269 192.168.0.9.81 > 192.168.0.7.57299: R 0:0(0) ack 2292115002 win 0
```

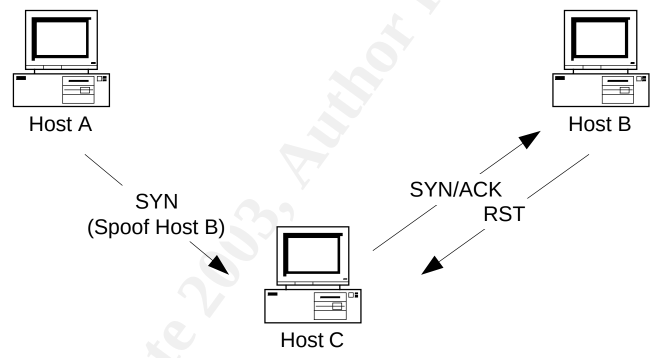
Nmap features a powerful and extremely fast SYN scanner. This is a great utility for finding remote services on machines in your network. As you can see, the packet transfer is much smaller than what would be required in a full connect scan. This makes SYN scans less network intensive than their connect scan counterpart.

Spoofing

The next part to understand is spoofing. Spoofing is forging a source address such that the remote host believes it came from someone else. This is highly effective in generating a denial of service attack or exploit while maintaining a level of anonymity. Spoofing is also used in sequence number prediction attacks.

Let's take a look at how spoofing works from a conceptual point of view. If a disgruntled employee wants to send a SYN flood (a type of Denial of Service attack) to the corporate email server, it can be done easily with some degree of anonymity. The employee simply forges the source address as the corporate web server and begins to flood SYN traffic to port 25 on the email server.

This scenario creates a flow of traffic as shown below:



The disgruntled employee (Host A) sends the spoofed SYN packet. The Email server (Host C) receives the packet, believing it came from the web server (Host B). Host C sends the corresponding SYN/ACK packet to Host B. Host B does not remember sending a SYN request to Host C, so therefore will likely send a RST packet to tear down the connection attempt. This process will continue as long as Host A continues to send spoofed SYN packets to Host C.

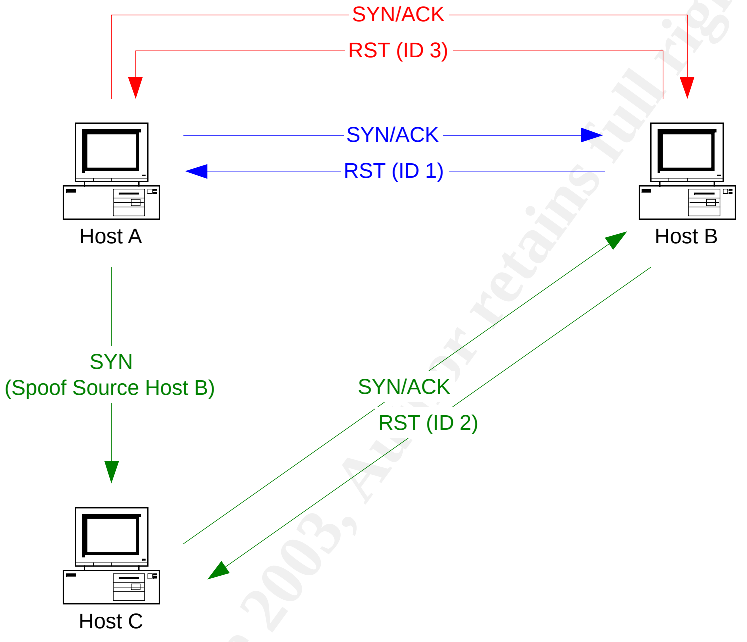
By looking at the above scenario, we see how we can divert the flow of traffic towards an innocent host by spoofing packets. This is where we begin to tie the pieces together. We know that IP ID numbers on Host B will increment sequentially. We know that we can spoof Host B while sending traffic to Host C. We know that host B will send out a RST packet if Host C sends an unsolicited SYN/ACK packet. What is stopping us from analyzing the ID numbers of Host B to see if a RST packet was sent in response to an incoming SYN/ACK packet?

Idlescanning

That above discovery is the basis of idlescanning. If Host B is not actively communicating with hosts on the network, we can sniff and analyze the incoming IP ID numbers and use them as a remote packet count tool with high accuracy. Let's take a look at how this might work.

(See the below diagram for the first example of Idlescanning: Open Port)

IDLESCANNING: OPEN PORT



BLUE:	First set of packets
GREEN:	Second set of packets
RED:	Third set of packets

We have now covered most of the concepts in the above diagram. Don't let the diagram appear more complicated than it is. We'll go through the process step by step to see how we can make idlescanning work for us. Continue to assume that Host A is the attacker in this example. This example also assumes that Host B uses incremental IP ID's and is not actively communicating with other hosts on the network.

STEP 1: BLUE

The first packets are sent to get the base IP ID number. We send a SYN/ACK packet to Host B so that we can see the IP ID number in the returning RST packet. We note that the ID number is 1. This means the next outbound packet from Host B will be ID 2.

STEP 2: GREEN

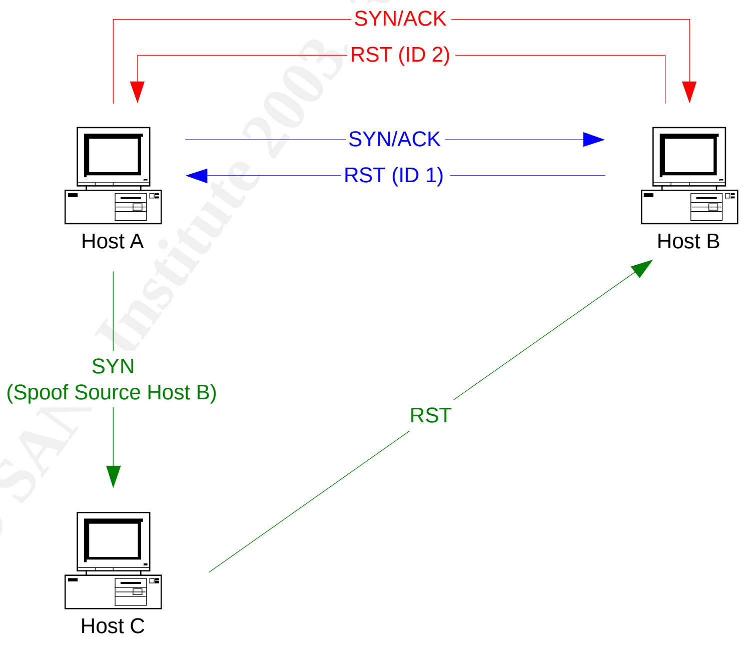
The second set of packets is where spoofing comes into play. We spoof a SYN packet to a port on Host C to make it appear that the source of the connection request is Host B. The above example assumes that the port on Host C is open, so a SYN/ACK is sent to Host B. Host B sends an outbound RST packet to Host C, therefore incrementing the ID to 2.

STEP 3: RED
Now we're at the most interesting part. If we send a SYN/ACK packet to Host B, we will see that the RST response has an ID of 3. Remember that the ID was 1 when we checked it in the first step. We never saw ID 2, which means that Host B must have sent one outbound packet since we first checked its original ID. We can conclude that a SYN/ACK packet was sent from Host C to Host B and that Host B sent back a RST response using ID 2. This means that the port we probed on Host C is in fact open and listening. Congratulations, you have just scanned a port on Host C without ever revealing your IP address!

That's how an idlescan works when the port is open…but what happens when the port is closed? The next example is very similar to the first, and shows how we know the port on Host C is closed.

(See the below diagram for the second example of Idlescanning: Closed Port)

IDLESCANNING: CLOSED PORT



BLUE:	First set of packets
GREEN:	Second set of packets
RED:	Third set of packets

You will find that the steps are very much the same as the above example. We will quickly work through each step highlighting the key points.

STEP 1: BLUE
Send a SYN/ACK packet to Host B and discover ID 1 in the RST response.

STEP 2: GREEN
Spoof a SYN packet to a port on Host C appearing as Host B making a request. The port on Host C is not open, so the result is a RST packet to the spoofed Host B. A RST packet requires no further attention, so host B does nothing when the RST packet arrives.

STEP 3: RED
We now go back and check the ID of Host B using our SYN/ACK request method. The ID is 2. We can conclude that Host C sent a RST packet to Host B since Host B generated no outbound traffic since we first checked its ID in step 1. The port must be closed on Host C.

This concludes the description of how idlescanning works. It's the ultimate stealth in port scanning. An attacker can actively scan systems on your network with high accuracy and never reveal the true source of the scan.

Idlescan Impact

Now that you see how the scan works, it's important to discuss how this impacts your network. First, the obvious is that an attacker can potentially do all necessary reconnaissance of your network without being truly identified. This eliminates the need for the attacker to use a cracked host as the scanning station.

Keep in mind that the host being used as the Idlescanning machine must be idle on the network. It is not difficult for an attacker to find an idle host on a network. Devices like network printers have built-in TCP/IP stacks and could quite easily be used for idlescan reconnaissance. This can be stopped by properly filtering source IP addresses at your border routers to eliminate some incoming spoofing.

Next, trust relationships on your local network can be established. If Host A has a conduit through the firewall to Host B, we can spoof Host A in an idlescan on the local segment to find open ports on Host B. These packets would easily find their way through a packet filtering firewall. This shows how powerful idlescan can be in "bypassing" firewall protection. Stateful firewalls will not be tricked quite as easily as packet filtering firewalls.

If we know it can happen, we need to know how to discover it. The most obvious sign of spoofing is a flood of SYN/ACK packets to a host with corresponding RST packets. This does not help us track the source, but it does let us know someone is spoofing one of our IP addresses. If the spoofing continues, a sniffer can be set up on the spoofed host to check for incoming SYN/ACK packets intended to check our ID in an outbound RST packet. Those incoming SYN/ACK packets are from the true source of the idlescan.

Testing the Nmap Idlescan on a live Network

I tested Nmap idlescan on my home network. The three machines involved in the scan are listed below.

192.168.0.201	- The true scanner – Linux 2.4.x kernel
192.168.0.4	- The scan target – Windows 2000
192.168.0.2	- The idlescan host – Windows 2000

The host 192.168.0.2 was used as the idlescan host because Windows 2000 uses incremental IP ID numbers. This makes it a prime candidate for our use. The Linux host 192.168.0.201 was the scanner because Nmap performs the best on a Unix/Linux platform. That left the other Windows 2000 host (192.168.0.4) as the target.

First, I'll display some helpful network information about each host.

192.168.0.4 Nmap results (Normal Connect Scan):

```
Interesting ports on 192.168.0.4:
(The 1601 ports scanned but not shown below are in state: closed)
Port      State      Service
135/tcp    open       loc-srv
139/tcp    open       netbios-ssn
1025/tcp   open       NFS-or-IIS
```

There are three open ports to target during our idlescan. We will go after port 135 and 1025 since we know they are open and listening.

192.168.0.2 Nmap results:

```
Interesting ports on 192.168.0.2:
(The 1603 ports scanned but not shown below are in state: closed)
Port      State      Service
135/tcp    open       loc-srv
```

As you can see, only port 135 is open to the outside world. This will be our target port in the idlescan for checking the ID.

The target host for the spoofed idle scans needs to be in a peaceful state. No active network connections can be running on the remote machine. This would make the IP ID's hard to follow, and therefore would not allow for a good scan. The following is a netstat output of the idlescan host (192.168.0.2) that shows there are no active network connections that would modify the IP ID numbers outside the scope of our test.

```
C:\>netstat -an
```

Active Connections

```
  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING
  UDP    0.0.0.0:445            *:*
```

Now we have some things established. We are going to scan 192.168.0.4 using 192.168.0.2 as our idlescan host. We are going to look for ports 135 and 1025 on 192.168.0.4. We will also include port 31337 to show how a closed port reacts in the Nmap idlescan. We know that 192.168.0.2 will work as the idlescan host because it is Windows 2000 and uses incremental IP ID numbers. It also has no active network connections.

The idlescan option (-sI) in Nmap must be run as root. It requires root for the use of raw sockets. Raw sockets are necessary for Nmap to create spoofed packets. The command line appears as follows:

```
#nmap –p135,1025,31337 192.168.0.4 –sI 192.168.0.2:135 –P0
```

This shows that we are scanning port 135, 1025, and 31337 on 192.168.0.4 using port 135 on 192.168.0.2 for our IP ID predictions. We are also disabling the Ping option. This is very important – if the ping option is not disabled, the true source IP address pings the destination IP. This gives away the true source of the scan. This takes the anonymity out of the idlescan.

A tcpdump output of the idlescan with commentary is listed below:
(The headers were trimmed for cleanliness – ACK information is not included)

The scanning PC checks 192.168.0.2 for IP ID sequence:

```
18:42:12.727434 192.168.0.201.46957 > 192.168.0.2.135: S (ttl 59, id 5699, len 40)
18:42:12.727511 192.168.0.2.135 > 192.168.0.201.46957: R (ttl 128, id 1637, len 40)
18:42:12.760637 192.168.0.201.46958 > 192.168.0.2.135: S (ttl 59, id 55518, len 40)
18:42:12.760715 192.168.0.2.135 > 192.168.0.201.46958: R (ttl 128, id 1638, len 40)
```

The IP's are sequential. This host will work for an Idlescan.

The scanning PC spoofs 4 packets, then checks if the IP ID is properly predicted:

```
18:42:12.760753 192.168.0.4.46952 > 192.168.0.2.135: (ttl 59, id 24662, len 40)
18:42:12.813371 192.168.0.4.46952 > 192.168.0.2.135: S (ttl 59, id 10635, len 40)
18:42:12.866105 192.168.0.4.46952 > 192.168.0.2.135: S (ttl 59, id 56793, len 40)
18:42:12.918839 192.168.0.4.46952 > 192.168.0.2.135: S (ttl 59, id 26915, len 40)
18:42:13.221594 192.168.0.201.47199 > 192.168.0.2.135: S (ttl 59, id 51022, len 40)
18:42:13.221675 192.168.0.2.135 > 192.168.0.201.47199: R (ttl 128, id 1643, len 40)
```

The ID is incremented properly. The Idlescan host works as planned.

Check for port 31337 using Idlescanning:

```
18:42:13.221719 192.168.0.2.135 > 192.168.0.4.31337: S (ttl 59, id 59953, len 40)
18:42:13.274349 192.168.0.201.47071 > 192.168.0.2.135: S (ttl 59, id 44292, len 40)
18:42:13.274434 192.168.0.2.135 > 192.168.0.201.47071: R (ttl 128, id 1645, len 40)
18:42:13.299718 192.168.0.201.47076 > 192.168.0.2.135: S (ttl 59, id 9569, len 40)
18:42:13.299800 192.168.0.2.135 > 192.168.0.201.47076: R (ttl 128, id 1646, len 40)
18:42:13.299849 192.168.0.2.135 > 192.168.0.4.31337: S (ttl 59, id 41123, len 40)
18:42:13.352453 192.168.0.201.47029 > 192.168.0.2.135: S (ttl 59, id 40765, len 40)
18:42:13.352533 192.168.0.2.135 > 192.168.0.201.47029: R (ttl 128, id 1647, len 40)
18:42:13.377843 192.168.0.201.47088 > 192.168.0.2.135: S (ttl 59, id 56780, len 40)
18:42:13.377923 192.168.0.2.135 > 192.168.0.201.47088: R  (ttl 128, id 1648, len 40)
```

The ID's are incremented only 1 each time. This is correct since port 31337 is NOT open on the remote host.

Check for port 135 using Idlescanning:

```
18:42:13.377962 192.168.0.2.135 > 192.168.0.4.135: S (ttl 59, id 43938, len 40)
18:42:13.432870 192.168.0.201.47076 > 192.168.0.2.135: S (ttl 59, id 42926, len 40)
18:42:13.432977 192.168.0.2.135 > 192.168.0.201.47076: R (ttl 128, id 1650, len 40)
```

The Idlescan host jumped over ID 1649 and is currently at 1650. This must mean that port 135 is open on the scanned host.

Check for port 1025 using Idlescanning:

```
18:42:13.433105 192.168.0.2.135 > 192.168.0.4.1025: S (ttl 59, id 57136, len 40)
18:42:13.485266 192.168.0.201.46982 > 192.168.0.2.135: S (ttl 59, id 18581, len 40)
18:42:13.485346 192.168.0.2.135 > 192.168.0.201.46982: R (ttl 128, id 1652, len 40)
```

The ID once again jumped 2 spots up to 1652. This means that port 1025 is also open on the remote host.

Nmap Output from Idlescan:

```
Idlescan using zombie 192.168.0.2 (192.168.0.2:135); Class: Incremental
Interesting ports on 192.168.0.4:
(The 1 port scanned but not shown below is in state: closed)
Port       State       Service
135/tcp    open        loc-srv
1025/tcp   open        NFS-or-IIS
```

The scan did in fact find port 135 and 1025 open. Port 31337 was not open, and was not found in the scan. This proves that finding a remote host with sequential IP ID's can definitely be used to anonymously scan a remote host.

In my tests, I also tried scanning with OpenBSD 3.2 and Windows XP as the Idlescan hosts. Both operating systems have protection against IP ID sequence. They were both rejected by the Nmap idlescan as they were identified as non-sequential IP ID hosts. A compiled list of vulnerable Operating Systems could not be found at the time of writing this paper. Programming for a non-sequential ID is an important step that software vendors can take in securing the TCP/IP stack on their products.

References:

"Nmap Idle Scan", 2002
http://www.insecure.org/nmap/idlescan.html

Antirez. "New TCP Scan Method", December 17, 1998
http://lists.insecure.org/bugtraq/1998/Dec/0082.html

Nmap Manpage
http://www.insecure.org/nmap/data/nmap_manpage.html

Olofsson, Thomas. "IPID Scanning", 2001
http://www.blackhat.com/presentations/bh-usa-01/ThomasOlofsson/bh-usa-01-Thomas-Oloffson.ppt

Arkin, Ofir. "Network Scanning Techniques", November 1999
http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf

Part 2 – Network Detects

Detect 1: Fragmented ICMP Traffic

Log trace:

```
Frame 1:
11:13:25.090000 IP (tos 0x0, ttl 49, len 28) 195.153.112.21 > 78.119.167.58:
icmp 8: echo request seq 768 (wrong icmp cksum 7dd0 (->30ff)!) (frag
50176:8@0+)
                        4500 001c c400 2000 3101 7c80 c399 7015
                        4e77 a73a 0800 7dd0 c400 0300 0000 0000
                        0000 0000 0000 0000 0000 0000 0000
Frame 2:
11:13:25.090000 IP (tos 0x0, ttl 49, len 76) 195.153.112.21 > 78.119.167.58:
icmp (frag 50176:56@8)
                        4500 004c c400 0001 3101 9c4f c399 7015
                        4e77 a73a 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 a6d8 993d 6818 0b00
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000
Frame 3:
11:13:25.090000 IP (tos 0x0, ttl 254, id 12755, len 84) 78.119.167.58 >
195.153.112.21: icmp 64: echo reply seq 768
                        4500 0054 31d3 0000 fe01 6175 4e77 a73a
                        c399 7015 0000 85d0 c400 0300 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        a6d8 993d 6818 0b00 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000
```

Source of trace:

This trace was recorded on a public section of the network at my workplace and was directed towards a DNS server in an internal DMZ environment. The capture took place directly off of the connection from the ISP and had not yet reached the gateway firewall. The fragmentation set off a series of "Misc: Tiny Fragments" alerts on one of the Snort Intrusion Detection System sensors. I was lucky enough to log into the IDS sensor quick enough to catch a binary log of the incoming data to use for this example. A very small amount of this traffic was generated and it was all ICMP protocol. ICMP echo requests are allowed through the gateway firewall to this specific server for functional reasons.

A conceptual view of the attack path:
Source Host ---> INTERNET ---> IDS sensor ---> Firewall ---> Destination Host

Detect generated by:

The network detect was collected with a binary tcpdump capture on an OpenBSD 3.0 system.

Command line: tcpdump -w frag-icmp.tcp -i eth1 –s 1514 &

The snort signature that alerted me to this fragmentation (from "misc.rules"):

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments"; fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)

The "fragbits:M" section of the snort rule means that snort looks for the More Fragments (MF) bit set as true in the flags section within the IP header. When that is true, the rule then checks for "dsize: < 25" which means the packet payload is less than 25 bytes in total size. When both of these checks are true, a Tiny Fragments alert is triggered by Snort.

After the binary tcpdump capture was created, the network addresses belonging to our network were cleared using the program "tcpmunge". TCPMunge is available for download from http://www.cyber-defense.org.

Probability that the address was spoofed:

Even though ICMP is a simple protocol to spoof since the protocol does not require a handshake, the probability of this packet being spoofed is very low. An ICMP echo request is usually a sign of reconnaissance work, and the source host wants to know if the destination host responds to the packet.

The main reason for spoofing a fragmented packet like this would be to create a Denial of Service attack. The amount of fragmentation traffic detected during this capture was too low to be even remotely considered a Denial of Service and the source was limited to a single IP address.

Description of attack:

This is an IP fragmentation attack used in conjunction with the ICMP protocol. The attack tries to fool simple perimeter filtering devices, bypass IDS sensors to find live hosts for reconnaissance purposes, or possibly (but not likely) identify remote operating systems.

The packets are fragmented in such a way that when reassembled the ICMP header comes from Frame 1 and some unknown data in Frame 2 becomes the payload. The packets are zero-padded to meet length requirements in RFC 894. The eight bytes from Frame 1 and 56 bytes from Frame 2 make the 64 byte ICMP packet that returns in the echo reply.

Attack mechanism:

The incoming attack is an IP packet with the “more fragments” bit set. The IP packet carries an ICMP echo request. Fragmentation on such a small packet is an obvious sign of packet crafting. The fragmentation in the IP packet is most likely used to confuse a simple perimeter packet filtering device and to attempt to bypass an IDS sensor (though it is not quite small enough). Simple packet filtering devices do not always reassemble fragmented traffic correctly, so the attacker is hoping that the use of fragmentation will let the packet pass through the perimeter. The fragmented packets will also fool a simple IDS that does not properly reassemble the packet. The attacker made nothing but problems since the IDS that sensed the attack was not monitoring for individual echo replies. A simple ping without fragmentation would have slipped under the radar.

The fragmentation itself is a bit interesting. The first frame submits 26 bytes of ICMP traffic, but only tells the IP stack that 8 bytes (the ICMP header) are coming in the first packet. This is actually caused by the TCP/IP stack of the remote host. RFC 894 states that the minimum size of an IP packet over Ethernet is 46 octets. The additional zero-padded bytes make the ICMP packet meet this minimum length requirement.

The attacker is using ICMP as the embedded protocol for reconnaissance purposes. The ICMP packet contains an echo request. The echo request is sent to the destination host in hopes of receiving an echo reply. If the packet fragmentation handled by the IP layer successfully bypasses a filtering device, the echo request will in fact reach the destination host (if it is alive). If the source host receives an echo reply, it means that the destination host did in fact receive the echo request and that the host is alive.

Correlations:

No previous alerts associated with the source host had been recorded on the IDS logs. A search through the DShield database did not find any previous attacks in their logs associated with the source host.

DShield link:
http://www.dshield.org/warning_explanation.php?fip=195.153.112.21+

The below information is from the RIPE database. The IP comes from a small net block in Great Britain. (See below)

```
inetnum:     195.153.112.0 - 195.153.112.127
netname:     WHSMITH-ONLINE
descr:       WHSMITH ONLINE
country:     GB
admin-c:     CH5539-RIPE
tech-c:      GC4616-RIPE
status:      ASSIGNED PA
notify:      ripe-notify@uk.psi.com
mnt-by:      PSINET-UK-SYSADMIN
changed:     sysadmin@uk.psi.com 20000503
source:      RIPE
```

The fragroute software package does not create the same fingerprint as found in these captured packets. My initial reaction was to research this software.

Some firewalls do pass traffic they cannot handle. This is a terrible practice for products that are supposed to enforce security. The link below states that this can happen. Sinus 0.2.9 was listed as a vulnerable product.

http://archives.neohapsis.com/archives/nfr-wizards/1998/11/0116.html

No CVE correlations were found specific to this attack.

Evidence of active targeting:

This traffic was directed at a single host on our network. Even though there is no previous record of the source host in the IDS logs, I'm going defend that it was a targeted attack.

First of all, there is normally no need for an attacker to use fragmentation when running a ping sweep. It is very rare to see ICMP sweeps using fragmented traffic. This leads me to believe that the attacker knew a firewall was in place on that subnet and suspected an IDS was monitoring the network traffic. This information was likely gathered through reconnaissance from a different source host on previous occasions.

Second of all, the targeted server was a DNS server. DNS servers have an important function and it would be quite serious if one were to be compromised.

Lastly, the fact that the attacking IP address does not show up in previous IDS logs or in a public IDS database (DShield) means that the attacker likely targets its attacks carefully. Since the IP source is from a corporate network and not a home connection, the IP was likely cracked and used as a launch pad for attacks. The fact that the source IP uses crafted packets means that they are in fact an attacker with malicious intent. If the source host was a careless script kiddie, the IP address would likely show up in all kinds of public IDS logs as an active scanner and general annoyance.

Severity:

Criticality: 5
The system is an external DNS system, so it is quite critical.

Lethality: 1
The ICMP echo reply response does not cause any damage to the target system.

System countermeasures: 4
The system is entirely up to date on patches and has minimal services running.

Network countermeasures: 5
The only data allowed through the firewall to the DNS server from the Internet is port 53 DNS traffic and ICMP echo requests. All remote administration of the server is done from inside the network.

Severity = (5 + 1) – (4 + 5) = -3

The severity of this attack is quite low due to the low lethality rating and high countermeasures.

Defensive recommendation:

The defenses against this attack were sufficient. The firewall allowed the ICMP echo request through according to its rule-set. Had the attacker returned to further probe the DNS server, they would have found high network protection and an up-to-date system. A check of appropriate DNS Zone Transfer settings would be necessary as well. Those defenses will be enough to thwart the common hacker.

Multiple choice question:

Take a look at the TTL in the following packet:

```
11:13:25.090000 IP (tos 0x0, ttl 254, id 12755, len 84) 78.119.167.58 >
195.153.112.21: icmp 64: echo reply seq 768
```

How many routing devices has the packet most likely traveled through?

a. None yet
b. 1
c. 2
d. 6

B. – With a maximum TTL of 255 (8 bit), we can assume that this device used the maximum TTL amount and has passed through one routing device since the TTL has been decremented to 254.

Detect 2: Scanning for Anonymity

Log Trace

```
Frame 1:
18:38:50.189864 IP (tos 0x0, ttl 48, id 24716, len 60) 216.226.129.209.4971 >
192.168.0.8.1080: S [tcp sum ok] 2453099773:2453099773(0) win 65535 <mss
1460,nop,wscale 0,nop,nop,timestamp 1106064116 0> (DF)
0x0000      4500 003c 608c 4000 3006 cecb d8e2 81d1    E..<`.@.0.......
0x0010      c0a8 0008 136b 0438 9237 54fd 0000 0000    .....k.8.7T.....
0x0020      a002 ffff bbea 0000 0204 05b4 0103 0300    ................
0x0030      0101 080a 41ed 32f4 0000 0000          ....A.2.....

Frame 2:
18:38:59.653838 IP (tos 0x0, ttl 48, id 26109, len 60) 216.226.129.209.4974 >
192.168.0.8.23: S [tcp sum ok] 4046605445:4046605445(0) win 65535 <mss
1460,nop,wscale 0,nop,nop,timestamp 1106065065 0> (DF)
0x0000      4500 003c 65fd 4000 3006 c95a d8e2 81d1    E..<e.@.0..Z....
0x0010      c0a8 0008 136e 0017 f132 4c85 0000 0000    .....n...2L.....
0x0020      a002 ffff 65d0 0000 0204 05b4 0103 0300    ....e...........
0x0030      0101 080a 41ed 36a9 0000 0000          ....A.6.....

Frame 3:
18:39:04.629103 IP (tos 0x0, ttl 48, id 26679, len 60) 216.226.129.209.4981 >
192.168.0.8.27374: S [tcp sum ok] 2288599591:2288599591(0) win 65535 <mss
1460,nop,wscale 0,nop,nop,timestamp 1106065562 0> (DF)
0x0000      4500 003c 6837 4000 3006 c720 d8e2 81d1    E..<h7@.0.......
0x0010      c0a8 0008 1375 6aee 8869 4227 0000 0000    .....uj..iB'....
0x0020      a002 ffff 6c28 0000 0204 05b4 0103 0300    ....l(..........
0x0030      0101 080a 41ed 389a 0000 0000          ....A.8.....

Frame 4:
18:39:09.624564 IP (tos 0x0, ttl 48, id 27389, len 60) 216.226.129.209.4990 >
192.168.0.8.3128: S [tcp sum ok] 3345313260:3345313260(0) win 65535 <mss
1460,nop,wscale 0,nop,nop,timestamp 1106066062 0> (DF)
0x0000      4500 003c 6afd 4000 3006 c45a d8e2 81d1    E..<j.@.0..Z....
0x0010      c0a8 0008 137e 0c38 c765 6dec 0000 0000    .....~.8.em.....
0x0020      a002 ffff 5e20 0000 0204 05b4 0103 0300    ....^...........
0x0030      0101 080a 41ed 3a8e 0000 0000          ....A.:.....

Frame 5:
18:39:15.346733 IP (tos 0x0, ttl 48, id 28010, len 60) 216.226.129.209.4992 >
192.168.0.8.80: S [tcp sum ok] 2895853204:2895853204(0) win 65535 <mss
1460,nop,wscale 0,nop,nop,timestamp 1106066634 0> (DF)
```

```
0x0000      4500 003c 6d6a 4000 3006 c1ed d8e2 81d1   E..<mj@.0.......
0x0010      c0a8 0008 1380 0050 ac9b 3694 0000 0000   .......P..6.....
0x0020      a002 ffff b9ec 0000 0204 05b4 0103 0300   ................
0x0030      0101 080a 41ed 3cca 0000 0000         ....A.<.....
```

Frame 6:

```
18:39:15.493342 IP (tos 0x0, ttl 48, id 28040, len 52) 216.226.129.209.4992 >
192.168.0.8.80: . [tcp sum ok] 2895853205:2895853205(0) ack 3549719734 win
65535 <nop,nop,timestamp 1106066649 1651123567> (DF)
0x0000      4500 0034 6d88 4000 3006 c1d7 d8e2 81d1   E..4m.@.0.......
0x0010      c0a8 0008 1380 0050 ac9b 3695 d394 6cb6   .......P..6...l.
0x0020      8010 ffff 1d6d 0000 0101 080a 41ed 3cd9   .....m......A.<.
0x0030      626a 256f                       bj%o
```

Frame 7:

```
18:39:15.495310 IP (tos 0x0, ttl 48, id 28041, len 90) 216.226.129.209.4992 >
192.168.0.8.80: P [tcp sum ok] 0:38(38) ack 1 win 65535 <nop,nop,timestamp
1106066649 1651123567> (DF)
0x0000      4500 005a 6d89 4000 3006 c1b0 d8e2 81d1   E..Zm.@.0.......
0x0010      c0a8 0008 1380 0050 ac9b 3695 d394 6cb6   .......P..6...l.
0x0020      8018 ffff 1b68 0000 0101 080a 41ed 3cd9   .....h......A.<.
0x0030      626a 256f 434f 4e4e 4543 5420 3130 2e31
        bj%oCONNECT.10.1
0x0040      3031 2e31 3636 2e31 3133 3a38 3020 4854   01.166.113:80.HT
0x0050      5450 2f31 2e30 0d0a 0d0a              TP/1.0....
```

Frame 8:

```
18:39:15.662391 IP (tos 0x0, ttl 48, id 28058, len 52) 216.226.129.209.4992 >
192.168.0.8.80: . [tcp sum ok] 38:38(0) ack 532 win 65469 <nop,nop,timestamp
1106066666 1651123567> (DF)
0x0000      4500 0034 6d9a 4000 3006 c1c5 d8e2 81d1   E..4m.@.0.......
0x0010      c0a8 0008 1380 0050 ac9b 36bb d394 6ec9   .......P..6...n.
0x0020      8010 ffbd 1b65 0000 0101 080a 41ed 3cea   .....e......A.<.
0x0030      626a 256f                       bj%o
```

Frame 9:

```
18:39:15.664346 IP (tos 0x0, ttl 48, id 28060, len 52) 216.226.129.209.4992 >
192.168.0.8.80: F [tcp sum ok] 38:38(0) ack 533 win 65535 <nop,nop,timestamp
1106066666 1651123567> (DF)
0x0000      4500 0034 6d9c 4000 3006 c1c3 d8e2 81d1   E..4m.@.0.......
0x0010      c0a8 0008 1380 0050 ac9b 36bb d394 6eca   .......P..6...n.
0x0020      8011 ffff 1b21 0000 0101 080a 41ed 3cea   .....!......A.<.
0x0030      626a 256f                       bj%o
```

Frame 10:

```
18:39:15.665582 IP (tos 0x0, ttl 48, id 28059, len 52) 216.226.129.209.4992 >
```

```
192.168.0.8.80: . [tcp sum ok] 38:38(0) ack 533 win 65535 <nop,nop,timestamp
1106066666 1651123567> (DF)
0x0000          4500 0034 6d9b 4000 3006 c1c4 d8e2 81d1    E..4m.@.0.......
0x0010          c0a8 0008 1380 0050 ac9b 36bb d394 6eca    .......P..6...n.
0x0020          8010 ffff 1b22 0000 0101 080a 41ed 3cea    ....."......A.<.
0x0030          626a 256f                          bj%o
```

Source of trace:

This trace was recorded on my home broadband connection. The traffic was directed towards my home server. My server is configured to be entirely publicly viewable through the DSL connection. The server runs OpenBSD 3.2 and has SSH, SMTP, and Web services running viewable to the Internet. Secure IMAP services are also running, but only viewable internally. There is also a wireless segment connected to the internal network.

FYI: The CONNECT string in Frame 7 has been modified in this analysis to use a non-public 10.x.x.x address to protect the IP address of my server. The snip from the Apache access_log has been modified as well.

Detect generated by:

The detect was collected using tcpdump 3.4.0 dumping a binary network capture to disk. The network dump ran for many days collecting traffic directed towards my IP address on the Qwest DSL network.

Command line: `tcpdump -w capture.tcp -s 1514 -i xl0 &`

During the days I was collecting network data, my Apache web logs showed some CONNECT requests that made me believe someone was scanning me for signs of an open proxy. My Apache server responded back to the CONNECT request with code 405: Method not allowed. The following is a snip from my httpd access_log:

```
216.226.129.209 - - [03/Jan/2003:18:39:15 -0600] "CONNECT
10.101.166.113:80 HTTP/1.0" 405 308
```

After running a filter ('host 216.226.129.209') against the tcpdump network capture file, I saw lots of activity from the source host. Most of it was proxy related.

Probability that the address was spoofed:

The probability of spoofing is extremely low.

There are incoming packets with only the SYN flag set (completely spoof-able traffic), but the destination ports were not listening on my server. The only port on my server that responded to a SYN packet was port 80. The result was a full TCP handshake and an established connection with the remote host. This leads me to believe that spoofing did not take place and the source host truly sent the packets.

Second of all, anyone scanning for a proxy is going to be interested in the scan's results. It would not make sense for someone to go through the trouble of spoofing SYN packets and make a sequence number prediction attack just to let the results of their scan disappear. It's also obvious that a sequence prediction attack did not take place since no ACK packets from the source host were sent back to a non-responding port on my server.

Description of attack:

This attack scans for a socks proxy (port 1080), a squid proxy (port 3128), a web server allowing the CONNECT method (port 80), a telnet server (port 23), and the SubSeven 2.1 Trojan (port 27374). The selection of ports makes me believe that the attacker is not centering their attack to a single operating system. For example, the telnet port is most likely going to be found on a Unix/Linux machine and the SubSeven port is going to be found on a Windows machine.

The scan seems to be from a specialized automated tool. The tool searches for services that will offer anonymous protection. Socks, Squid, and web proxies will allow traffic to pass through them and have the ability to mask the true source of network traffic. This is valuable because it protects an attacker's identity. The scans for a telnet server and a SubSeven backdoor make sense for this scan because those services can also offer anonymity. Telnet and SubSeven offer the destination system up for remote control. Those remote control systems can then be used as a launch pad to attack more systems and therefore protect the attacker's true identity.

The IP ID field is sequential in nature. At times there are many gaps in the sequential ID's which leads me to believe that the attacker is scanning multiple hosts at once. Proof that the ID field is sequential comes with Frame 6 and Frame 7. The two packets were both sent within 1/100 of a second and the ID fields are 28040 and 28041 in frames 6 and 7 respectively.

Attack mechanism:

The remote program sends out SYN packets to ports 23, 80, 1080, 3128, and 27374. If the destination host responds with a SYN/ACK on any of the ports, the source host completes a three-way handshake. Once a handshake is completed, the source host uses the established connection to test the destination port for proxy capabilities. Port 23 and 27374 are a bit unique for this scan because they will likely require further authentication if the remote service is running. Default passwords will be the next likely attack on those ports.

One thing that is unique about the remote tool is that it actually tears down the TCP connection in a normal fashion using a FIN flag. Most scanning utilities will send a reset packet to quickly tear down the connection. The remote program actually shows very few signs of packet crafting since normal IP and TCP rules are being followed.

The only listening port that this scan found was port 80. This is the only full three-way handshake and application data captured in this network dump. The attack on port 80 uses the CONNECT command and tries to establish a remote connection through the web server. If the CONNECT method is enabled on a remote server, this proxy attempt may actually work. If the remote web server is running Apache with mod_proxy enabled, the CONNECT command will most definitely be enabled. This is why I think that the port 80 probe was specifically looking for a poorly configured Apache server with mod_proxy enabled.

Correlations:

This attacker showed up on my network many times within my network capture over the period of a few days. Every visit shows use of the same scanning utility and the same results. The access_log from my Apache web server also showed many CONNECT attempts from this IP address. This scan is by no means limited to my network. These are very common ports that are checked in most common port scanning utilities.

Surprisingly, a quick check of the Dshield database does not show any alerts from this IP address:
http://www.dshield.org/warning_explanation.php?fip=216.226.129.209

Dshield does show a large amount of scanning destined for these ports. The following two links show examples of scans for port 1080 and 3128.
http://isc.sans.org/port_details.html?port=1080
http://isc.sans.org/port_details.html?port=3128

The below 'Whois' information shows that the source IP belongs to a web hosting company:

```
OrgName:    CompleteWeb.Net
OrgID:      CWBN

NetRange:   216.226.128.0 - 216.226.159.255
CIDR:       216.226.128.0/19
NetName:    COMPLETEWEB
NetHandle:  NET-216-226-128-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: DNS1.COMPLETEWEB.NET
NameServer: DNS2.COMPLETEWEB.NET
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    1999-05-13
Updated:    2001-02-13

TechHandle: GA96-ARIN
TechName:   Anagnostis, Gary
TechPhone:  +1-614-471-2965
TechEmail:  garya@completeweb.net
```

A quick look for Unix-based scanners on PacketStorm's web site shows an abundance of utilities.
http://www.packetstormsecurity.com/UNIX/scanners/

Evidence of active targeting:

This scan did show up multiple times on my network, but I do not believe that it was active targeting. The gaps in the IP ID fields between some of the packets makes me believe that many hosts were being scanned at once; not just me. Also, the attacker had scanned me many times with no positive results. That leads me to believe it was an automated scan left to sift through entire networks without user interaction. No smart attacker would sit at the computer purposely wasting time scanning me many times with no beneficial results. This scan was most definitely not targeted solely at my system.

I also believe the attacker was launching the scan from a cracked box. The IP block belonged to a web hosting company. There are likely a large group of servers on this network that are scanned by attackers for vulnerabilities quite often. This is definitely not an ISP that offers home broadband or dial-up access.

Severity:

Criticality: 2
The system is for personal use to run my mail and web server. It's not extremely critical.

Lethality: 2
There would be no personal, financial, or data damage if the attacker found an open proxy server on my machine. It could, however, cause problems to other systems if the attacker used my server as a launch pad to attack a major network. Not having an open proxy is a “good neighbor” policy.

System countermeasures: 4
The system is entirely up to date on patches. It uses the OpenBSD packet filter to disallow all Internet traffic except SSH, SMTP, and Web. TCP Wrappers are also used.

Network countermeasures: 2
The system has the ability to be protected by a firewall on my network. For the purpose of collecting network data, it was placed in the open network with no network protection.

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

The severity of this attack is quite low.

Defensive recommendation:

The defense against this attack was sufficient. The packet filter was enabled and running on my server and TCP wrappers were also running. Only necessary network services are allowed to communicate with the outside world. My Apache web server was configured properly so that the attack was not effective. Proper configuration is the key if you are running a proxy server that is viewable to the public. Unauthorized proxy servers can be found and abused very quickly.

Multiple choice question:

Which of the following ports is associated with a Squid proxy server?

a.) 1080
b.) 27374
c.) 515
d.) 3128

Answer: D - port 3128

Detect 3: SYN/ACK Scan

Log Trace

```
Frame 1:
04:41:32.414488 IP (tos 0x0, ttl 45, id 0, len 48) 195.232.60.24.6000 >
46.5.97.171.1024: S [bad tcp cksum 7dd4 (->75cf)!] 1889749918:1889749918(0)
ack 620587536 win 5840 <mss 1460,nop,nop,sackOK> (DF)bad cksum c61c (-
>be17)!
                         4500 0030 0000 4000 2d06 c61c c3e8 3c18
                         2e05 61ab 1770 0400 70a3 4b9e 24fd 6a10
                         7012 16d0 7dd4 0000 0204 05b4 0101 0402

Frame 2:
04:41:36.174488 IP (tos 0x0, ttl 45, id 0, len 48) 195.232.60.24.6000 >
46.5.97.171.1024: S [bad tcp cksum 7dd4 (->75cf)!] 1889749918:1889749918(0)
ack 620587536 win 5840 <mss 1460,nop,nop,sackOK> (DF)bad cksum c61c (-
>be17)!
                         4500 0030 0000 4000 2d06 c61c c3e8 3c18
                         2e05 61ab 1770 0400 70a3 4b9e 24fd 6a10
                         7012 16d0 7dd4 0000 0204 05b4 0101 0402

Frame 3:
04:41:42.394488 IP (tos 0x0, ttl 45, id 0, len 48) 195.232.60.24.6000 >
46.5.97.171.1024: S [bad tcp cksum 7dd4 (->75cf)!] 1889749918:1889749918(0)
ack 620587536 win 5840 <mss 1460,nop,nop,sackOK> (DF)bad cksum c61c (-
>be17)!
                         4500 0030 0000 4000 2d06 c61c c3e8 3c18
                         2e05 61ab 1770 0400 70a3 4b9e 24fd 6a10
                         7012 16d0 7dd4 0000 0204 05b4 0101 0402

Frame 4:
04:41:54.404488 IP (tos 0x0, ttl 45, id 0, len 48) 195.232.60.24.6000 >
46.5.97.171.1024: S [bad tcp cksum 7dd4 (->75cf)!] 1889749918:1889749918(0)
ack 620587536 win 5840 <mss 1460,nop,nop,sackOK> (DF)bad cksum c61c (-
>be17)!
                         4500 0030 0000 4000 2d06 c61c c3e8 3c18
                         2e05 61ab 1770 0400 70a3 4b9e 24fd 6a10
                         7012 16d0 7dd4 0000 0204 05b4 0101 0402
```

Source of trace:

The trace was downloaded from the Incidents.org web site:

http://www.incidents.org/logs/Raw/2002.5.10

Detect generated by:

The detection was generated by tcpdump v3.7.0 on a RedHat Linux 7.3 system. A filter looking for SYN ACK packets was applied. The only IP address found through the filter was 195.232.60.24.

Command line: tcpdump -r 2002.5.10 -nnvvX 'tcp[13] = 0x12'

I used Snort v1.9.0 to analyze the dump file (applying the same filter) and no alerts were generated

Command line: snort -r 2002.5.10 -c snort.conf 'tcp[13] = 0x12'.

Probability that the address was spoofed:

It's not likely that these packets had a spoofed source address. First of all, it is definitely not a SYN flood since the ACK flag is also set. Second, the scan comes in at a very slow rate. If you look at the timestamps you will see that the remote host sent them using a normal fallback pattern. The second packet came roughly 3 seconds after the first, the next 6 seconds later, and the last 12 seconds later. The incoming packet rate would have to be much higher to be a spoofed Denial of Service attack. Lastly, the source host is likely looking for some kind of response to the packet. This also supports that spoofing was not used.

Even if the source address wasn't spoofed in these packets, SYN ACK packets can a sign that someone has spoofed a SYN packet using your IP address. I also do not believe that these SYN ACK packets are the responses to such a spoofed packet. I believe this because the IP ID field is 0. The likelihood of a 0 value in the ID field is so small that I can safely say the incoming packet was crafted and not the result of a SYN packet spoofed as the destination host.

The clip from the IP header shows the ID 0:
04:41:32.414488 IP (tos 0x0, ttl 45, id 0, len 48)

Description of attack:

The attack appears to be a scanning tool doing reconnaissance. The tool looks for remote computers running the KDE windows manager (kdm) on TCP port 1024. KDM is a service that operates on Unix/Linux machines and runs with root privileges. This makes it an attractive service to locate and exploit. The links below show known exploitable vulnerabilities associated with kdm.

pam_console vulnerability:
http://online.securityfocus.com/bid/1513/discussion/

RedHat 6.0 KDM vulnerability
http://online.securityfocus.com/bid/310/discussion/

If any of the remote systems scanned by the source host respond on port 1024, the source host then believes they are running kdm. After a list of kdm-enabled machines is found, a kdm exploit will likely be launched in the future. The end result of a successful kdm exploit is root privileges.

If KDM isn't the desired scan target, the scan may also be looking for a variety of Trojans or remote utilities. Jade, Latinus, NetSpy, and RAT are all known to run on this port as well. My analysis assumes that the target is in fact kdm because I believe the attacker is using an X port and a kdm port (6000 and 1024 respectively) to appear to be legitimate traffic.

Attack mechanism:

The source host crafts an initial TCP packet with the SYN and ACK flags set. This is the first sign of packet crafting. The source port is 6000 and the destination port is 1024. The attacker used the source port 6000 to try and give the packet an appearance of Unix X traffic and KDE display manager traffic. The specific source port 6000 is another sign of packet crafting.

Since KDM runs on Unix/Linux, it is safe to assume that Unix/Linux systems have a response to SYN ACK packets that will acknowledge an open port. If this were not true, the scan would be pointless. A check of the tcpdump file did not show a response to this packet.

Another possible reason for the SYN ACK flags is that dumb stateless packet filters will allow that traffic through to the internal network. Stateless packet filters will assume that a host behind their firewall sent a SYN packet and will therefore happily let a SYN ACK packet into the network.

The packet's main sign of crafting is the 0 valued IP ID field. It would be so extremely rare to see a true instance of an IP ID 0 that we can consider it an anomaly. If it weren't for the ID 0 and the normal back-down timing of incoming connections, this packet would probably be identified as the result of spoofing.

The window size of 5840 (15d0 hex) gives a hint to the source OS. A look at the ettercap OS fingerprint database shows that Windows and Linux 2.4.x systems mostly commonly use window size 5840. With all of the packet crafting, the remote OS is likely Linux or possibly Windows 2000. Win2k is the only Windows OS in the list capable of raw packets that would allow such packet crafting.

Ettercap database:
http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/~checkout~/ettercap/ettercap/etter.passive.os.fp?rev=HEAD&con

tent-type=text/plain

Correlations:

No other instances of this IP address are found within the same network dump file. This IP address belongs to a dial-up pool in Frankfurt, Germany. This means a correlation to this IP address in unlikely since the IP will change nearly every time the source host dials up to the ISP.

RIPE database output:

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

A correlation between the scan and vulnerability was stated above in the "description of attack" section. The scan looks for destination port 1024. That port belongs to the kdm process that has known remote root exploits.

According to Dshield’s database, port 1024 is not highly scanned:
http://isc.sans.org/port_details.html?port=1024

Evidence of active targeting:

There is no evidence of active targeting. This appears to be a scanning utility that just by chance happened to hit this specific destination IP address. No other hosts in the network were probed by this source.

Severity:

Criticality: 1
The criticality of the remote system is unknown. The tcpdump file contained no clues to the function of the system.

Lethality: 1
A probe of a single port is highly unlikely to cause problems. It is a reconnaissance scan and is not meant to be damaging.

System countermeasures: 3
First of all, the system did not respond to this probe. This means the port was either not open or filtered in some fashion. Second, the vulnerabilities associated with this probe are at least 2 years old. Any recent patching would fix this vulnerability.

Network countermeasures: 1
It appears that the network let this packet through. There is no evidence in the tcpdump file that shows any kind of network protection. I am assuming that no firewall is present.

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

The severity is quite low.

Defensive recommendation:

Placing this network segment behind a stateful firewall would block packets like this. A host-based firewall like ipchains or iptables could be used to limit access as well. If a firewall does allow an exploit to port 1024 through, applying applicable patches would make any known kdm exploit fail. Any of these defensive measures would stop this scan from yielding beneficial results to the attacker.

Multiple choice question:

What is anomalous about this packet header?

```
04:41:54.404488 IP (tos 0x0, ttl 45, id 0, len 48) 195.232.60.24.6000 >
46.5.97.171.1024: S 1889749918:1889749918(0) ack 620587536 win 5840
<mss 1460,nop,nop,sackOK> (DF)
```

a. The timestamp
b. The sequence number
c. The id
d. The window size

C - the ID value of 0 is anomalous.

Feedback Section

The above analysis (*Detect 3: SYN/ACK Scan*) was submitted to intrusions@incidents.org for comment. Below is the body of an email with feedback I received along with my defense.

You can also find this message online:
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00522.html

---

Feedback #1 to SYN/ACK Scan

Online location:
http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00522.html

> > Even if the source address
> > wasn't spoofed in these packets, SYN ACK packets can a sign that
> > someone has spoofed a SYN packet using your IP address.

>What makes You certain that the initiating packet didn't come from
>46.5.97.171 and that this is not a normal response? You do not have the
>full tcpdump files, rather only the things that an unknown snort ruleset
>decided to keep. If one of those rules was sport: 6000; flags: SA to
>indicate successful connects to an X server, You would only get the
>second packet in the TCP three-way handshake..
>   Your defense may now be that there was the four packet retry, and if
>this were a valid connection, that would not happen. This is not
>necessarily true, either. I had that exact effect yesterday. Weird problem.

You're right - my defense is that there was a set of retry packets in the dump. According to your response, you have seen this before...but seeing it once does not make it accepted, normal TCP/IP behavior. I do not believe that 46.5.97.171 sent a SYN packet. SYN ACK scans do work and are effective. The retry pattern in the remote utility is likely designed to make the scan appear to be normal traffic. Note the link below outlining the normal response to a SYN ACK scan:

Section 3.1.3.1.1 SYN/ACK
http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf

> > I also do
> > not believe that these SYN ACK packets are the responses to such a
> > spoofed packet. I believe this because the IP ID field is 0.

>This is not a valid assumption. Please see
>http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00188.html

>You even fingerprint the machine below and decide that it may be Linux.

I just tested and verified that ID 0 does in fact happen in normal traffic on the 2.4.x kernel in both TCP and ICMP packets. ID 0 does not prove or disprove spoofing. The below link shows that ID 0 can be used as a fingerprinting method:

http://www.sys-security.com/archive/bugtraq/ofirarkin2001-03.txt

> > The attack appears to be a scanning tool doing reconnaissance. The
> > tool looks for remote computers running the KDE windows manager (kdm)
> > on TCP port 1024.

>This is also a normal ephemeral port. If this is the first network
>connection the machine makes...

Yes, unfortunately, kdm runs a root level service on an ephemeral port. Not a good idea if you ask me... I realize 1024 would be used in the first outbound connection, but my difference in opinion is based on my belief that this is a SYN ACK scan.

Thanks for the comments and feedback!!!

-Jared

Feedback #2 to SYN/ACK Scan

Online location:
http://cert.uni-stuttgart.de/archive/intrusions/2003/02/msg00010.html

>>I used Snort v1.9.0 to analyze the dump file (applying the same filter) and
\>>no alerts were generated
>>
\>>Command line: snort -r 2002.5.10 -c snort.conf 'tcp[13] = 0x12'.
>
\>What version of the rules were applied? The README file in the directory
\>states that only traffic generating an alert is logged.
>
\>http://www.incidents.org/logs/Raw/README

The rules that were used were current as of the night of January 30th, 2003.
http://www.snort.org/dl/rules/snortrules-stable.tar.gz

\>>Since KDM runs on Unix/Linux, it is safe to assume that Unix/Linux systems
\>>have a response to SYN ACK packets that will acknowledge an open port. If

>>this were not true, the scan would be pointless. A check of the tcpdump
>>file showed that the destination host did not respond to this packet.
>
>Is there any evidence to support this?

Yes, Ofir Arkin has an excellent paper on scanning techniques:
http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf
"3.1.3.1.1 SYN/ACK
This scan intentionally disregards the TCP three-way handshake. We send a SYN/ACK packet, which is step two in the TCP three-way handshake, while there is no SYN packet sent for step one.

Sending SYN/ACK packet to a closed port:

Because TCP is stateful, it knows no SYN has been sent, which is the first step in the three-way TCP handshake. TCP figures this packet must be a mistake and sends a RESET to tear down the connection. This is what we wished for - any kind of response to give away the existence of the system and the fact that the probed port is closed. If we send the SYN/ACK to an open port, it will ignore any such packet."

Thanks for the feedback!

-Jared

Feedback #3 to SYN/ACK Scan

Online location:
http://cert.uni-stuttgart.de/archive/intrusions/2003/02/msg00025.html

>> If we send the SYN/ACK to an open port, it will ignore any such packet."
>
>This is incorrect. A RST will be sent. RFC 761 page 34 figure 14 steps 3
>& 4.

As a matter of fact...

(Hping version 2.0.0 rc1 run on kernel 2.4.x with OpenBSD 3.2 target system)

```
#/usr/sbin/hping --syn -p 22 192.168.0.8
HPING 192.168.0.8 (eth1 192.168.0.8): S set, 40 headers + 0 data bytes
len=46 ip=192.168.0.8 flags=SA DF seq=0 ttl=64 id=58450 win=16384 rtt=2.7 ms
len=46 ip=192.168.0.8 flags=SA DF seq=1 ttl=64 id=57265 win=16384 rtt=3.1 ms
len=46 ip=192.168.0.8 flags=SA DF seq=2 ttl=64 id=45880 win=16384 rtt=3.0 ms
```

```
--- 192.168.0.8 hping statistic ---
3 packets tramitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2.7/2.9/3.1 ms

#/usr/sbin/hping --syn --ack -p 22 192.168.0.8
HPING 192.168.0.8 (eth1 192.168.0.8): SA set, 40 headers + 0 data bytes

--- 192.168.0.8 hping statistic ---
3 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Don't put too much trust in the RFC's alone - ultimately it's up to the programmer how the system will react.

-Jared

Part 3 – Analyze This

Executive Summary

The University has requested a security audit of five days worth of network data. There is five days of "Snort" Intrusion Detection System (IDS) data accompanied by port scan data, and out of spec data. The port scan data shows potential attackers on the Internet that are "rattling the door handle" on the front door of the university's network to see what's inside. The out of spec data shows anomalous network traffic that should not happen under normal circumstances. The analyzed data runs from January 18, 2003 up to January 22, 2003.

The university does not appear to have a great deal of security currently in place. The report will outline sub-par network protection along with out of date software. These two shortfalls combined create a perfect attack scenario for hackers. The Intrusion Detection System in place is a good start since it shows that the university is serious about improving security.

An analysis of the data showed that over 200,000 alerts were generated during these five days. Hosts coming into the university network from the Internet generated a majority of the alerts. There were a few alerts created by internal hosts that raised some suspicion. Overall, there is a large amount of malicious traffic entering and leaving this network.

Each of the top alerts is listed within the "prioritized detects" section followed up by the "top talkers" section where the alerts are analyzed according to active hosts. A set of external hosts is then analyzed based on the alerts generated on the IDS. It is highly recommended to contact these ISP's to report network abuse. This is followed up by a link graph that visually displays the relationships between a large number of hosts based on the IDS data.

Finally, defensive recommendations are listed within the report. They include the recommendation of a firewall, vulnerability assessments, software updating policies, and an improved Intrusion Detection System. After analyzing the available data, I feel all of these steps are important and should be highly considered when moving forward in securing the university network.

List of Files Analyzed

Alerts:	*Scans:*	*Out of Spec:*
alert.030118	scan.030118	OOS_Report_2003_01_18_6261
alert.030119	scan.030119	OOS_Report_2003_01_19_19130
alert.030120	scan.030120	OOS_Report_2003_01_20_10420
alert.030121	scan.030121	OOS_Report_2003_01_21_8590
alert.030122	scan.030122	OOS_Report_2003_01_22_27894

These files will be available for a limited amount of time online:
http://www.incidents.org/logs/

For the purpose of proper data correlation and compatibility, all instances of "MY.NET" were replaced with "172.16".

Prioritized Detects

The following records set off the top 10 largest numbers of alerts in the output data. The data was compiled using SnortSnarf perl script from Silicon Defense.

Signature	# Alerts	# Sources	# Dests
High port 65535 tcp – possible Red Worm – traffic	46946	145	144
Russia Dynamo – SANS Flash 28-jul-00	43523	2	2
Watchlist 000220 ILISDNNET-990517	35314	82	93
SMB Name Wildcard	26803	849	919
TFTP – External UDP connection to internal tftp server	15794	7	3
Spp_http_decode: IIS Unicode attack detected	15184	441	631
Incomplete Packet Fragments Discarded	6113	12	14
Spp_http_decode: CGI Null Byte attack detected	2806	37	57
SNMP public access	2752	1	861
EXPLOIT x86 NOOP	1353	40	45

High port 65535 tcp – possible Red Worm – traffic

The Red Worm is actually the Adore Worm. This is a collection of programs and scripts used as a root kit once a flaw in the LPRng, rcp-statd, wu-ftpd, or BIND has been exploited on the remote host. Once infected, a host sends out its ftpusers file, ifconfig information, a list of running processes, command history, the hosts file, and the encrypted password file. The root kit then listens for a specially crafted ICMP packet and then opens a remote backdoor on port 65535. The Adore worm is based on attributes from the Ramen and Lion worms and at least one variant is in the wild as well. Defensive recommendations include proper filtering at the network perimeter.

Adore Worm Analysis
http://www.sans.org/rr/threats/mutation.php

There were thousands of scans for port 65535 incoming to MY.NET from 217.136.73.54. A search for the IP addresses as well as TCP port 65535 were not found in the scan logs. No alerts were found in the OOS Reports either.

```
01/18-06:39:03.408400 [**] High port 65535 tcp - possible Red Worm - traffic [**]
217.136.73.54:4198 -> 172.16.84.151:65535
```

There were two major MY.NET IP addresses sending outbound traffic with a source port of 65535. Both 172.16.84.151 and 172.16.88.193 displayed this traffic. This leads me to believe that at least two hosts on MY.NET were infected with the Adore worm and have the backdoor opened and listening on port 65535.

```
01/18-00:00:34.562841 [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.16.84.151:65535 -> 80.13.214.198:3256

01/18-12:06:23.339182 [**] High port 65535 tcp - possible Red Worm - traffic [**]
172.16.88.193:65535 -> 80.13.100.129:2297
```

Russia Dynamo – SANS Flash 28-jul-00
SANS sent out a news flash back in July of 2000 pointing out a large amount of malicious traffic coming from Russia. The source net blocks were 194.87 and 194.87.6 more specifically. Scans originating from these net blocks had probed nearly the entire Internet by the time this news flash had been released. Traffic was headed back to that IP block at a high rate which probably meant a Trojan had been installed on victim PC's. It was highly recommended to block all inbound and outbound traffic to these networks.

It was extremely hard to find any more information on this rule. Miika Turkia came to the same conclusion about "Russia Dynamo" in a different GCIA practical.
http://www.giac.org/practical/Miika_Turkia_GCIA.html

SANS Flash July 28
http://www.sans.org/y2k/072818.htm
http://archives.neohapsis.com/archives/sans/2000/0068.html

There was a large set of alerts (25,000+) for outbound traffic from MY.NET to 194.87.6.86. This leads me to believe that there was a possible compromise on this host. The above link to neohapsis shows that it is likely that Trojaned computers are responding back to this 194.86 netblock. The source port and destination port remain the same throughout the alerts, so this appears to be an established TCP connection. The source IP 172.16.105.204 needs to be checked immediately for possible compromise.

```
01/20-17:52:23.489025 [**] Russia Dynamo - SANS Flash 28-jul-00 [**]
172.16.105.204:4657 -> 194.87.6.86:2244
```

```
01/20-17:52:23.489060 [**] Russia Dynamo - SANS Flash 28-jul-00 [**]
172.16.105.204:4657 -> 194.87.6.86:2244
01/20-17:52:23.772706 [**] Russia Dynamo - SANS Flash 28-jul-00 [**]
172.16.105.204:4657 -> 194.87.6.86:2244
…….
01/21-00:53:34.340738 [**] Russia Dynamo - SANS Flash 28-jul-00 [**]
172.16.105.204:4657 -> 194.87.6.86:2244
```

The 194.86 net block does not appear in the scan logs or the OOS reports.

Watchlist 000220 IL-ISDNNET-990517
There wasn't much information available on what this rule refers to. The attacks all appear to be associated with the 212.179 net block. A check of the registration information listed below in the "External Address Information" section shows that these IP addresses are mostly associated with a cable network overseas in Israel. This leads me to believe that a good deal of scanning activity had been associated with this network. This would be the reason for an IDS alert to be generated when that specific net block was spotted. It is highly recommended to block these net blocks at your perimeter inbound and outbound.

MY.NET did have a large number of alerts either inbound or outbound associated with this net block. Over 10,000 alerts were triggered between 212.179.1.145 and 172.16.113.4 port 1214. A check of port 1214 showed that the traffic was most likely generated by a peer-to-peer file trading network application. This likely means that the alerts generated were false positives. Any application with an active connection to a host in the watch list would generate a large number of alerts even if the traffic is not malicious. This makes these alerts a low priority.

```
01/18-00:00:25.310487 [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.1.145:3638 -> 172.16.113.4:1214
01/18-00:04:03.225758 [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.1.145:3908 -> 172.16.113.4:1214
01/18-00:05:44.833325 [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.1.145:4046 -> 172.16.113.4:1214
```

Port 1214 – Kazaa
http://www.portsdb.org/bin/portsdb.cgi?portnumber=1214&protocol=ANY

Below is the snip of a port 1214 alert in the OOS report from January 21st:

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

01/20-09:59:17.625097 62.46.251.177:3343 -> MY.NET.91.104:1214
TCP TTL:109 TOS:0x0 ID:7350 IpLen:20 DgmLen:48 DF
******** Seq: 0x7041FD  Ack: 0x8E93002F  Win: 0x34  TcpLen: 16
80 0F F0 F2 3E 2E FB B1                          ....>...
```

=+=

Many hits were found in the scan logs to port 1214. Below is a sample:

```
Jan 18 00:41:04 130.85.114.45:2917 -> 196.39.76.13:1214 UDP
Jan 18 00:41:14 130.85.114.45:2917 -> 65.28.41.35:1214 UDP
Jan 18 00:41:19 130.85.114.45:2917 -> 24.74.180.134:1214 UDP
Jan 18 00:41:19 130.85.114.45:2917 -> 24.88.23.243:1214 UDP
Jan 18 00:41:32 130.85.114.45:2917 -> 67.26.38.136:1214 UDP
Jan 18 00:41:33 130.85.114.45:2917 -> 65.173.105.89:1214 UDP
Jan 18 00:41:34 130.85.114.45:2917 -> 24.88.23.243:1214 UDP
Jan 18 00:41:41 130.85.114.45:2917 -> 65.173.105.89:1214 UDP
Jan 18 00:41:51 130.85.114.45:2917 -> 65.28.41.35:1214 UDP
Jan 18 00:41:52 130.85.114.45:2917 -> 65.28.41.35:1214 UDP
```

SMB Name Wildcard
This alert is generated by a query to a Windows PC. A wildcard will return a good deal of machine information if the remote host responds. This kind of activity is a good practice for reconnaissance for an attacker. If Null Sessions are allowed on the remote PC, it will reveal information such as user name, machine name, domain name, workgroup name, hints to remote services running, among other things. This can lead to the harvesting of user names, so password policy enforcement is quite important. Block all NetBIOS and SMB ports at your perimeter to stop this reconnaissance.

This traffic is also popular for new viruses and worms that spread through Windows file sharing. An example of such a virus is Bugbear.

Bugbear@mm Virus:
http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html

The alerts inbound to MY.NET are spread out quite evenly over an extremely large number of hosts. Once again, this traffic is generated mostly be reconnaissance tools and potentially new viruses and worms.

There are no instances of outbound traffic from MY.NET triggering this alert. This means that MY.NET does not have any virus infections that propagate through Windows file sharing.

Scans to UDP port 137 are quite common. Below is a clip from the scan files:

```
Jan 21 05:37:58 212.228.38.120:61331 -> 130.85.134.43:137 UDP
Jan 21 05:37:58 212.228.38.120:61798 -> 130.85.134.44:137 UDP
Jan 21 05:37:58 212.228.38.120:6958 -> 130.85.134.45:137 UDP
Jan 21 05:37:58 212.228.38.120:7814 -> 130.85.134.46:137 UDP
Jan 21 05:37:58 212.228.38.120:20478 -> 130.85.134.48:137 UDP
Jan 21 05:37:58 212.228.38.120:30205 -> 130.85.134.52:137 UDP
```

No instances of port 137 UDP or TCP were found in the OOS logs.

TFTP – External UDP connection to internal tftp server

TFTP is a file transfer protocol that communicates through UDP port 69 and requires no authentication. This makes a TFTP server a good target if left unprotected. It is to be used in environments where authentication is not necessary. Many times, it is used to upload and download configuration files of network hardware on a protected subnet. TFTP can also be used for diskless workstations.

The majority of these alerts on the internal network were confusing. Five internal addresses from MY.NET each triggered roughly 3100 alerts each. Every one of these alerts was directed towards 192.168.0.253. The net block 192.168 is used for private IP space as stated in RFC 1918. This address should not get routed on the Internet. This leads me to two possible conclusions; the MY.NET addresses were either part of a separate internal subnet, or spoofed packets were sent into MY.NET with the source address of 192.168.0.253. I'm guessing it was an internal network because a spoofed private address will lead to no gain for an attacker.

An internal subnet makes the most sense. It is likely that the Snort sensor was just monitoring in a portion of the network where this traffic crossed. Private networks are popular for use in the management of network equipment so that access may be limited. Private IP addresses would be used since there is never a need to make those networks public. The cause of the alerts is likely a diskless workstation boot or the upload/download of configuration files.

There were a very few amount of FTFP scans in the scan files. Below is every single scan recorded to port 69 UDP over the five-day period:

```
Jan 18 02:26:24 130.85.83.146:6257 -> 217.153.10.97:69 UDP
Jan 18 02:26:39 130.85.83.146:6257 -> 217.153.10.97:69 UDP
Jan 18 02:26:40 130.85.83.146:6257 -> 217.153.10.97:69 UDP
Jan 18 07:24:33 130.85.150.213:6257 -> 169.233.6.71:69 UDP
Jan 18 07:24:35 130.85.150.213:6257 -> 169.233.6.71:69 UDP
Jan 19 02:42:46 130.85.150.213:6257 -> 169.233.6.71:69 UDP
Jan 19 02:43:34 130.85.150.213:6257 -> 169.233.6.71:69 UDP
Jan 19 03:03:06 130.85.150.213:6257 -> 169.233.6.71:69 UDP
Jan 19 07:05:56 130.85.70.176:6257 -> 169.233.6.71:69 UDP
Jan 19 10:04:56 130.85.84.147:7736 -> 62.198.98.63:69 UDP
Jan 20 07:53:49 130.85.84.147:7736 -> 64.170.154.92:69 UDP
Jan 20 08:23:39 130.85.84.147:7736 -> 64.170.154.92:69 UDP
Jan 20 08:53:54 130.85.84.147:7736 -> 64.170.154.92:69 UDP
```

No instances of port 69 UDP were found within the OOS logs.

spp_http_decode: IIS Unicode attack detect
The http_decode preprocessor built into Snort generates these alerts. These alerts are not found within any rules files. The logic behind this preprocessor looks for Unicode-encoded strings, checks them for potential malicious activity, and normalizes the string to be sent to the snort rule engine.
http://archives.neohapsis.com/archives/snort/2001-08/0075.html

This is an important preprocessor since Unicode attacks pose a couple of different problems. First, a normal attack can be encoded in Unicode in an attempt to bypass an IDS that does not normalize traffic. Second, there are Unicode-specific attacks related to a flaw in Microsoft IIS. This flaw is the largest cause of Unicode attacks by far. The Nimda worm attacked this flaw. Updating your IIS servers to the latest patches will eliminate this problem.

Unicode vulnerability in IIS:
http://online.securityfocus.com/bid/1806/discussion/

The alerts (15,000+) are all spread out over a large number of hosts. Many are MY.NET hosts triggering an outbound alert. Though the Nimda worm uses the Unicode attack, I doubt it is an infected Nimda host. If it truly were Nimda, a number of other Snort alerts (root.exe, cmd.exe, etc.) would accompany the Unicode attack alert. The total numbers are also too small to be generated by a worm like Nimda. The most likely answer is that the http_decode preprocessor is generating false positives. This is not uncommon with this module. As stated in Pedro Bueno's GCIA practical, this preprocessor causes a large number of false positives, especially when users are browsing the Internet with Netscape.

Pedro Bueno's GCIA Practical:
http://www.giac.org/practical/Pedro_Bueno_GCIA.doc

spp_http_decode False Positives
http://www.geocrawler.com/archives/3/281/2001/3/0/5435119/

The below snip from the scan file shows scans to port 80 that are quite common:

```
Jan 18 01:09:10 65.214.36.150:43313 -> 130.85.99.85:80 SYN 12****S*
RESERVEDBITS
Jan 18 01:34:54 130.85.114.45:2564 -> 207.46.150.20:80 SYN ******S*
Jan 18 01:43:22 65.214.36.150:41507 -> 130.85.99.85:80 SYN 12****S*
RESERVEDBITS
Jan 18 02:33:56 65.214.36.150:46763 -> 130.85.70.113:80 SYN 12****S*
RESERVEDBITS
Jan 18 02:38:27 130.85.91.252:2796 -> 63.175.146.25:80 SYN ******S*
Jan 18 02:43:27 130.85.91.252:2802 -> 63.175.146.25:80 SYN ******S*
Jan 18 02:55:03 24.201.156.194:2292 -> 130.85.150.207:80 SYN ******S*
Jan 18 02:55:04 24.201.156.194:2291 -> 130.85.150.89:80 SYN ******S*
Jan 18 02:55:04 24.201.156.194:2292 -> 130.85.150.207:80 SYN ******S*
Jan 18 02:55:04 24.201.156.194:2258 -> 130.85.150.107:80 SYN ******S*
Jan 18 02:55:05 24.201.156.194:2297 -> 130.85.150.210:80 SYN ******S*
Jan 18 02:55:06 24.201.156.194:2292 -> 130.85.150.207:80 SYN ******S*
Jan 18 02:55:07 24.201.156.194:2297 -> 130.85.150.210:80 SYN ******S*
```

<u>Incomplete Packet Fragments Discarded</u>
Any fragmentation on a network should be analyzed with suspicion. Fragmentation rarely occurs “naturally” in a network environment. It can appear when different hardware or network architectures communicate with each other and have differing MTU’s, but it is not common.

One possibility is that this is an attempted Denial of Service attack. This particular pattern of fragmentation attack does not submit all fragments necessary to reassemble the packet. This attack is a system resource hog for the remote device since memory is allocated to store each incoming fragment for eventual reassembly. Fragmentation attacks like this can potentially tie up the remote device or cause a system crash. The alerts generated do not show enough speed on the incoming packets to be an effective DoS attack.

Another possibility is IDS evasion. Certain old intrusion detection products can easily be fooled by an encoding scheme. Unicode encoding a normal web exploit will easily bypass systems without normalization tools, such as Snort’s spp_http_decode plug-in. The patterns of collected traffic support this as a possible answer.

A final possibility might be a firewall-bypassing attempt. Simple packet filtering devices can be fooled by specific fragmentation. If a first fragmented packet with an allowed inbound port is overwritten a second packet that overlaps the port section of the header, it’s possible to bypass packet filtering. This is not a likely possibility since this alert is triggered by incomplete reassembly. An attacker would like to receive a response from such a crafted packet.

There were three major offenders to this alert; 202.109.80.82, 202.109.80.83, and 61.166.111.117. Both 202.109 addresses accounted for nearly 4000 alerts. They both sent a slow stream of incoming traffic with a source and destination port of 0. The traffic was not sent quickly enough to be an effective Denial of Service attack. The third host sent the exact same pattern of traffic. This makes the intent of the attackers unknown since there isn’t enough information available.

*01/20-23:05:19.356196 [**] Incomplete Packet Fragments Discarded [**]
202.109.80.83:0 -> 172.16.168.152:0
01/20-23:05:31.511706 [**] Incomplete Packet Fragments Discarded [**]
202.109.80.83:0 -> 172.16.168.152:0
01/20-23:05:33.263131 [**] Incomplete Packet Fragments Discarded [**]
202.109.80.83:0 -> 172.16.168.152:0*

01/20-23:05:34.699779 [**] Incomplete Packet Fragments Discarded [**] 202.109.80.83:0 -> 172.16.168.152:0

The below clip from the Out Of Spec file shows a packet with “Don’t Fragment” and “More Fragment” bits set::

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

01/17-17:57:22.765256 195.158.106.87 -> MY.NET.133.1
TCP TTL:111 TOS:0x0 ID:16056 IpLen:20 DgmLen:20 DF MF
Frag Offset: 0x0   Frag Size: 0x0

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

01/18-14:51:29.935853 65.42.92.54 -> MY.NET.153.210
TCP TTL:112 TOS:0x0 ID:3991 IpLen:20 DgmLen:752 DF MF
Frag Offset: 0x0   Frag Size: 0x2DC

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

spp_http_decode: CGI Null Byte attack detected
Null bytes can be used as attacks against online applications. A null byte denotes the end of a character string in languages like C and Java. If an unexpected null byte is sent to a remote application, the flow of control can potentially be modified on the remote server. With a bit of manipulation, this could potentially lead to remote access granted to the attacker. To my knowledge this cannot be used as an evasion technique.

There were nearly 3000 alerts triggered by this rule. Out of those 3000, only 5 did not originate from MY.NET. This rule is known for its high false positives. Without a packet capture, it is difficult to say if an attacker with malicious intent triggered any of the alerts. The following link gives examples of how this rule triggers false positives:

http://archives.neohapsis.com/archives/snort/2000-11/0244.html

SNMP public access
SNMP is a network management protocol that runs over UDP port 161. This protocol allows administrators to remotely view things like machine health and network health from a remote console. The authentication in SNMP is based on a shared “community string”. This string is set to ‘public’ by default on most systems. If the SNMP-enabled device is not configured to use your own organization’s secret string, then attackers easily have the ability to monitor and modify your hardware settings. It is important to block SNMP traffic at your border.

Another possibility would be the recent problems with SNMP implementations:
http://www.cert.org/advisories/CA-2002-03.html

Every single inbound alert was generated by the external host 130.206.173.31. This host scanned 861 hosts in the MY.NET network looking for the default 'public' community string. Most of the scanning went in sequential order, which is an obvious sign of an automated scanning utility. It is unknown if any of the devices responded. It would be a good idea to download an SNMP scanner of our own to check MY.NET for any default configurations using the 'public' string.

```
01/21-07:39:13.621508 [**] SNMP public access [**] 130.206.173.31:1035 ->
172.16.132.15:161
01/21-07:39:15.891622 [**] SNMP public access [**] 130.206.173.31:1035 ->
172.16.132.30:161
01/21-07:39:15.896648 [**] SNMP public access [**] 130.206.173.31:1035 ->
172.16.132.31:161
01/21-07:39:15.919443 [**] SNMP public access [**] 130.206.173.31:1035 ->
172.16.132.32:161
01/21-07:39:15.930956 [**] SNMP public access [**] 130.206.173.31:1035 ->
172.16.132.33:161
```

SNMP Scans are quite common. Public strings are a known default configuration in hardware. Below is a clip of 161 UDP scanning from the scan log files:

```
Jan 18 22:31:45 130.85.70.176:6257 -> 61.24.26.221:161 UDP
Jan 21 07:39:15 130.206.173.31:1035 -> 130.85.132.30:161 UDP
Jan 21 07:39:15 130.206.173.31:1035 -> 130.85.132.31:161 UDP
Jan 21 07:39:15 130.206.173.31:1035 -> 130.85.132.32:161 UDP
```

EXPLOIT x86 NOOP

NOOP is an instruction on Intel x86 systems. It tells the processor that there are no current operations to process, so go on to the next instruction. These NOOP instructions are quite common in buffer overflows. A proper overrun exploit will place exploit machine op codes (called shellcode) preceded by NOOP instructions in the remote buffer and overflow the buffer until the remote instruction pointer register is overwritten. The exploit will set the instruction pointer to reference the memory location where your shellcode is stored. The problem is finding exactly where in the memory of the remote server your shellcode is stored. NOOP's are used to pad the exploit code in memory so that there is an acceptable margin of error. If the instruction pointer lands anywhere within the NOOP pad, it will slide through the NOOP's until it hits your remote shellcode. This means that seeing a large number of NOOP commands is a sign of a remote exploit being launched.

Quite a few alerts were triggered from external hosts. This alert has the tendency to trigger a large number of false positives. Without a network capture, it's difficult to determine if any were true exploits. An example shown below shows a false positive. The NOOP alert is triggered many times and moves source and destination ports. A smart attacker would not randomly pick ports to try and hack since buffer overflow exploits are OS, software version, and architecture specific.

Note the change in source and destination port numbers:

```
01/19-11:47:42.845694 [**] EXPLOIT x86 NOOP [**] 217.128.167.158:1862 ->
172.16.82.239:3836
01/19-11:51:02.741992 [**] EXPLOIT x86 NOOP [**] 217.128.167.158:1941 ->
172.16.82.239:3879
01/19-12:10:18.803493 [**] EXPLOIT x86 NOOP [**] 217.128.167.158:2379 ->
172.16.82.239:1804
```

The host 217.128.167.158 (shown above in the NOOP alert) was found in my scan logs as being an active Windows scanner. There were plenty of outbound SYN scans for ports 135 and 445 originating from this IP address.

```
Jan 19 13:56:01 217.128.167.158:3996 -> 130.85.150.83:135 SYN ******S*
Jan 19 13:56:02 217.128.167.158:4014 -> 130.85.150.83:445 SYN ******S*
Jan 19 13:56:02 217.128.167.158:3997 -> 130.85.150.84:135 SYN ******S*
Jan 19 13:56:01 217.128.167.158:3923 -> 130.85.150.14:135 SYN ******S*
Jan 19 13:56:01 217.128.167.158:3999 -> 130.85.150.86:135 SYN ******S*
Jan 19 13:56:02 217.128.167.158:4020 -> 130.85.150.86:445 SYN ******S*
```

Top 10 Brief Summary

There is a great deal of questionable traffic entering the university network. A great deal of this traffic is malicious in nature. A great deal of the alerts are also false positives. There are a couple of alerts in the top 10 that do raise some concern. I believe that some hosts are possibly compromised. There are a couple of hosts that are sending outbound traffic to 194.87.6.86. That subnet is a known malicious network segment connecting to Trojaned workstations. Both 172.16.84.151 and 172.16.88.193 showed outbound Adore worm traffic. These hosts need to be analyzed immediately.

Top Talkers

1.) 172.16.105.204 – Internal IP Address
This IP address generated 25,179 alerts. All but 9 of these alerts were triggered by the "Russian Dynamo" rule. The intent of this outbound traffic is unknown.

```
01/21-00:53:34.340738 [**] Russia Dynamo - SANS Flash 28-jul-00 [**]
172.16.105.204:4657 -> 194.87.6.86:2244
```

2.) 194.87.6.86 – External IP Address

This host accounted for 18,372 alerts. This host was the destination IP address to over 25,000 alerts generated by Top Talker #1. This IP address is in a net block that triggers the "Russian Dynamo" alert. There was also one SMB Wildcard alert attributed to this host.

01/20-18:03:23.294984 [**] Russia Dynamo - SANS Flash 28-jul-00 [**] 194.87.6.86:2244 -> 172.16.105.204:4657

3.) <u>172.16.84.151 – Internal IP Address</u>
This host appears to be infected with the Red Worm (Adore worm). Every one of its 17,043 alerts originate from port 65535 which leads me to believe it makes the top talker list because it is infected and scanning the Internet.

01/18-00:00:34.562841 [**] High port 65535 tcp - possible Red Worm - traffic [**] 172.16.84.151:65535 -> 80.13.214.198:3256

4.) <u>212.179.1.145 – External IP Address</u>
This host set off 10,378 alerts. There were 10,376 instances of the "Watchlist" alert, 1 TFTP scan connection, and 1 IIS Buffer overflow attempt.

01/18-00:00:25.310487 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.1.145:3638 -> 172.16.113.4:1214

TFTP - External UDP connection to internal tftp server [**] 172.16.111.21901/19-05:42:37.721867 [**] Watchlist 000220 IL-ISDNNET-990517

IDS552/web-iis_IIS ISAPI Overflow ida nosize [**] 163.24.157.801/19-09:54:13.659054 [**] Watchlist 000220 IL-ISDNNET-990517

5.) <u>217.136.73.54 – External IP Address</u>
There were 6,780 alerts triggered by this host. Every single alert was generated as possible Red Worm traffic. This host is very likely to be infected with the Red Worm and is trying to propagate to MY.NET.

01/18-06:39:03.408400 [**] High port 65535 tcp - possible Red Worm - traffic [**] 217.136.73.54:4198 -> 172.16.84.151:65535

6.) <u>212.179.56.252 – External IP Address</u>
This is a Watchlist IP address as well. The host generated 5,891 alerts triggered by the Watchlist rule. It appears to be harmless Kazaa traffic, but it's hard to tell for sure without packet dump information to accompany the alerts.

01/19-05:30:12.283528 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.56.252:61659 -> 172.16.113.4:1214

7.) <u>212.179.83.66 – External IP Address</u>

Yet another IP address in the Watchlist. This host triggered 3,752 alerts over this five-day period. The intent of this network connection is unknown. Not enough data is available to make an assumption one way or the other.

01/21-05:42:36.696895 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.83.66:1235 -> 172.16.88.238:1974

8.) <u>212.179.5.161 – External IP Address</u>
There were 3,486 alerts generated by this host. Only 1 of those alerts appeared to contain malicious intent. This was a fragmentation alert generated by the IDS. All of the other alerts appear to be Kazaa traffic.

Incomplete Packet Fragments Discarded [**] 202.109.80.8301/22-16:01:11.925017 [**] Watchlist 000220 IL-ISDNNET-990517

01/22-15:56:15.324072 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.5.161:16549 -> 172.16.113.4:1214

9.) <u>212.179.107.228 – External IP Address</u>
The Watchlist alert triggered every one of the 3,206 alerts generated by this host. This is yet another IP address associated with the Israeli net block full of malicious traffic. This does appear to be full of false positives, though. It looks like hosts on the MY.NET network connected to a web server running on the remote IP address.

01/21-15:15:53.699202 [**] Watchlist 000220 IL-ISDNNET-990517 [**] 212.179.107.228:80 -> 172.16.153.175:1665

10.) <u>172.16.111.230 – Internal IP Address</u>
This host generated 3,173 alerts from the internal network. Every alert was triggered by a TFTP connection. The remote host was 192.168.0.253, which is a private address. This was likely the result of spoofing or misconfiguration.

01/18-00:04:05.623018 [**] TFTP - External UDP connection to internal tftp server [**] 172.16.111.230:69 -> 192.168.0.253:1297

External Address Information

A very large number of alerts were generated from the 212.179 net block. A "Watchlist" alert logged these hosts. This leads me to believe these net blocks were actively scanning the Internet. It made me interested where in particular these hosts came from.

212.179.100.0, 212.179.83-94

http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=212.179.100.0

```
inetnum:     212.179.100.0 - 212.179.124.255
netname:     CABLES-CONNECTION
mnt-by:      INET-MGR
descr:       CABLES-CUSTOMERS-CONNECTION
country:     IL
admin-c:     MR916-RIPE
tech-c:      ZV140-RIPE
status:      ASSIGNED PA
remarks:     please send ABUSE complains to abuse@bezeqint.net
remarks:     INFRA-AW
notify:      hostmaster@bezeqint.net
changed:     hostmaster@bezeqint.net 20021029
source:      RIPE
```

I looked up the information on this below IP address after seeing all of the SNMP alerts associated with the host. This information below would be useful in reporting network abuse information.

130.206.173.31
http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=130.206.173.31

```
inetnum:     130.206.0.0 - 130.206.255.255
netname:     IRIS
descr:       RedIRIS
descr:       Spanish National R&D Network
descr:       Madrid, Spain
country:     ES
admin-c:     VC63
tech-c:      IRIS1-RIPE
status:      ASSIGNED PI
remarks:     mail spam reports: abuse@rediris.es
remarks:     security incidents: cert@rediris.es
mnt-by:      REDIRIS-NMC
changed:     miguel.sanz@rediris.es 19980429
changed:     iris-nic@rediris.es 19990824
source:      RIPE
```

I picked the below IP address because it seemed to be up to no good. It was either in a busy connection to an Adore infected server or was sending a DoS attack. Either way, the traffic was not normal. The information below shows it was a home DSL user launching their attack against MY.NET.

217.136.73.54
http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=217.136.73.54

```
inetnum:     217.136.0.0 - 217.136.127.255
netname:     BE-SKYNET-ADSL1
descr:       Belgacom Skynet SA/NV
descr:       ADSL Access
country:     BE
admin-c:     SN2068-RIPE
tech-c:      SN2068-RIPE
rev-srv:     ns.ripe.net
rev-srv:     ns1.skynet.be
rev-srv:     ns2.skynet.be
rev-srv:     ns3.skynet.be
rev-srv:     ns4.skynet.be
status:      ASSIGNED PA
mnt-by:      SKYNETBE-MNT
changed:     ripe@skynet.be 20021125
source:      RIPE
```

The below host was checked because it had some strange activity logged. The host was performing null scans using a source and destination port of 0. It was also scanning for the Adore worm.

```
01/20-15:47:37.254289 [**] Null scan! [**] 200.67.25.7:0 -> 172.16.168.98:0
01/20-15:48:08.953001 [**] High port 65535 tcp - possible Red Worm - traffic [**]
200.67.25.7:65535 -> 172.16.168.98:65535
```

200.67.25.7
http://lacnic.net/cgi-bin/lacnic/whois?lg=EN&qr=200.67/16

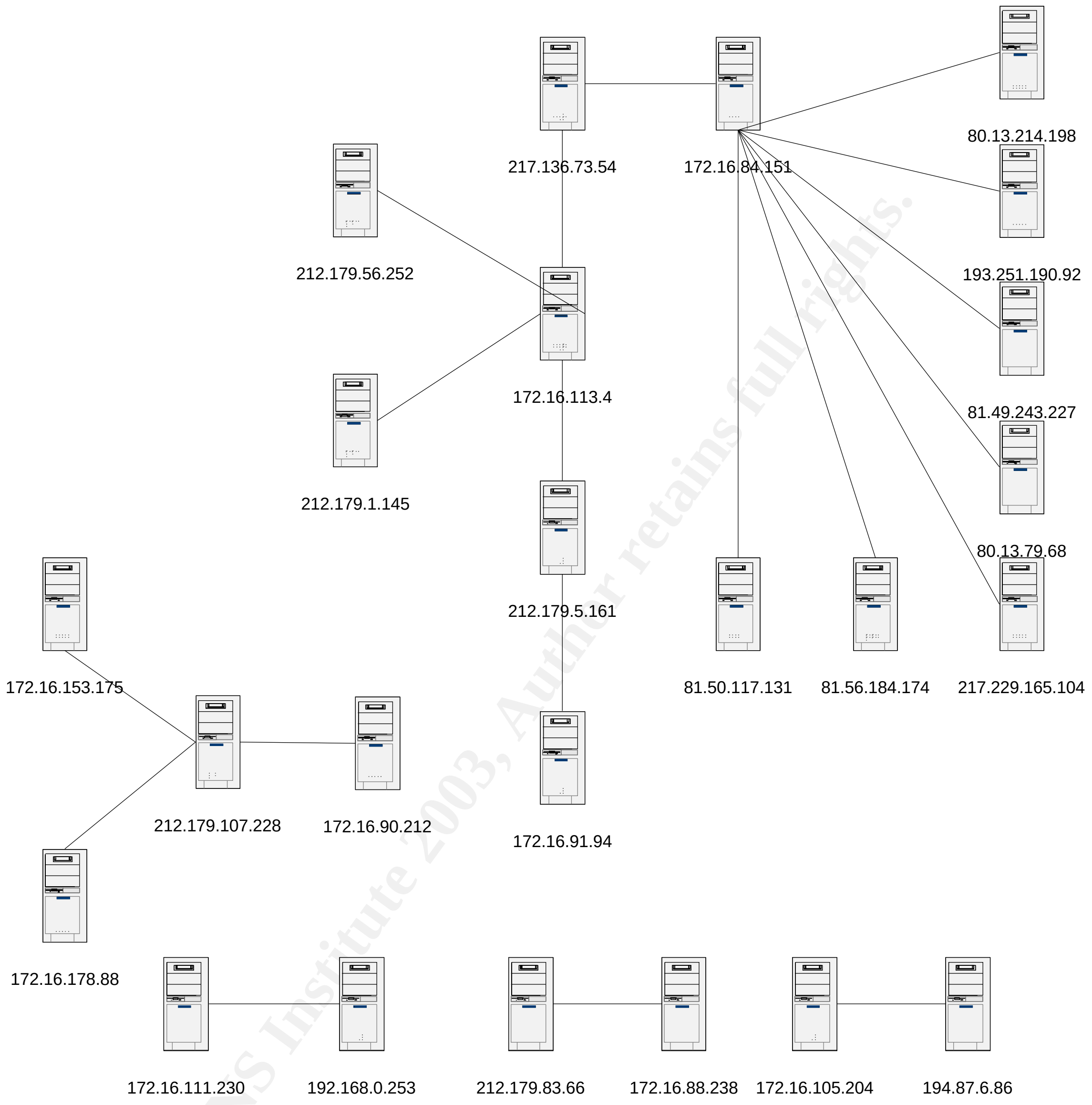
```
inetnum:     200.67/16
status:      reallocated
owner:       Uninet S.A. de C.V.
ownerid:     MX-USCV4-LACNIC
responsible: David Chávez Alba
address:     Periferico Sur, 3190,
address:    01900 - Ciudad de México - DF
country:    MX
phone:      +52 5 54907079 []
owner-c:    DCA
tech-c:     DCA
created:    20010823
changed:    20010823
inetnum-up: 200.64/14
```

The 194.86.6 net block was specifically noted in the July 28th, 2000 SANS news flash. This block was involved in scanning the entire Internet for vulnerabilities. This made me curious about who operated within this net block. As suspected, it is an ISP block allocated to users. I'm surprised how much impact a dial-up ISP had on the Internet.

194.87.6.86
http://www.ripe.net/perl/whois?form_type=simple&full_query_string=&searchtext=194.87.6.86

```
inetnum:    194.87.6.0 - 194.87.6.255
netname:    DEMOS-DOL-DIALUP
descr:      DEMOS-Online Dialup
descr:      Demos-Internet Co.
descr:      Moscow, Russia
country:    RU
admin-c:    DNOC-ORG
tech-c:     DNOC-ORG
status:     ASSIGNED PA
mnt-by:     AS2578-MNT
remarks:    ********************************************
remarks:    Please send abuse reports to abuse@demos.su
remarks:    ********************************************
changed:    rvp@demos.net 20020911
source:     RIPE
```

Link Graph



The above link graph visually shows the relationships between a large number of the hosts within the Snort alert data. As you can see, 172.16.84.151 has a large number of connections. This link graph is a good tool showing that this host is likely infected with the Adore worm and trying to spread itself to other hosts. We can also see how 172.16.113.4 interacts with the multiple 212.179.x.x addresses through the Kazaa file-sharing network. The host 212.179.107.228 is another popular host. This appeared to be a web server that happened to be within a net block that triggered a Snort alert.

Defensive Recommendations

First of all, it's very important to have a firewall between your organization and the Internet. This is a very basic first step to clean up incoming traffic. A DMZ environment should be established for those servers that need to be available for public access (web servers, email servers, dns servers, etc…). This protects your internal network from being attacked if your external servers do get compromised. Also implement rules that limit outbound network traffic. One simple outbound rule could have solved the Adore worm infection from spreading to other external hosts. Ingress and Egress filtering should be implemented at the borders to be a good Internet neighbor.

If a firewall is put in place, explicitly blocking net blocks known for malicious traffic is a good idea. The "Watchlist" and "Russian Dynamo" alerts were both based on known bad net blocks. Actively blocking potentially malicious traffic before it hits you is a very good idea.

Next, the internal network's software needs to be managed. Systems need to be implemented that keep the software patches up to date. Running an internal SUS server from Microsoft will keep the Microsoft systems up to date. Software like RedCarpet can be installed on the Linux servers and desktops to keep their software updated. This is another very basic and effective step to defend your network.

The Intrusion Detection System must be modified to capture packet information. Packet dump information is extremely important when determining if alerts are real attacks or just false positives. A good intrusion detection system will not only show inbound attacks, but outbound traffic to discover if you have in fact been compromised. This could be accomplished with adding more software to the current IDS without requiring more hardware. Using free tools like MySQL and ACID can allow you to capture packet data on alerts without requiring large amounts of hard disk space.

A vulnerability assessment should be performed against the network. There was a large amount of malicious traffic directed at the systems. It is important to assess the situation and get a grip on how vulnerable your network appears to attackers. A SNMP scan is highly recommended based on the SNMP sweeps discussed in the top alerts. There are many misconfigurations that will likely be discovered when performing a vulnerability assessment. Fixing these discovered misconfigurations does nothing but improve security of your organization.

Once those projects are complete, the big steps can be taken. Security is not a one-time job; it is a process. Security must be implemented at all levels through policy. Policy needs to surround things like proper PC use, physical security, software management, and passwords. Proper policy can go a long way in creating an active security culture and keeping your facilities and systems safe.

Analysis Process

There was a great deal of data to sort through when doing all of the analysis. I took a look at the SANS GCIA practical study tips and took the recommendation of concatenating all of the files into one large file. All of my analysis took place on a Linux laptop, so all commands listed below are Linux based.

SANS GCIA Study Guide
http://www.giac.org/gcia_study_guide_v33.pdf

Concatenate all the files:
cat alert > all.alerts*
cat scan > all.scans*
cat OOS_* > all.reports

After that, I took a lot of tips from a previous GCIA practical. Jeff Holland's tips helped me out quite a bit. I would have wasted plenty of time if I had not taken the time to read through his analysis. Appendix A in his practical goes over the data manipulation to get it working in SnortSnarf. The steps are listed below:

Jeff Holland GCIA Practical:
http://www.giac.org/practical/Jeff_Holland_GCIA.doc

1.) MY.NET was replaced with 172.16
vi all.alerts -- :1,$s/MY.NET/172.16/g

2.) A new concatenated file was created with no portscan alerts:
(The > character is found in every non-portscan alert)
cat all.alerts | grep –E ">" > all.alerts-non-portscan

3.) The non-portscan log is now analyzed with SnortSnarf (this takes some time):
./snortsnarf.pl all.alerts-non-portscan

The SnortSnarf output is now available for use. The output correlates all kinds of alerts and IP addresses. I used the top 20 lists to start a list of IP addresses to analyze. Next to each IP I marked the amount of alerts and my initial reaction to what kind of traffic they sent. For every interesting IP address, I ran a 'grep' against the scan and alert files. The analysis went pretty smoothly once everything was organized through SnortSnarf.

References:

Informational sites:

Latin America and Caribbean Registry:
http://lacnic.net/en/index.html

ARIN Whois database:
http://whois.arin.net

Dshield attack correlation:
http://www.dshield.org/warning_explanation.php?fip=195.153.112.21
http://www.dshield.org/warning_explanation.php?fip=216.226.129.209

Snort Port Search
http://www.snort.org/ports.html

Ettercap database:
http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/~checkout~/ettercap/ettercap/etter.passive.os.fp?rev=HEAD&content-type=text/plain

Arkin, Ofir. "Network Scanning Techniques", November 1999
http://www.sys-security.com/archive/papers/Network_Scanning_Techniques.pdf

Arkin, Ofir. "Fingerprinting Linux Kernel 2.4.x", May 2001
http://www.sys-security.com/archive/bugtraq/ofirarkin2001-03.txt

Holland, French, Tan. "SANS GCIA Study Guide"
http://www.giac.org/gcia_study_guide_v33.pdf

Jeff Holland GCIA Practical:
http://www.giac.org/practical/Jeff_Holland_GCIA.doc

Miika Turkia GCIA Practical:
http://www.giac.org/practical/Miika_Turkia_GCIA.html

Pedro Bueno GCIA Practical:
http://www.giac.org/practical/Pedro_Bueno_GCIA.doc

Snort User's Archive – spp_http_decode
http://archives.neohapsis.com/archives/snort/2001-08/0075.html

Incidents.org downloads
http://www.incidents.org/logs/Raw/2002.5.10
http://www.incidents.org/logs/

Horning, Charles. "RFC 894: Standard for the transmission of IP datagrams over Ethernet", April 1984
http://www.faqs.org/rfcs/rfc894.html

Alert Information:

Dell, J. Anthony. "Adore Worm -- Another Mutation", April 6, 2001
http://www.sans.org/rr/threats/mutation.php

“SANS Flash July 28”, July 2000
http://www.sans.org/y2k/072818.htm
http://archives.neohapsis.com/archives/sans/2000/0068.html

SecurityFocus, “Real Server DOS in template.html”, June 2000
http://online.securityfocus.com/bid/1288

“Bugbear@mm Virus”, September 2002
http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear@mm.html

SecurityFocus, “pam_console vulnerability”, July 2000
http://online.securityfocus.com/bid/1513/discussion/

SecurityFocus, “RedHat 6.0 KDM vulnerability”, June 1999
http://online.securityfocus.com/bid/310/discussion/

SecurityFocus, “Unicode vulnerability in IIS”, October 2000
http://online.securityfocus.com/bid/1806/discussion/

Stewart, Joe. “Null Byte False Positives”, November 2000
http://archives.neohapsis.com/archives/snort/2000-11/0244.html

Tools:

SnortSnarf from Silicon Defense
http://www.silicondefense.com/software/snortsnarf/

TCPdump
http://www.tcpdump.org

Snort
http://www.snort.org

TCPMunge
http://www.cyber-defense.org

Hping
http://www.hping.org