Global Information Assurance Certification Paper

GCIA CERTIFICATION PRACTICAL ASSIGNMENT

Version 3.3

ANTHONY NEIL

May 30, 2003

Assignment #1 – Mingsweeper – Network Reconnaissance Tool

Assignment #2 – Network Detects

Summary

This report is culmination of weeks worth of research based on training received from a SANS conference in Toronto, Ontario, Canada. Within this report you will find the research performed on a network reconnaissance tool called Mingsweeper, an analysis of three different type of detects/scans, and an analysis of five days worth of alerts, scans, and out of spec files generated by traffic to and from a university network

Assignment #1 – MingSweeper – Network Reconnaissance Tool

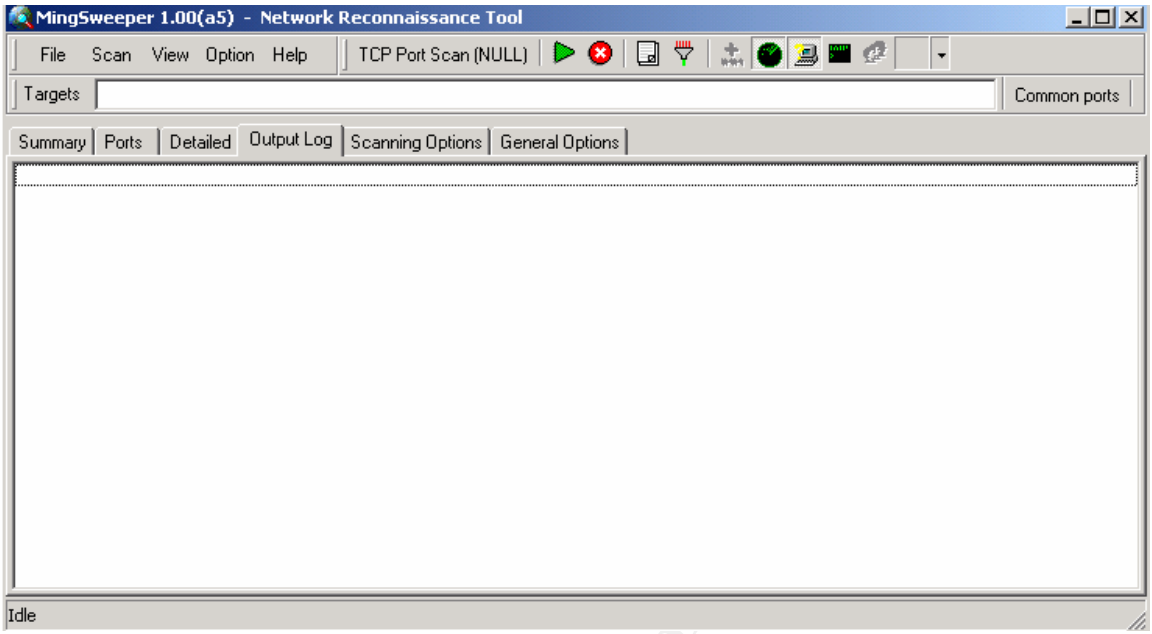
1.1 Mingsweeper

One of the most misunderstood network activities by the uninitiated is network scanning. Ever day there are literally thousands of packets sent by crackers and script kiddies looking for exploitable hosts in someone else's network. Generally speaking the scan should not be the source of concern for network administrators. However, how your network responses to the scans is where your efforts should lie. I believe understanding the reconnaissance tools and techniques used would greatly assist network administrator in protecting their networks. In an effort to help broaden that understanding I have taken the next few pages to describe one such network reconnaissance tool called MingSweeper.

MingSweeper can be downloaded for testing purposes at HTTP://www.hoobie.net/mingsweeper . Included with the download is a readme.txt file that has the installation information.

MingSweeper is a Windows based GUI program that, according to the documentation, works best with Windows 2000, XP, and NT 4 with service pack 4 and above installed. The main functions of MingSweeper are to fingerprint the operating system on the target and scan the target for ports that are open. This program is very easy to configure and operate. I used MingSweeper on an XP system and discovered that I had to turn off my McAfee firewall in order to get any results.

Once installed a simple double click of the MingSweeper icon starts the GUI.



1.2 The Basics

Although there are a plenty of possible options, setting up the basics required to get immediate results with MingSweeper is really quite simple.

1. First pick the target address, network, or networks you wish to scan. In the "Targets" block you can enter a single ip address, block of ips, or any combination of ips and blocks.
   For example.
   - enter 192.168.10.1 to scan a single host
   - enter 10.21.*.* to scan a class B network
   - enter 10.10.0.1-100 to scan a range of host from 10.10.10.1 to 10.10.10.100
   - a comma can be used to separate each of the above entries to perform multiple scan at the same time. Such as:
     10.21.1-35.* ,192.168.0.*, www.google.ca
     This example will request a scan of all hosts from 10.21.1.0 to 10.21.35.255, 192.168.0 class c network, and google.ca
   - enter *.*.*.* and you'll be scan the world. However the author of MingSweeper does not recommend trying this.

2. Next choose the type of scan you wish to perform. From the dropdown menu left of the green arrow you can select one of the following:
   - Ping Sweep

- TCP Port Scan (Syn)
- TCP Port Scan (Fin)
- TCP Port Scan (Xmas)
- TCP Port Scan (Null)
- TCP Filter Scan (Ack)
- UDP Port Scan

3. Finally select the ports you wish to scan from the dropdown menu located to at the right side of the "Targets" block. Pick one of the following options
   - Common ports
   - Services ports
   - All ports
   - Specified ports (this option is configurable)

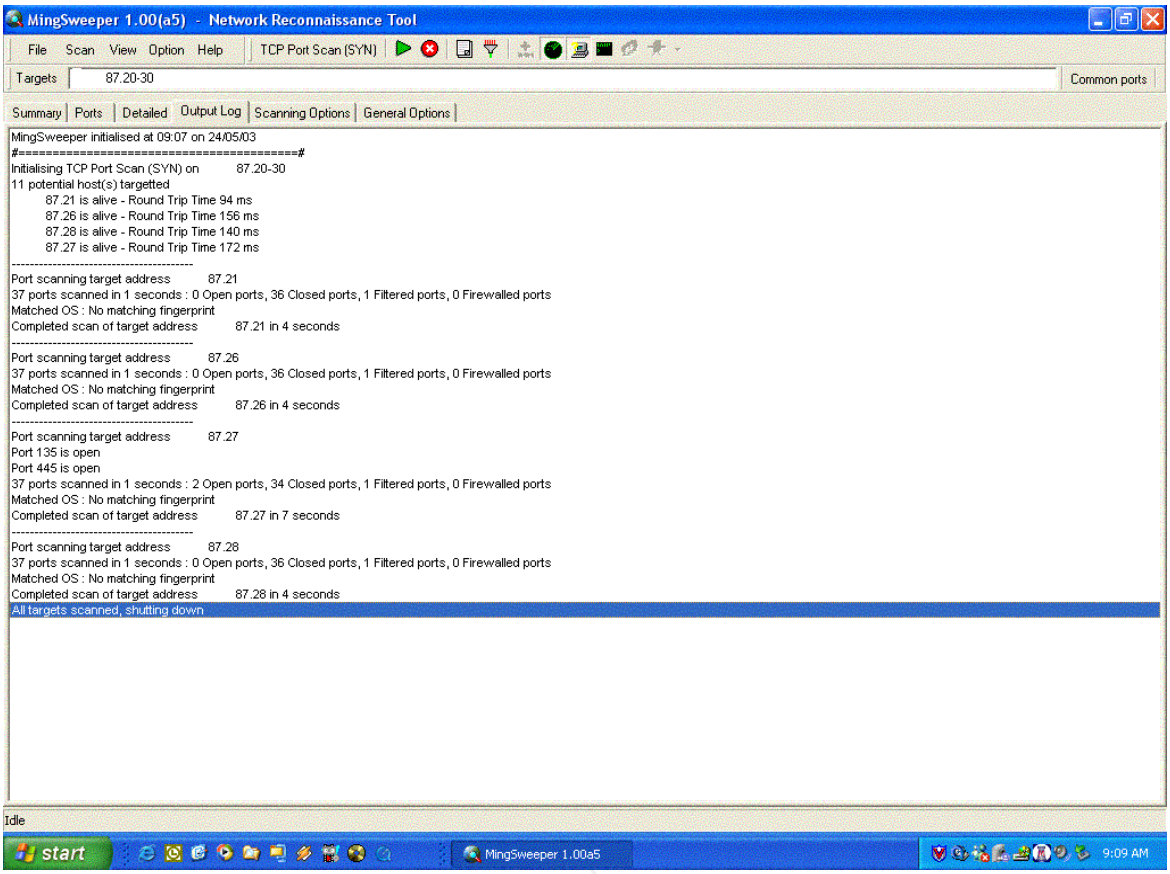
4. Press the Green start arrow and the results will start to appear on the screen if the "Output Log" tab is selected. To stop the scan press the red X.

5. That 's it. The results from your scan will start to print out on the screen in the "Output Log" tab.

Note: In all the screen captures used for testing the first 2 octets of the IP address and all references to the service provider have been removed.

For my test I used a used a Syn scan on hosts .20 to .30 within the network that I chose to scan. As you can see from the results there were on 4 hosts alive at the time and only 1 of those hosts had any common ports open.

A more detailed analysis of these results is available at the end of the more in-depth description of the option that Mingsweeper has to offer.



1.3 More in-depth configuration

In addition to the basics configurations there are two "Option" tabs accessible from either the main screen next to the "Output Log" tab or through the options dropdown menu. The first is the Scanning Options and the second is the General Options

1.3.1 Scanning Options:

This section allows MingSweeper to be configured to the way it performs it search and how it "appears" on the network to IDS. There are three sections under this tab.

1. General Options.
   - "On Banner Timeout Send"
   - "Skip Network and Broadcast Wildcard Addresses". If you select this option MingSweeper will not ping host *.*.*.0 and .255
   - "Automatic Scan Rate Adjustment". This option will vary the rate at which MingSweeper scans the targets that you have selected. This could be used in

an attempt to hide the scans from an IDS by varying the rate at which Mingsweeper requests each scan.

2. Raw Scan and OS Detection Options
   - "Force Scan Source Port" This option we selected allows you to select the source port you want to run the scans on.
   - "Force Source Address" This option allows the user to "spoof" their ip.
   - "Retry Unresponsive Ports" When selected MingSweeper will retry any ports that failed to answer to the scan on the first attempt.
   - "Force Local Interface" will force MingSweeper to use the computer known local interface as its connection to the Internet.
   - The "Decoy List" and "Ping Type" option are not selectable.

3. Performance Options

   The top button is the Simple parameters of the scan performance that the user can set. There are two variable under the performance options. Aggressiveness and Timeouts.

   1. Aggressiveness can be altered to set to one of five rates at which the scan will be performed. The setting ranges from Very low – Maximum Stealth to Merciless – Fast network/Stable targets.
   2. Timeouts can also be set to one of five values. This setting is used to set that time the scan will wait depending on the transmission speed of the network being scanned. The settings range from Very low – low latency/Fast networks to Very high – High latency/Slow networks.

   The second button under the performance options is the advanced setting. The user to fine-tune the way Mingsweeper scan its targets can use this area. There are six variables that can be adjusted under this section.

   1. Packet/Group. The number of packets sent per scan to each target host
   2. Packet Delay. The rate at which the packets are sent
   3. Concurrent Resolvers. The number of host that Mingsweeper will attempt to resolve at one time (concurrently).
   4. Timeout. The length of time Mingsweeper will attempt to connect to a port before timing out
   5. Banner Timeout. The length of time Mingsweeper will attempt to fingerprint the OS
   6. TCP Connect Sockets.

   By adjusting the Packets/Group to a low number and the Packet Delay to a high number an attacker to use MingSweeper to scan a number of hosts while attempting to perform the network scan without detection by the IDS.

1.3.2 General Options

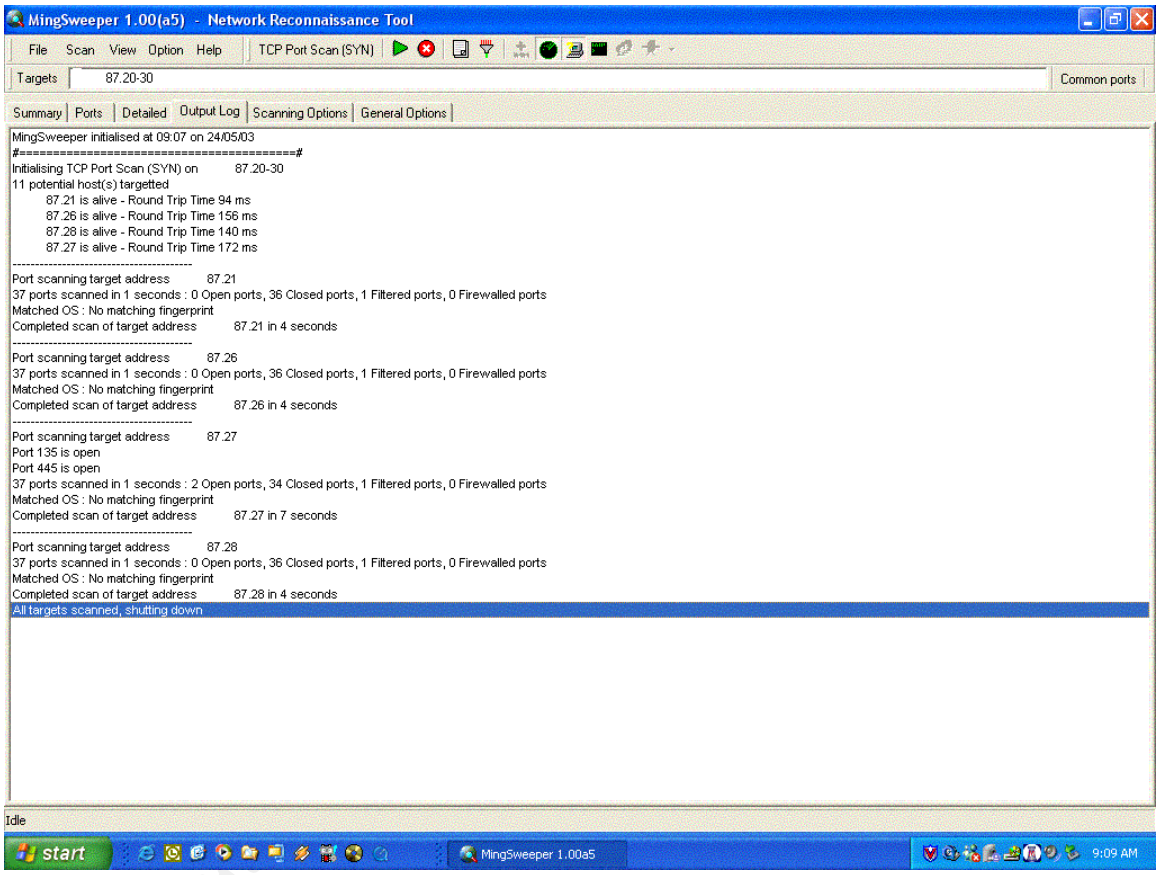
Under this tab the user can configure where MingSweeper will find the files it requires to perform its searches and the location of where to output the results. There are two configurable section contained within this section.

1. File Options
    - "Log Activity" When selected MingSweeper will place a copy of the scan in the logs subdirectory of MingSweeper. It saves the file in the format 110503-0049 (11 May 2003 at 0049Hrs local). The browse button does not work.
    - ""Log Unknown Fingerprints" According to the author all unresolved fingerprints are written to the "Unknown Fingerprints" folder. The problem is the folder doesn't exist and there was nothing in the documentation about the "oversight". I created the directory and performed a couple of scans where Mingsweeper was unable to determine the OS of the target. Mingsweeper did create of file using the IP as the file name noting the fact that it was unable to fingerprint the host. Included in the file are the types of attempts used to fingerprint the host and the responses to those attempts.
    - "OS Fingerprint File". MingSweeper utilizes both NMAP and ICMP styles to fingerprint the IP stack target host. This section has the location of the tcpfp file that MingSweeper uses to perform it's fingerprinting. The Author mentions in the readme file that NMAP fingerprinting files can be imported and used by MingSweeper.
    - "Application ID File" To determine what ports are open on the target host MingSweeper searches for possible applications running. The file listed in this field has a list of applications and ports to search.

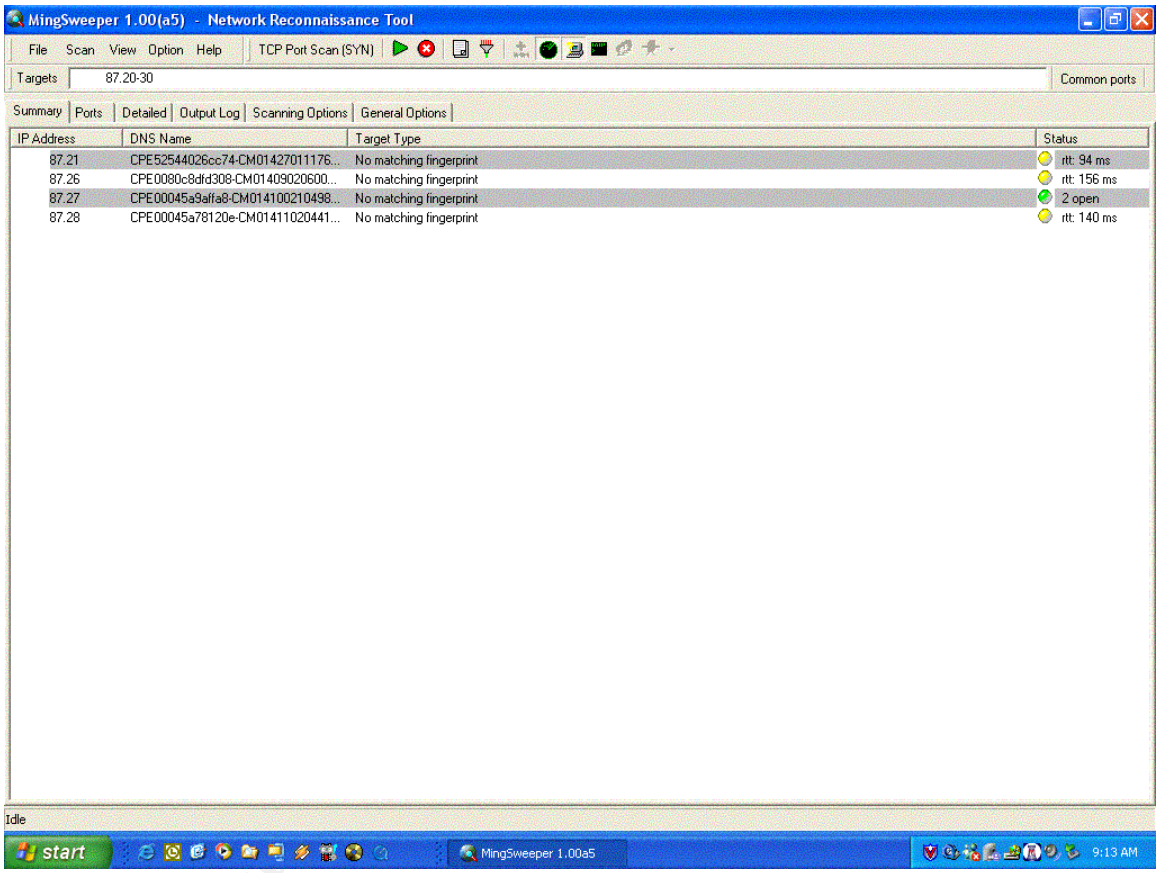
    Note: Both the tcpfp and application files are written in Notepad and can be modified by the user to add and update new search strings.

Results from my test.

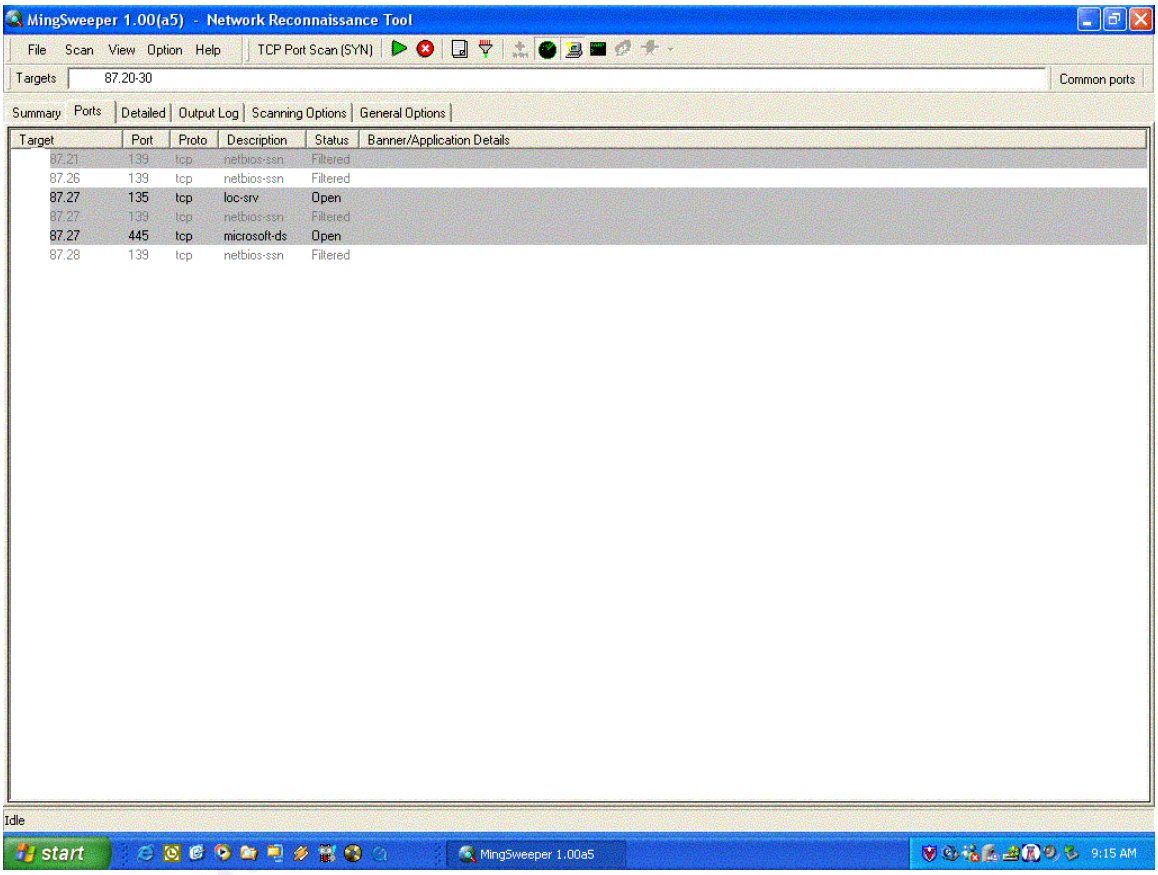
I performed a little SYN scan on 11 possible hosts by starting a scan using the target range ***.***.87.20-30. Below are the results that were produced in approximately 20 seconds. The Output Log is a quick report that the user can have displayed to watch the results in real time as Mingsweeper performs its scan.



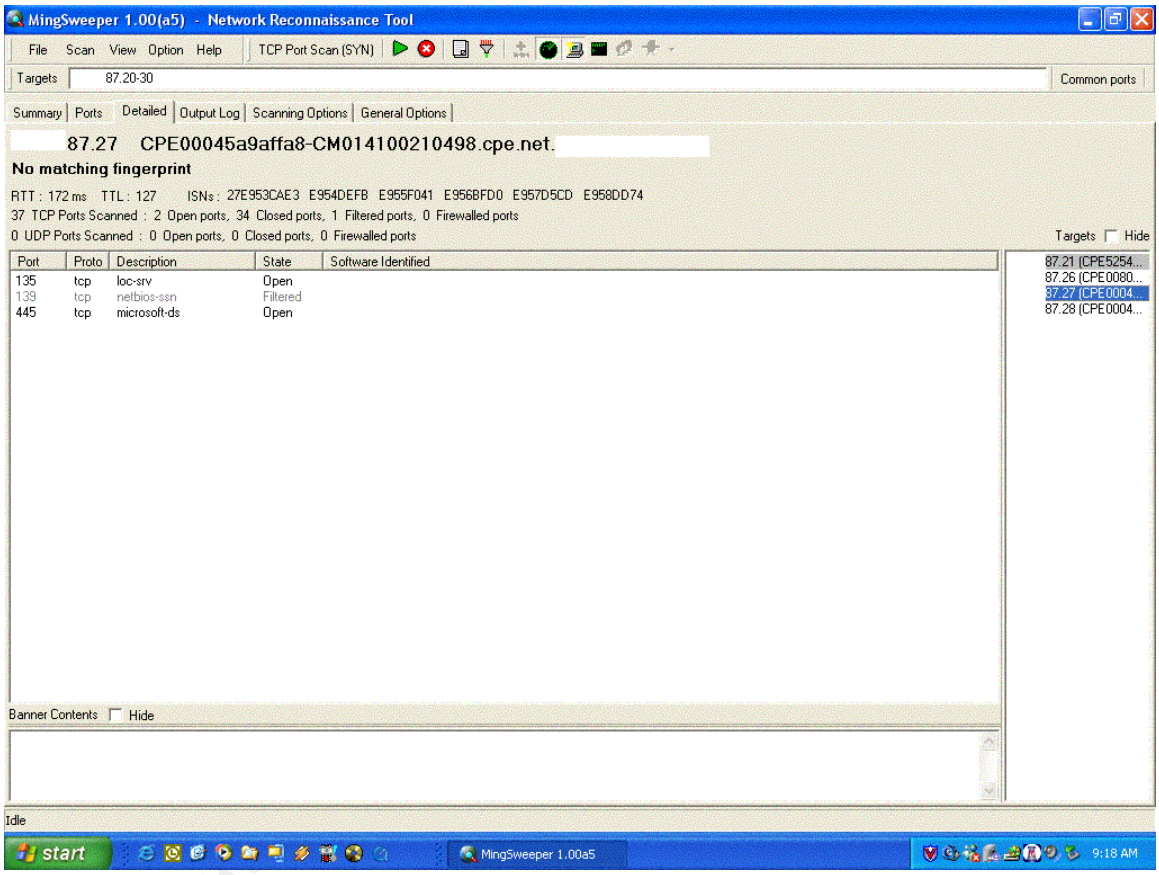
The Summary tab. This tab displays a summary of the hosts that were scanned, the DNS name (including the service provider), the OS type, and either the length of time Mingsweeper took to perform the scan or the number of ports found open.



The Port tab. This displays shows the ports that Mingsweeper either found open, filter, or in some cases closed due to the fact they require password authentication.



The Detailed tab. The first thing to do on this page is to deselect the "Targets Hide" button located on the right side of the screen. The list of all hosts scan will now be displayed. The user can then select the host to be displayed in the main portion of the screen. As you can see from the display a detailed analysis of the ip scanned is displayed.



As I mentioned in the beginning, I believe network administrator and analysts can make their network lives a little easy if they understand what is happening when their networks are scanned. Network administrators could use tools such as Mingsweeper to perform scans on their own networks to see what hosts are responding and what information these hosts are giving up to possible attackers. Mingsweeper can be easily used to scan single hosts within the network to ensure the firewall or latest patch to an OS is properly installed. Scans are a part of everyday networking. Learn to live with them but don't ignore them.

1.4 References:

http://www.hoobie.net/mingsweeper
http://www.securityfocus.com/tools/2141 - discussion
http://clan.cyaccess.com/?menunet&mingsweeper
http://www.insecure.org/nmap/

From within the folder of the MingSweeper download:
readme document
services document
Tcpfp document
application document

The number one source of information I found on this tool was from my online testing.

Assignment #2 - Network Detects

2.1 Detect 1 DF and MF flags set

This detect was posted for comment to intrusions@incidents.org on Mar 26 2003 at 12:17 EST.

```
06:57:23.796507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 2917574263:2917575691(1428)
ack 1019127594 win 17520 (frag 44710:1448@0+)bad cksum ee0 (->22f7)!
0x0000       4500 05bc aea6 6000 6e06 0ee0 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

06:58:12.326507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 46582:1448@0+)bad cksum 790 (->1ba7)!
0x0000       4500 05bc b5f6 6000 6e06 0790 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

06:59:50.366507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 50305:1448@0+)bad cksum f904 (->d1c)!
0x0000       4500 05bc c481 6000 6e06 f904 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

07:07:03.516507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 1638:1448@0+)bad cksum b720 (->cb37)!
0x0000       4500 05bc 0666 6000 6e06 b720 3efd b54b  E....f`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

11:08:15.546507 IP (tos 0x0, ttl 111, len 1468) 62.253.181.75.4137 >
32.245.188.197.80: . [bad tcp cksum a9a7 (->c07)!]
```

```
3916287836:3916289264(1428) ack 4063164739 win 17520 (frag 35249:1448@0+)bad
cksum 756f (->8a87)!
0x0000       4500 05bc 89b1 6000 6f06 756f 3efd b54b  E.....`.o.uo>..K
0x0010       20f5 bcc5 1029 0050 e96d cf5c f22e f943  .....).P.m.\...C
0x0020       5010 4470 a9a7 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P.
```

2.1.1 Source of the Trace

This detect was extracted from the following raw log downloaded off the GIAC assignment site as per the instructions in the current assignment 3.3. Raw/2002.9.17. From the readme file at incidents.org/logs/Raw/README I noted that the checksum errors found in the Windump displays are a result of the sanitization process these raw logs were put through prior to being posted.

For this assignment the following network topology was presumed

Internet >> router >> IDS >> firewall >> host (internal network)

From my posting for comments on incidents.org; Andrew Rucker Jones asked why I presumed the above network configuration?

In response to Andrew's question: I made that presumption to negate the possibility that the target is a host in the DMZ or a network left "unprotected" for testing or research purposes such as a honeypot.

2.1.2 Detect generated by

This detect was generated by Snort IDS version 1.9 using the standard rule set. The following command was used to produce the alerts.

Snort -c /<path>/snort.conf -r /<path>/2002.9.17 -l /<path>/log

After running this command I searched through alert files and found the following alarm:

```
[**] [1:1322:4] BAD TRAFFIC bad frag bits [**]
[Classification: Misc activity] [Priority: 3]
10/17-06:57:23.796507 62.253.181.75 -> 32.245.0.97
TCP TTL:110 TOS:0x0 ID:44710 IpLen:20 DgmLen:1468 DF MF
Frag Offset: 0x0000   Frag Size: 0x05A8
```

The DF and MF bits being set triggered an alert in the BAD TRAFFIC rule set.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC bad frag bits";
fragbits:MD; sid:1322; classtype:misc-activity; rev:4;).
```

After looking through the 62.253.181.75 folder that was produced by the alert, I saw the first four entries destined to the same target ip. The fifth entry was detected four hours later to a different ip within the same class B network. I ran Windump on the raw log file to produce the results noted in Section 2.1 Detect 1 using the following command:

Windump -r /<path>/2002.9.17 -nXv ip and host 62.253.181.75 > <log file name>

I would have added **ip [6] "& 0x60 ! = 60"** to my command line if there had been any non-essential results (i.e.: those without the DF and MF bits set). I choose not to use this filter to start with in order to see all packets from this source. This result may have produced a "bigger picture" of what the attacker may have been attempting to do.

I also ran windump using the ip of all the targets generated by the above command. There were no alarms involving those ip as source.

2.1.3 Possibility the IP was spoofed

It is not likely that this is a spoofed ip. I believe the attacker has used a tool to set the DF and MF bits to 1 in an attempt to elicit a response from a target. If the attacker spoofed their ip, than any responses the attacker was hoping to receive would go to the spoofed address, thus defeating their intended purpose.

ripe.net provided the following information about the attacking ip.

```
inetnum:        62.253.180.0 - 62.253.183.255
netname:        NTL
descr:          NTL Internet
descr:          Renfrew site
country:        GB
admin-c:        NNMC1-RIPE
tech-c:         NNMC1-RIPE
status:         ASSIGNED PA
mnt-by:         AS5089-MNT
changed:        hostmaster@ntli.net 20010330
changed:        hostmaster@ntli.net 20020815
source:         RIPE
route:          62.252.0.0/14
descr:          NTL-UK-IP-BLOCK-3
origin:         AS5089
mnt-by:         AS5089-MNT
changed:        bob.procter@ntli.net 20010205
source:         RIPE
```

2.1.4 Description of the attack.

Below is a copy of packets 2 from Section 2.1 Detect1. Since packets 3 and 4 are identical , except for the times, I choose to show only the first 3 lines of those packet.

```
06:58:12.326507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 46582:1448@0+)bad cksum 790 (->1ba7)!
0x0000       4500 05bc b5f6 6000 6e06 0790 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
```

```
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

06:59:50.366507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 50305:1448@0+)bad cksum f904 (->d1c)!

07:07:03.516507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 1638:1448@0+)bad cksum b720 (->cb37)!
```

First take a look at IP[6]. You will see the Hex value of 60 (binary 0110 0000). This shows that the DF and MF bits are set.

Next, note TCP [4-7] in all three packets have a hex value of **ade6 a677 3cbe a72a.**. Converting the hex to decimal results in the following sequence number **2917574263:2917575691**. This is the same sequence number in the first packet sent. (Packet 1 shown below.)

```
06:57:23.796507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 2917574263:2917575691(1428)
ack 1019127594 win 17520 (frag 44710:1448@0+)bad cksum ee0 (->22f7)!
0x0000       4500 05bc aea6 6000 6e06 0ee0 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...
```

After the 1st packet was sent, with a proper sequence number displayed, the remaining three packets Windump displayed a sequence number of **0:1428**

Third, note the time interval.
Packet 1 **06:57:23.796507**
Packet 2 **06:58:12.326507**
Packet 3 **06:59:50.366507**
Packet 4 **07:07:03.516507**

The times are too far apart for these packets to be a resent failed connection attempt, denial of service, or a continuation of data transfer from an original connection.

As a final point of interest note the dst port of 80. This port is left wide open and firewalls will pass the traffic. The src port is not that important, since tool such as hping are totally configurable.

Packet 5 appears to be the beginning of a new set of attempts to a different target within the same network but using src port 4137.

Snort triggered on the fact that the DF and MF bits were both set. Rfc791 does not allow for the simultaneous setting of these bits. The fact that the sequence numbers in all four packets are identical provides additional evidence that these are crafted packets.

2.1.5 Attack Mechanism

Using the topology that I have presumed for this assignment

Internet >> router >> IDS >> firewall >> host (internal network)

If the target happens to be a properly configured firewall the attacker's packet should be accepted for reassembly or dropped completely. If however there is a misconfiguration in the firewall, the attacker may receive a service message from the firewall that could provide a network configuration "picture" for the attacker.

If the target is an individual host behind a firewall, maybe a server, the response to the packet may be a reset or a denial of the packet with a standard prohibited banner. Depending on the configuration or patched level, the server could give details such as services running and version numbers. This information could allow the attacker to try other known exploits against the target.

In either case, the IDS with a standard rule set will produce an alarm due to fact that the DF and MF bits both set. This attack appears to be a simple reconnaissance with the expressed purpose of being rejected or denied. The attacker would then analysis whatever data they received in response to their crafted packet.

As mentioned earlier hping is a tool totally capable of scanning or mapping a network. Setting the DF and MF flags is simply accomplished by using the following arguments:

-x more fragments
-y dont fragment

I believe this is a scan of a network. The attacker appears to be trying to perform a reconnaissance on a single host.

A second possibility that I though was the attacker was attempting to "push" traffic to a host through fragmentation. The first fragment is sent trying to spoof a connection with the ACK bit set. The target holds the fragment waiting for more to arrive. The other fragments then arrive carrying a "harmful" payload and the target reassembles the packets. Thus infecting itself. This could only happen if the firewall does not reassemble the fragments before passing the packets to the host.

2.1.6 Correlation

http://www.geocrawler.com/archives/3/6752/2002/2/50/7755302/

```
Giles Coochey wrote about this detect on 02/06/2002 03:32:15

SUBJECT:  [Snort-sigs] BAD Traffic entries for database
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD TRAFFIC bad frag
bits"; fragbits: MD; sid:1322; classtype:misc-activity; rev:4;)=20
Sid:1322
```

"This traffic could be part of reconnaissance on your network by an intruder. Different IP stacks will respond to traffic in different ways; the intruder may use this information to identify hosts on your network. Some attack scenarios involve out-of-order fragmentation, where the attack omits first and/or last fragments and sends multiple instances of other fragments, these fragments may also overlap - This activity is normally part of a Denial of Service attack.
Ease of Attack:
The intruder needs to be able to craft and insert the bad packets on your network. If the intruder spoofs the source IP of the traffic then they will also need to be able to intercept returned traffic."

2.1.7 Evidence of active targeting

The attacker has targeted a single addressee at a time. By performing a reconnaissance against this target the attacker could be looking to run other known exploits against the target later.

2.1.8 Severity

Criticality: 2

If the target was that of a properly configured firewall, then the packet should have been dropped. If the target was that of an improperly configured firewall, then the attacker may have been able to learn some valuable information about the network depending on how the firewall responded to the packet.

If the target was a host behind the firewall, then the fragmented packet should have been reassembled before being passed to the host. Since the IDS is before the firewall, the analyst will see the packets arriving. The analyst will have to see the traffic behind the firewall before making a more accurate assessment.

Lethality: 1

A successful attack could result in the target inadvertently providing information about itself that could be used for a later more lethal attack.

From my post for comments Andrew Rucker Jones questioned my Lethality level at 3 for a scan. After review the course material I must agree that I gave this scan to high a rating and have since adjusted it accordingly

System counter measures: 3

This will depend above whether not there is anti-virus software installed and it is running.

Network counter measures: 3

Since an alarm was produced, this shows that the network is using Snort with a standard set of rules. I am also counting on a properly configured stateful firewall being used to protect

the network. As stated before, the analyst should be doing some research to ensure that this is the case and the network is secure.

Severity is calculated using the following formula:

Severity = (criticality + Lethality) - (System counter measures + Network Counter measures)

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

From my post for comments Andrew Rucker Jones asked whether or not this scan was severe or not.

Although the severity level is low, system administrators should be constantly aware of any attempts by unwanted host to map their networks or gain unauthorized access.

2.1.9 Defensive recommendations.

A properly configure firewall should not allow any traffic with the DF and MF both set. The firewall should first reassemble fragmented packets for virus checking prior to passing data to host within the network. Providing little or no information to improperly configured or crafted packets must be a priority to safeguarding networks. More importantly ensuring the data being passed through the firewall is "clean" is vital to the safe operation of the network.

2.1.10 Multiple choice question.

Chose the correct answer.

```
06:57:23.796507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 2917574263:2917575691(1428)
ack 1019127594 win 17520 (frag 44710:1448@0+)bad cksum ee0 (->22f7)!
0x0000       4500 05bc aea6 6000 6e06 0ee0 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

06:58:12.326507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 46582:1448@0+)bad cksum 790 (->1ba7)!
0x0000       4500 05bc b5f6 6000 6e06 0790 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

06:59:50.366507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 50305:1448@0+)bad cksum f904 (->d1c)!
0x0000       4500 05bc c481 6000 6e06 f904 3efd b54b  E.....`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
```

```
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

07:07:03.516507 IP (tos 0x0, ttl 110, len 1468) 62.253.181.75.4377 >
32.245.0.97.80: . [bad tcp cksum d213 (->3372)!] 0:1428(1428) ack 1 win 17520
(frag 1638:1448@0+)bad cksum b720 (->cb37)!
0x0000       4500 05bc 0666 6000 6e06 b720 3efd b54b  E....f`.n...>..K
0x0010       20f5 0061 1119 0050 ade6 a677 3cbe a72a  ...a...P...w<..*
0x0020       5010 4470 d213 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P...

11:08:15.546507 IP (tos 0x0, ttl 111, len 1468) 62.253.181.75.4137 >
32.245.188.197.80: . [bad tcp cksum a9a7 (->c07)!]
3916287836:3916289264(1428) ack 4063164739 win 17520 (frag 35249:1448@0+)bad
cksum 756f (->8a87)!
0x0000       4500 05bc 89b1 6000 6f06 756f 3efd b54b  E.....`.o.uo>..K
0x0010       20f5 bcc5 1029 0050 e96d cf5c f22e f943  .....).P.m.\...C
0x0020       5010 4470 a9a7 0000 feff ff69 d28d 66f0  P.Dp.......i..f.
0x0030       5089 9574 feff ff8b 4508 8b8d 50fe ffff  P..t....E...P.
```

These packets are most likely:

A) Legitimate traffic passed by a router.
B) A scan of a class B network.
C) A reconnaissance of a host.
D) An attempt to infect a host with a virus.

Answer: C

2.2 Detect 2 Socks Proxy Scan

```
08:39:07.754488 IP (tos 0x0, ttl 50, id 52810, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 1681 (->f7a)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12596744 0,nop,wscale 0> (DF)bad cksum 87e0
(->80d9)!
0x0000       4500 003c ce4a 4000 3206 87e0 40e4 6b5b  E..<.J@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 1681 0000 0204 05b4 0402 080a  ................
0x0030       00c0 3608 0000 0000 0103 0300            ..6.........

08:39:10.744488 IP (tos 0x0, ttl 50, id 52811, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 1555 (->e4e)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12597044 0,nop,wscale 0> (DF)bad cksum 87df
(->80d8)!
0x0000       4500 003c ce4b 4000 3206 87df 40e4 6b5b  E..<.K@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 1555 0000 0204 05b4 0402 080a  .....U..........
0x0030       00c0 3734 0000 0000 0103 0300            ..74........

08:39:16.744488 IP (tos 0x0, ttl 50, id 52812, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 12fd (->bf6)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12597644 0,nop,wscale 0> (DF)bad cksum 87de
(->80d7)!
0x0000       4500 003c ce4c 4000 3206 87de 40e4 6b5b  E..<.L@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 12fd 0000 0204 05b4 0402 080a  ................
```

```
0x0030       00c0 398c 0000 0000 0103 0300           ..9.........

08:39:17.964488 IP (tos 0x0, ttl 50, id 34377, len 60) 64.228.107.91.48454 >
46.5.31.84.1080: S [bad tcp cksum 3cf2 (->35eb)!] 1170980793:1170980793(0)
win 5840 <mss 1460,sackOK,timestamp 12597744 0,nop,wscale 0> (DF)bad cksum
cfe1 (->c8da)!
0x0000       4500 003c 8649 4000 3206 cfe1 40e4 6b5b  E..<.I@.2...@.k[
0x0010       2e05 1f54 bd46 0438 45cb bfb9 0000 0000  ...T.F.8E.......
0x0020       a002 16d0 3cf2 0000 0204 05b4 0402 080a  ....<...........
0x0030       00c0 39f0 0000 0000 0103 0300           ..9.........

08:39:20.824488 IP (tos 0x0, ttl 50, id 34378, len 60) 64.228.107.91.48454 >
46.5.31.84.1080: S [bad tcp cksum 3bc6 (->34bf)!] 1170980793:1170980793(0)
win 5840 <mss 1460,sackOK,timestamp 12598044 0,nop,wscale 0> (DF)bad cksum
cfe0 (->c8d9)!
0x0000       4500 003c 864a 4000 3206 cfe0 40e4 6b5b  E..<.J@.2...@.k[
0x0010       2e05 1f54 bd46 0438 45cb bfb9 0000 0000  ...T.F.8E.......
0x0020       a002 16d0 3bc6 0000 0204 05b4 0402 080a  ....;...........
0x0030       00c0 3b1c 0000 0000 0103 0300           ..;.........
08:39:26.904488 IP (tos 0x0, ttl 50, id 34379, len 60) 64.228.107.91.48454 >
46.5.31.84.1080: S [bad tcp cksum 396e (->3267)!] 1170980793:1170980793(0)
win 5840 <mss 1460,sackOK,timestamp 12598644 0,nop,wscale 0> (DF)bad cksum
cfdf (->c8d8)!
0x0000       4500 003c 864b 4000 3206 cfdf 40e4 6b5b  E..<.K@.2...@.k[
0x0010       2e05 1f54 bd46 0438 45cb bfb9 0000 0000  ...T.F.8E.......
0x0020       a002 16d0 396e 0000 0204 05b4 0402 080a  ....9n..........
0x0030       00c0 3d74 0000 0000 0103 0300           ..=t........
```

There were 3 pages of data similar to the 6 packets I've shown above. Instead of including the entire packet I choose to include only the first four lines of the Windump output for the next 6 attempted connections to show the repeating pattern.

```
08:39:27.844488 IP (tos 0x0, ttl 50, id 7221, len 60) 64.228.107.91.51747 >
46.5.31.84.1080: S [bad tcp cksum c89e (->c197)!] 1193681389:1193681389(0)
win 5840 <mss 1460,sackOK,timestamp 12598744 0,nop,wscale 0> (DF)bad cksum
39f6 (->32ef)!

08:39:30.794488 IP (tos 0x0, ttl 50, id 7222, len 60) 64.228.107.91.51747 >
46.5.31.84.1080: S [bad tcp cksum c772 (->c06b)!] 1193681389:1193681389(0)
win 5840 <mss 1460,sackOK,timestamp 12599044 0,nop,wscale 0> (DF)bad cksum
39f5 (->32ee)!

08:39:36.884488 IP (tos 0x0, ttl 50, id 7223, len 60) 64.228.107.91.51747 >
46.5.31.84.1080: S [bad tcp cksum c51a (->be13)!] 1193681389:1193681389(0)
win 5840 <mss 1460,sackOK,timestamp 12599644 0,nop,wscale 0> (DF)bad cksum
39f4 (->32ed)!

08:39:37.744488 IP (tos 0x0, ttl 50, id 12577, len 60) 64.228.107.91.55293 >
46.5.31.84.1080: S [bad tcp cksum 6754 (->604d)!] 1203400929:1203400929(0)
win 5840 <mss 1460,sackOK,timestamp 12599744 0,nop,wscale 0> (DF)bad cksum
250a (->1e03)!

08:39:40.744488 IP (tos 0x0, ttl 50, id 12578, len 60) 64.228.107.91.55293 >
```

```
46.5.31.84.1080: S [bad tcp cksum 6628 (->5f21)!] 1203400929:1203400929(0)
win 5840 <mss 1460,sackOK,timestamp 12600044 0,nop,wscale 0> (DF)bad cksum
2509 (->1e02)!

08:39:46.744488 IP (tos 0x0, ttl 50, id 12579, len 60) 64.228.107.91.55293 >
46.5.31.84.1080: S [bad tcp cksum 63d0 (->5cc9)!] 1203400929:1203400929(0)
win 5840 <mss 1460,sackOK,timestamp 12600644 0,nop,wscale 0> (DF)bad cksum
2508 (->1e01)!

Ten hours later the same attacker performed the same proxy scan on a
different target. There were another 3 pages of data similar to what is
included above. Once again, I only included the first three packets to show
the re-occurring pattern.

18:19:41.034488 IP (tos 0x0, ttl 50, id 65370, len 60) 64.228.107.91.60304 >
46.5.95.162.1080: S [bad tcp cksum 39af (->31aa)!] 3565396959:3565396959(0)
win 5840 <mss 1460,sackOK,timestamp 16080005 0,nop,wscale 0> (DF)bad cksum
1780 (->f7b)!

18:19:44.024488 IP (tos 0x0, ttl 50, id 65371, len 60) 64.228.107.91.60304 >
46.5.95.162.1080: S [bad tcp cksum 3883 (->307e)!] 3565396959:3565396959(0)
win 5840 <mss 1460,sackOK,timestamp 16080305 0,nop,wscale 0> (DF)bad cksum
177f (->f7a)!

18:19:50.034488 IP (tos 0x0, ttl 50, id 65372, len 60) 64.228.107.91.60304 >
46.5.95.162.1080: S [bad tcp cksum 362b (->2e26)!] 3565396959:3565396959(0)
win 5840 <mss 1460,sackOK,timestamp 16080905 0,nop,wscale 0> (DF)bad cksum
177e (->f79)!
```

2.2.1 Source of the Trace

This detect was extracted from the raw log file 2002.5.20 downloaded from incidents.org/logs/Raw as per the instructions in the current GIAC 3.3 assignment. From the README file at the same location I noted that the checksum errors found in the Windump displays are a result of the sanitization process for these raw logs to be posted.

For this example the following network topology is being presumed:

Internet >> router >> IDS >> firewall >> host (internal network)

This topology negates the possibility that the target is a host in the DMZ or a network left "unprotected" for testing or research purposes.

2.2.2 Detect generated by

This detect was generated by Snort IDS version 1.9 using the standard rule set. The following command was used to produce the alerts.

Snort -c /<path>/snort.conf -r /<path>/2002.5.20 -l /<path>/log

After running this command I searched through alert files and found the following alarm:

```
[**] [1:615:3] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/20-08:39:07.754488 64.228.107.91:44905 -> 46.5.31.84:1080
TCP TTL:50 TOS:0x0 ID:52810 IpLen:20 DgmLen:60 DF
******S* Seq: 0x4578F842  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 12596744 0 NOP WS: 0
[Xref => url help.undernet.org/proxyscan/]

The destination port of 1080 with the syn flag set triggered this alert.

alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS Proxy
attempt"; flags:S; reference:url,help.undernet.org/proxyscan/;
classtype:attempted-recon; sid:615; rev:3;)
```

I searched the folder produced by snort on the source ip and found that 12 separate alarms had been generated. I ran Windump on the raw log file using the following command to produce the results noted in Section 2.2 Detect 2.

Windump -r /<path>/2002.5.20 -nXv ip and host 64.228.107.91 > <log file name>

2.2.3 Possibility the IP was spoofed

There is a very good chance that this is not a spoofed ip. This is a proxy port scan on a single target ip. The attacker appears to be scanning a couple of hosts within a class B network trying to find an open port 1080. If the attacker establishes a connection, then they would probably use that host to hide their true ip and launch more serious attacks against other targets.

arin.net resolves the attacking ip as follows.

```
Bell Canada BELLCANADA-5 (NET-64-228-0-0-1)
                                  64.228.0.0 - 64.231.255.255
Sympatico SYMP20002-CA (NET-64-228-96-0-1)
                  64.228.96.0 - 64.228.127.255
```

Swhois.net provides a little more detail.

```
Name:    Toronto-ppp221528.sympatico.ca
Address:  64.228.107.91
```

2.2.4 Description of the attack.

In a span of one minute the attacker attempt to connect to a single target ip's port 1080 a total of 18 times using 6 different src ports. If you look at the first three packets you will note a

very distinct pattern in the time interval. After the first packet is sent there is a 3 second delay to the second packet then a 6 second delay to the third packet. Once the third is sent there is a 1 second delay before the scan starts again using a new src port. The regular time intervals show that the attacker is using a script to attempt to connect to the proxy port. The attacker sends three packet with the syn flag set to the target then changes the src port and tries again..

This is a simple scan by an attacker looking to see is the target is running proxy services. A proxy server is commonly used quick and cheap method of providing Internet access for a private networks. Programs such as Wingate provide the ability for even home users to create their own proxy server. If improperly configured the proxy server can become an easy target for most hackers.

2.2.5 Attack Mechanism

The attacker is attempting to connect to a host through proxy port 1080. If the attacker can establish a connection to the proxy port they could launch attacks against other networks using the proxy server's ip.

A simple search for "socks" on the sans.org finds an article that Christopher Misra wrote:

> "Port 1080 is used by the SOCKS networking proxy protocol. It is designed to allow a host outside of a firewall to connect transparently and securely through the firewall. As a consequence, some sites may have port 1080 opened for incoming connections to a system running a socks daemon. One of the more common uses of SOCKS seems to be allowing ICQ traffic to hosts that are behind a firewall. One common package that provides this function is Wingate (wingate.deerfield.com). "

2.2.6 Correlation

Typing a search string like "port 1080" or " proxy scan" into Google.ca or similar search engines will produce a large list of posts and sites related to this scan. One of the more interesting sites I found was Undernet.org. They are seeing so much abuse with port 1080 that they are scanning any users for proxy services they may be running before the use can connect to their site.

> "Due to the overwhelming abuse of misconfigured Wingate, Socks and Proxy servers being exploited daily, the UnderNet network is now checking all users upon connection to any of the UnderNet IRC Servers."

This is a very common scan seen every day on the Internet. There so many proxy scans happening that isc.incidents.org maintains a chart that tracks port 1080 scans.

2.2.7 Evidence of active targeting

The attacker chooses a single target to attempt to connect to dst port 1080. After three connection attempts the attacker retried with a through a new src port. When the attacker returned they tried a new target ip but kept the rest of the characteristics of the sent packets the same as used in the first attack.

2.2.8 Severity

Criticality: 4

This scan is normally run against a host that would require proxy services. I.e. a small company may use a proxy as their single point of presence to the Internet. If the attacker can gain unauthorized access to the proxy, then the attacker could use the company's ip to launch lethal attacks against other host and make it look like the company is the responsible party.

Lethality: 1

If the attacker can connect to a misconfigured proxy then the attacker can mask their ip with that of the targets and launch more serious attacks on other hosts or networks. Other than the bandwidth consumption that the attacker may steal, this network would in all likeliness be left untouched.

System counter measures: 3

Since I am only presuming the network topology I am making a guess as to the need for the proxy server. I would that inbound request for connects to port 1080 would require a password.

Network counter measures: 3

The use of the proxy service would indicate to me that there was not a lot of expense placed into the set up of this network. However since Snort is running with a standard rule set, the system admin would be able to see and deal with any authorized connection to the proxy.

Severity is calculated using the following formula:

Severity = (criticality + Lethality) - (System counter measures + Network Counter measures)

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

This is one of those scans that could quickly elevate in severity if the system administrator is not maintaining their proxy services properly. Sys admin have to constantly aware of all users who are accessing these services.

2.2.9 Defensive recommendations.

First and foremost. If the proxy services are not required ensure that they are turned off. If the proxy services are being used enforce strong password protection for all users. Perform regular checks on the alarms and usage to ensure that only authorized users are connected to the proxy services.

2.2.10 Multiple choice question.

```
08:39:07.754488 IP (tos 0x0, ttl 50, id 52810, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 1681 (->f7a)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12596744 0,nop,wscale 0> (DF)bad cksum 87e0
(->80d9)!
0x0000       4500 003c ce4a 4000 3206 87e0 40e4 6b5b  E..<.J@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 1681 0000 0204 05b4 0402 080a  ................
0x0030       00c0 3608 0000 0000 0103 0300            ..6.........

08:39:10.744488 IP (tos 0x0, ttl 50, id 52811, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 1555 (->e4e)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12597044 0,nop,wscale 0> (DF)bad cksum 87df
(->80d8)!
0x0000       4500 003c ce4b 4000 3206 87df 40e4 6b5b  E..<.K@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 1555 0000 0204 05b4 0402 080a  .....U..........
0x0030       00c0 3734 0000 0000 0103 0300            ..74........

08:39:16.744488 IP (tos 0x0, ttl 50, id 52812, len 60) 64.228.107.91.44905 >
46.5.31.84.1080: S [bad tcp cksum 12fd (->bf6)!] 1165555778:1165555778(0) win
5840 <mss 1460,sackOK,timestamp 12597644 0,nop,wscale 0> (DF)bad cksum 87de
(->80d7)!
0x0000       4500 003c ce4c 4000 3206 87de 40e4 6b5b  E..<.L@.2...@.k[
0x0010       2e05 1f54 af69 0438 4578 f842 0000 0000  ...T.i.8Ex.B....
0x0020       a002 16d0 12fd 0000 0204 05b4 0402 080a  ................
0x0030       00c0 398c 0000 0000 0103 0300            ..9.........
```

Choose the correct answer:

A proxy scan is easily identified by:

1) The repeating time interval of 1-3-6 seconds between each packet.
2) The src port changes after three connection attempts.
3) The dst port of 1080 never changes.
4) The sequence number doesn't change.
5) The total length of the packet is always 60

Answer: 3

2.3 Detect 3 DNS Named Version Attempt

```
21:41:58.294488 IP (tos 0x0, ttl 45, id 8629, len 58) 203.155.237.144.4861 >
46.5.136.248.53: [bad udp cksum f9f9!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 1db (->fbd4)!
0x0000       4500 003a 21b5 0000 2d11 01db cb9b ed90  E..:!...-.......
0x0010       2e05 88f8 12fd 0035 0026 4df3 1234 0080  .......5.&M..4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
```

```
21:48:13.254488 IP (tos 0x0, ttl 45, id 30898, len 58) 203.155.237.144.3735 >
46.5.226.234.53: [bad udp cksum f9f9!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 50eb (->4ae5)!
0x0000       4500 003a 78b2 0000 2d11 50eb cb9b ed90  E..:x...-.P.....
0x0010       2e05 e2ea 0e97 0035 0026 f866 1234 0080  .......5.&.f.4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
22:29:31.314488 IP (tos 0x0, ttl 45, id 39113, len 58) 203.155.237.144.1465 >
46.5.42.61.53: [bad udp cksum f8f8!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum ea82 (->e37b)!
0x0000       4500 003a 98c9 0000 2d11 ea82 cb9b ed90  E..:....-.......
0x0010       2e05 2a3d 05b9 0035 0026 baf3 1234 0080  ..*=...5.&...4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
22:33:36.844488 IP (tos 0x0, ttl 45, id 44473, len 58) 203.155.237.144.3166 >
46.5.189.184.53: [bad udp cksum f9f9!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 4116 (->3b10)!
0x0000       4500 003a adb9 0000 2d11 4116 cb9b ed90  E..:....-.A.....
0x0010       2e05 bdb8 0c5e 0035 0026 1fd2 1234 0080  .....^.5.&...4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
23:14:16.504488 IP (tos 0x0, ttl 45, id 53498, len 58) 203.155.237.144.3315 >
46.5.75.218.53: [bad udp cksum faf7!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 91b2 (->89ad)!
0x0000       4500 003a d0fa 0000 2d11 91b2 cb9b ed90  E..:....-.......
0x0010       2e05 4bda 0cf3 0035 0026 931a 1234 0080  ..K....5.&...4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
02:15:50.944488 IP (tos 0x0, ttl 45, id 31610, len 58) 203.155.237.144.3562 >
46.5.14.249.53: [bad udp cksum faf7!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 2414 (->1c0f)!
0x0000       4500 003a 7b7a 0000 2d11 2414 cb9b ed90  E..:{z..-.$.....
0x0010       2e05 0ef9 0dea 0035 0026 cf04 1234 0080  .......5.&...4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
```

2.3.1 Source of the Trace

This detect was extracted from the raw log file 2002.5.20 downloaded from incidents.org/logs/Raw as per the instructions in the current GIAC 3.3 assignment. From the README file at the same location I noted that the checksum errors found in the Windump displays are a result of the sanitization process for these raw logs to be posted.

For this example the following network topology is being presumed:

Internet >> router >> IDS >> firewall >> Domain Name Server (DNS)

This topology negates the possibility that the target is a host in the DMZ or a network left "unprotected" for testing or research purposes.

2.3.2 Detect generated by

This detect was generated by Snort IDS version 1.9 using the standard rule set. The following command was used to produce the alerts.

Snort -c /<path>/snort.conf -r /<path>/2002.5.20 -l /<path>/log

After running this command I searched through alert files and found the following alarm:

```
[**] [1:1616:3] DNS named version attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
06/19-21:41:58.294488 203.155.237.144:4861 -> 46.5.136.248:53
UDP TTL:45 TOS:0x0 ID:8629 IpLen:20 DgmLen:58
Len: 38
[Xref => arachnids 278][Xref => nessus 10028]
```

A UDP packet destined for port 53 with the words " version" and "bind" in the traffic cause this alarm to trigger.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version attempt";
content:"|07|version"; offset:12; content:"|04|bind"; nocase; offset: 12;
reference:nessus,10028; reference:arachnids,278; classtype:attempted-recon;
sid:1616; rev:3;)
```

I searched the source ip folder produced by snort and then ran Windump on the raw log file using the following command to produce the results noted in Section 2.3 Detect 3.

Windump -r /<path>/2002.5.20 -nXv ip and host 203.155.237.144 > <log file name>

2.3.3 Possibility the IP was spoofed

It is not likely that the attacker is spoofing their ip. The packets are separated by a far bit of time probably in an attempt to avoid detection. Plus the attacker appears to be trying to find out the bind version running on the DNS and would want to see the responses to their request. Spoofing their ip would defeat the purpose of the scan.

Swhois.net/ provides the following info about this ip.

```
Server:  localhost
Address:  127.0.0.1

Name:    l237ppp144.ksc.net.th
Address:  203.155.237.144
```

2.3.4 Description of the attack.

This is part of a reconnaissance leading up to a potential intrusion attempt. An attacker may attempt to determine the BIND version that the target is running in hopes of finding an un-patched version on bind running on the DNS.

2.3.5 Attack Mechanism

The attacker sends a simple inverse DNS query to determine the version of BIND running on the target. If the attacker discovers the version to be vulnerable, the attacker will then attempt to compromise the server.

2.3.6 Correlation

This is a well-known attack. For interest sake alone you can see the amount of DNS traffic report daily at Isc.incidents.org. This chart doesn't differentiate between regular or scan traffic destined to port 53. As of the date this was written there are 31 CVE s related to vulnerabilities in a variety of releases of BIND.

2.3.7 Evidence of targeting

The attacker is specifically attempting to find the BIND version running on any DNS. They have chosen destination port 53 (DNS) with a "version bind" request within there packets.

2.3.8 Severity

Criticality: 5.

The host targeted is the DNS.

Lethality: 2.

This is only a reconnaissance.

System counter measures: 3.

This will be higher if all the proper patches have been installed on the DNS.

Network counter measures: 1

The firewall will allow all dst port 53 traffic due to the required function of the dns. Therefore this kind of traffic is "free flowing".

Severity is calculated using the following formula:

Severity = (criticality + Lethality) - (System counter measures + Network Counter measures)

Severity = (5 + 2) - (3 + 1) = 3

These types of scan are considered more severe due to the services provided by a DNS servers and the fact that they are generally easily accessible on networks.

2.3.9 Defensive recommendations.

Ensuring that the latest patches and upgrades have been applied to the version of BIND running on the DNS will greatly enhance the security of the DNS. If possible, system administrators should deny all packets at the firewall that contain the content "version.bind" within their dst port 53 traffic.

2.3.10 Multiple choice question.

Choose the correct answer.

```
21:41:58.294488 IP (tos 0x0, ttl 45, id 8629, len 58) 203.155.237.144.4861 >
46.5.136.248.53: [bad udp cksum f9f9!]  4660 [b2&3=0x80] TXT CHAOS?
version.bind. (30)bad cksum 1db (->fbd4)!
0x0000       4500 003a 21b5 0000 2d11 01db cb9b ed90  E..:!...-.......
0x0010       2e05 88f8 12fd 0035 0026 4df3 1234 0080  .......5.&M..4..
0x0020       0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0x0030       0462 696e 6400 0010 0003                 .bind.....
```

This packet is:
A) a user is trying to download a text file from TXT CHOAS website
B) an attempt to upload a text file named chaos to the DNS via port 53.
C) an attempt to discover the bind version on the DNS
D) a user is asking the DNS for the ip address of the version.bind website.
Answer: C

3.0 - Analyze This

3.1 - Executive Summary

The following section is an analysis of five days worth of Alerts, Scans, and Out of Spec files downloaded from the GIAC website as per instructions in assignment 3.3. The list of files downloaded can be viewed in section 3.3 – File selection.

Before continuing, I wish to thank Al Williams for all his help. He gave me the direction I required to get the data sorted. From his direction I used the methodology detailed below.

Once the five days worth of files were downloaded they had to be sorted into such a way that correlations between the files could be found. The lack of packet traffic and lack of a network diagram made this task all that much more difficult.

However, through the use of program such as Microsoft Access, SnortSnarf and Perl the data was sorted and what was found is distributed over the next few pages.

3.2 - File Selection

The files analyzed were for the time period March 25th to 29th, 2003. These files were downloaded for the GIAC webpage as per the instructions in assignment 3.3

The files analyzed were as follows:

March 25 - Alert030325.ids	Scans030325	Oos_Mar26-2003
March 26 - Alert030326.ids	Scans030326	Oos_Mar27-2003
March 27 - Alert030327.ids	Scans030327	Oos_Mar28-2003
March 28 - Alert030328.ids	Scans030328	Oos_Mar29-2003
March 29 - Alert030329.ids	Scans030329	Oos_Mar30-2003

3.3 – Methodology

Using Windows I required four programs; ActivePerl 5.6.1.633, SED for Windows, Microsoft Access, and SnortSnarf. I also borrowed two perl script files from Harry Halladay's GIAC practical (0516) and a third parse-oos file from Ricky Smiths practical (0595). I used SED to change the MY.NET and the university Class B address block to 10.10 to make correlation between the alert, scan and out of spec files more visible. I then used the two Perl scripts to change the alert and scan files into comma separated value (csv) files. The csv files were then imported into one Microsoft Access database. Access then provided the means for a number of queries to be run against the database. I then combined the five Oos files and ran the parse-oos perl script to change this new file into a tab-delimited format. Finally I ran SnortSnarf on the tab-delimited file to produce the results shown in section 3.3.4.

3.3.1 Preparing the alert files for Microsoft Access.

The first thing I did was run SED to change the MY.NET entries in the five alert files to 10.10 using the following command.

C:\sed> sed –s/MY.NET/10.10/g *alertfilename* > *sedfilename*

Then run Perl using the perl script shown below to change the five *sedfilename* files into comma separated value files so that could be imported into Access.

C:\Perl> perl5.6.1 c:\perl\bin\script1.pl sedfilename > perlfilename

Perl script # 1 – Converts the **Alert files** into comma-separated values

```
#!/usr/bin/perl
# Convert alert.DDDDDD files to csv

while(<>) {
   next unless m/^\d/;
   next if m/spp_portscan/;

   chomp;
   ($date_time,$alert,$addrs) = split(/\s+\Q[**]\E\s+/);

   ($source, $dest) = ($addrs =~ m/(.*)\s+->\s+(.*)/);
   ($date,$time) = split(/-/,$date_time);
   ($source_ip, $source_port) = split(/:/,$source);
   ($dest_ip, $dest_port) = split(/:/,$dest);

   print "$date,$time,$alert,$source_ip,$source_port,$dest_ip,$dest_port\n";
```

3.3.2 Preparing the scan alerts for Microsoft Access.

If you look in the scan file you will note that there is a lot of activity from one specific class B network. Using SWHOIS.NET I was able to confirm that this class B network belonged to the University. In order to make correlations between the scans and the alerts files I changed the University addresses to 10.10 with SED using the following command.

C:\sed> sed –s/*university.ip*/10.10/g scan*filename* > *sedscanfilename*

Once again run Perl to change the five *sedscanfilename* into comma separated value files using the script shown below.

C:\Perl> perl5.6.1 c:\perl\bin\script1.pl sedscanfilename > perlscanfilename

Perl script # 2 – Converts the **Scan files** into comma separated values

```
#!/usr/bin/perl
# Convert scans.DDDDDD files to csv

%months = ("Mar" => 3);

while (<>)
{
   next unless m/^[A-Z]/;
   chomp;
   ($month,$day,$time,$source,$dir,$dest,$proto,$flags) = split;
```

```
    $month = $months{$month};
    $date = sprintf("%02d/%02d", $month, $day);
    ($src_ip,$src_port) = split(/:/,$source);
    ($dst_ip,$dst_port) = split(/:/,$dest);

    print "$date,$time,$src_ip,$src_port,$dst_ip,$dst_port,$proto,$flags\n";
}
```

3.3.3 Using Microsoft Access.

Once the five alert files were changed into comma separated values I imported them into one combined alert file. I also created one combined scan file from the five scan files. Finally I ran a variety of queries on the combined files to sort the data by volume and alerts to determine the top 10 talkers, top detects, a range of scans, and then the results were used for further research some selected external hosts. This analysis will be mainly concerned with system compromise and potential weakness that could be exploited using attack trending. Defensive recommendations were then suggested.

3.3.4 Preparing the Out of Spec files for SnortSnarf.

The first thing I did was change the MY.NET to 10.10 using the same method as above.

C:\sed> sed –s/MY.NET/10.10/g oosfilename > sedoosfilename

Next I ran the parse-oos file from Rick Smith's (0595) practical using the following command.

C:\perl\bin> parse-oos <*sedoosfilename*>

```
#Perl script: parse-oos.pl
#! /usr/bin/perl
# use strict;
##########
#       Author:  Rick Smith
#         Date:  13 Dec 2002
#      Version:  1.0
#      Purpose:  Parse the OOS log files and convert to tab-delimited
#                for import into Excel spreadsheets.
#  Description:  1. Uses the specified oos log file to creates a parsed-<oos-file>.txt
#                   in the current directory which is ready for use with SnortSnarf.
```

```
#       Usage:  parse-oos.pl <path_to_log>/<oos file>
#   Parameters:  1. The oos file to parse including the full path
## MAIN ##
##Initialize variables.
$logfile = 0;
$resultsfilename = 0;
$linecount = 0;
# Check for null input
if (! $ARGV[0])  {
        die "Usage: parse-oos.pl <path_to_logs>/<oos file>\n\n";
};  #if
## Get the oos file name and path to the log files from command line.
$logfile = @ARGV[0];
open (LOG, $logfile) || warn "Cannot open $logfile : $!";
$resultsfile = ">parsed-". $logfile ."oos.txt";
open RESULTS, $resultsfile || die "Cannot open $resultsfile : $!";

## Parse through the log files.
## Sequentially read in the lines from the log file and print to the RESULTS
## file if the line in the correct format for SnortSnarf
while (<LOG>) {
        $linecount++;  ## increment the line count.
        if (/(\=\+)+$/||/(\w\w\s\w\w\s)+.+$/)
        {  ## throw away the line if a separator line or if hex dump
        }  #if
        elsif (/(\d\d\/\d\d-\d\d:\d\d:\d\d\.\d\d\d\d\d\d\s)(\w.+?\:.+?->\s.+?\s)(.*)/)
        {
                ## put the entry in the correct format
                $line = $1 . " [**] OOS [**]  " . $2 . $3;
                chomp($line);
                ## finalize the previous entry and output the beginning of the entry
                print RESULTS "\n", $line, " ";
        }  #elsif
        else
        {
                chomp($_);
                print RESULTS $_, " ";  ## add the remaining lines of the entry to the same line
        }  #else
};  #while
1;
```

The last thing I did was download and installed SnortSnarf program in the BIN subdirectory of the PERL directory and run the following command.

C:/Perl/bin> snortsnarf –rs <*parsedfilename*> –d <*/path to output directory*>

3.4 - Top Talkers

3.4.1. Top ten Alert.

The first query I ran was on my combined five-day Alert file looking for all source ips and the number of time they appeared in relationship to an alert. The first column is what I consider the top ten talkers. The second column in the list shows the number of times the "talker" generated an alert and the third column shows the alert that was generated.

Source IP	Number of Alerts	Type of Alert
68.49.35.0	15977	10.10.30.3 activity
61.56.247.174	8924	External RPC call
10.10.105.204	6559	Russia Dynamo - SANS Flash 28-jul-00
10.10.222.194	6100	High port 65535 udp - possible Red Worm - traffic
212.179.48.177	5808	Watchlist 000220 IL-ISDNNET-990517
63.148.150.226	5589	External RPC call
194.87.6.230	5405	Russia Dynamo - SANS Flash 28-jul-00
66.95.149.154	4510	High port 65535 udp - possible Red Worm - traffic
10.10.222.122	3945	High port 65535 tcp - possible Red Worm - traffic
128.8.10.18	3610	SUNRPC highport access!

3.4.2 Top ten scanned ports.

I then ran a query on my combined five-day scan file to find the top ten destination ports. I choose to use the destination port as a starting point to look for possible known attacks. It is important to note that these ports could have been targeted from either inbound or outbound hosts. Of interest there were a total of 3,249,153 scans during the five-day period. The top ten list below represent 2,698,278 or 83% of those scans.

Dst port	Total number of times
445	1381324
80	339128
137	292456
6257	178106
53	177928
135	100890
22321	92075

Dst port	Total number of times
1433	59099
2303	42706
7674	34566

3.4.3 Top Out of Specification alerts (Oos)

Over the five day 738 source ips generated a total of 23500 Oos alert. Below are two tables, one showing the top ten source ips and the second showing the top ten destination ips. Approximately 80% of the alerts were generated by many external ips talking to many internal ips with the CWR,ENC, and Syn flags set. Without the packet traffic to analyze there is no real way to confirm what the source ips are trying to do. However, according to Security Focus, both these flags being set is indicative of QUESO fingerprinting. A further 19% of the alerts were generated by peer-to-peer software.

3.4.4. Top ten Source ips

Rank	Total # Alerts	**Source IP**	# Signatures triggered	Destinations involved
rank #1	4007 alerts	**68.54.93.181**	1 signatures	10.10.6.7
rank #2	754 alerts	**212.186.78.246**	1 signatures	10.10.226.206, 10.10.202.50
rank #3	582 alerts	**216.95.201.25**	1 signatures	(9 destination IPs)
rank #4	510 alerts	**81.218.95.86**	1 signatures	10.10.240.62
rank #5	500 alerts	**216.95.201.22**	1 signatures	(8 destination IPs)
rank #6	494 alerts	**148.64.20.178**	1 signatures	10.10.235.202
rank #7	483 alerts	**216.95.201.32**	1 signatures	(11 destination IPs)
rank #8	458 alerts	**216.95.201.29**	1 signatures	(10 destination IPs)
rank #9	440 alerts	**62.75.157.99**	1 signatures	10.10.24.47
		216.95.201.24	1 signatures	(12 destination IPs)

3.4.5 Top ten destination ips.

Rank	Total # Alerts	**Destination IP**	# Signatures triggered	Originating sources
rank #1	4213 alerts	**10.10.6.7**	1 signatures	(41 source IPs)
rank #2	1968 alerts	**10.10.24.22**	1 signatures	(62 source IPs)
rank #3	1680 alerts	**10.10.24.23**	1 signatures	(64 source IPs)
rank #4	1673 alerts	**10.10.24.21**	1 signatures	(71 source IPs)
rank #5	1650 alerts	**10.10.6.40**	1 signatures	(67 source IPs)
rank #6	1616 alerts	**10.10.6.47**	1 signatures	(59 source IPs)
rank #7	1332 alerts	**10.10.24.44**	1 signatures	(106 source IPs)
rank #8	1035 alerts	**10.10.202.50**	1 signatures	(4 source IPs)
rank #9	594 alerts	**10.10.211.106**	1 signatures	(7 source IPs)
rank #10	559 alerts	**10.10.24.47**	1 signatures	(14 source IPs)

3.5 – Detects

I then ran an Access query to find all of the alert types generated and the number of times the alert appeared over the five days. This list was based only on the alert files and does not take into account any of the scan or Out of spec files. The list is meant to show events that may be attacks, intrusion attempts, or signs of any compromised hosts. Over the five days 103,656 total hosts generated a total of 719,162 alerts divided into 52 different categories. Of those, 430 hosts from the University's network generated 35155, or 4.88%, of the alerts. The table below list all alerts that were reported during the five days and is in order from the most alerts generated to the least. Of interest is that the top ten alerts generated 95.4% of the total alerts.

Following the table is a description of the top ten alerts plus of couple of the more interesting ones. Without the actual packets to analyze, most of the information provided below is based upon the presumption that there is no exclusion rules enforced at the border router or the firewall. I am making my recommendations based on the lack packet traffic and the lack of a network diagram.

Type of Alert	Internal network (outbound traffic)	External addresses (inbound traffic)	Internal to Internal	Total Number of Alerts
SMB Name Wildcard	53	456659	58	456770
TCP SRC and DST outside	0	65622	0	65622

Type of Alert	Internal network (outbound traffic)	External addresses (inbound traffic)	Internal to Internal	Total Number of Alerts
network				
CS WEBSERVER - external web traffic	0	49547	0	49547
Watchlist 000220 IL-ISDNNET-990517	0	21911	0	21911
10.10.30.3 activity	0	20064	0	20064
High port 65535 tcp - possible Red Worm - traffic	7872	12110	0	19982
External RPC call	0	14789	0	14789
High port 65535 udp - possible Red Worm - traffic	7534	6178	0	13712
Russia Dynamo - SANS Flash 28-jul-00	6559	5405	0	11964
spp_http_decode: IIS Unicode attack detected	9320	2244	0	11564
SUNRPC highport access!	0	5706	0	5706
10.10.30.4 activity	0	5325	0	5325
TFTP - Internal TCP connection to external tftp server	2985	1998	0	4983
Queso fingerprint	0	3921	0	3921
spp_http_decode: CGI Null Byte attack detected	2787	11	0	2798
Watchlist 000222 NET-NCFC	0	1728	0	1728
CS WEBSERVER - external ftp traffic	0	1413	0	1413
Incomplete Packet Fragments Discarded	52	1336	0	1388
IDS552/web-iis_IIS ISAPI Overflow ida nosize	0	1331	0	1331
Possible trojan server activity	54	718	0	772
Null scan!	0	681	0	681
SNMP public access	0	520	0	520
connect to 515 from outside	0	519	0	519
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	404	0	0	404
NMAP TCP ping!	1	401	0	402
EXPLOIT x86 stealth noop	0	282	0	282
NIMDA - Attempt to execute cmd from campus host	221	0	0	221
EXPLOIT x86 NOOP	0	198	0	198
Notify Brian B. 3.56 tcp	0	109	0	109
Notify Brian B. 3.54 tcp	0	94	0	94
Tiny Fragments - Possible Hostile Activity				90
IRC evil - running XDCC	74	0	0	74

Type of Alert	Internal network (outbound traffic)	External addresses (inbound traffic)	Internal to Internal	Total Number of Alerts
EXPLOIT x86 setuid 0	0	72	0	72
SMB C access				55
EXPLOIT x86 setgid 0	0	49	0	49
FTP passwd attempt				35
TFTP - External TCP connection to internal tftp server	6	6	0	12
Fragmentation Overflow Attack	0	12	0	12
RFB - Possible WinVNC - 010708-1	6	2	0	8
NIMDA - Attempt to execute root from campus host	6	0	0	6
TFTP - Internal UDP connection to external tftp server	2	3	0	5
EXPLOIT NTPDX buffer overflow	0	5	0	5
Probable NMAP fingerprint attempt	0	4	0	4
External FTP to HelpDesk 10.10.70.49	0	3	0	3
External FTP to HelpDesk 10.10.70.50	0	3	0	3
NETBIOS NT NULL session	0	3	0	3
External FTP to HelpDesk 10.10.53.29	0	1	0	1
DDOS TFN Probe	0	1	0	1
SMB CD...	0	1	0	1
SYN-FIN scan!	0	1	0	1
Attempted Sun RPC high port access	0	1	0	1
External FTP to HelpDesk 10.10.83.197	0	1	0	1

3.5.1 - SMB Name Wildcard. This was the number one alert that was recorded during the five days. There were 456,770 alert generated accounting for 63.5% of the total number of alerts. Of those alerts 53 were generated by traffic outbound from the University network and another 58 were generated between hosts within the network. Although the source port varied the destination port was always 137 (Netbios). Netbios is the method used by Windows machines to find out about file shares and ip addresses within their network. Internal traffic is usually normal traffic. External traffic inbound to port 137 more than likely is due to an attacker trying to discover the NETBIOS hostname by sending a NetBIOS SMB Name Wildcard query to a host. If successful the attacker may also receive information such as a list of some services running, the domain, current user, and the MAC address. This would be a genuine concern if the target host responded to the query.

Recommendations: Block port 137 to inbound traffic at the border router. The information gained by hackers will only prove harmful to the University's network. Blocking this port would remove unnecessary noise from network and improve availability with the freed up bandwidth.

3.5.2 - TCP SRC and DST outside network. With 65622 records this is the second most common alert. The largest amount of the alerts (99.7%) were to dst ip, 202.54.1.93. This ip appears to have been the target ip for the responses to a class A network scan. There were 65428 hosts in the 93.X.X.X class A network that responded with a single packet to the target ip. A further search finds that 202.54.1.93 never generated an alarm as a source ip. Of interest is that fact the neither of the two ips involved in generating this alert belongs to the University network. By TCP rules this not suppose to be possible. Since the IDS would be located on the network side of the router, it appears that the router is mal-configured and passing the traffic.

Recommendations: Check the router configuration and ensure that the access control list has not been corrupted and allowing this traffic into the network.

3.5.3 - CS WEBSERVER - external web traffic and CS WEBSERVER - external ftp traffic. These alert were generated by numerous external ips accessing the computer science and electrical engineering web server (10.10.100.165) on port 80 and ftp services on port 21. With 49533 of the 49547 port 80 alerts and all the 1413 port 21 alerts this department's Web Page seems to be quite popular. All of this traffic is most likely a false positive.

Recommendations: Ensure the server has the most resent patches installed and the virus software is up to date. Also check the ftp connections to confirm that they were legitimate file transfers.

3.5.4 - Watchlist 000220 IL-ISDNNET-990517 and Watchlist 000222 NET-NCFC. These are two Class B net blocks of ips that the University are interested in keeping on eye on. The first watch list is counting traffic generated from the 212.179.X.X network. Swhois resolves this class b network as belonging to Israel. The second is the 159.226.X.X class B network belonging to the Institute of Computing Technology Chinese Academy of Science.

Recommendations: Since this is a local rule, I am sure Network administrators are already checking the traffic associated with these ips.

3.5.5 - 10.10.30.3 activity and 10.10.30.4 activity. This alert is set to trigger on all traffic sent to either 10.10.30.3 or 10.10.30.4. These two ips are associated with Novell Netware. This is a product that provides secure Internet access to the Universities file storage. The University is monitoring all connections to these two ips. The login page to the Universities Novell NetWare site is 10.10.30.4 and 10.10.30.3 is the NetWare 6 home page that explains the features of product.

Recommendations: Since this service requires users to log in, administrator need to confirm only legitimate users are using the site and make sure that strong passwords use is enforced.

3.5.6 - High port 65535 tcp - possible Red Worm – traffic. (Included here is the UDP version of the same alarm). This rule was written to trigger on traffic to and from port 65535. Although a legitimate ephemeral port, as a rule normal traffic should not use such a high port. Hence the title of the rule: "possible Red Worm". There were a total of 13712 UDP alarm 7534 of those were outbound and the number one ip of interest is shown below. Something to take into account is the fact that the src port is 27005 for all of those connections, which may support the notion that these 7534 alarms are false positives. The TCP version of this alarm generated 19982 alarms. Of those 7872 were outbound alarms and 3942 of those were generated by the ip shown below. In this case 27 different source ports were used.

alert	Count of alert	dst port	src ip
High port 65535 udp - possible Red Worm - traffic	6095	65535	10.10.222.194
High port 65535 tcp - possible Red Worm - traffic	3942	65535	10.10.222.122

Recommendations: Due to the possible destructive nature of worms, viruses, and Trojans, Network administrators have to ensure that all host are protected by the most up to date dat files for the anti virus software. The two ips in the table above need to be scanned for possible infection.

3.5.7 - External RPC call. This alert triggers on inbound traffic to dst port 111, the Remote Procedure Call (RPC). RPC is a UNIX service that allows distant hosts to query a target host for a list of services and ports that may be open for the target host to connection on.

Recommendations: This port should be block to all inbound traffic

3.5.8 - Russia Dynamo - SANS Flash 28-jul-00. This rule is set to trigger on all traffic to and from the194.87.X.X class B network. Over this five-day period the 11964 alert were generated between 194.87.6.230 and 10.10.105.204. The src port used was 4559 and the dst port was the CONNECT port 2137. A facsimile software program called HylaFax commonly uses these ports.

Recommendations: Continue to monitor traffic that triggers this alert. I am quite sure most of this is legitimate traffic.

3.5.9 - spp_http_decode: IIS Unicode attack detected. Snort Preprocessor Plugin (spp). There were 11564 alerts in this category and 9230 of these were internal. This alert is based on the detection of Unicode-encoded characters "\", "/", and ".". These are common strings found in HTTP traffic. However there is a known vulnerability in Microsoft IIS 4 and 5 web server that could allow an attacker to sent a specially crafted URL string that could provide the attacker access to files and folders on the web server. It is also possible for the attacker to upload a cmd.exe file to the web server that could provide the attacker an ftp and telnet session on the server. If an attacker could gain full access and control on even one host it could be disastrous to the University's network. There were 312 internal host that generated the 9230 alerts. Every one of these hosts must be scanned for a potential virus. 10.10.242.250 produced 1775 of these alerts; this would be a good one to start with.

Recommendations: This is a real concern for administrators. Ensure that all servers all patched to the appropriate level and the anti virus software is kept up to date. The response to these requests will confirm if the servers are vulnerable.

3.5.10 - SUNRPC highport access! This alert triggers on traffic to dst port 32771. This port is listed as being the listener port for the portmapper in many Solaris platforms. Since high ports are seldom blocked or filtered an attacker may use this port in the same way they would use port 111. There were 5706 of these alerts detected. The table below shows three internal ips accounting for 5564 or 97.5% of these alerts.

src ip	alert	Count Of alert	dst port	dst ip
24.131.151.47	SUNRPC highport access!	688	32771	10.10.70.198
216.179.62.107	SUNRPC highport access!	1267	32771	10.10.239.254
128.8.10.18	SUNRPC highport access!	3609	32771	10.10.24.8

Recommendations: These three dst ips in the table above should be scanned for possible infection. Blocking this port to inbound traffic should be considered.

3.5.11- Queso fingerprint. This is an attempt by a remote host to gather information about the operating system (os) running on the target host. An attacker would send a crafted packet with various TCP flags set in attempt to gather information about the target host through the responses to the bogus packet. Although the "fingerprinting" of the os is not critical it could be the prelude to a more serious attack coming.

Recommendations: Not much can be done here. These scans are an everyday part of network life. Just ensure that all patches and updates are installed on all hosts within the network.

3.5.12- Incomplete Packet Fragments Discarded. As a rule fragmented packets are dropped if the host does not receive the entire sequence. There were a total of 1388 of these alerts and every one of those was src and dst port 0. Of the total number of alerts there were only 52 outbound alert generated to three different ips. Swhois.net resolved those three ips as belonging to ICQ. With this info the 52 outbound alert should be considered normal traffic.

67% of the inbound fragments were destined to five internal ips. This traffic could be a possible scan or denial of service by opening half-open connections.. A search of the database found no alerts of any kind were generated from these five ips. The other 33 % of this inbound traffic was scattered among 63 internal ips.

Destination IP	Number of alerts	Notes
10.10.194.182	259	Attempts from multiple src ips
10.10.207.13	248	Attempts from multiple src ips
10.10.249.134	206	Attempts from multiple src ips
10.10.207.2	98	Attempts from multiple src ips
10.10.247.174	89	Attempts from multiple src ips

Recommendations: Use a stateful firewall to limit half-open connections. Or have the packets reassembled and virus checked at the firewall prior to sending the packets to the intended hosts. Either way this will prevent hackers from using this type of attack against the network.

3.5.13- Possible trojan server activity. This rule has been written to trigger on any traffic to of from port 27374. This port has been associated with many Trojans such as DefCon8, multiple versions of SubSeven, BadBlood, and CIAC lists Linux as being vulnerable to the Ramen Worm on this port. There were a total of 52 outbound. 10.10.24.34 generated 28 of those alarms and interestingly was the only ip to use src port 80. This host should be scanned for a possible infection.

Recommendation: There is obviously a genuine concern here. Since this is a well known default Trojan port blocking 27374 inbound at the router could be considered. This may be not be necessary if the AV products is maintained and kept current.

3.5.14 NMAP TCP ping! This is another simple scan of hosts with the University's network. During the five days there were only 402 of these alarms generated. The attacker is using a program called NMAP in an attempt to find out what hosts are up, what services are running, and the operating system running on the host. This could be the prelude to a more serious attack.

Recommendations: Scans are everyday occurrences on ever network everywhere. Your best defense against compromise is ensuring a properly configured firewall, up to date anti virus product, all appropriate patches for software products are applied, and strong user passwords are in place. An attacker may discover some info about the network but if the above is procedures are followed then the usefulness of the information will be mitigated.

3.5.15- NIMDA - Attempt to execute cmd from campus host. There were 221 alarms produced by internal addresses attempt to connect to external ip with a packet content of "cmd.exe.". All of these alarms where destined for port 80. What should be noted from the table below is that the 10.10.97.X and 98.X class C blocks are the University's dial in accounts. This would indicate the computer owned by the dial in user is either playing with packet crafting tools or possibly infected with Nimda. The 10.10.168.65 and the 10.10.198.79 ips accounted for a total of 4 alarms over the five days and are more than likely a false positive but should be checked for a possible infection.

src ip	alert	Number of alerts
10.10.198.79	NIMDA - Attempt to execute cmd from campus host	1
10.10.168.65	NIMDA - Attempt to execute cmd from campus host	3
10.10.97.45	NIMDA - Attempt to execute cmd from campus host	4
10.10.97.30	NIMDA - Attempt to execute cmd from campus host	10
10.10.97.70	NIMDA - Attempt to execute cmd from campus host	26

src ip	alert	Number of alerts
10.10.98.102	NIMDA - Attempt to execute cmd from campus host	41
10.10.97.43	NIMDA - Attempt to execute cmd from campus host	142

Recommendations: User awareness seems to be the culprit here. The apparent infected boxes look as though they belong to dial up users. If there is a login record kept network administrators should be able to find out who was logged in and get the computer scanned. "Encouraging" users to use and update their AV product could be enforced with the loss of account access should they fail to do so.

3.5.16- IRC evil - running XDCC. There were only 74 total alarms produced by 15 different hosts. Internal hosts contacting a variety of external ip generated all alarms. The dst ports were 6665, 6667, 6668, 6669, and 7000. Although these ports are associated with a variety of IRC Nets, it should be noted that there are quite a few Trojans and exploits that also use these ports. These alarms definitely would warrant some further investigation.

Recommendations: Ensure that this traffic is legitimate.

3.6 - Selected External Hosts

The external source ip addresses were selected based on the number of times they produced a specific alert. These five ips generated 5.8% of the total number of alerts.

src ip	alert	Count Of src ip
68.49.35.0	10.10.30.3 activity	15977
61.56.247.174	External RPC call	8924
212.179.48.177	Watchlist 000220 IL-ISDNNET-990517	5808
63.148.150.226	External RPC call	5589
194.87.6.230	Russia Dynamo - SANS Flash 28-jul-00	5405

3.6.1. – 68.49.35.0. This ip was the most active host when it comes to inbound alerts generated. RIPE.NET used ARIN to resolve this ip as shown below:

```
inetnum:      68.32.0.0 - 68.63.255.255
netname:      JUMPSTART-1
descr:        Comcast Cable Communications, Inc.
country:      US
admin-c:      ARIN1-RIPE
tech-c:       ARIN1-RIPE
remarks:      ***************************************************
remarks:      *    This IPv4 network is assigned by the ARIN.     *
remarks:      * Please query WHOIS.ARIN.NET for more information. *
remarks:      ***************************************************
mnt-by:       ARIN-MNT
changed:      hostmaster@arin.net 20011129
changed:      ripe-dbm@ripe.net 20030423
source:    ARIN
```

3.6.2. – 61.65.247.174. This host accounted for over 60% of the external RPC call alerts. RIPE.NET resolve this ip using APNIC as belonging to:

```
inetnum:        61.56.0.0 - 61.67.255.255
netname:        TWNIC-TW
descr:          Taiwan Network Information Center
descr:          4F-2, No. 9 Section 2, Roosevelt Road,
descr:          Taipei, Taiwan, R.O.C.
country:        TW
admin-c:        SO12-AP
tech-c:         NS10-AP
mnt-by:         APNIC-HM
mnt-lower:      MAINT-TW-TWNIC
changed:        hostmaster@apnic.net 20010221
status:         ALLOCATED PORTABLE
source:         APNIC
```

3.6.3. – 212.179.48.177. Of the watch list traffic this ip was the most active host, accounting for 26.5% of the alert. RIPE.NET resolve this ip as belonging to:

```
inetnum:        212.179.48.128 - 212.179.48.191
netname:        ADZ
descr:          ADZ-LAN
country:        IL
admin-c:        MZ4647-RIPE
tech-c:         MZ4647-RIPE
status:         ASSIGNED PA
notify:         hostmaster@isdn.net.il
mnt-by:         RIPE-NCC-NONE-MNT
changed:        hostmaster@isdn.net.il 20010123
source:         RIPE
```

3.6.4. – 63.148.150.226. This ip was the fourth most active host, producing a further 37.8% of the external RPC traffic. RIPE.NET used ARIN and RADB to resolve this ip as below:

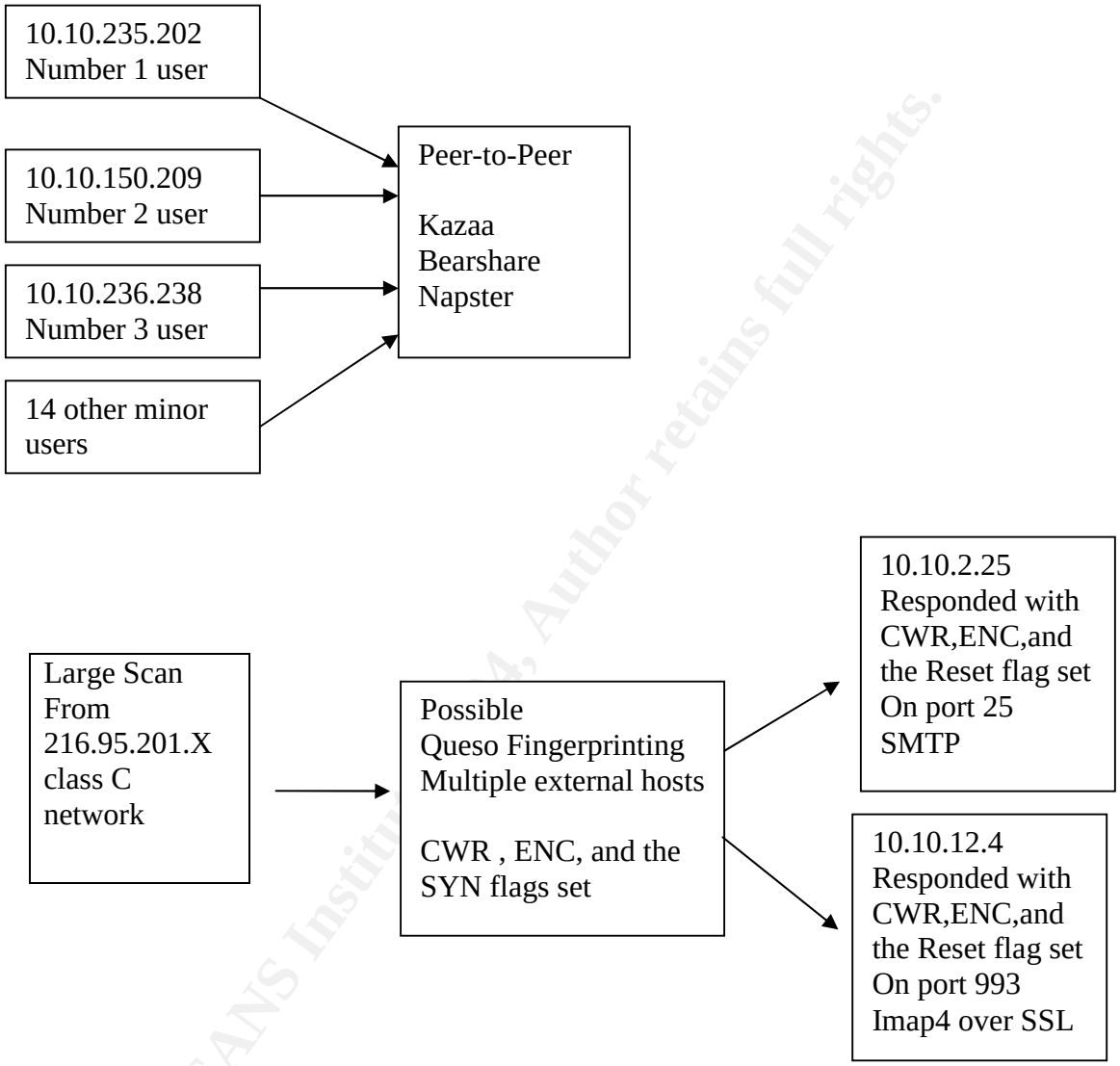
```
inetnum:        63.144.0.0 - 63.151.255.255
netname:        NET-QWEST-BLKS-2
descr:          Qwest Communications
country:        US
admin-c:        ARIN1-RIPE
tech-c:         ARIN1-RIPE
remarks:        ***************************************************
remarks:        *    This IPv4 network is assigned by the ARIN.     *
remarks:        * Please query WHOIS.ARIN.NET for more information. *
remarks:        ***************************************************
mnt-by:         ARIN-MNT
changed:        hostmaster@arin.net 20000313
changed:        ripe-dbm@ripe.net 20030423
source:         ARIN
```

```
route:          63.144.0.0/13
descr:          Qwest Communications
                950 17th Street Suite 1900
                Denver, CO 80202
origin:         AS209
mnt-by:         MAINT-QWEST
changed:        dgassen@qwest.com 20020504
source:         RADB
```

3.6.5. – 194.87.6.230. This ip was the fifth most active ip with respect to alarms generated and was responsible for 45.2% of the alert set to trigger on traffic from Russia. RIPE.NET resolve this ip as belonging to:

```
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
```

3.7 - Link Graph of OOS



3.8 - Summary & Defensive Recommendations

It is an unfortunate statement of the computing world. As long is there a Internet connection to a network there will always be an individual looking to exploit the services or attempt to destroy the operation of a host or network. As administrators and analyst we accept the reality that our network will be scanned, probed, and, if not protected, hacked. It is our responsibility to use and do whatever we can to protect our network and continue to provide uninterrupted service to our users. Looking at the alerts and scans to the University's network and without the benefit of the packets, I am making the following recommendations.

1) Ensure that a stateful firewall with a strong up to date anti virus product is protecting the network.
2) Enforce the use of up to date anti virus software on all hosts within the network. Have users sign an agreement prior to being granted access stating that they comply with this rule or they will loss their access to the network.
3) To improve the bandwidth and serviceability I would block all inbound traffic to port 1433 at the border router. This traffic is well known as nothing more than "noise". I would also monitor outbound traffic for signs of any possibly compromised hosts.
4) At the border router block inbound requests to ports 137,139,443,445. There are numerous vulnerabilities related to these ports. If there is a legitimate requirement for any host to have these ports accessible then build exclusions to the rule at the router and monitor the traffic.
5) Block port 111 to inbound requests. This is the listener port for the portmapper and there is no requirement for anyone outside your own network to be requesting info related to open services on your hosts.
6) Ensure all your servers have the latest patches installed.
7) Authorized use policy. In resent months there has been an increase in questions related to the legalities behind Peer-to-Peer file sharing through programs such as Kazaa, Winmx, and Gnutella. If the courts deem that this type of "file sharing" is illegal the University may want to take a proactive active position and prohibit this type of traffic before they become the focus of a lawsuit. One of the best ways to control what software can be installed on your host is to base line the network. As a self-protective move the University may want to remove the ability for users to download or install software on any of the University owned computers. Once again users could be required to sign an agreement that they will not use these "questionable" produces of the network. Although configurable, the default ports 1214 (kazaa) , 6346 (gnutella), and 6699 (winmx) would then have to be monitored and violators of the agreement would have their access to the network removed. An additional feature of base lining is the ability to "push" updates and patches to software whenever a user logs in.
8) Finally, a daily check of the IDS logs in a must. Administrators will be able to confirm and correct any unusually traffic trigger by their IDS rule sets.

3.9 References

Nothcutt, Stephen et al. Intrusion Detection Signatures and Analysis. Indianapolis, IN New Riders, January 2001

Scambray, Joel, et al Hacking Exposed: Network Security Secrets and Solutions 2nd Edition. Berkely, CA: Osborne/Mcgraw-Hill, 2001

http://www.securityfocus.com
http://aris.securityfocus.com/
http://www.dshield.org/ipinfo.php
http://www.sans.org
http://www.incidents.org
http://www.cert.org
http://www.practicallynetworked.com/sharing/app_port_list.htm
http://www.iana.org/assignments/port-numbers
http://www.iss.net/security_center/alerts/
www.simovits.com/sve/nyhetsarkiv/1999/nyheter9902.html
http://Ripe.net
http://www.google.ca
http://www.cve.mitre.org
http://www.giac.org/GCIA_600.php Ricky Smith #0595
http://www.giac.org/GCIA_600.php Harry Halladay #0516 Document #3