Global Information Assurance Certification Paper

Assignment 1 – Network Detects

Detect 1 – NetBIOS Name port scan (port 137)

```
7-16-2000 (GMT-0500)
09:18:08.779551  24.163.31.51.137 > my.box.137: udp 50 (ttl 106, id 7464)
09:18:08.782194  192.168.1.20.137 > my.box.137: udp 50 (ttl 106, id 7720)
09:18:10.278349  24.163.31.51.137 > my.box.137: udp 50 (ttl 106, id 7976)
09:18:10.280268  192.168.1.20.137 > my.box.137: udp 50 (ttl 106, id 8232)
09:18:11.776314  24.163.31.51.137 > my.box.137: udp 50 (ttl 106, id 8488)
09:18:11.782210  192.168.1.20.137 > my.box.137: udp 50 (ttl 106, id 8744)

07/16-09:18:08.779551 24.163.31.51:137 -> my.box:137
UDP TTL:106 TOS:0x0 ID:7464
Len: 58
48 7E 00 10 00 01 00 00 00 00 00 00 20 43 4B 41 H~.......... CKA
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 41 00 00 21 AAAAAAAAAAAAA..!
00 01                                           ..
```

1. Source of trace:

Home computer connected to @home cable modem network.

2. Detect was generated by:

A. Windump, (win32 port of tcpdump).

Log format:

09:18:08.779551	192.168.1.20.137 >	my.box.137:	udp	50	(ttl 106,	id 7720)
timestamp	source IP and port	dest and port	protocol	payload bytes	time-to-live	IP ID#

B. Snort, (win32 port of Snort).

Log format:

07/16-09:18:08.779551	24.163.31.51:137->my.box.137
date/timestamp	source IP and port dest and port

UDP	TTL: 106	TOS:OxO	ID:7464
protocol	time-to-live	type of service	IP ID#

Len: 58
packet length

48 7E 00 10 00 01 00 00 00 00 00 00 20 43 4B 41	H~.........CKA
hexadecimal	ascii decode

At the time of the detect, both windump and snort captured all packets, dropping none. Due to the isolated nature of this computer on a large public network, only ARP packets were filtered in both windump and snort while the computer is idle. The output is then reviewed by the analyst, and further filters or rulesets may be utilized to isolate traffic if necessary.

3. Probability the source address was spoofed:

It is highly unlikely the source addresses are spoofed. There is no indication that this detect is a denial of

service against either the source or destination hosts, nor is there any evidence of TCP hijacking as the two hosts have no trust relationship. Additionally, there is evidence of a poorly configured network at the source host. Finally, reconnaissance typically requires a non-spoofed source IP to receive an informative response.

4. Description of attack:

This scan of port 137 is likely targeted at the NetBIOS Name Service commonly found on Microsoft Windows operating systems. A windows operating system command, nbtstat.exe is a common source for this type of scan. The IP ID numbers increment by a steady 256, indicative of a script or other tool to automate the scanning of an entire Class C address space. The source host is on a different Class C subnet than the destination host.

Currently, there are no CVE entries but there are two relevant CVE candidates. CAN-1999-0520 is described as "A system-critical NETBIOS/SMB share has inappropriate access control". CAN-1999-0621 is described as "A component service related to NETBIOS is running".

5. Attack mechanism:

This attack is the stimulus in a reconnaissance effort to the NetBIOS Name service port. The program can be run from a command prompt in Windows and is supplied with the operating system. Nbtstat.exe can provide detailed information about the host's shares and Microsoft's WINS database. This diagnostic command displays protocol statistics and current TCP/IP connections using NBT (NetBIOS over TCP/IP). This command is available only if the TCP/IP protocol has been installed on the source host. Options for nbtstat.exe on Microsoft's Windows 2000 implementation are as follows:

nbtstat [-a remotename] [-A IP address] [-c] [-n] [-R] [-r] [-S] [-s] [interval]

Parameters

-a remotename Lists the remote computer's name table using its name.

-A IP address Lists the remote computer's name table using its IP address.

-c Lists the contents of the NetBIOS name cache giving the IP address of each name.

-n Lists local NetBIOS names. Registered indicates that the name is registered by broadcast (Bnode) or WINS (other node types).

-R Reloads the Lmhosts file after purging all names from the NetBIOS name cache.

-r Lists name resolution statistics for Windows networking name resolution. On a Windows 2000 computer configured to use WINS, this option returns the number of names resolved and registered via broadcast or via WINS.

-S Displays both client and server sessions, listing the remote computers by IP address only.

-s Displays both client and server sessions. It attempts to convert the remote computer IP address to a name using the Hosts file.

interval Redisplays selected statistics, pausing interval seconds between each display. Press CTRL+C to stop redisplaying statistics. If this parameter is omitted, nbtstat prints the current configuration information once.

An intruder can use inappropriately secured shares to acquire files or other data from a victim's computer and even more severe, plant Trojans or other malicious code on the users computer. Additionally, if the targets Lmhosts file has been compromised, nbstat.exe can be used to great effect to reload the Lmhosts file at will.

Interesting to note is the proximity of a private IP address sending a similar packet immediately after the first scan. This repeats after each packet from the public IP source address. This is typical of a poorly configured Windows host and/or network.

This is described in Microsoft's Knowledge Base article Q166159:

The Windows NT 4. 0 (and earlier) redirector used the following logic to establish a NetBIOS session:

Place a call to the destination name on all bound transports, in the order they are bound. Waft for the 'Primary" (first-bound) transport to complete, and if it was successful in reaching the target, set up a session on it and disconnect (cancel) the other calls.

In the case of a computer that is multihomed on two connected (by a router) networks, if the target system is on only one of those networks, there are two paths to the target system. In this case, the call succeeds on both paths, and the "primary" transport (for example, the local source IP address associated with that binding of NetBT) is the chosen one. The other call is cancelled.

Also of interest is the ascii data (CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA) shown in the snort output. When NetBIOS names are sent over the network, they are coded. What happens with an nbtstat query is as follows:

Each character in the NetBIOS name is divided into two hex characters;
Normally blank padded to 16 characters;
Each hex character added to ASCII value 0x41 (uppercase "A").

Now, if a wildcard name is used "*", the formula is a little different:

Each character in NetBIOS name is divided into two hex characters;
Null padded to 16 characters;
Each hex character added to ASCII value 0x41 (uppercase "A").

The value of "*" in hex is 2A. So the formula is as follows:

```
2 A (divided into two hex characters)
2 A (null padded = no change)
   2  A
+ 41 41 41 41 41 41 41 41 41 41 41 41 41 41, etc.
------------------------------------------------------------------
  43 4B 41 41 41 41 41 41 41 41 41 41 41 41 = Hex result
   C  K  A  A  A  A  A  A  A  A  A  A  A  A = ASCII result
```

6. Correlations:

A detect from almost exactly one week prior was logged on the same destination host from a different source:

```
7-2-2000 EST (GMT-0500)
17:06:08.433073 192.168.0.1.137 > my.box.137: udp 50 (ttl 115, id 63480)
17:06:08.437464 24.189.78.72.137 > my.box.137: udp 50 (ttl 115, id 63736)
17:06:09.934894 192.168.0.1.137 > my.box.137: udp 50 (ttl 115, id 63992)
17:06:09.940614 24.189.78.72.137 > my.box.137: udp 50 (ttl 115, id 64248)
17:06:11.433232 192.168.0.1.137 > my.box.137: udp 50 (ttl 115, id 64504)
17:06:11.437664 24.189.78.72.137 > my.box.137: udp 50 (ttl 115, id 64760)
```

Once again, we see a private address perform the same scan, at an interval of 256 between IP ID numbers. In this case, however, the private (local source) IP packet arrives before the public interface packet. Further, the source host is again on a different subnet than the destination host.

Scans of port 137 have become common as of late (originating from numerous different source IP's). Similar scans have been reported by others and can be found at:

http://www.sans.org/y2k/011900.htm
http://www.sans.org/y2k/031600.htm
http://www.sans.org/y2k/033000.htm
http://www.sans.org/y2k/040500-1000.htm
http://www.sans.org/y2k/041100.htm
http://www.sans.org/y2k/042400.htm
http://www.sans.org/y2k/042700.htm
http://www.sans.org/y2k/042900.htm
http://www.sans.org/y2k/051300.htm
http://www.sans.org/y2k/052300-0800.htm
http://www.sans.org/y2k/060900-1030.htm
http://www.sans.org/y2k/081300.htm

7. Evidence of active targeting:

These detects were targeted at a subnet on the @home cable modem network. The @home cable network currently has a block of IP address space from 24.0.0.0 through 24.23.255.255. The IP ID intervals of 256 are indicative of scanning an entire Class C address space at a time.

8. Severity:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality = 2 (User's home computer.)

Lethality = 1 (The recon attack is very unlikely to succeed.)

System Countermeasures = 5 (The target is a Windows 2000 machine with all patches installed AND NetBIOS over TCP/IP is DISABLED.)

Network Countermeasures = 5 (Restrictive firewall with only one way in or out.)

Severity = (2 + 1) - (5 + 5) = -7

9. Defensive recommendation:

Defenses are more than adequate. Firewall blocks port 137 and NetBIOS is disabled at the Operating System level. All patches are installed.

10. Multiple choice test question:

The detect above is targeting:

a) Macintosh users
b) Windows users
c) UNIX NFS users
d) all of the above

answer: b

Detect 2 – Unknown NetBIOS Session packet flood (port 139)

```
08/22-11:33:23.929689 client1.mynet:1038 -> mail.server.mynet:139
TCP TTL:128 TOS:0x0 ID:1360  DF
*****PA* Seq: 0x1B0D6C   Ack: 0x735DD7D5   Win: 0x2100
...X.SMB2.........................@t.......(.............D......
..............\.V.I.R.U.S...

08/22-11:33:23.959438 client2.mynet:1042 -> mail.server.mynet:139
TCP TTL:128 TOS:0x0 ID:32462  DF
*****PA* Seq: 0x185111   Ack: 0x735F31F0   Win: 0x2030
...X.SMB2.........................@@.......(.............D......
.....Y........\.V.I.R.U.S...

08/22-11:33:23.987863 client3.mynet:1033 -> mail.server.mynet:139
TCP TTL:128 TOS:0x0 ID:47915  DF
*****PA* Seq: 0x3455E   Ack: 0x73D6D5F3   Win: 0x1EF8
...X.SMB2.................................(.............D......
.....Y........\.V.I.R.U.S...

08/22-11:33:23.990375 mail.server.mynet:139 -> client1.mynet:1038
TCP TTL:128 TOS:0x0 ID:2461  DF
*****PA* Seq: 0x735DD7D5   Ack: 0x1B0DC8   Win: 0x2010
...d.SMB2.........................@t...(.....8...(.<.....-......
...0........I...@.......@...r...........

08/22-11:33:23.990588 mail.server.mynet:139 -> client2.mynet:1042
TCP TTL:128 TOS:0x0 ID:2717  DF
*****PA* Seq: 0x735F31F0   Ack: 0x18516D   Win: 0x2010
...d.SMB2.........................@@...(.....8...(.<.....-......
...0........I...@.......@...r...........
```

1. Source of trace:

Windows NT Server v4.0 Email/Internet server on a Class C LAN.

2. Detect was generated by:

Snort, (win32 port of Snort).

Log format:

08/22-11:33:23.929689	client1.mynet:1038 ->	mail.server.mynet:139
date/timestamp	source IP and port	dest and port

TCP	TTL: 128	TOS:0x0	ID:1360	DF
protocol	time-to-live	type of service	IP ID#	do not fragment

*****PA*	Seq: 0x1B0D6C	Ack: 0x735DD7D5	Win: 0x2100
TCP flags	TCP sequence #	ACK sequence #	max. window size

...X.SMB2.........................@t.......(.............D......
..............\.V.I.R.U.S...
ascii decode

Snort had just been installed on the server and was run with the following command line options to test the installation:
Snort –v –d -C
Immediately it was noticed that hundreds of packets per minute were arriving with the text string “virus” in them and with a destination port of 139. The command line was quickly modified to:

Snort –v –d –C port 139
This option helped to isolate NetBIOS session traffic.

3. Probability the source address was spoofed:

The source addresses were not spoofed, the packets were generated by as many as 80 workstations on the internal Class C subnet. TTL's were all 128, indicating no hops. Further, proxy server software is installed on the server and a router is also in place. At the conclusion of the investigation, it was determined that the packets were generated because of misconfigured anti-virus client software.

4. Description of attack:

It was discovered that approximately 80 workstations on the internal Class C network were sending almost 100 TCP packets per minute each to the Email/Internet/Intranet server. Each packet contained the ascii data "virus" in its payload. The sheer quantity, widespread nature, and curious payload got my immediate attention. In addition, the server was responding to the workstations with it's own PUSH ACK. The workstations were sending packets to the NetBIOS session port (port 139).

Currently, there are no CVE entries but there are two somewhat relevant CVE candidates. CAN-1999-0520 is described as "A system-critical NETBIOS/SMB share has inappropriate access control". CAN-1999-0621 is described as "A component service related to NETBIOS is running".

5. Attack mechanism:

Each workstation had established a NetBIOS session with the server. The initial 3-way handshakes were not discovered. The workstations were sending TCP packets with a payload including the ascii data "virus." A closer look revealed that "virus" was preceded by a "\". Immediately, a file search for "*virus*.*" was performed on the NT 4.0 SP6a server. A subdirectory "..\virus\" was found in one of the Trend Micro Corp. anti-virus program directories. This Email/Internet server had a number of anti-virus products manufactured by Trend Micro installed on it. All incoming emails are virus scanned as well as Internet access via the proxy server software installed on the server. One other program from Trend Micro, OfficeScan NT, was based on the server as a virtual directory on the company Intranet. This program distributed updates of the desktop virus scan program to workstations.

At this point, there were two concerns. The first, was this a denial of service caused by the nearly 8,000 packets per minute sent to the server (and the 8,000 responses)? Secondly, was there some type of virus or Trojan being passed back and forth between the server and the workstations? A complete virus scan of one of the impacted workstations was performed. The virus pattern files were current and the machine was found to be clean of known viruses and Trojans. This result moved the focus of my concern to the DoS implications. The server was a bit sluggish but functioning properly.

An email was placed to Trend Micro technical support describing the Trend Micro applications installed on the server and a request was made of them as to whether one of these software packages were responsible for creating these packets. Snort traces were included in the email as well. Responses were received from three different Trend Micro tech support personnel (one for each of the server applications). The email and Internet scanning products were ruled out early on, as they typically have little or no communication with client machines.

I reported to the OfficeScan NT technician that a 30-day demo was installed and then an updated version of the application was purchased and installed. The demo program was a file-based application, whereas the newer version was http based. I was informed that an incremental update was available (v3.50 to v3.52), and was instructed to remove the old version, remove the virtual directory, and that a restart of the server would be required. Then an installation of OfficeScan NT v3.52 should be performed.

The instructions were followed and the restart was performed. The events that followed make the earlier

frustrations pale in comparison. Upon bootup, the server was not sluggish anymore, it was downright crawling! It took 5 minutes just to logon to the server and CPU utilization was pegged at 100%. My determination of the cause of this is that the client attempts at the 3-way handshakes were being replied to with resets, and the initial SYN's were being repeated constantly. I briefly disabled the port on the switch the server was connected to. The CPU utilization dropped to nominal. Unfortunately, it was early afternoon and I couldn't kill the email connection for the required time to install OfficeScan NT 3.52 and reapply NT Service Pack 6a. I enabled the switch port again. 100% CPU utilization returned. Email managed to trickle through in spite of this. It took 10 minutes to simply click START…RUN…BROWSE on the server. I backed our of browse and typed the full network path to the new setup files for OfficeScan NT. After a half-hour, the setup program hung at approximately 19% of the installation. I cancelled the install process. I shut down all unnecessary services to try to get some speed out of the server, but the effort was made in vain. From my workstation, I copied the new setup files to a local directory on the server. This consumed about an hour. I ran the install at the server from the local drive and it actually completed successfully after about an hour. I configured the application and noticed CPU utilization back to nominal again after clients were sent instructions to logoff and back on to the network (I had inserted a call statement to the client anti-virus update location in the logon scripts). Problem solved (almost). Running snort again on the server indicated there were still 4 workstations sending the curious packets. The client software was manually uninstalled at the 4 workstations and the client setup program was run successfully for the 3.52 version.

6. Correlations:

I am unaware of any public correlations to this activity. Trend Micro had nothing in their database either. As seen in the trace submitted, similar packets were sent by multiple workstations, thereby confirming that the problem was widespread and not isolated.

7. Evidence of active targeting:

The packets in question were obviously sent deliberately to a specific target (the server). However, considering that this detect was actually a poorly configured internal application, the use of the word "attack" may be inappropriate.

8. Severity:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality = 4 (Email/Internet server.)

Lethality = 4 (The server experienced a limited, one-time DoS during the repair process.)

System Countermeasures = 3 (The server had all OS patches. The application in question was not fully patched.)

Network Countermeasures = 3 (Restrictive firewall and proxy server on Internet connection. Router on WAN connection.)

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

9. Defensive recommendation:

The install of the original demo version was later updated to a significantly enhanced version. Hindsight clearly shows that the clients should have been uninstalled prior to the upgrade to the new version with its http based (vs. file based) approach. The logistics of uninstalling approximately 100 workstations was somewhat of a deterrent however. Also noteworthy, the server application did not have up to date patched installed the time of the detect. Application patches should be applied as soon as possible after initial availability.

10. Multiple choice test question:

The above detect is an example of:

a) a NetBIOS name port scan
b) the transmission of a virus
c) NetBIOS session port scan
d) data transfer after a completed 3-way handshake

answer: d

Detect 3 – PcAnywhere Scan (port 5632)

```
FWIN,2000/08/18,18:03:36 -5:00 GMT,24.xxx.xxx.130:1042,my.box:5632,UDP
FWIN,2000/08/18,18:17:38 -5:00 GMT,24.xxx.xxx.130:1062,my.box:5632,UDP

08/19-06:17:34.250344 24.xxx.xxx.130:1043 -> my.box:5632
UDP TTL:127 TOS:0x0 ID:333
Len: 10
4E 51 00 00 00 00 60 D1 06 02 FF FF FF FF FF FF  NQ....`.........
08 00                                            ..

FWIN,2000/08/19,06:50:20 -5:00 GMT,24.xxx.xxx.130:1061,my.box:5632,UDP

08/19-10:13:16.276436 24.xxx.xxx.130:1169 -> my.box:5632
UDP TTL:127 TOS:0x0 ID:17067
Len:10
4E 51 00 00 76 11 FF FF FF FF FF FF 08 00 3E 0F  NQ..v.........>.
4E 12                                            N.

FWIN,2000/08/19,12:01:42 -5:00 GMT,24.xxx.xxx.130:1496,my.box:5632,UDP
FWIN,2000/08/19,12:35:02 -5:00 GMT,24.xxx.xxx.130:1505,my.box:5632,UDP
FWIN,2000/08/19,14:37:44 -5:00 GMT,24.xxx.xxx.130:1513,my.box:5632,UDP
FWIN,2000/08/19,15:54:08 -5:00 GMT,24.xxx.xxx.130:1627,my.box:5632,UDP
```

1. Source of trace:

Home computer connected to @home cable modem network.

2. Detect was generated by:

A. ZoneAlarm Personal Edition Firewall

Log format:

FWIN,	2000/08/19,12:01:42 -5:00 GMT,	24.xxx.xxx.130:1496,	my.box:5632,	UDP
Incoming blocked	date/timestamp	source IP and port	dest and port	Protocol

B. Snort, (win32 port of Snort).

Log format:

08/19-10:13:16.276436	24.xxx.xxx.130:1169->	my.box:5632
date/timestamp	source IP and port	dest and port

UDP	TTL:127	TOS:OxO	ID:17067
protocol	time-to-live	type of service	IP ID#

```
Len: 10
```
packet length

```
4E 51 00 00 76 11 FF FF FF FF FF FF 08 00 3E 0F
4E 12
```
hexadecimal

```
NQ..v.........>.
N.
```
ascii decode

At the time of the detect, snort captured all packets, dropping none. Due to the isolated nature of this computer on a large public network, only ARP packets were filtered in snort while the computer is idle. The output is then reviewed by the analyst, and further filters or rulesets may be utilized to isolate traffic if necessary. ZoneAlarm security settings were "High" for both local network and Internet zones.

3. Probability the source address was spoofed:

It is highly unlikely the source addresses are spoofed. There is no indication that this detect is a denial of service against either the source or destination hosts, nor is there any evidence of TCP hijacking as the two hosts have no trust relationship. Finally, reconnaissance typically requires a non-spoofed source IP to receive the informational response.

4. Description of attack:

These scans of port 5632 are looking for hosts running the remote control software application pcAnywhere.

Currently, there are no CVE entries or CVE candidates relating to pcAnywhere.

5. Attack mechanism:

Symantec's pcAnywhere client versions 7.5x and higher can scan an entire subnet for a host by setting the last octet of its host's TCP/IP address to 255. Entering multiple subnets is possible. The application provides remote control capability of remote computers. Symantec's original intent in providing this functionality was likely to aid network administrators in discovering installed clients on their own network. A demo of this application can be downloaded off the Internet at:

https://enterprisesecurity.symantec.com/Content/TrialwareForm.cfm?PromoCode=ESTrialware&ProductID=2&PID=521995&SSL=Yes

This provides a quick thrill apparently for some script kiddies and hacker wannabes. Once the perpetrator finds a host, weak password policies can be detected by attempting to logon with the host. Full control of the victims computer can be obtained if compromised.

6. Correlations:

This type of scanning is common on the Internet and some correlations can be found at the links below:

http://www.sans.org/y2k/122699.htm
http://www.sans.org/y2k/010200-0900.htm
http://www.sans.org/y2k/051000.htm
http://www.sans.org/y2k/070400.htm

7. Evidence of active targeting:

These scans were actively targeted at the destination computer. The question is whether this was malicious in intent or caused by the incompetence of an amateur computer professional. This is an impossible question to answer with the information available. If one had to make a judgment call, I would say script kiddie (amateur).

8. Severity:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality = 2 (User's home computer.)

Lethality = 1 (The recon attack is very unlikely to succeed.)

System Countermeasures = 5 (The target is a Windows 2000 machine with all patches installed and pcAnywhere is not installed.)

Network Countermeasures = 5 (Restrictive firewall with only one way in or out.)

Severity = (2 + 1) - (5 + 5) = -7

9. Defensive recommendation:

Defenses are more than adequate. Firewall blocks port 5632 and the application in question is not installed. All patches are installed.

10. Multiple choice test question:

The detect above is targeting:

a) Microsoft SMS remote control hosts
b) Secure shell hosts
c) pcAnywhere hosts
d) none of the above

answer: c

Detect 4 – Back Orifice

```
Wed Mar 01 00:39:04    BO PING sweep attempted by 216.67.21.103
Wed Mar 01 00:39:12    BO TYPE_PROCESSLIST attempted by 216.67.21.103
Wed Mar 01 00:39:12    BO TYPE_APPLIST attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_SYSINFO attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_SYSLISTPASSWORDS attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_SYSENDKEYLOG attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_FILEDELETE attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_SYSLOGKEYS attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_HTTPDISABLE attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_HTTPENABLE attempted by 216.67.21.103
Wed Mar 01 00:39:13    BO TYPE_REDIRLIST attempted by 216.67.21.103
Wed Mar 01 00:39:14    BO TYPE_NETCONNECTIONS attempted by 216.67.21.103
Wed Mar 01 00:39:14    BO NETVIEW attempted by 216.67.21.103
Wed Mar 01 00:39:14    BO DIR C:\*.* attempted by 216.67.21.103
Wed Mar 01 00:39:26    BO TYPE_SYSINFO attempted by 216.67.21.103
Wed Mar 01 00:39:26    BO TYPE_SYSLISTPASSWORDS attempted by 216.67.21.103
Wed Mar 01 00:39:38    BO TYPE_SYSINFO attempted by 216.67.21.103
Wed Mar 01 00:39:38    BO TYPE_SYSLISTPASSWORDS attempted by 216.67.21.103
Wed Mar 01 00:39:44    BO TYPE_SYSENDKEYLOG attempted by 216.67.21.103
Wed Mar 01 00:39:44    BO TYPE_FILEDELETE attempted by 216.67.21.103
```

1. Source of trace:

Home computer connected to local ISP via ISDN.

2. Detect was generated by:

NFR's Back Officer Friendly version 1.0.1.

Log format:

Wed Mar 01 00:39:04	BO	PING sweep	attempted by 216.67.21.103
Date/timestamp	program	command	Source IP address

At the time of the detect, there was no other intrusion detection software or firewalls installed on this connection.

3. Probability the source address was spoofed:

It is highly unlikely the source address is spoofed. The extensive running of Back Orifice commands after the initial scan is indicative that the attacker was receiving the fake replies from Back Officer Friendly. Typica scans for Back Orifice (BO) consist of 3 or 4 packets (see correlations section). In this case, the initial scan was followed by 19 separate BO commands.

4. Description of attack:

The initial "BO ping sweep" scan is of port 31337. The scanner is looking for hosts running the stealthy remote control software application Back Orifice which has been created by the hacker group Cult of the Dead Cow. Fake replies sent by Back Officer Friendly provided false confirmation of the existence of a Back Orifice host, and subsequently, numerous commands were run by the attacker thinking he/she had found a vulnerable target.

Currently, there are no CVE entries or CVE candidates relating to Back Orifice.

5. Attack mechanism:

Back Orifice is a remote control application. It can install itself on Windows 95 and 98 machines. BO has been distributed in similar fashion to computer viruses. It has been embedded as part of downloadable shareware and executable greeting card programs. When the infected program is run, BO quietly installs itself. Hackers can than obtain complete access to the compromised machine, including the ability to:
View files;
Capture keystrokes to a log file;
Set network file sharing properties;
Update registry entries;
Enable a web server and upload and download files;
Reboot or crash the machine; and
Use the compromised machine to scan or control other machines on your network.

Your entire network is made vulnerable with just one infected machine. The compromised computer(s) can be made relay points thereby making the origin of the perpetrator almost impossible to trace.

The Cult of the Dead Cow claims that over 300,000 copies of BO have been downloaded from their primary and mirror web sites. A newer version, Back Orifice 2000 (BO2K) was released in July 1999. This version runs on Windows NT in addition to Windows 95 and 98. Additionally, a more powerful plug in architecture was incorporated in BO2K, and now works with TCP in addition to UDP.

BO has two primary components, the server application, and the client application. An infected machine contains the server software. Computers running the client software can have complete control of the server machine.

The detect indicates the following commands and procedures were run:

PING- initial detection of the BO server. Back Officer Friendly was set to send Fake Replies and to indicate the server is a BO server.

PROCESSLIST- returns the process list of the server machine with process names and ID numbers.

APPLIST- returns the applications list for the server.

SYSINFO- returns information about the system including machine name and capacity of storage devices attached to the server.

SYSLISTPASSWORDS- returns list of passwords stored in the Internet Explorer password cache.

SYSENDKEYLOG- stops logging keystrokes on the server.

FILEDELETE- deletes a file via its full pathname on the server.

SYSLOGKEYS- captures the keystrokes that the user of the server types at the keyboard to a disk file. It tells what windows they typed the keystrokes into as an aid to understand what the user is doing.

HTTPDISABLE- disable web server on the BO server.

HTTPENABLE- enable web server on the BO server.

REDIRLIST- it is assumed this is a request for a directory list but cannot confirm.

NETCONNECTIONS- returns a list of which machines are connected to the server.

NETVIEW- provides the functionality of the net view Windows command.

DIR C:*.* - file command.

The SYSINFO, SYSLISTPASSWORDS, SYSSENDKEYLOG and FILEDELETE commands were repeated.

The use of the Back Officer Friendly Fake Replies option appears to have kept the attacked entertained for the good part of a minute until the attacker determined it was a dead end.

Back Orifice 2000 can be downloaded at:
http://www.bo2k.com/warez.html

6. Correlations:

This type of scanning is common on the Internet and some correlations with differing log types can be found at the links below:

http://www.sans.org/y2k/081600-1500.htm

```
Aug 11 08:54:51 [my.firewall.ip] %PIX-2-106006:
  Deny inbound UDP from 4.35.128.159/2829 to 192.168.254.2/31337
Aug 11 08:54:51 %PIX-7-106011: Deny self route udp src
  outside:4.35.128.159/2829 dst outside:ext.net.230.102/31337
Aug 11 08:54:51 %PIX-7-106011: Deny self route udp src
  outside:4.35.128.159/2829 dst outside:ext.net.230.104/31337
Aug 11 08:54:52 %PIX-7-106011: Deny self route udp src
```

```
  outside:4.35.128.159/2829 dst outside:ext.net.230.105/31337
Aug 11 08:54:52 [my.firewall.ip] %PIX-2-106006:
  Deny inbound UDP from 4.35.128.159/2829
  to internal.ip.net.2/31337
```

http://www.sans.org/y2k/062700.htm

```
Datestamp 2000-06-23
Timestamp: 06:27:16
Type: Back Orifice ping
Source IP: 200.238.99.165
Destination IP: 4.35.89.174
Data: PING(1)&passwd=0x7A69&length=19&xid=0x0&iport=0x7A6A&vport=0x7A69
```

http://www.sans.org/y2k/061000.htm

```
Jun 07 2000 13:13:36 : Deny inbound UDP from
  212.211.0.27/1000 to x.x.x.2/31337
Jun 07 2000 13:13:36 : Deny inbound UDP from
  212.211.0.27/1000 to x.x.x.31/31337
Jun 07 2000 13:13:36 : Deny inbound UDP from
  212.211.0.27/1000 to x.x.x.31/31337
Name: mfs-pci-bqe-vty27.as.wcom.net
Address: 212.211.0.27
```

http://www.sans.org/y2k/053000-1100.htm

```
Back Orifice probes [NFR Back Officer Friendly providing fake replies]
Sun May 28 00:02:09 BO PING sweep attempted by 216.244.12.4
00:02:10.038459 ip 61: pool0004.cvx35-bradley.dialup.earthlink.net.2806 >
my.box.31337: udp 19 (ttl 47, id 24930)
00:02:10.097812 ip 84: my.box.31337 > pool0004.cvx35-
bradley.dialup.earthlink.net.2806: udp 42 (ttl 128, id 19)
Sun May 28 21:37:57 BO PING sweep attempted by 216.244.12.13
21:37:57.407399 ip 61: pool0013.cvx35-bradley.dialup.earthlink.net.1188 >
my.box.31337: udp 19 (ttl 47, id 22984)
21:37:57.537572 ip 84: my.box.31337 >   pool0013.cvx35-
bradley.dialup.earthlink.net.1188: udp 42 (ttl 128, id 4363)
Sun May 28 22:38:51 BO TYPE_REDIRADD attempted by 216.244.12.13
22:38:52.159781 ip 82: pool0013.cvx35-bradley.dialup.earthlink.net.1272 >
my.box.31337: udp 40 (ttl 47, id 26050)
22:38:52.160405 ip 138: my.box.31337 >   pool0013.cvx35-
bradley.dialup.earthlink.net.1272: udp 96 (ttl 128, id 4364)
```

http://www.sans.org/y2k/053000-1000.htm

```
Sun May 28 00:02:09 BO PING sweep attempted by 216.244.12.4
  00:02:10.038459 ip 61: pool0004.cvx35-bradley.dialup.earthlink.net.2806 >
  my.box.31337: udp 19 (ttl 47, id 24930)
```

http://www.sans.org/y2k/052800.htm

```
[**] IDS188/probe-back-orifice [**] 05/24-19:34:10.961503
168.191.127.159:1182 -> 24.x.x.x:31337 UDP TTL:117 TOS:0x0 ID:28750
Len: 26
```

http://www.sans.org/y2k/051500.htm

```
May 11 20:12:05 cc1014244-a kernel: securityalert: udp if=ef0 from
24.112.117.50:3650 to 24.3.21.199 on unserved port 31337
```

http://www.sans.org/y2k/051200.htm

```
May 7 14:12:44 myhost snort[1939]: Back Orifice:
```

```
user-38ld29b.dsl.mindspring.com:1567 -> my.ip.addr:31337
```

http://www.sans.org/y2k/042100.htm

```
From Port Date - Time (CST) From  To   To Port  EventName
Information
1746  4/17/00  11:20:06AM     200.42.140.45     163.186.32.92     31337
BackOrifice COMMAN      Ping host
1746  4/17/00  11:20:06AM     200.42.140.45     163.186.32.91     31337
BackOrifice COMMAN      Ping host
1746  4/17/00  11:20:06AM     200.42.140.45     163.186.32.93     31337
BackOrifice COMMAN      Ping host
```

http://www.sans.org/y2k/041600.htm

```
From Port       Priority         Date     To       To Port Event
31338   High    4/15/00   7:39:26AM      c.183.1   31337   BackOrifice
31338   High    4/15/00   7:39:26AM      c.183.2   31337   BackOrifice
31338   High    4/15/00   7:39:26AM      c.183.3   31337   BackOrifice
31338   High    4/15/00   7:39:26AM      c.183.4   31337   BackOrifice
```

http://www.sans.org/y2k/032800-2000.htm

```
Mar 27 22:07:59 cc1014244-a kernel: securityalert: udp if=ef0 from
38.32.11.6:1044 to 24.3.21.199 on unserved port 31337
```

http://www.sans.org/y2k/032100-2000.htm

```
Mar 20 13:48:21.150739 212.217.21.232,2888 -> 10.1.8.55,31337 PR udp len 20 47
```

7. Evidence of active targeting:

The scan was actively targeted at the destination computer. After the apparent successful scan, extremely active targeting was performed.

8. Severity:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)

Criticality = 2 (User's home computer.)

Lethality = 1 (The attack is very unlikely to succeed.)

System Countermeasures = 5 (The target was a Windows NT v4.0 server with all patches installed and Back Orifice server could not run on the Windows NT platform. Additionally, NFR's Back Officer Friendly spoofing and detection software was installed. Note that BO2K does run on NT.)

Network Countermeasures = 2 (Computer utilized a dial up connection to an ISP with a limited firewall.)

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

9. Defensive recommendation:

Defenses are more than adequate to fend off any Back Orifice scans or commands. General Internet security was extremely poor, however. Personal firewall software should be put in place..

10. Multiple choice test question:

The detect above is targeting:

a) Microsoft 95/98 computers
b) Microsoft NT Workstation computers
c) Microsoft NT Server computers
d) all of the above

answer: a

Assignment 2 – Evaluate an Attack *nMapNT*

1. nmapNT from eEye Digital Security

eEye Digital Security has ported nmap to the Windows NT platform. It can be downloaded at: http://www.eeye.com/html/Databases/software/nmapNT/nmapNT.zip

2. Attack Description

nmapNT is a very customizable network scanner. It has various options to perform the following types of scans:

Vanilla TCP connect() scanning,
TCP SYN (half open) scanning,
TCP FIN, Xmas, or NULL (stealth) scanning,
TCP ftp proxy (bounce attack) scanning
SYN/FIN scanning using IP fragments (bypasses some packet filters),
TCP ACK and Window scanning,
UDP raw ICMP port unreachable scanning,
ICMP scanning (ping-sweep)
TCP Ping scanning
Direct (non portmapper) RPC scanning
Remote OS Identification by TCP/IP Fingerprinting, and
Reverse-ident scanning.

The following excerpt from nmap documentation (http://www.insecure.org/nmap/nmap-fingerprinting-article.html) clearly defines the different methods used in OS identification:

> The FIN probe -- Here we send a FIN packet (or any packet without an ACK or SYN flag) to an open port and wait for a response. The correct RFC793 behavior is to NOT respond, but many broken implementations such as MS Windows, BSDI, CISCO, HP/UX, MVS, and IRIX send a RESET back. Most current tools utilize this technique.
>
> The BOGUS flag probe -- Queso is the first scanner I have seen to use this clever test. The idea is to set an undefined TCP "flag" (64 or 128) in the TCP header of a SYN packet. Linux boxes prior to 2.0.35 keep the flag set in their response. I have not found any other OS to have this bug. However, some operating systems seem to reset the connection when they get a SYN+BOGUS packet. This behavior could be useful in identifying them.
>
> TCP ISN Sampling -- The idea here is to find patterns in the initial sequence numbers chosen by TCP implementations when responding to a connection request. These can be categorized in to many groups such as the traditional 64K (many old UNIX boxes), Random increments (newer versions of Solaris, IRIX, FreeBSD, Digital

UNIX, Cray, and many others), True "random" (Linux 2.0.*, OpenVMS, newer AIX, etc). Windows boxes (and a few others) use a "time dependent" model where the ISN is incremented by a small fixed amount each time period. Needless to say, this is almost as easily defeated as the old 64K behavior. Of course my favorite technique is "constant". The machines ALWAYS use the exact same ISN :). I've seen this on some 3Com hubs (uses 0x803) and Apple LaserWriter printers (uses 0xC7001).

You can also subclass groups such as random incremental by computing variances, greatest common divisors, and other functions on the set of sequence numbers and the differences between the numbers.

It should be noted that ISN generation has important security implications. For more information on this, contact "security expert" Tsutomu "Shimmy" Shimomura at SDSC and ask him how he was owned. Nmap is the first program I have seen to use this for OS identification.

Don't Fragment bit -- Many operating systems are starting to set the IP "Don't Fragment" bit on some of the packets they send. This gives various performance benefits (though it can also be annoying -- this is why nmap fragmentation scans do not work from Solaris boxes). In any case, not all OS's do this and some do it in different cases, so by paying attention to this bit we can glean even more information about the target OS. I haven't seen this one before either.

TCP Initial Window -- This simply involves checking the window size on returned packets. Older scanners simply used a non-zero window on a RST packet to mean "BSD 4.4 derived". Newer scanners such as queso and nmap keep track of the exact window since it is actually pretty constant by OS type. This test actually gives us a lot of information, since some operating systems can be uniquely identified by the window alone (for example, AIX is the only OS I have seen which uses 0x3F25). In their "completely rewritten" TCP stack for NT5, Microsoft uses 0x402E. Interestingly, that is exactly the number used by OpenBSD and FreeBSD.

ACK Value -- Although you would think this would be completely standard, implementations differ in what value they use for the ACK field in some cases. For example, lets say you send a FIN|PSH|URG to a closed TCP port. Most implementations will set the ACK to be the same as your initial sequence number, though Windows and some stupid printers will send your seq + 1. If you send a SYN|FIN|URG|PSH to an open port, Windows is very inconsistent. Sometimes it sends back your seq, other times it sends S++, and still other times is sends back a seemingly random value. One has to wonder what kind of code MS is writing that changes its mind like this.

ICMP Error Message Quenching -- Some (smart) operating systems follow the RFC 1812 suggestion to limit the rate at which various error messages are sent. For example, the Linux kernel (in net/ipv4/icmp.h) limits destination unreachable message generation

to 80 per 4 seconds, with a 1/4 second penalty if that is exceeded. One way to test this is to send a bunch of packets to some random high UDP port and count the number of unreachables received. I have not seen this used before, and in fact I have not added this to nmap (except for use in UDP port scanning). This test would make the OS detection take a bit longer since you need to send a bunch of packets and wait for them to return. Also dealing with the possibility of packets dropped on the network would be a pain.

ICMP Message Quoting -- The RFCs specify that ICMP error messages quote some small amount of an ICMP message that causes various errors. For a port unreachable message, almost all implementations send only the required IP header + 8 bytes back. However, Solaris sends back a bit more and Linux sends back even more than that. The beauty with this is it allows nmap to recognize Linux and Solaris hosts even if they don't have any ports listening.

ICMP Error message echoing integrity -- I got this idea from something Theo De Raadt (lead OpenBSD developer) posted to comp.security.unix. As mentioned before, machines have to send back part of your original message along with a port unreachable error. Yet some machines tend to use your headers as 'scratch space' during initial processing and so they are a bit warped by the time you get them back. For example, AIX and BSDI send back an IP 'total length' field that is 20 bytes too high. Some BSDI, FreeBSD, OpenBSD, ULTRIX, and VAXen fuck up the IP ID that you sent them. While the checksum is going to change due to the changed TTL anyway, there are some machines (AIX, FreeBSD, etc.) which send back an inconsistent or 0 checksum. Same thing goes with the UDP checksum. All in all, nmap does nine different tests on the ICMP errors to sniff out subtle differences like these.

Type of Service -- For the ICMP port unreachable messages I look at the type of service (TOS) value of the packet sent back. Almost all implementations use 0 for this ICMP error although Linux uses 0xC0. This does not indicate one of the standard TOS values, but instead is part of the unused (AFAIK) precedence field. I do not know why this is set, but if they change to 0 we will be able to keep identifying the old versions _and_ we will be able to identify between old and new.

Fragmentation Handling -- This is a favorite technique of Thomas H. Ptacek of Secure Networks, Inc (now owned by a bunch of Windows users at NAI). This takes advantage of the fact that different implementations often handle overlapping IP fragments differently. Some will overwrite the old portions with the new, and in other cases the old stuff has precedence. There are many different probes you can use to determine how the packet was reassembled. I did not add this capability since I know of no portable way to send IP fragments (in particular, it is a bitch on Solaris). For more information on overlapping fragments, you can read their IDS paper (www.secnet.com).

TCP Options -- These are truly a gold mine in terms of leaking

information. The beauty of these options is that:
1) They are generally optional (duh!) :) so not all hosts implement them.
2) You know if a host implements them by sending a query with an option set. The target generally show support of the option by setting it on the reply.
3) You can stuff a whole bunch of options on one packet to test everything at once.

Nmap sends these options along with almost every probe packet:

Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops;

When you get your response, you take a look at which options were returned and thus are supported. Some operating systems such as recent FreeBSD boxes support all of the above, while others, such as Linux 2.0.X support very few. The latest Linux 2.1.x kernels do support all of the above. On the other hand, they are more vulnerable to TCP sequence prediction. Go figure.

Even if several operating systems support the same set of options, you can sometimes distinguish them by the _values_ of the options. For example, if you send a small MSS value to a Linux box, it will generally echo that MSS back to you. Other hosts will give you different values.

And even if you get the same set of supported options AND the same values, you can still differentiate via the _order_ that the options are given, and where padding is applied. For example Solaris returns 'NNTNWME' which means:
<no op><no op><timestamp><no op><window scale><echoed MSS>

While Linux 2.1.122 returns MENNTNW. Same options, same values, but different order!

I have not seen any other OS detection tools utilizes TCP options, but it is very useful.

There are a few other useful options I might probe for at some point, such as those that support T/TCP and selective acknowledgements.

Exploit Chronology -- Even with all the tests above, nmap is unable to distinguish between the TCP stacks of Win95, WinNT, or Win98. This is rather surprising, especially since Win98 came out about 4 years after Win95. You would think they would have bothered to improve the stack in some way (like supporting more TCP options) and so we would be able to detect the change and distinguish the operating systems. Unfortunately, this is not the case. The NT stack is apparently the same crappy stack they put into '95. And they didn't bother to upgrade it for '98.

But do not give up hope, for there is a solution. You can simply start with early Windows DOS attacks (Ping of Death, Winnuke, etc) and move up a little further to attacks such as Teardrop and Land.

After each attack, ping them to see whether they have crashed. When you finally crash them, you will likely have narrowed what they are running down to one service pack or hotfix.

I have not added this functionality to nmap, although I must admit it is very tempting :).

SYN Flood Resistance -- Some operating systems will stop accepting new connections if you send too many forged SYN packets at them (forging the packets avoids trouble with your kernel resetting the connections). Many operating systems can only handle 8 packets. Recent Linux kernels (among other operating systems) allow various methods such as SYN cookies to prevent this from being a serious problem. Thus you can learn something about your target OS by sending 8 packets from a forged source to an open port and then testing whether you can establish a connection to that port yourself. This was not implemented in nmap since some people get upset when you SYN flood them. Even explaining that you were simply trying to determine what OS they are running might not help calm them.

NMAP IMPLEMENTATION AND RESULTS

I have created a reference implementation of the OS detection techniques mentioned above (except those I said were excluded). I have added this to my Nmap scanner which has the advantage that it already _knows_ what ports are open and closed for fingerprinting so you do not have to tell it. It is also portable among Linux, *BSD, and Solaris 2.51 and 2.6, and some other operating systems.

The new version of nmap reads a file filled with Fingerprint templates that follow a simple grammar. Here is an example:

```
FingerPrint  IRIX 6.2 - 6.4 # Thanks to Lamont Granquist
TSeq(Class=i800)
T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=C000|EF2A%ACK=O%Flags=A%Ops=NNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

Lets look at the first line (I'm adding '>' quote markers):

> FingerPrint IRIX 6.2 - 6.3 # Thanks to Lamont Granquist

This simply says that the fingerprint covers IRIX versions 6.2 through 6.3 and the comment states that Lamont Granquist kindly sent me the IP addresses or fingerprints of the IRIX boxes tested.

> TSeq(Class=i800)

This means that ISN sampling put it in the "i800 class". This means

that each new sequence number is a multiple of 800 greater than the last one.

> T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)

The test is named T1 (for test1, clever eh?). In this test we send a SYN packet with a bunch of TCP options to an open port. DF=N means that the "Don't fragment" bit of the response must not be set. W=C000|EF2A means that the window advertisement we received must be 0xC000 or EF2A. ACK=S++ means the acknowledgement we receive must be our initial sequence number plus 1. Flags = AS means the ACK and SYN flags were sent in the response. Ops = MNWNNT means the options in the response must be (in this order):

<MSS (not echoed)><NOP><Window scale><NOP><NOP><Timestamp>

> T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)

Test 2 involves a NULL with the same options to an open port. Resp=Y means we must get a response. Ops= means that there must not be any options included in the response packet. If we took out '%Ops=' entirely then any options sent would match.

> T3(Resp=Y%DF=N%W=400%ACK=S++%Flags=AS%Ops=M)

Test 3 is a SYN|FIN|URG|PSH w/options to an open port.

> T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

This is an ACK to an open port. Note that we do not have a Resp= here. This means that lack of a response (such as the packet being dropped on the network or an evil firewall) will not disqualify a match as long as all the other tests match. We do this because virtually any OS will send a response, so a lack of response is generally an attribute of the network conditions and not the OS itself. We put the Resp tag in tests 2 and 3 because some operating systems _do_ drop those without responding.

> T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
> T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
> T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)

These tests are a SYN, ACK, and FIN|PSH|URG, respectively, to a closed port. The same options as always are set. Of course this is all probably obvious given the descriptive names 'T5', 'T6', and 'T7' :).

> PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

This big sucker is the 'port unreachable' message test. You should recognize the DF=N by now. TOS=0 means that IP type of service field was 0. The next two fields give the (hex) values of the IP total length field of the message IP header and the total length given in the IP header they are echoing back to us. RID=E means the RID value we got back in the copy of our original UDP packet was expected (ie the same as we sent). RIPCK=E means they didn't fuck up the checksum (if they did, it would say RIPCK=F). UCK=E means the UDP checksum is

also correct. Next comes the UDP length which was 0x134 and DAT=E means they echoed our UDP data correctly. Since most implementations (including this one) do not send any of our UDP data back, they get DAT=E by default.

(excerpt from: Remote OS detection via TCP/IP Stack FingerPrinting
by Fyodor <fyodor@insecure.org> (www.insecure.org)
Written: October 18, 1998
Last Modified: April 10, 1999)

3. Annotated network trace

Three TCP scans were performed against ports 1-65355 against the target machine "workstation" running Windows NT 4.0 operating system. The computer running nmapNT was a Windows 2000 SP1 server named "the-brickyard".

The first scan, utilizing the TCP "Xmas tree" sets flags for FIN URG and PUSH. This pattern is used to evade routers and other filtering devices. Microsoft's implementation (or lack thereof) of RFC 793 pp64 renders this scan useless against Windows NT. The result is no ports are indicated as open. (Note that the options –vv and –dd do not operate properly in the command line below, they must be entered as –v –v and –d –d respectively. These options are for verbosity of output for the screen and the file dump.)

```
# Nmap (V. nmap) scan initiated 2.53 as: nmapnt -vv -sX -O -p 1-65535 -dd -oN hacker111 xxx.xxx.69.91
Warning:  No TCP ports found open on this machine, OS detection will be MUCH less reliable
All 65535 scanned ports on WORKSTATION (xxx.xxx.69.91) are: closed
Too many fingerprints match this host for me to give an accurate OS guess
# Nmap run completed at Tue Sep 19 12:10:04 2000 -- 1 IP address (1 host up) scanned in 35 seconds
```

The second scan, utilizing a "half open" TCP SYN scan indicated that three ports were open. Of particular concern was open port 12345, identified by nmapNT as NetBus.

```
E:\Secure\NMapNT>nmapnt -v -v -sS -O -p 1-65535 -d -oN hacker112 xxx.xxx.69.91
based on nmap by fyodor@insecure.org  ( www.insecure.org/nmap/ )

The first host is xxx, and the last one is xxx
The first host is xxx, and the last one is xxx
The first host is 69, and the last one is 69
The first host is 91, and the last one is 91
Packet capture filter: (icmp and dst host xxx.xxx.69.142)
We got a ping packet back from xxx.xxx.69.91: id = 3888 seq = 0 checksum = 61647
Hostupdate called for machine xxx.xxx.69.91 state UNKNOWN/COMBO -> HOST_UP (trynu
m 0, dotimeadj: yes time: 0)
Finished block: srtt: 0 rttvar: 0 timeout: 300000 block_tries: 1 up_this_block:
1 down_this_block: 0 group_sz: 1
massping done:  num_hosts: 1  num_responses: 1
Host WORKSTATION (xxx.xxx.69.91) appears to be up ... good.
Starting pos_scan
Packet capture filter: (icmp and dst host xxx.xxx.69.142) or (tcp and src host xx
x.xxx.69.91 and dst host xxx.xxx.69.142)
Initiating SYN half-open stealth scan against WORKSTATION (xxx.xxx.69.91)
Adding TCP port 139 (state open).
Adding TCP port 12345 (state open).
Adding TCP port 135 (state open).
Done with round 0
```

The SYN scan took 22 seconds to scan 65535 ports.
Wait time is 300
Packet capture filter: (icmp and dst host xxx.xxx.69.142) or (tcp and src host 19
6.69.69.91 and dst host xxx.xxx.69.142)
For OSScan assuming that port 135 is open and port 1 is closed and neither are f
irewalled
Interesting ports on WORKSTATION (xxx.xxx.69.91):
(The 65532 ports scanned but not shown below are in state: closed)
Port State Service
135/tcp open unknown
139/tcp open unknown
12345/tcp open NetBus

TCP Sequence Prediction: Class=trivial time dependency
Difficulty=10 (Easy)

Sequence numbers: A4245C A4247A A42490 A424AE A424F4 A42532
Remote operating system guess: Windows NT4 / Win95 / Win98
OS Fingerprint:
TSeq(Class=TD%gcd=2%SI=A)
T1(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)

Final times for host: srtt: 433 rttvar: 853 to: 300000
Nmap run completed -- 1 IP address (1 host up) scanned in 24 seconds

A complete virus scan of the target was performed immediately. No viruses or Trojans were found on the system. Nevertheless, this odd port was open and the investigation continued. Netstat.exe was run and provided the information below:

```
Proto  Local Address          Foreign Address        State
  TCP    workstation:135        0.0.0.0:0              LISTENING
  TCP    workstation:135        0.0.0.0:0              LISTENING
  TCP    workstation:161        0.0.0.0:0              LISTENING
  TCP    workstation:1029       0.0.0.0:0              LISTENING
  TCP    workstation:12345      0.0.0.0:0              LISTENING
  TCP    workstation:1026       0.0.0.0:0              LISTENING
  TCP    workstation:1026       localhost:1029         ESTABLISHED
  TCP    workstation:1029       localhost:1026         ESTABLISHED
  TCP    workstation:137        0.0.0.0:0              LISTENING
  TCP    workstation:138        0.0.0.0:0              LISTENING
  TCP    workstation:nbsession  0.0.0.0:0              LISTENING
  TCP    workstation:1027       0.0.0.0:0              LISTENING
  TCP    workstation:1030       0.0.0.0:0              LISTENING
  TCP    workstation:1030       SERVER1:nbsession      ESTABLISHED
  TCP    workstation:1031       0.0.0.0:0              LISTENING
  TCP    workstation:1031       SERVER2:nbsession      ESTABLISHED
  UDP    workstation:135        *:*
  UDP    workstation:snmp       *:*
```

```
  UDP    workstation:nbname      *:*
  UDP    workstation:nbdatagram  *:*
  UDP    workstation:1027        *:*
```

The services applet of Control Panel was opened on the workstation. Nothing unexpected was seen. It was noted that the OfficeScan NT listener service was running (see Detect 2). This service was stopped and netstat.exe was run again:

```
 Proto  Local Address          Foreign Address        State
  TCP    workstation:135        0.0.0.0:0              LISTENING
  TCP    workstation:135        0.0.0.0:0              LISTENING
  TCP    workstation:161        0.0.0.0:0              LISTENING
  TCP    workstation:1029       0.0.0.0:0              LISTENING
  TCP    workstation:1026       0.0.0.0:0              LISTENING
  TCP    workstation:1026       localhost:1029         ESTABLISHED
  TCP    workstation:1029       localhost:1026         ESTABLISHED
  TCP    workstation:137        0.0.0.0:0              LISTENING
  TCP    workstation:138        0.0.0.0:0              LISTENING
  TCP    workstation:nbsession  0.0.0.0:0              LISTENING
  TCP    workstation:1027       0.0.0.0:0              LISTENING
  TCP    workstation:1030       0.0.0.0:0              LISTENING
  TCP    workstation:1030       SERVER1:nbsession      ESTABLISHED
  TCP    workstation:1037       0.0.0.0:0              LISTENING
  TCP    workstation:1037       SERVER3:nbsession      ESTABLISHED
  TCP    workstation:1038       0.0.0.0:0              LISTENING
  TCP    workstation:1038       SERVER2:nbsession      ESTABLISHED
  UDP    workstation:135        *:*
  UDP    workstation:snmp       *:*
  UDP    workstation:nbname     *:*
  UDP    workstation:nbdatagram *:*
```

Port 12345 was not open any longer. Back to the nmapNT port scanning. An excerpt from windump follows.

Note the crafted TTL of the nmapNT source packet.

```
12:12:16.984184 xxx.xxx.69.142.42973 > xxx.xxx.69.91.139: S 1846888533:1846888533(
0) win 1024 (ttl 44, id 28988)
12:12:16.985211 xxx.xxx.69.91.139 > xxx.xxx.69.142.42973: S 1977333:1977333(0) ack
 1846888534 win 8576 <mss 1460> (DF) (ttl 128, id 19227)
12:12:16.985600 xxx.xxx.69.142.42973 > xxx.xxx.69.91.139: R 1846888534:1846888534(
0) win 0 (ttl 128, id 10080)
```

In the half open connection we see the first part of a three way handshake, the SYN from the source:

```
12:12:17.482997 xxx.xxx.69.142.42973 > xxx.xxx.69.91.135: S 1846888533:1846888533(
0) win 1024 (ttl 44, id 7744)
```

We follow with the SYN ACK:

```
12:12:17.483415 xxx.xxx.69.91.135 > xxx.xxx.69.142.42973: S 1977477:1977477(0) ack
 1846888534 win 8576 <mss 1460> (DF) (ttl 128, id 49954)
```

Then the half open connection is closed by nmapNT with a RST:

```
12:12:17.483559 xxx.xxx.69.142.42973 > xxx.xxx.69.91.135: R 1846888534:1846888534(
0) win 0 (ttl 128, id 10081)
```

Here we see a closed port RST the nmapNT's port scan SYN:

```
12:12:24.237816 xxx.xxx.69.142.42973 > xxx.xxx.69.91.136: S 1846888533:1846888533(
0) win 1024 (ttl 44, id 50459)
12:12:24.238545 xxx.xxx.69.91.136 > xxx.xxx.69.142.42973: R 0:0(0) ack 1846888534
win 0 (ttl 128, id 20855)
12:12:26.575772 xxx.xxx.69.142.42973 > xxx.xxx.69.91.12345: S 1846888533:184688853
3(0) win 1024 (ttl 44, id 11620)
12:12:26.576730 xxx.xxx.69.91.12345 > xxx.xxx.69.142.42973: S 1986580:1986580(0) a
ck 1846888534 win 8576 <mss 1460> (DF) (ttl 128, id 29330)
12:12:26.577133 xxx.xxx.69.142.42973 > xxx.xxx.69.91.12345: R 1846888534:184688853
4(0) win 0 (ttl 128, id 10082)
```

Port 135 is used for the OS fingerprinting and we see nmapNT sending different combinations of flags to port 135:

```
12:12:38.702447 xxx.xxx.69.142.42980 > xxx.xxx.69.91.135: S 403131312:403131312(0)
 win 1024 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 44, id 3183)
12:12:38.702821 xxx.xxx.69.142.42981 > xxx.xxx.69.91.135: . win 1024 <wscale 10,no
p,mss 265,timestamp 1061109567 0,eol> (ttl 44, id 36986)
12:12:38.702959 xxx.xxx.69.142.42982 > xxx.xxx.69.91.135: SFP 403131312:403131312(
0) win 1024 urg 0 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 44, id
 14616)
12:12:38.703108 xxx.xxx.69.142.42983 > xxx.xxx.69.91.135: . ack 0 win 1024 <wscale
 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 44, id 53608)
12:12:38.703959 xxx.xxx.69.91.135 > xxx.xxx.69.142.42980: S 1998676:1998676(0) ack
 403131313 win 8215 <mss 1460> (DF) (ttl 128, id 14856)
12:12:38.704000 xxx.xxx.69.91.135 > xxx.xxx.69.142.42981: R 0:0(0) ack 403131312 w
in 0 (ttl 128, id 15112)
12:12:38.704039 xxx.xxx.69.91.135 > xxx.xxx.69.142.42982: S 1998738:1998738(0) ack
 403131313 win 8215 <mss 1460> (DF) (ttl 128, id 15368)
12:12:38.704076 xxx.xxx.69.91.135 > xxx.xxx.69.142.42983: R 0:0(0) win 0 (ttl 128,
 id 15624)
12:12:38.704593 xxx.xxx.69.142.42980 > xxx.xxx.69.91.135: R 403131313:403131313(0)
 win 0 (ttl 128, id 10083)
12:12:38.704819 xxx.xxx.69.142.42982 > xxx.xxx.69.91.135: R 403131313:403131313(0)
 win 0 (ttl 128, id 10084)
12:12:38.704957 xxx.xxx.69.142.42974 > xxx.xxx.69.91.135: S 403131313:403131313(0)
 win 1024 (ttl 44, id 31777)
12:12:38.705226 xxx.xxx.69.91.135 > xxx.xxx.69.142.42974: S 2126759:2126759(0) ack
 403131314 win 8576 <mss 1460> (DF) (ttl 128, id 16904)
12:12:38.705514 xxx.xxx.69.142.42974 > xxx.xxx.69.91.135: R 403131314:403131314(0)
 win 0 (ttl 128, id 10085)
12:12:38.733203 xxx.xxx.69.142.42975 > xxx.xxx.69.91.135: S 403131314:403131314(0)
 win 1024 (ttl 44, id 46899)
12:12:38.733547 xxx.xxx.69.91.135 > xxx.xxx.69.142.42975: S 2126822:2126822(0) ack
 403131315 win 8576 <mss 1460> (DF) (ttl 128, id 17160)
12:12:38.734824 xxx.xxx.69.142.42975 > xxx.xxx.69.91.135: R 403131315:403131315(0)
 win 0 (ttl 128, id 10086)
12:12:38.764438 xxx.xxx.69.142.42976 > xxx.xxx.69.91.135: S 403131315:403131315(0)
 win 1024 (ttl 44, id 12097)
12:12:38.764774 xxx.xxx.69.91.135 > xxx.xxx.69.142.42976: S 2126744:2126744(0) ack
 403131316 win 8576 <mss 1460> (DF) (ttl 128, id 17416)
12:12:38.766104 xxx.xxx.69.142.42976 > xxx.xxx.69.91.135: R 403131316:403131316(0)
 win 0 (ttl 128, id 10087)
12:12:38.795705 xxx.xxx.69.142.42977 > xxx.xxx.69.91.135: S 403131316:403131316(0)
 win 1024 (ttl 44, id 15967)
12:12:38.796051 xxx.xxx.69.91.135 > xxx.xxx.69.142.42977: S 2126786:2126786(0) ack
 403131317 win 8576 <mss 1460> (DF) (ttl 128, id 17672)
12:12:38.797315 xxx.xxx.69.142.42977 > xxx.xxx.69.91.135: R 403131317:403131317(0)
 win 0 (ttl 128, id 10088)
12:12:38.827063 xxx.xxx.69.142.42978 > xxx.xxx.69.91.135: S 403131317:403131317(0)
 win 1024 (ttl 44, id 55608)
12:12:38.827433 xxx.xxx.69.91.135 > xxx.xxx.69.142.42978: S 2126820:2126820(0) ack
 403131318 win 8576 <mss 1460> (DF) (ttl 128, id 17928)
```

```
12:12:38.828444 xxx.xxx.69.142.42978 > xxx.xxx.69.91.135: R 403131318:403131318(0)
 win 0 (ttl 128, id 10089)
12:12:38.859005 xxx.xxx.69.142.42979 > xxx.xxx.69.91.135: S 403131318:403131318(0)
 win 1024 (ttl 44, id 8763)
12:12:38.859376 xxx.xxx.69.91.135 > xxx.xxx.69.142.42979: S 2126846:2126846(0) ack
 403131319 win 8576 <mss 1460> (DF) (ttl 128, id 18184)
12:12:38.860649 xxx.xxx.69.142.42979 > xxx.xxx.69.91.135: R 403131319:403131319(0)
 win 0 (ttl 128, id 10090)
```

The third scan, utilizing the TCP FIN packet yielded similar results to the XMAS tree packet, no ports are indicated as open. Snort, tcpdump and windump can be utilized to detect the odd TCP flag combinations that have shown mixed scanning results on Windows NT.

Nmap (V. nmap) scan initiated 2.53 as: nmapnt -sF -O -p 1-65535 -oN hacker113 xxx.xxx.69.91
Warning: No TCP ports found open on this machine, OS detection will be MUCH less reliable
All 65535 scanned ports on WORKSTATION (xxx.xxx.69.91) are: closed
Too many fingerprints match this host for me to give an accurate OS guess
Nmap run completed at Tue Sep 19 13:24:23 2000 -- 1 IP address (1 host up) scanned in 31 seconds

Assignment 3 – "Analyze This" Scenario

Intrusion Detection Summary

The subject network was equipped with a snort based intrusion detection system. Snort was configured with a standard rule set in an effort to determine the type and extent of malicious traffic on the network.

A number of vulnerabilities and poor system administration practices and policies were discovered. The items are detailed in the sections that follow. Of primary concern is traffic on ports 34555 and 35555 on a handful of machines on the network. These ports are utilized by Wintrinoo, a distributed denial of service (DDos) tool. The size of the network makes it an extremely effective army of zombies if compromised.

http://www.sans.org/dosstep/index.htm should be reviewed and implemented immediately in an effort to reduce the chances of the network being used to harm other networks. The Consensus Roadmap for Defeating Distributed Denial of Service Attacks http://www.sans.org/ddos_roadmap.htm should be reviewed and implemented for the more complete long-term effort required to reduce the risks of DDos.

Napster (port 6699)

On August 8 over 8500 packets were detected in 30 minutes to the same machine. The destination port of 6699 is commonly used by Napster (see www.napster.com). Napster.com is a site where users share MP3 files. These files can be quite large and consume a lot of bandwidth. There are also security concerns as the user has allowed his computer to share files with anyone on napster. Furthermore, this detect may be a denial of service against the client computer. Two CVE candidates are relevant to napster:

CAN-2000-0281
** CANDIDATE (under review) ** Buffer overflow in the Napster client beta 5 allows remote attackers to cause a denial of service via a long message.
CAN-2000-0412
** CANDIDATE (under review) ** The gnapster and knapster clients for Napster do not properly restrict access only to MP3 files, which allows remote attackers to read arbitrary files from the client by specifying the full pathname for the file.

```
08/01-01:21:06.844268  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:07.240098  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
```

```
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:09.517656  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:09.944275  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:10.102194  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:10.678676  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:10.685720  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
08/01-01:21:10.989105  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
```

This pattern continues until the final packet detected below:

```
08/01-01:52:38.166986  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.38.141:2792 -> MY.NET.217.38:6699
```

The origin of these packets was from ISDNNET based in Israel. The contact information is as follows:

```
route:        212.179.0.0/17
descr:        ISDN Net Ltd.
origin:       AS8551
notify:       hostmaster@isdn.net.il
mnt-by:       AS8551-MNT
changed:      hostmaster@isdn.net.il 19990610
source:       RIPE

person:       Nati Pinko
address:      Bezeq International
address:      40 Hashacham St.
address:      Petach Tikvah  Israel
phone:        +972 3 9257761
e-mail:       hostmaster@isdn.net.il
nic-hdl:      NP469-RIPE
changed:      registrar@ns.il 19990902
source:       RIPE
```

The source network is known to the Intrusion Detection community and thus is on our watchlist. Suspicious activity from ISDNNET can be found at:
http://www.sans.org/y2k/090500-1200.htm http://www.sans.org/y2k/052000.htm
http://www.sans.org/y2k/051900.htm
http://www.sans.org/y2k/043000.htm http://www.sans.org/y2k/033000-2300.htm
http://www.sans.org/y2k/032700-2000.htm
http://www.sans.org/y2k/032500-2200.htm http://www.sans.org/y2k/032200-1700.htm

Watchlists

Known networks were found targeting the network. In this case, users from the Institute of Computing Technology Chinese Academy of Sciences are probing the network.

```
08/01-01:15:12.412095  [**] Watchlist 000222 NET-NCFC [**] 159.226.91.37:2367 ->
MY.NET.100.230:25
```

```
08/01-01:15:18.808184  [**] Watchlist 000222 NET-NCFC [**] 159.226.91.37:113 ->
MY.NET.100.230:53364
08/01-04:01:24.105795  [**] Watchlist 000222 NET-NCFC [**] 159.226.228.1:2280 ->
MY.NET.100.230:25
08/01-05:13:42.497549  [**] Watchlist 000222 NET-NCFC [**] 159.226.67.61:3580 ->
MY.NET.253.43:25
08/01-05:13:50.727397  [**] Watchlist 000222 NET-NCFC [**] 159.226.67.61:113 ->
MY.NET.253.43:41155
08/01-08:14:33.739196  [**] Watchlist 000222 NET-NCFC [**] 159.226.116.129:113 ->
MY.NET.253.41:45763
08/01-08:14:35.196633  [**] Watchlist 000222 NET-NCFC [**] 159.226.116.129:2662 -
> MY.NET.253.41:25
08/01-09:55:22.017658  [**] Watchlist 000222 NET-NCFC [**] 159.226.224.1:58505 ->
MY.NET.100.230:113
```

Distributed Denial of Service "Zombies"

The following communications to ports 34555 and 35555 are indicative of Wintrinoo, a distributed denial of service (DDos) tool. If the network is compromised by Wintrinoo, you may unwittingly become a participant in a DDos attack against a large organization, similar to recent attacks against eBay and Yahoo.

```
08/01-01:22:53.802533  [**] GIAC 000218 VA-CIRT port 34555 [**] 24.2.2.66:113 ->
MY.NET.6.47:34555
08/01-02:04:37.380617  [**] GIAC 000218 VA-CIRT port 35555 [**] 63.211.173.17:25
-> MY.NET.6.34:35555
08/01-03:51:11.666649  [**] GIAC 000218 VA-CIRT port 35555 [**] 128.11.68.209:25
-> MY.NET.6.47:35555
08/01-06:03:56.320188  [**] GIAC 000218 VA-CIRT port 34555 [**] 169.226.1.24:113
-> MY.NET.253.42:34555
08/01-10:24:53.489225  [**] GIAC 000218 VA-CIRT port 34555 [**] 209.27.89.5:25 ->
MY.NET.253.51:34555
```

There is one relevant CVE candidate relevant to Wintrinoo:

CAN-2000-0138
** CANDIDATE (under review) ** A system has a distributed denial of service (DDOS) attack master, agent, or zombie installed, such as (1) Trinoo, (2) Tribe Flood Network (TFN), (3) Tribe Flood Network 2000 (TFN2K), (4) stacheldraht, (5) mstream, or (6) shaft.

http://www.sans.org/newlook/resources/IDFAQ/trinoo.htm provides more information on Trinoo and Wintrinoo and the following excerpt provides a link for you to immediately scan for Wintrinoo:

```
The best defense against DDos attacks is to prevent initial system compromises.
Generally, this involves installing patches, anti virus software, using a
firewall and monitoring for intruders. However, even vigilant hosts can become
targets because of lesser prepared, less security aware hosts (especially if
these hosts have always-on high-speed internet connections). Many systems are
compromised because patches for vulnerabilities reported and fixed months
beforehand were never installed. Similarly, such systems have anti-virus software
that is not up to date.

It is difficult to specifically defend against becoming the ultimate target of a
DDos attack but protection against being used as a daemon or master system is
more easily attainable. To this end, the following measures should be met (Gary
Flynn, 2000):
```

Check for frequent patches and subscribe to automatic vendor notifications
Attempt to understand the vulnerabilities in your software and configuration
Disable unnecessary network software
Only accept program files from trusted sources (or at least be cautious)

For Unix operators:

Limit accessibility with network access control tools e.g. TCP Wrappers
Use file system integrity checks e.g. Tripwire
Download programs to test for common DDos attacks. For example:
http://www.fbi.gov/nipc/trinoo.htm for Sun and Linux boxes *(NOTE: this link is unreliable or not currently working...comment by Stephan Odak)*
http://www.theorygroup.com/Software/RID for all unix platforms. (Remote Intrusion Detector for detecting trinoo, TFN and stacheldraht DDos tools).

For Windows operators:

Keep anti-virus (e.g. Norton) and anti-trojan (e.g. BOClean) software up to date
Disable scripting on browsers and e-mail clients
Run a desktop firewall
Download Wtrinscan.exe which scans for wintrinoo
http://www.jmu.edu/info-security/engineering/tools/wtrinscan.exe

Active Targeting

A number of narrow, targeted portscans have been made to individual computers. In the case of the scanning by 207.0.62.254 (address block managed by Cybermall Enterprises), 3 IP addresses were targeted on two different occasions. Most of the scans below are likely checking to see if machines with known vulnerabilities (or otherwise compromised systems) are on line. It is probable that they have been identified by a broad scan previously performed and the attacker found a vulnerability or has compromised the system. Note that the SYN-FIN and Null scans are stealth attempts to pass through firewalls and packet filters.

```
08/01-00:18:16.304736  [**] spp_portscan: End of portscan from 207.0.62.254
(TOTAL HOSTS:3 TCP:3 UDP:0) [**]
08/01-00:14:32.088386  [**] SYN-FIN scan! [**] 207.0.62.254:9704 ->
MY.NET.1.5:9704
08/01-00:23:05.674383  [**] spp_portscan: End of portscan from 207.0.62.254
(TOTAL HOSTS:3 TCP:3 UDP:0) [**]
08/01-02:02:24.308734  [**] spp_portscan: End of portscan from 209.138.191.140
(TOTAL HOSTS:1 TCP:29 UDP:0) [**]
08/01-02:32:42.999181  [**] spp_portscan: End of portscan from 24.108.186.212
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-07:03:20.295356  [**] spp_portscan: End of portscan from MY.NET.1.3 (TOTAL
HOSTS:1 TCP:0 UDP:27) [**]
08/01-09:03:15.626667  [**] spp_portscan: End of portscan from 202.99.227.100
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-10:55:49.736342  [**] spp_portscan: End of portscan from 212.95.73.171
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-11:02:28.059772  [**] spp_portscan: End of portscan from 212.95.73.171
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-12:10:02.130309  [**] spp_portscan: End of portscan from 172.166.156.239
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-12:17:22.751538  [**] spp_portscan: End of portscan from MY.NET.1.3 (TOTAL
HOSTS:1 TCP:0 UDP:8) [**]
08/01-12:53:59.764631  [**] spp_portscan: End of portscan from 24.30.146.190
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
```

```
08/01-13:08:27.172543  [**] spp_portscan: End of portscan from 24.30.146.190
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-13:54:22.451173  [**] spp_portscan: End of portscan from 24.24.32.244
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-13:55:14.259821  [**] spp_portscan: End of portscan from 24.24.32.244
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-13:55:20.955898  [**] spp_portscan: End of portscan from 24.24.32.244
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-13:56:24.769657  [**] spp_portscan: End of portscan from 24.24.32.244
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-13:56:57.130833  [**] spp_portscan: End of portscan from 24.24.32.244
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-14:25:02.583950  [**] Null scan! [**] 207.50.54.6:2558 -> MY.NET.53.28:4362
08/01-14:37:58.039182  [**] spp_portscan: End of portscan from 207.50.54.6 (TOTAL
HOSTS:1 TCP:1 UDP:0) [**]
08/01-15:19:37.268970  [**] spp_portscan: End of portscan from 134.147.63.214
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
08/01-15:23:04.657117  [**] spp_portscan: End of portscan from 194.251.100.177
(TOTAL HOSTS:1 TCP:1 UDP:0) [**]
```

Broad Network scans

There are many broad scans of the network to as many as 5,595 hosts per scan. The source of most of these scans is from an address block assigned to the University of California at Berkley (169.229.88.203). Michigan State University (35.10.82.105) was the source of a number of large scans as well.

```
08/01-00:18:09.554918  [**] spp_portscan: End of portscan from 213.188.8.45
(TOTAL HOSTS:51 TCP:47 UDP:0) [**]
08/01-04:31:13.449828  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:506 TCP:508 UDP:0) [**]
08/01-04:31:17.165285  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:506 TCP:508 UDP:0) [**]
08/01-04:31:21.062433  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:1053 TCP:1058 UDP:0) [**]
08/01-04:31:22.014691  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:2276 TCP:2289 UDP:0) [**]
08/01-04:31:26.337702  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:1156 TCP:1162 UDP:0) [**]
08/01-04:31:26.947849  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:609 TCP:614 UDP:0) [**]
08/01-04:31:28.917058  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:296 TCP:298 UDP:0) [**]
08/01-04:31:30.073422  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:599 TCP:604 UDP:0) [**]
08/01-04:31:32.326283  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:399 TCP:401 UDP:0) [**]
08/01-04:31:32.675299  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:356 TCP:358 UDP:0) [**]
08/01-04:31:36.659479  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:409 TCP:412 UDP:0) [**]
08/01-04:31:56.556335  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:503 TCP:506 UDP:0) [**]
08/01-04:32:01.183289  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:451 TCP:453 UDP:0) [**]
08/01-04:32:03.357418  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:193 TCP:194 UDP:0) [**]
08/01-04:32:08.288872  [**] spp_portscan: End of portscan from 169.229.88.203
```

```
(TOTAL HOSTS:224 TCP:225 UDP:0) [**]
08/01-04:32:17.606917  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:840 TCP:845 UDP:0) [**]
08/01-04:32:26.318566  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:875 TCP:880 UDP:0) [**]
08/01-04:32:26.661263  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:364 TCP:366 UDP:0) [**]
08/01-04:32:32.122801  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:447 TCP:449 UDP:0) [**]
08/01-04:32:36.068283  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:253 TCP:254 UDP:0) [**]
08/01-04:32:39.612911  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:253 TCP:254 UDP:0) [**]
08/01-04:32:45.827456  [**] spp_portscan: End of portscan from 35.10.82.105
(TOTAL HOSTS:171 TCP:172 UDP:0) [**]
08/01-04:32:52.402828  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:230 TCP:231 UDP:0) [**]
08/01-04:33:02.031408  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:811 TCP:816 UDP:0) [**]
08/01-04:33:19.440990  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:1837 TCP:1849 UDP:0) [**]
08/01-04:33:22.672704  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:210 TCP:211 UDP:1) [**]
08/01-04:33:27.667904  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:215 TCP:216 UDP:0) [**]
08/01-04:33:37.446331  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:205 TCP:206 UDP:0) [**]
08/01-04:33:51.111527  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:1829 TCP:1840 UDP:0) [**]
08/01-04:33:57.428899  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:888 TCP:893 UDP:0) [**]
08/01-04:34:07.967382  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:1826 TCP:1838 UDP:0) [**]
08/01-04:34:33.474693  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:2569 TCP:2584 UDP:0) [**]
08/01-04:34:34.977051  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:253 TCP:254 UDP:0) [**]
08/01-04:46:23.856457  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:5959 TCP:5993 UDP:0) [**]
08/01-04:46:39.157472  [**] spp_portscan: End of portscan from 169.229.88.203
(TOTAL HOSTS:299 TCP:301 UDP:0) [**]
```

Wingate Telnet Proxy

The Wingate Telnet proxy is a common target for hackers. The are a number of vulnerabilities associated with Wingate and poor installation and administration can allow a system compromise. There are four relevant CVE entries and one CVE candidate relevant to Wingate:

CVE-1999-0290
The WinGate telnet proxy allows remote attackers to cause a denial of service via a large number of connections to localhost.
CVE-1999-0291
The WinGate proxy is installed without a password, which allows remote attackers to redirect connections without authentication.
CVE-1999-0441
Remote attackers can perform a denial of service in WinGate machines using a buffer overflow in the Winsock

Redirector Service.
CVE-1999-0494
Denial of service in WinGate proxy through a buffer overflow in POP3.
CAN-1999-0657
** CANDIDATE (under review) ** WinGate is being used.

```
08/01-00:07:32.008919  [**] WinGate 1080 Attempt [**] 161.58.8.77:20 ->
MY.NET.98.117:1080
08/01-00:43:09.660050  [**] WinGate 1080 Attempt [**] 216.35.103.80:35926 ->
MY.NET.100.203:1080
08/01-01:12:45.392236  [**] WinGate 1080 Attempt [**] 202.212.5.30:46984 ->
MY.NET.100.203:1080
08/01-01:13:00.154420  [**] WinGate 1080 Attempt [**] 216.67.50.180:1476 ->
MY.NET.60.11:1080
08/01-01:57:35.656300  [**] WinGate 1080 Attempt [**] 212.72.75.236:2104 ->
MY.NET.217.162:1080
08/01-02:01:26.731988  [**] WinGate 1080 Attempt [**] 216.35.103.81:58816 ->
MY.NET.100.203:1080
08/01-02:16:28.468476  [**] WinGate 1080 Attempt [**] 212.72.75.236:3892 ->
MY.NET.60.16:1080
08/01-02:41:01.744921  [**] WinGate 1080 Attempt [**] 194.75.152.237:46627 ->
MY.NET.98.152:1080
08/01-04:39:53.760486  [**] WinGate 1080 Attempt [**] 63.168.242.5:19617 ->
MY.NET.98.147:1080
08/01-04:39:53.875562  [**] WinGate 1080 Attempt [**] 216.198.1.4:3898 ->
MY.NET.98.147:1080
08/01-04:39:54.030175  [**] WinGate 1080 Attempt [**] 216.217.216.5:4772 ->
MY.NET.98.147:1080
08/01-04:39:54.701659  [**] WinGate 1080 Attempt [**] 24.165.34.160:3531 ->
MY.NET.98.147:1080
08/01-05:18:36.799981  [**] WinGate 1080 Attempt [**] 216.152.64.153:36634 ->
MY.NET.60.11:1080
08/01-06:11:35.595816  [**] WinGate 1080 Attempt [**] 216.35.103.62:53451 ->
MY.NET.100.203:1080
08/01-07:30:58.052565  [**] WinGate 1080 Attempt [**] 209.10.218.250:3880 ->
MY.NET.60.8:1080
08/01-07:41:23.346605  [**] WinGate 1080 Attempt [**] 206.204.3.253:20 ->
MY.NET.98.202:1080
08/01-07:57:09.351795  [**] WinGate 1080 Attempt [**] 207.114.4.46:1342 ->
MY.NET.98.129:1080
08/01-07:57:27.376354  [**] WinGate 1080 Attempt [**] 195.139.15.242:1544 ->
MY.NET.98.129:1080
08/01-08:05:34.761407  [**] WinGate 1080 Attempt [**] 216.179.0.37:2618 ->
MY.NET.60.11:1080
08/01-08:29:06.644292  [**] WinGate 1080 Attempt [**] 216.179.0.37:2880 ->
MY.NET.60.8:1080
08/01-09:11:40.552636  [**] WinGate 1080 Attempt [**] 216.152.64.153:60626 ->
MY.NET.60.11:1080
08/01-09:16:03.278066  [**] WinGate 1080 Attempt [**] 194.75.152.237:41301 ->
MY.NET.60.11:1080
08/01-09:44:20.740906  [**] WinGate 1080 Attempt [**] 207.175.201.247:1780 ->
MY.NET.60.11:1080
08/01-09:59:32.718291  [**] WinGate 1080 Attempt [**] 216.179.0.37:4224 ->
MY.NET.60.11:1080
08/01-10:06:12.405915  [**] WinGate 1080 Attempt [**] 24.177.5.45:20 ->
MY.NET.217.26:1080
08/01-10:15:16.225937  [**] WinGate 1080 Attempt [**] 216.67.50.203:1226 ->
```

```
MY.NET.60.8:1080
08/01-10:15:17.214203  [**] WinGate 1080 Attempt [**] 207.175.201.247:1983 ->
MY.NET.60.8:1080
08/01-10:34:46.643575  [**] WinGate 1080 Attempt [**] 216.179.0.37:4812 ->
MY.NET.60.16:1080
08/01-10:40:02.055996  [**] WinGate 1080 Attempt [**] 205.252.89.153:4882 ->
MY.NET.97.168:1080
08/01-10:47:42.620855  [**] WinGate 1080 Attempt [**] 24.0.96.240:4540 ->
MY.NET.70.93:1080
08/01-10:55:58.194922  [**] WinGate 1080 Attempt [**] 207.175.201.247:2129 ->
MY.NET.97.228:1080
08/01-11:18:17.158370  [**] WinGate 1080 Attempt [**] 216.67.82.224:1160 ->
MY.NET.60.16:1080
08/01-11:20:36.499623  [**] WinGate 1080 Attempt [**] 216.67.82.224:1178 ->
MY.NET.60.8:1080
08/01-11:45:06.282638  [**] WinGate 1080 Attempt [**] 216.67.82.224:1302 ->
MY.NET.60.11:1080
08/01-11:45:06.300192  [**] WinGate 1080 Attempt [**] 207.175.201.247:2282 ->
MY.NET.60.11:1080
08/01-11:58:40.868228  [**] WinGate 1080 Attempt [**] 212.72.75.236:3012 ->
MY.NET.217.162:1080
08/01-12:04:50.782498  [**] WinGate 1080 Attempt [**] 216.198.1.4:1383 ->
MY.NET.97.234:1080
08/01-12:04:55.509498  [**] WinGate 1080 Attempt [**] 63.168.242.5:6240 ->
MY.NET.97.234:108
08/01-12:20:07.081484  [**] WinGate 1080 Attempt [**] 63.168.242.5:5212 ->
MY.NET.97.234:1080
08/01-12:20:07.465662  [**] WinGate 1080 Attempt [**] 24.165.34.160:1213 ->
MY.NET.97.234:1080
08/01-12:20:09.008815  [**] WinGate 1080 Attempt [**] 216.198.1.4:2125 ->
MY.NET.97.234:1080
08/01-12:28:00.051999  [**] WinGate 1080 Attempt [**] 212.72.75.236:2054 ->
MY.NET.217.162:1080
08/01-12:39:14.345581  [**] WinGate 1080 Attempt [**] 207.114.4.46:4513 ->
MY.NET.98.202:1080
08/01-12:48:19.334370  [**] WinGate 1080 Attempt [**] 216.179.0.37:3532 ->
MY.NET.60.11:1080
08/01-12:51:37.019353  [**] WinGate 1080 Attempt [**] 216.179.0.37:3589 ->
MY.NET.60.11:1080
08/01-13:11:49.778094  [**] WinGate 1080 Attempt [**] 207.175.201.247:2590 ->
MY.NET.98.135:1080
08/01-13:11:50.135059  [**] WinGate 1080 Attempt [**] 216.67.82.5:1171 ->
MY.NET.98.135:1080
08/01-13:14:08.803265  [**] WinGate 1080 Attempt [**] 207.175.201.247:2596 ->
MY.NET.60.16:1080
08/01-13:14:11.702188  [**] WinGate 1080 Attempt [**] 216.67.82.5:1213 ->
MY.NET.60.16:1080
08/01-13:17:16.551829  [**] WinGate 1080 Attempt [**] 206.50.68.20:3270 ->
MY.NET.60.11:1080
08/01-14:04:19.889365  [**] WinGate 1080 Attempt [**] 207.114.4.46:2649 ->
MY.NET.53.48:1080
08/01-14:11:55.762255  [**] WinGate 1080 Attempt [**] 207.175.201.247:2837 ->
MY.NET.60.16:1080
08/01-14:11:56.786286  [**] WinGate 1080 Attempt [**] 216.67.82.5:1749 ->
MY.NET.60.16:1080
08/01-14:33:27.669534  [**] WinGate 1080 Attempt [**] 64.86.6.250:4576 ->
MY.NET.53.29:1080
```

```
08/01-14:34:23.153586  [**] WinGate 1080 Attempt [**] 63.160.118.40:2226 ->
MY.NET.53.29:1080
08/01-14:37:05.580451  [**] WinGate 1080 Attempt [**] 216.165.129.80:2679 ->
MY.NET.98.183:1080
08/01-14:37:47.611639  [**] WinGate 1080 Attempt [**] 205.251.208.49:3868 ->
MY.NET.98.183:1080
08/01-15:25:52.684900  [**] WinGate 1080 Attempt [**] 206.50.68.20:4865 ->
MY.NET.97.154:1080
08/01-15:31:49.004186  [**] WinGate 1080 Attempt [**] 193.166.0.134:1283 ->
MY.NET.60.8:1080
08/01-15:35:22.546485  [**] WinGate 1080 Attempt [**] 207.25.80.132:57950 ->
MY.NET.97.201:1080
08/01-15:53:25.149150  [**] WinGate 1080 Attempt [**] 207.175.201.247:3223 ->
MY.NET.60.11:1080
08/01-15:56:34.728761  [**] WinGate 1080 Attempt [**] 216.67.50.65:1228 ->
MY.NET.60.11:1080
08/01-15:57:22.103686  [**] WinGate 1080 Attempt [**] 207.175.201.247:3234 ->
MY.NET.60.8:1080
08/01-15:57:24.143662  [**] WinGate 1080 Attempt [**] 216.67.50.65:1230 ->
MY.NET.60.8:1080
08/01-16:02:13.062332  [**] WinGate 1080 Attempt [**] 207.175.201.247:3255 ->
MY.NET.60.16:1080
08/01-16:02:13.659280  [**] WinGate 1080 Attempt [**] 216.67.50.65:1254 ->
MY.NET.60.16:1080
08/01-16:19:57.825357  [**] WinGate 1080 Attempt [**] 168.120.16.250:47893 ->
MY.NET.97.221:1080
08/01-17:09:07.397787  [**] WinGate 1080 Attempt [**] 63.160.116.165:1088 ->
MY.NET.53.29:1080
```

SNMP policies

Internal network traffic indicates that the default community string “public” has not been renamed. This can allow internal users to map out network devices and obtain detailed information about the devices. There are four relevant CVE entries and nine CVE candidates relevant to SNMP:

CVE-1999-0294
All records in a WINS database can be deleted through SNMP for a denial of service.
CVE-1999-0472
The SNMP default community name "public" is not properly removed in NetApps C630 Netcache, even if the administrator tries to disable it.
CVE-2000-0221
The Nautica Marlin bridge allows remote attackers to cause a denial of service via a zero length UDP packet to the SNMP port.
CVE-2000-0379
The Netopia R9100 router does not prevent authenticated users from modifying SNMP tables, even if the administrator has configured it to do so.
CAN-1999-0186
** CANDIDATE (under review) ** In Solaris, an SNMP subagent has a default community string that allows remote attackers to execute arbitrary commands as root, or modify system parameters.
CAN-1999-0254
** CANDIDATE (under review) ** A hidden SNMP community string in HP OpenView allows remote attackers to modify MIB tables and obtain sensitive information.
CAN-1999-0499
** CANDIDATE (under review) ** NETBIOS share information may be published through SNMP registry keys in NT.
CAN-1999-0516

** CANDIDATE (under review) ** An SNMP community name is guessable.
CAN-1999-0517
**** CANDIDATE (under review) ** An SNMP community name is the default (e.g. public), null, or missing.**
CAN-1999-0615
** CANDIDATE (under review) ** The SNMP service is running.
CAN-1999-0792
** CANDIDATE (under review) ** ROUTERmate has a default SNMP community name which allows remote attackers to modify its configuration.
CAN-2000-0147
** CANDIDATE (under review) ** snmpd in SCO OpenServer has an SNMP community string that is writable by default, which allows local attackers to modify the host's configuration.
CAN-2000-0515
** CANDIDATE (under review) ** The snmpd.conf configuration file for the SNMP daemon (snmpd) in HP-UX 11.0 is world writable, which allows local users to modify SNMP configuration or gain privileges.

```
08/01-09:00:25.247478  [**] SNMP public access [**] MY.NET.98.148:1082 ->
MY.NET.101.192:161
08/01-09:00:25.494896  [**] SMB Name Wildcard [**] MY.NET.101.160:137 ->
MY.NET.101.192:137
08/01-09:21:47.490084  [**] SNMP public access [**] MY.NET.97.176:1143 ->
MY.NET.101.192:161
08/01-09:21:49.421022  [**] SMB Name Wildcard [**] MY.NET.101.160:137 ->
MY.NET.101.192:137
08/01-12:18:32.492268  [**] SNMP public access [**] MY.NET.97.186:1601 ->
MY.NET.101.192:161
08/01-12:18:34.422347  [**] SMB Name Wildcard [**] MY.NET.101.160:137 ->
MY.NET.101.192:137
08/01-13:37:59.716824  [**] SNMP public access [**] MY.NET.97.191:1835 ->
MY.NET.101.192:161
```

Assignment 4 – Analysis Process

The process used to analyze the data in assignment 3 was quite rudimentary. I have yet to implement an appropriate intrusion detection system in my organization due to fiscal limitations by my organization. With the recent port of snort to a win32 platform, I will be able to implement an IDS in my organization when I come up with a sensor or two. I would refer you to detect #2 of my practical for an in depth analyses of the thought process to evaluate what resulted in a false alarm, and how that was determined as well as assignment #2 which also contains an in depth look at the thought process and tools used to reach a final determination of another false alarm.

As for assignment #3, and the detects analyzed, a number of tools were utilized. The first was a whois search of the source address, and in this case I used the whois search engine located at:
http://www.geektools.com/cgi-bin/proxy.cgi .
This would provide the information on the source address including the registrant of a partial address block of a larger address block.

I also utilized the search link at http://www.sans.org/search_query to search for reported detects, IP addresses of attackers and ports relating to the detects in question. This is my favorite tool as it has the input of all our community that participates in sharing information.

To investigate whether a vulnerability or compromise is known to our community, I run a search at http://www.cve.mitre.org/cve/ to see if there are any CVE entries or candidates regarding the problem. Additionally, CVE often references Bugtraq messages that inform our community of vulnerabilities. A search of the title for Bugtraq archives can be performed from http://www.securityfocus.com/bugtraq/archive .
Select -BUGTRAQ from the drop down list box the initially indicates Entire Site, and then input the title of the item as indicated by CVE and click Search.