Mike Strube
Practical Assignment for SANS Network Security 2000, Monterey
GIAC Intrusion Detection Curriculum
22 November, 2000

Assignment 1 – Network Detects

Detect #1
RFC 1918 Source Addresses

```
12 Nov 2000 15:45:48 CST list 101 denied tcp 192.168.4.223(1617) -> my.mail.svr.65(25), 1 packet
12 Nov 2000 15:51:14 CST list 101 denied tcp 192.168.4.223(1617) -> my.mail.svr.65(25), 3 packets
12 Nov 2000 16:17:26 CST list 101 denied tcp 192.168.4.223(3499) -> my.mail.svr.65(25), 1 packet
12 Nov 2000 16:23:14 CST list 101 denied tcp 192.168.4.223(3499) -> my.mail.svr.65(25), 3 packets
13 Nov 2000 13:43:28 CST list 101 denied tcp 192.168.4.223(1256) -> my.mail.svr.65(25), 1 packet
13 Nov 2000 13:48:31 CST list 101 denied tcp 192.168.4.223(1256) -> my.mail.svr.65(25), 3 packets
14 Nov 2000 11:03:05 CST list 101 denied tcp 10.1.4.53(1408) -> my.mail.svr.65(25), 1 packet
14 Nov 2000 18:25:06 CST list 101 denied tcp 192.168.4.223(2401) -> my.mail.svr.65(25), 1 packet
14 Nov 2000 18:30:55 CST list 101 denied tcp 192.168.4.223(2401) -> my.mail.svr.65(25), 3 packets
15 Nov 2000 08:31:31 CST list 101 denied tcp 192.168.4.223(1403) -> my.mail.svr.65(25), 1 packet
15 Nov 2000 08:37:06 CST list 101 denied tcp 192.168.4.223(1403) -> my.mail.svr.65(25), 3 packets
15 Nov 2000 12:25:01 CST list 101 denied tcp 10.1.10.76(1041) -> my.mail.svr.65(25), 1 packet
15 Nov 2000 12:30:09 CST list 101 denied tcp 10.1.10.76(1041) -> my.mail.svr.65(25), 3 packets
------------------------ -------- ------ --- ---------- ----     -------------- --   ---------
        timestamp        list ID  action  |    src IP    src        dest IP       |    count
                                          |              port                   |
                                      protocol                              dest port
```

1. **Source of Trace** – My network, collected as part of my normal duties. Internal IP addresses have been obscured.

2. **Detect was generated by** – one of our perimeter Cisco routers that is directly connected to the Internet. This Cisco router was configured with an "ingress" access-list based on recommendations in a Cisco white paper (http://cisco.com/warp/public/cc/pd/iosw/ioft/iofwft/tech/firew_wp.pdf). Access-list 101 is a filter applied to inbound traffic on the Internet interface to block unwanted traffic of three types:
    a. Allow ICMP traffic that is deemed "useful" and block all other ICMP traffic that could be harmful.
    b. Block inbound traffic with illegal source addresses including loopback, RFC 1918 (http://www.ietf.org/rfc/rfc1918.txt), multicast addresses and other reserved addresses.
    c. Block inbound traffic with a source address of internal networks (anti-spoofing)

```
access-list 101 permit icmp any any echo
access-list 101 permit icmp any any echo-reply
access-list 101 permit icmp any any administratively-prohibited
access-list 101 permit icmp any any packet-too-big
access-list 101 permit icmp any any time-exceeded
access-list 101 permit icmp any any unreachable
access-list 101 deny   icmp any any log
access-list 101 deny   ip 127.0.0.0 0.255.255.255 any log
access-list 101 deny   ip 224.0.0.0 31.255.255.255 any log
access-list 101 deny   ip 0.0.0.0 0.255.255.255 any log
access-list 101 deny   ip 10.0.0.0 0.255.255.255 any log
access-list 101 deny   ip 172.16.0.0 0.15.255.255 any log
```

```
access-list 101 deny   ip 192.168.0.0 0.0.255.255 any log
access-list 101 deny   ip cidr.blk.addr.128 0.0.0.127 any log
access-list 101 permit ip any any
```

3. **Probability the source address was spoofed** – not likely. It is certainly possible that these packets are crafted, but since these source addresses are all in the RFC 1918 private address range (http://www.ietf.org/rfc/rfc1918.txt) and really shouldn't have been routed over the Internet in the first place, there shouldn't be any expectation of the packets being delivered to the target, much less of getting a response back.

4. **Description of attack** – This is a set of packets from various machines that appear to be attempting to start SMTP connections to my mail server. The packets are not coming in fast enough to constitute a denial of service attack, so the most likely explanation is that we have three machines that are misconfigured or behind some sort of malfunctioning NAT (Network Address Translation) gateway. CVE-2000-0181 (Checkpoint FW-1 may leak packets with private address) may apply, and CAN-1999-0529 is the more generic description of this detect. The pattern displayed of a lone packet followed 5-6 minutes later with a set of 3 packets with the same source port may serve as a fingerprint pattern for whatever is going wrong upstream.

5. **Attack Mechanism** – these packets seem to be relatively normal connection attempts, they just happen to come from illegal source addresses. No other evidence of malicious intent is seen.

6. **Correlations** – none of my other sensors can correlate these events since the packets were blocked at the perimeter router. Unfortunately, the number of incidents of packets arriving with a source address in the RFC 1918 address space is increasing (http://www.securityfocus.com/archive/47/71563, http://www.securityfocus.com/archive/75/58773).

7. **Evidence of active targeting** – Yes, every one of these packets was specifically addressed to my mail server.

8. **Severity** = (5 + 0) – (5 + 0) = 0
   (System criticality + Attack lethality) – (Network countermeasures + System countermeasures)

   System criticality: 5 – mail server
   Attack lethality: 0 – no chance to complete a 3-way handshake, not a denial of service attack
   Network countermeasures: 5 – router access-list rejected the packets
   System countermeasures: 0 – not applicable since packets never reached mail server

9. **Defensive recommendation** – Defenses were fine because the perimeter router rejected all of the packets. Contact the upstream ISP and request that they stop forwarding packets that have a source address in the RFC 1918 ranges.

10. **Multiple choice test question** – answer is d.

    Which of these is a valid source address on the Internet?
    a) 192.168.4.223
    b) 10.1.4.53

c) 172.25.210.14
d) 24.3.21.199

Detect #2
DNS Server Probe

```
WTsyslog[2000-11-17 15:37:56 ip=pix1.int.addr.10 pri=6] <162>%PIX-2-106006: Deny
inbound UDP from 12.43.88.5/10931 to int.dns.svr.15/37852
WTsyslog[2000-11-17 15:37:56 ip=pix1.int.addr.10 pri=6] <166>%PIX-6-106015: Deny TCP
(no connection) from 12.43.88.5/80 to int.dns.svr.15/53 flags ACK
WTsyslog[2000-11-17 15:37:56 ip=pix1.int.addr.10 pri=6] <166>%PIX-6-302001: Built
inbound TCP connection 1194991 for faddr 12.43.88.5/10929 gaddr ext.dns.svr.65/53
laddr int.dns.svr.15/53
WTsyslog[2000-11-17 15:37:56 ip=pix1.int.addr.10 pri=6] <166>%PIX-6-302002: Teardown
TCP connection 1194991 faddr 12.43.88.5/10929 gaddr ext.dns.svr.65/53 laddr
int.dns.svr.15/53 duration 0:00:00 bytes 0 (TCP Reset-O)
```

1. **Source of trace** – My network, collected as a part of my normal duties. Destination IP addresses have been obscured.

2. **Detect was generated by** – Cisco PIX firewall, logs entries collected by WebTrends Firewall Suite.

3. **Probability the source address was spoofed** – Negligible, a TCP three-way handshake was completed during the course of the detect.

4. **Description of attack** – The first log entry is of a denied UDP packet sent to UDP port 37852 on my secondary name server. This packet didn't raise any initial alarm, it could be just a lost packet. Here is a quick rundown of the fields in this log file entry:

```
Wtsyslog                      Label inserted by WebTrends Syslog collector service
[2000-11-17 15:37:56          Timestamp
ip=pix1.int.addr.10           Firewall internal address
pri=6]                        Syslog priority (informational)

<162>%PIX-2-106006: Deny inbound UDP from 12.43.88.5/10931 to int.dns.svr.15/37852
------------------  --------------------- ---------- -----    ------------- -----
  PIX message ID     Message Action        source     src       destination    dest
                                           IP         port      IP             port
```

The second log entry is of a denied packet that is obviously crafted. It shows a TCP packet with a source port of 80 (HTTP), a destination port of 53 (domain) and only the ACK flag set sent to my DNS server. That combination of source port/destination port/ACK bit is clearly designed to slip through a packet-filter screening device such as a Cisco router that has an access-list that allows inbound packets with a source port of 80 for established connections (ACK bit set).

The third log entry records that a successful TCP connection (three-way handshake completed) was established by the attacker to my DNS server.

The fourth log entry records that the TCP connection was terminated by a TCP reset after an elapsed time of 0:00:00 with 0 bytes transferred.

An analysis of the timestamps for these log entries shows that these packets all arrived within 1 second of each other, further evidence that this was a scripted attack.

5. **Attack mechanism** – First, it checks to see if anything is listening on UDP port 37852 (unknown, possible backdoor?). Then it checks to see if there is an active DNS server by trying to sneak in a TCP packet with a source port of 80, a destination port of 53 and the ACK bit set (a live server would respond with a reset packet and a host not running a DNS server would respond with an ICMP Port Unreachable packet). It then establishes a TCP connection to the name server and then immediately closes the connection by sending a TCP reset (no data transferred). This looks like a reconnaissance scan of my DNS server for the presence of a program listening on UDP/37852 and to ascertain if a TCP connection to the name server is allowed.

6. **Correlations** – similar scans from the same source were also recorded that were aimed at my primary name server:

```
Nov 17 07:07:40 pix2.int.addr.20 %PIX-2-106006: Deny inbound UDP from
12.43.88.5/10245 to int.dns.svr1.26/37852
Nov 17 07:07:40 pix2.int.addr.20 %PIX-6-106015: Deny TCP (no connection) from
12.43.88.5/80 to int.dns.svr1.26/53 flags ACK
Nov 17 07:07:40 pix2.int.addr.20 %PIX-6-302001: Built inbound TCP connection
952079878 for faddr 12.43.88.5/10243 gaddr ext.dns.svr1.1/53 laddr
int.dns.svr1.26/53
Nov 17 07:07:40 pix2.int.addr.20 %PIX-6-302002: Teardown TCP connection 952079878
faddr 12.43.88.5/10243 gaddr ext.dns.svr1.1/53 laddr int.dns.svr1.26/53 duration
0:00:00 bytes 0 (TCP Reset-O)

Nov 17 15:45:02 pix2.int.addr.20 %PIX-2-106006: Deny inbound UDP from
12.43.88.5/10398 to int.dns.svr1.26/37852
Nov 17 15:45:02 pix2.int.addr.20 %PIX-6-106015: Deny TCP (no connection) from
12.43.88.5/80 to int.dns.svr1.26/53 flags ACK
Nov 17 15:45:02 pix2.int.addr.20 %PIX-6-302001: Built inbound TCP connection
956719638 for faddr 12.43.88.5/10396 gaddr ext.dns.svr1.1/53 laddr
int.dns.svr1.26/53
Nov 17 15:45:03 pix2.int.addr.20 %PIX-6-302002: Teardown TCP connection 956719638
faddr 12.43.88.5/10396 gaddr ext.dns.svr1.1/53 laddr int.dns.svr1.26/53 duration
0:00:00 bytes 0 (TCP Reset-O)
```

Nothing appeared in the logs of the DNS servers – we do log unapproved zone transfer requests and unapproved updates, but not individual queries. There has been a recent CERT alert regarding multiple denial of service attacks against ISC BIND software (http://www.cert.org/advisories/CA-2000-20.html). One of the vulnerabilities listed involves zone transfer requests. To test the logging facility of my nameserver, I generated a zone transfer request from a test box that should be denied. Here is the named log entry:

```
Nov 17 16:27:39 ns2.domain.com named[6455]: unapproved AXFR from [my.test.box.8].515
72 for "domain.com" (acl)
```

Good, the server is properly rejecting zone transfer requests from unauthorized hosts and logging the event. Here are the log entries from the PIX firewall for that zone transfer request:

```
WTsyslog[2000-11-17 16:27:40 ip=pix1.int.addr.10 pri=6] <166>%PIX-6-302001: Built
outbound TCP connection 1197584 for faddr int.dns.svr.15/53 gaddr
my.test.box.8/51572 laddr my.test.box.8/51572
WTsyslog[2000-11-17 16:27:40 ip=pix1.int.addr.10 pri=6] <166>%PIX-6-302002: Teardown
TCP connection 1197584 faddr int.dns.svr.15/53 gaddr my.test.box.8/51572 laddr
my.test.box.8/51572 duration 0:00:01 bytes 54 (TCP FIN)
                                        --------
```

Note that there were 54 bytes transferred during the connection even though the zone transfer request was denied by the DNS server. This contrasts with the 0 bytes transferred during the connection in the detect. One possible explanation for the packets aimed at my machines is a search for DNS servers that allow TCP connections to port 53 (required for zone transfers) in preparation for future attacks.

7. **Evidence of active targeting** – Yes, these packets are specifically addressed to my name servers.

8. **Severity** = (5 + 1) – (4 + 3) = -1
   (System criticality + Attack lethality) – (Network countermeasures + System countermeasures)

   System criticality: 5 – name server (DNS)
   Attack lethality: 1 – reconnaissance
   Network countermeasures: 4 – Firewall rejected the Trojan probe and crafted packets, but did allow the TCP connection.
   System countermeasures: 3 – latest patches have been applied to DNS daemon, and it is configured to not allow DNS zone transfers to unauthorized sites.

9. **Defensive recommendation** – Defense against the UDP port probe is fine at the firewall, but it couldn't hurt to make sure there isn't anything listening on that port on the target host. Close attention needs to be paid to the patch level and configuration of the DNS server daemon if TCP connections are allowed from all outside sources.

10. **Multiple choice test question** – answer is c.

    Which of the following Protocol/Port combinations is used during a DNS zone transfer?

    a) UDP/53
    b) TCP/23
    c) TCP/53
    d) UDP/37852

Detect #3

Portmapper Stealth Scan

```
Frame Status Source Address    Dest. Address        Size Rel. Time     Delta
```

```
Time     Abs. Time              Summary
     1 M      [164.58.70.236]   [00A.00B.00C.1]       60 0:00:00.000
0.000.000     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     2        [164.58.70.236]   [00A.00B.00C.3]       60 0:00:00.041
0.041.103     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     3        [164.58.70.236]   [00A.00B.00C.4]       60 0:00:00.063
0.022.058     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     4        [164.58.70.236]   [00A.00B.00C.8]       60 0:00:00.139
0.076.544     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     5        [164.58.70.236]   [00A.00B.00C.14]      60 0:00:00.259
0.119.926     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     6        [164.58.70.236]   [00A.00B.00C.15]      60 0:00:00.278
0.018.664     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     7        [164.58.70.236]   [00A.00B.00C.16]      60 0:00:00.301
0.022.874     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     8        [164.58.70.236]   [00A.00B.00C.21]      60 0:00:00.398
0.097.719     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     9        [164.58.70.236]   [00A.00B.00C.22]      60 0:00:00.419
0.020.853     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
```

1. **Source of trace** – GIAC Website, http://www.sans.org/y2k/111300.htm (Luis Mendoza). Destination IP addresses have been obscured.

2. **Detect was generated by** – unidentified IDS system

3. **Probability the source address was spoofed** – unlikely. This appears to be a host scan of the target network looking for machines with an active portmapper (111/TCP). This type of reconnaissance is pointless if the answers to these probe packets don't go back to the bad guy's machine. The IP address of the attacking host (164.58.70.236) does not have a resolvable name associated with it. The source network of the attacking host is:

```
[Server: whois.arin.net]

Oklahoma State Regents for Higher Education (NET-ONENET)
Oklahoma State Regents for Higher Education
500 Education Building
State Capitol Complex
Oklahoma City, OK 73105

Netname: ONENET
Netnumber: 164.58.0.0
```

4. **Description of attack** – Quick and dirty scan of victim's address space for machines running RPC portmapper using crafted packets. There are many known vulnerabilities of portmapper

(CVE-1999-0168, CAN-1999-1095, CAN-1999-0632) as well as the RPC services it supports such as rpc.statd (CVE-1999-0018, CVE-1999-0019, CVE-1999-0493, CVE-2000-0666), rpc.sadmind (CVE-1999-0977), rpc.cmsd (CVE-1999-0320, CVE-1999-0696), etc.

The target IP addresses show some gaps in the sequence of machines being scanned. This may indicate that the scanner has prior knowledge of which machines are alive on that subnet.

The program generating these packets should be readily identifiable due to the many elements in common between these packets. The signature for this specific attack is: Source Port = 111/TCP, Destination Port = 111/TCP, both SYN and FIN flags set, and Sequence Number = 1861598755.

5. **Attack mechanism** -- The combination of both the SYN and FIN flags set at the same time is not "legal." The purpose of having both flags set is to evade logging or filter machines that check only for packets with the SYN flag set. Also, packets generated by a normal TCP/IP stack would have different sequence numbers for each new connection attempt – these packets all have the same sequence number. The last bit of evidence that these packets come from a crafted attack is that they all arrive within a very short time (sum of delta times ~ 0.42 seconds).

   Since the SYN/FIN combination is not supposed to occur in normal traffic, the response of the target machine is not defined by the standards. It is possible that the scanner could determine system information from the target machine's reply as well as whether or not the target system is running RPC portmapper.

6. **Correlations** – No information was supplied in the posting as to whether the scan was blocked by network defenses or if the target machines responded to the probes. A similar attack was attributed to a Mac web server in Washington by Arrigo (http://www.sans.org/y2k/110900.htm):

```
4. Quick report, two scans going round 195.0.0.0/8 from top to bottom (I have both ends):

1) root@cx32801-a.wwck1.ri.home.com [24.0.242.170]
   You-know-who...
2) 207.221.31.73
   ICG NetAhead, Inc. (NET-ICG-BLK-BLK7)
      532 Race St.
      Sana Jose, CA 95126      US

   Netname: ICG-BLK-BLK7
   Netblock: 207.220.0.0 - 207.223.255.255

Examples (both rather "dirty, esp. Mr. ICG):

Nov  8 00:35:13 24.0.242.170:2230 -> 192.168.120.100:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2231 -> 192.168.120.98:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2233 -> 192.168.120.99:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2235 -> 192.168.120.102:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2237 -> 192.168.120.105:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2236 -> 192.168.120.103:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2242 -> 192.168.120.109:111 SYN **S*****
Nov  8 00:35:13 24.0.242.170:2243 -> 192.168.120.110:111 SYN **S*****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.103:111 SYNFIN**SF****
Nov  8 08:59:14 207.221.31.73:608 -> 192.168.120.103:111 SYN **S*****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.105:111 SYNFIN**SF****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.109:111 SYNFIN**SF****
Nov  8 08:59:14 207.221.31.73:609 -> 192.168.120.109:111 SYN **S*****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.110:111 SYNFIN**SF****
Nov  8 08:59:16 207.221.31.73:3558 -> 192.168.120.110:111 SYN **S*****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.99:111 SYNFIN **SF****
Nov  8 08:59:14 207.221.31.73:611 -> 192.168.120.99:111 SYN **S*****
Nov  8 08:59:11 207.221.31.73:111 -> 192.168.120.100:111 SYNFIN**SF****
Nov  8 08:59:14 207.221.31.73:612 -> 192.168.120.100:111 SYN **S*****

5.  I think FTP and portmap are on this week's specials menu... So far I have pretty
definite correlations between a portscan, normally SF, and then an RPC info query.  What
is even more interesting is that I can definitely follow the pattern through 195.0.0.0/0.
So far I've seen it on 195.82.x.y/28, 195.89.x.y/29 and 195.212.x.y/27 at times which
correlate with a sweep of 195.0.0.0/0 started a couple of days ago.  So far I have
two "culprits":

202.185.200.8 (mail.tpm.com.my) - Malaysia.

207.221.31.73 (www.milepost1.com, via investigation)  no PTR, somewhere on ICG NetAhead,
Inc.,  traceroute points to Seattle, WA, USA. Ah, replies to Web: milepost1.com in Kent,
WA... their main web server is the source of the scan :-( Running MacOS + Apache/WebTen.
We learn something every day, never thought you could portscan from a Mac!  Fundamentally
the pattern is a portscan plus a "followup" connection to either SUNRPC or FTPD.
Example patterns for both sites:

[mail.tpm.com.my]
Nov  8 11:43:07 scylla snort: spp_portscan: PORTSCAN DETECTED from
207.221.31.73
Nov  8 11:43:07 scylla snort: SCAN-SYN FIN: 207.221.31.73:111 ->
192.168.178.229:111
Nov  8 11:43:07 scylla snort: SCAN-SYN FIN: 207.221.31.73:111 ->
192.168.178.230:111
Nov  8 11:43:08 scylla snort: RPC Info Query: 207.221.31.73:757 ->
192.168.178.229:111
Nov  8 11:43:36 scylla snort: spp_portscan: portscan status from
207.221.31.73: 3 connections across 2 hosts: TCP(3), UDP(0) STEALTH
Nov  8 11:43:42 scylla snort: spp_portscan: End of portscan from
207.221.31.73
Nov  8 11:52:21 bishop-rock tcplogd: sunrpc connection attempt from
unknown@[207.221.31.73]
```

7. **Evidence of active targeting** – not really. This is a reconnaissance scan searching for machines running RPC portmapper. The gaps in the sequence of machines being scanned may indicate that the scanner already knows which machines are alive on the subnet.

8. **Severity** = (2 + 2) – (1 + 1) = 2
(System criticality + Attack lethality) – (Network countermeasures + System countermeasures)

System criticality: 2 – Arbitrary value since knowledge of the systems involved was not included in the posting. On a shotgun scan like this, most of the targets are normally workstations (criticality = 1) but there are almost always a few machines with a higher importance. A generic "average" criticality figure of 2 seems reasonable.

Attack lethality: 2 – Normally assign a value of 1 for reconnaissance, but due to the large number of root compromises immediately available if a system is running unpatched RPC services, increasing the lethality a notch seems reasonable.

Network countermeasures: 1 – Arbitrary value since knowledge of the systems involved was not included in the posting.

System countermeasures: 1 – Arbitrary value since knowledge of the systems involved was not included in the posting
.

9. **Defensive recommendation** – Block incoming connection attempts to 111/TCP (and 111/UDP) at the perimeter router and/or firewall. Do not run RPC services on machines that can be accessed form the Internet. See also, Information Security Paper: "Rpcbind and Portmapper" (http://www.sans.org/newlook/resources/IDFAQ/blocking.htm) for more information.

10. **Multiple choice test question** – answer is b.

```
Frame Status Source Address    Dest. Address        Size Rel. Time     Delta
Time    Abs. Time                Summary
     1 M      [164.58.70.236]   [00A.00B.00C.1]       60 0:00:00.000
0.000.000     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
     2        [164.58.70.236]   [00A.00B.00C.3]       60 0:00:00.041
0.041.103     11/06/2000 02:49:28 PM TCP: D=111 S=111 SYN FIN SEQ=1861598755
LEN=0 WIN=1028
```

Which of the following characterize the above packets as a RPC portmapper connection attempt?

a) Source Port = 111/TDP
b) Destination Port = 111/TCP
c) Both SYN & FIN flags set
d) Sequence number = 1861598755

Detect #4

(A day in the life of @Home)

```
Nov  3 01:31:11 cc1014244-a kernel: securityalert: udp if=ef0
from 24.27.196.95:137 to 24.3.21.199 on unserved port 137
Nov  3 03:55:23 cc1014244-a kernel: securityalert: tcp if=ef0
from 24.180.71.16:3761 to 24.3.21.199 on unserved port 1243
Nov  3 04:40:00 cc1014244-a kernel: securityalert: udp if=ef0
from 158.252.141.73:137 to 24.3.21.199 on unserved port 137

Nov  3 05:28:21 cc1014244-a kernel: securityalert: tcp if=ef0
from 216.62.230.73:9704 to 24.3.21.199 on unserved port 9704
Nov  3 05:48:28 cc1014244-a kernel: securityalert: udp if=ef0
from 24.108.4.126:137 to 24.3.21.199 on unserved port 137
```

```
Nov  3 07:06:58 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2643 to 24.3.21.199 on unserved port 8080
Nov  3 07:06:58 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2644 to 24.3.21.199 on unserved port 80
Nov  3 07:06:59 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2645 to 24.3.21.199 on unserved port 3128
Nov  3 07:06:59 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2646 to 24.3.21.199 on unserved port 1080
Nov  3 07:06:59 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2643 to 24.3.21.199 on unserved port 8080
Nov  3 07:06:59 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2644 to 24.3.21.199 on unserved port 80
Nov  3 07:07:00 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2645 to 24.3.21.199 on unserved port 3128
Nov  3 07:07:00 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2646 to 24.3.21.199 on unserved port 1080
Nov  3 07:07:00 cc1014244-a kernel: securityalert: tcp if=ef0
from 213.8.0.51:2643 to 24.3.21.199 on unserved port 8080
Nov  3 07:33:45 cc1014244-a kernel: securityalert: udp if=ef0
from 24.92.193.112:61229 to 24.3.21.199 on unserved port 137
Nov  3 07:56:18 cc1014244-a kernel: securityalert: tcp if=ef0
from 63.21.41.156:1143 to 24.3.21.199 on unserved port 8080

Nov  3 13:49:23 cc1014244-a kernel: securityalert: udp if=ef0
from 208.194.193.125:137 to 24.3.21.199 on unserved port 137
Nov  3 13:49:25 cc1014244-a kernel: securityalert: udp if=ef0
from 24.160.66.32:137 to 24.3.21.199 on unserved port 137
Nov  3 13:49:25 cc1014244-a kernel: securityalert: udp if=ef0
from 208.194.193.125:137 to 24.3.21.199 on unserved port 137
Nov  3 13:49:26 cc1014244-a kernel: securityalert: udp if=ef0
from 24.160.66.32:137 to 24.3.21.199 on unserved port 137
Nov  3 14:32:52 cc1014244-a kernel: securityalert: tcp if=ef0
from 63.248.89.132:1669 to 24.3.21.199 on unserved port 1080

Nov  3 15:56:13 cc1014244-a kernel: securityalert: udp if=ef0
from 24.0.206.107:137 to 24.3.21.199 on unserved port 137
Nov  3 18:56:43 cc1014244-a kernel: securityalert: tcp if=ef0
from 65.33.61.200:4624 to 24.3.21.199 on unserved port 1080
Nov  3 19:23:59 cc1014244-a kernel: securityalert: tcp if=ef0
from 202.92.69.208:1370 to 24.3.21.199 on unserved port 27374
Nov  3 19:32:19 cc1014244-a kernel: securityalert: tcp if=ef0
from 24.68.56.206:1812 to 24.3.21.199 on unserved port 1243
Nov  3 22:23:47 cc1014244-a kernel: securityalert: tcp if=ef0
from 24.68.56.206:1268 to 24.3.21.199 on unserved port 1243
```

1. **Source of trace** – GIAC Website, http://www.sans.org/y2k/110700.htm, (binette@home).

2. **Detect was generated by** – xNIX host reporting connection attempts to unoccupied ports. Here is a description of the fields:

```
                                                Protocol
Timestamp       hostname            function        |  Interface
--------------- ------------------- ------------- --- ---------
Nov  3 01:31:11 cc1014244-a kernel: securityalert: udp if=ef0
```

```
                Source                                       Destination
     Source IP    Port   Dest. IP                            Port
     ------------ ---    -----------                         ---
from 24.27.196.95:137 to 24.3.21.199 on unserved port 137
```

3. **Probability the source addresses were spoofed** – unlikely. Most of these are reconnaissance probes searching for well known Trojans/backdoors and wouldn't serve any function without the replies. There isn't enough volume of traffic to constitute a denial of service attack.

4. **Description of attacks** – A day in the life of an @Home user at IP address 24.3.21.199 (cc1014244-a.hwrd1.md.home.com). There are multiple machines from all over the world constantly trolling the address space of the @Home cable network looking for vulnerable or already compromised systems ("low hanging fruit").

24.27.196.95	ubr-27.196.95.satellitebeach.cfl.rr.com
24.180.71.16	cc515072-c.union1.nj.home.com
158.252.141.73	sdn-ar-002arfayeP271.dialsprint.net
216.62.230.73	adsl-216-62-230-73.dsl.snantx.swbell.net
24.108.4.126	c11034-001.powersurfr.com
213.8.0.51	Ramat-Gan-0-51.access.net.il (Italy)
24.92.193.112	dt141n70.tampabay.rr.com
63.21.41.156	1Cust156.tnt2.elizabethtown.ky.da.uu.net
208.194.193.125	208-194-193-125.flash.net
24.160.66.32	cs16066-32.houston.rr.com
63.248.89.132	3ff85984.dsl.flashcom.net
24.0.206.107	cc806005-a.slbch1.occa.home.com
65.33.61.200	ubr-33.61.200.unionpark.cfl.rr.com
202.92.69.208	goconsyd464.goconnect.net (Australia)
24.68.56.206	24.68.56.206.on.wave.home.com

In order to facilitate the analysis, I've reformatted the detect into a more compact form and added my description of the attack:

Number	Timestamp (all Nov 3)	Protocol	Source IP	Src. port	Destination IP	Dest. port	Description of Attack
1	1:31:11	udp	24.27.196.95	137	24.3.21.199	137	NetBIOS Name Service
2	3:55:23	tcp	24.180.71.16	3761	24.3.21.199	1243	SubSeven, version 1
3	4:40:00	udp	158.252.141.73	137	24.3.21.199	137	NetBIOS Name Service
4	5:28:21	tcp	216.62.230.73	9704	24.3.21.199	9704	xNIX backdoor search
5	5:48:28	udp	24.108.4.126	137	24.3.21.199	137	NetBIOS Name Service
6	7:06:58	tcp	213.8.0.51	2644	24.3.21.199	80	80/3128/8080 pattern = RingZero 1080 is SOCKS or Wingate
7	7:06:58	tcp	213.8.0.51	2643	24.3.21.199	8080	
8	7:06:59	tcp	213.8.0.51	2644	24.3.21.199	80	
9	7:06:59	tcp	213.8.0.51	2646	24.3.21.199	1080	

10	7:06:59	tcp	213.8.0.51	2645	24.3.21.199	3128	80 is the common web server port 3128 is Squid proxy 8080 is a common proxy port
11	7:06:59	tcp	213.8.0.51	2643	24.3.21.199	8080	
12	7:07:00	tcp	213.8.0.51	2646	24.3.21.199	1080	
13	7:07:00	tcp	213.8.0.51	2645	24.3.21.199	3128	
14	7:07:00	tcp	213.8.0.51	2643	24.3.21.199	8080	
15	7:33:45	udp	24.92.193.112	61229	24.3.21.199	137	NetBIOS Name Service probe
16	7:56:18	tcp	63.21.41.156	1143	24.3.21.199	8080	Web proxy or RingZero
17	13:49:23	udp	208.194.193.125	137	24.3.21.199	137	NetBIOS Name Service
18	13:49:25	udp	208.194.193.125	137	24.3.21.199	137	
19	13:49:25	udp	24.160.66.32	137	24.3.21.199	137	NetBIOS Name Service
20	13:49:26	udp	24.160.66.32	137	24.3.21.199	137	NetBIOS Name Service
21	14:32:52	tcp	63.248.89.132	1669	24.3.21.199	1080	SOCKS or Wingate probe
22	15:56:13	udp	24.0.206.107	137	24.3.21.199	137	NetBIOS Name Service
23	18:56:43	tcp	65.33.61.200	4624	24.3.21.199	1080	SOCKS or Winhole probe
24	19:23:59	tcp	202.92.69.208	1370	24.3.21.199	27374	SubSeven, version 2.1
25	19:32:19	tcp	24.68.56.206	1812	24.3.21.199	1243	SubSeven, version 1
26	22:23:47	tcp	24.68.56.206	1268	24.3.21.199	1243	

Packets 1, 3, 5, 15, 17, 18, 19, 20 and 22:
NetBIOS Name Service probes could be from use of NBTSTAT command for reconnaissance or after effect of network.vbs virus infection (http://www.sans.org/newlook/resources/IDFAQ/port_137.htm). Note that packet 15 does not have a source port of 137. This could mean that the packet may have originated from a non-Windows box.

Packets 2, 24, 25 and 26:
The SubSeven Trojan allows remote control of the infected Windows machine (http://xforce.iss.net/alerts/advise30.php and CAN-1999-066). It is one of the more popular and powerful backdoor Trojans because it is being continually updated (http://subseven.slak.org).

Packet 4:
Port 9704 has been reported as a backdoor (shell attached to the port at root privileges) for xNIX boxes that have had a rpc.statd buffer overflow exploit (http://www.sans.org/y2k/110900.htm & http://www.cert.org/advisories/CA-2000-17.html) or a wu-ftpd exploit (http://lists.insecure.org/incidents/2000/Sep/0054.html).

Packets 6 – 14:
RingZero is a scan for proxies/possible Trojans on TCP Ports 80, 3128 and 8080 (http://www.sans.org/newlook/resources/IDFAQ/ring_zero.htm). This variation also scans for SOCKS/Wingate on port 1080.

Packets 21 and 23:
Wingate or SOCKS proxy allows a host outside of a firewall to connect transparently and securely through the firewall (http://www.sans.org/newlook/resources/IDFAQ/socks.htm).

5. **Attack mechanism** – All of these packets appear to be reconnaissance, either looking for system information (NetBIOS Name Service), searching for backdoors on systems that have been previously compromised (RingZero, SubSeven, shell on port 9704) or searching for services that have known exploits (SOCKS, Wingate).

   Most of these packets represent the only packet seen from a particular host. The exceptions include packets 6-14 from 213.8.0.51 (Ramat-Gan-0-51.access.net.il) that appear to be simple TCP retries as evidenced by the non-changing source ports between packets sent to a particular destination port (e.g. packets 11 & 14 are retries of packet 6). Packet 18 appears a to be a simple retry of packet 17. The time differential between packets 25 and 26 (2:59:28) is far too large for the second packet to be a simple TCP retry. Perhaps the user at 24.68.56.206 (24.68.56.206.on.wave.home.com) forgot he scanned this host 3 hours previously?

6. **Correlations** – Reports of each of these types of scans, especially on the @Home cable network, are a regular occurrence in the various security incidence sites and mailing lists (e.g. http://www.securityfocus.com/archive/75/61759, http://www.sans.org/y2k/111500.htm, etc.). The RingZero + SOCKS/Wingate probe seen in this detect was mentioned as a known variation in the SecurityFocus Incidents list (http://www.securityfocus.com/archive/75/52279).

7. **Evidence of active targeting** – perhaps. The RingZero + SOCKS scan represented by packets 6-15 is directly targeting this machine. The NetBIOS Name Service packets from the 24.x.x.x network might be targeted probes or they could just be Windows machines on the @Home network trying to do normal (for Microsoft) name resolution. The other 1 or 2 packet hits are probably part of larger host scans of the @Home address space.

8. **Severity** = (3 + 4) – (0 + 4) = 3
   (System criticality + Attack lethality) – (Network countermeasures + System countermeasures)

   System criticality: 3 – Arbitrary value since knowledge of the systems involved was not included in the posting.

   Attack lethality: 4 – The backdoors being probed for (RingZero, SubSeven, shell on port 9704) would result in complete control of the target system by taking advantage of a previous compromise of that system. However, these probes would not directly cause the compromise of the targeted system. The NetBIOS probes are normally just reconnaissance (lethality = 1), although some older, unpatched Windows systems might still be vulnerable to DoS attacks on port 137 such as WinNuke (CVE-1999-0153).

   Network countermeasures: 0 – All these packets reached the targeted host system.

   System countermeasures: 4 – Target host system rejected these packets because it did not have a service listening on any of the probed ports. What we don't know is what services the host system is listening for.

9. **Defensive recommendation** – The fact that these packets were rejected by the target host is good. What we don't see is what packets were accepted by the target host. It is recommended to

put a firewall between the @Home network and this host. Also, hardening the system by removing all unnecessary services is recommended.

10. **Multiple choice test question** – answer is a.

Which of the following is a characteristic of a scan for machines infected with the SubSeven version 2.1 Trojan?

a) Destination port = TCP/27374
b) Source port = UDP/137
c) Destination port = TCP/12345
d) Destination ports of TCP/80, TCP/3128 & TCP/8080

Assignment 2 – Evaluate an Attack

On November 13, 2000, the Computer Emergency Response Team Coordination Center (CERT/CC) of the Software Engineering Institute at Carnegie Mellon University issued CERT Advisory CA-2000-20 titled "Mulitple Denial-of-Service Problems in ISC BIND." (http://www.cert.org/advisories/CA-2000-20.html) The following is an excerpt from CERT Advisory CA-2000-20:

The CERT Coordination Center has recently learned of two serious denial-of-service vulnerabilities in the Internet Software Consortium's (ISC) BIND software.

The first vulnerability is referred to by the ISC as the "zxfr bug" and affects ISC BIND version 8.2.2, patch levels 1 through 6. The second vulnerability, the "srv bug", affects ISC BIND versions 8.2 through 8.2.2-P6. Derivatives of the above code sets should also be presumed vulnerable unless proven otherwise.

The Internet Software Consortium, the maintainer of BIND, the software used to provide domain name resolution services, has recently posted information about several denial-of-service vulnerabilities. If exploited, any of these vulnerabilities could allow remote intruders to cause site DNS services to be stopped.

For more information about these vulnerabilities and others, please see

http://www.isc.org/products/BIND/bind-security.html

Two vulnerabilities in particular have been categorized by both the ISC and the CERT/CC as being serious.

The "zxfr bug" (CVE CAN-2000-0888)

Using this vulnerability, attackers on sites which are permitted to request zone transfers can force the *named* daemon running on vulnerable DNS servers to crash, disrupting name resolution service until the *named* daemon is restarted. The only preconditions for this attack to succeed is that a compressed zone transfer (ZXFR) request be made from a site allowed to make any zone transfer request (not just ZXFR), and that a subsequent name service query of an authoritative and non-cached record be made. The time between the attack and the crash of *named* may vary from system to system.
This vulnerability has been discussed in public forums. The ISC has confirmed that all platforms running version 8.2.2 of the BIND software prior to patch level 7 are vulnerable to this attack.

The "srv bug" (CVE CAN-2000-0887)

This vulnerability can cause affected DNS servers running *named* to go into an infinite loop, thus preventing further name requests to be handled. This can happen if an SRV record (defined in RFC2782) is sent to the vulnerable server.
Microsoft's Windows 2000 Active Directory service makes extensive use of SRV records and is reportedly capable of triggering this bug in the course of normal operations. This is not, however, a vulnerability in Microsoft Active Directory. ***Any network client capable of sending SRV records to vulnerable name server systems can exercise this vulnerability.***

The CERT/CC has not received any direct reports of either of these vulnerabilities being exploited to date. Both vulnerabilities can be used by malicious users to break the DNS services being offered at all exposed sites on the Internet.

This document will explore what an exploit of the "zxfr bug" would look like over the Internet and attempt to develop a signature of the attack so that it can be recognized by the intrusion detection analyst.

The original identification of the zxfr bug is attributed to Fabio Pietrosanti (naif) from a posting on the BugTraq mailing list (http://www.securityfocus.com/archive/1/143843). He noted that when attempting to do a "compressed" zone transfer from a BIND 8.2.2-P5 server, the daemon crashed. The method used to cause the crash was the named-xfer utility application that is a part of the BIND code distribution. The command and arguments he used were:

```
named-xfer  -z zone.pippo.com  -d 9 -f pics -Z dns.pippo.com
```

where "-z zone.pippo.com" specifies the zone to be transferred, "-d 9" specifies debug level 9, "-f pics" specifies the filename in which to store the results of the query, "-Z" specifies to compress the zone transfer data before transfer, and "dns.pippo.com" specifies the name of the server from which to request the zone transfer. The nameserver daemon (named) on dns.pippo.com was reported to not support compressed transfers and did not have any restrictions placed on zone transfers (anyone was authorized to request a zone transfer), both of which are default configuration options.

I have ready access to the BIND 8.2.2-P5 code distribution since that is the software used to provide domain name resolution services within our company. A quick search of the code for the named-xfer utility was performed using the following:

```
grep -i zxfr /usr/local/src/bind-8.2.2p5/src/bin/named-xfer/named-xfer.c
```

which returned:

```
                        xfr_qtype = ns_t_zxfr;
                            (query_type == ns_t_zxfr) ? "ZXFR" :
                            (query_type == ns_t_zxfr) ? "ZXFR" :/*XXX ZXFR*/
```

This allowed me to zero in on the following code fragment in the section that parses the command line arguments:

```
                case 'Z':
                        xfr_qtype = ns_t_zxfr;
                        break;
```

What this piece of the code does is assign the value of the global constant "ns_t_zxfr" to the variable "xfr_qtype" if the "-*Z*" option is included on the command line. (The other hits on the string "zxfr" were in portions of the code related to debugging.) Further down in the named-xfer.c code is

```
                                if ((soa_cnt == 0) || (zp->z_type == Z_STUB)) {
                                        if (zp->z_type == Z_STUB) {
                                                if (soa_cnt == 1 &&
                                                    ns_cnt == 0)
                                                        query_type = T_NS;
                                                else
                                                        query_type = T_SOA;
                                        } else if (methode == ISIXFR)
                                                query_type = T_IXFR;
                                        else
                                                query_type = xfr_qtype;
                                        n = res_nmkquery(&res, QUERY,
                                                         zp->z_origin,
                                                         curclass, query_type,
                                                         NULL, 0,
                                                         NULL, buf, bufsize);
```

which I interpret to mean that if all of the conditions are right, the variable "query_type" is assigned the value of the "xfr_qtype" variable. The "query_type" variable is then used as an argument to the "res_nmkquery" routine that builds the actual query packet that is sent out over the network.

I then did a search to find where the global constant "ns_t_zxfr" was defined:

```
find /usr/local/src/bind-8.2.2p5/src -name "*.h" -print -exec grep ns_t_zxfr {} \;
```

(output too long to include here). The above command starts a recursive search at the top of the BIND 8.2.2-P5 distribution for all files that end with ".h" (header files), prints the filename, and then executes a grep command to search for the string ns_t_zxfr in each file found. (There may be better ways to do a recursive search across a directory structure, but that's the way I learned to do it.) The following list of definitions was found in /usr/local/src/bind-8.2.2p7/src/include/arpa/nameser.h:

```
        ns_t_invalid = 0,       /* Cookie. */
        ns_t_a = 1,             /* Host address. */
        ns_t_ns = 2,            /* Authoritative server. */
        ns_t_md = 3,            /* Mail destination. */
        ns_t_mf = 4,            /* Mail forwarder. */
        ns_t_cname = 5,         /* Canonical name. */
        ns_t_soa = 6,           /* Start of authority zone. */
        ns_t_mb = 7,            /* Mailbox domain name. */
        ns_t_mg = 8,            /* Mail group member. */
        ns_t_mr = 9,            /* Mail rename name. */
        ns_t_null = 10,         /* Null resource record. */
        ns_t_wks = 11,          /* Well known service. */
        ns_t_ptr = 12,          /* Domain name pointer. */
        ns_t_hinfo = 13,        /* Host information. */
        ns_t_minfo = 14,        /* Mailbox information. */
        ns_t_mx = 15,           /* Mail routing information. */
        ns_t_txt = 16,          /* Text strings. */
        ns_t_rp = 17,           /* Responsible person. */
        ns_t_afsdb = 18,        /* AFS cell database. */
        ns_t_x25 = 19,          /* X_25 calling address. */
        ns_t_isdn = 20,         /* ISDN calling address. */
        ns_t_rt = 21,           /* Router. */
        ns_t_nsap = 22,         /* NSAP address. */
        ns_t_nsap_ptr = 23,     /* Reverse NSAP lookup (deprecated). */
        ns_t_sig = 24,          /* Security signature. */
        ns_t_key = 25,          /* Security key. */
        ns_t_px = 26,           /* X.400 mail mapping. */
        ns_t_gpos = 27,         /* Geographical position (withdrawn). */
        ns_t_aaaa = 28,         /* Ip6 Address. */
        ns_t_loc = 29,          /* Location Information. */
        ns_t_nxt = 30,          /* Next domain (security). */
        ns_t_eid = 31,          /* Endpoint identifier. */
        ns_t_nimloc = 32,       /* Nimrod Locator. */
        ns_t_srv = 33,          /* Server Selection. */
        ns_t_atma = 34,         /* ATM Address */
        ns_t_naptr = 35,        /* Naming Authority PoinTeR */
        ns_t_kx = 36,           /* Key Exchange */
        ns_t_cert = 37,         /* Certification record */
        ns_t_a6 = 38,           /* IPv6 address (deprecates AAAA) */
        ns_t_dname = 39,        /* Non-terminal DNAME (for IPv6) */
        ns_t_sink = 40,         /* Kitchen sink (experimentatl) */
        ns_t_opt = 41,          /* EDNS0 option (meta-RR) */
        ns_t_tsig = 250,        /* Transaction signature. */
        ns_t_ixfr = 251,        /* Incremental zone transfer. */
        ns_t_axfr = 252,        /* Transfer zone of authority. */
        ns_t_mailb = 253,       /* Transfer mailbox records. */
        ns_t_maila = 254,       /* Transfer mail agent records. */
        ns_t_any = 255,         /* Wildcard match. */
        ns_t_zxfr = 256,        /* BIND-specific, nonstandard. */
        ns_t_max = 65536
```

Now that we know that the query type used in a "compressed" zone transfer request is 256 (0x0100), we can go to the network and see what it looks like on the wire. I setup a test server running Solaris 7 (mmstest.xyz.com) with the BIND 8.2.2-P5 name server code installed and loaded a copy of a fictional zone database. I had a second server (tester.xyz.com) with the BIND 8.2.2-P5 code installed to initiate the query and my NT Workstation running Microsoft Network Monitor on the network between the two servers to capture the traffic. I initiated a "normal" transfer with the following command from tester:

```
named-xfer -z xyz.com -s 0 -f /tmp/output1 -S mmstest.xyz.com
```

-z – zone to transfer = xyz.com
-s 0 – serial number = 0 (force transfer)
-f /tmp/output1 = output filename
-S = transfer just the start of authority record (to limit the amount of data transferred)

Here is the relevant packet as displayed by Microsoft Network Monitor:

```
Frame    Time         Src MAC Addr    Dst MAC Addr    Protocol    Description
Src Other Addr     Dst Other Addr     Type Other Addr
```

```
5        297.164000   SUN MI7CFCC7    SUN MI856C7C    DNS         0x61D6:Std Qry
for xyz.com. of type SOA on class  10.11.65.10        10.11.65.8

  IP: ID = 0x2FF2; Proto = TCP; Len: 67
  TCP: .AP..., len:   27, seq:3130908176-3130908203, ack:4209501259, win: 8760,
src:40139  dst:   53
  DNS: 0x61D6:Std Qry for xyz.com. of type SOA on class INET addr.
      DNS: TCP Length = 25 (0x19)
      DNS: Query Identifier = 25046 (0x61D6)
      DNS: DNS Flags = Query, OpCode - Std Qry, RCode - No error
          DNS: 0............... = Request
          DNS: .0000........... = Standard Query
          DNS: .....0.......... = Server not authority for domain
          DNS: ......0......... = Message complete
          DNS: .......0........ = Iterative query desired
          DNS: ........0....... = No recursive queries
          DNS: .........000.... = Reserved
          DNS: ............0000 = No error
      DNS: Question Entry Count = 1 (0x1)
      DNS: Answer Entry Count = 0 (0x0)
      DNS: Name Server Count = 0 (0x0)
      DNS: Additional Records Count = 0 (0x0)
      DNS: Question Section: xyz.com. of type SOA on class INET addr.
          DNS: Question Name: xyz.com.
          DNS: Question Type = Start of zone of authority
          DNS: Question Class = Internet address class

00000:  08 00 20 85 6C 7C 08 00 20 7C FC C7 08 00 45 00   .. ?l|.. |üÇ..E.
00010:  00 43 2F F2 40 00 FF 06 B5 9A 0A 0B 41 0A 0A 0B   .C/ò@.ÿ.µ?..A...
00020:  41 08 9C CB 00 35 BA 9D DE 10 FA E7 E4 4B 50 18   A.?Ë.5° Þ.úçäKP.
00030:  22 38 8B D6 00 00 00 19 61 D6 00 00 00 01 00 00   "8?Ö....aÖ......
00040:  00 00 00 00 03 78 79 7A 03 63 6F 6D 00 00 06 00   .....xyz.com....
00050:  01                                                .
```

I have highlighted in bold the relevant section of the packet where the query type is defined. As expected, the query type field has a value of 0x0006 which corresponds to the ns_t_soa = 6, Start of authority zone defined above. I then attempted the "compressed zone transfer" request:

```
named-xfer –z xyz.com –s 0 –f /tmp/output2 –Z mmstest.xyz.com
```

The only difference between this command and the previous one is the –Z switch instead of the –S switch. Here is the relevant packet:

```
Frame    Time         Src MAC Addr    Dst MAC Addr    Protocol    Description
Src Other Addr     Dst Other Addr     Type Other Addr
9        297.413000   SUN MI7CFCC7    SUN MI856C7C    DNS         0x61D7:Std Qry
for xyz.com. of type Unknown Type  10.11.65.10        10.11.65.8

  ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD Internet Protocol
      ETHERNET: Destination address : 080020856C7C
      ETHERNET: Source address : 0800207CFCC7
      ETHERNET: Frame Length : 81 (0x0051)
      ETHERNET: Ethernet Type : 0x0800 (IP:  DOD Internet Protocol)
      ETHERNET: Ethernet Data: Number of data bytes remaining = 67 (0x0043)
  IP: ID = 0x2FF4; Proto = TCP; Len: 67
  TCP: .AP..., len:   27, seq:3130908203-3130908230, ack:4209501453, win: 8760,
src:40139  dst:   53
  DNS: 0x61D7:Std Qry for xyz.com. of type Unknown Type on class INET addr.
      DNS: TCP Length = 25 (0x19)
      DNS: Query Identifier = 25047 (0x61D7)
      DNS: DNS Flags = Query, OpCode - Std Qry, RCode - No error
          DNS: 0............... = Request
          DNS: .0000........... = Standard Query
          DNS: .....0.......... = Server not authority for domain
          DNS: ......0......... = Message complete
          DNS: .......0........ = Iterative query desired
          DNS: ........0....... = No recursive queries
          DNS: .........000.... = Reserved
          DNS: ............0000 = No error
      DNS: Question Entry Count = 1 (0x1)
      DNS: Answer Entry Count = 0 (0x0)
      DNS: Name Server Count = 0 (0x0)
      DNS: Additional Records Count = 0 (0x0)
      DNS: Question Section: xyz.com. of type Unknown Type on class INET addr.
          DNS: Question Name: xyz.com.
          DNS: Question Type = 0x0100
          DNS: Question Class = Internet address class

00000:  08 00 20 85 6C 7C 08 00 20 7C FC C7 08 00 45 00   .. ?l|.. |üÇ..E.
00010:  00 43 2F F4 40 00 FF 06 B5 98 0A 0B 41 0A 0A 0B   .C/ô@.ÿ.µ?..A...
00020:  41 08 9C CB 00 35 BA 9D DE 2B FA E7 E5 0D 50 18   A.?Ë.5º Þ+úçå.P.
00030:  22 38 90 F7 00 00 00 19 61 D7 00 00 00 01 00 00   "8 ÷....a×......
00040:  00 00 00 00 03 78 79 7A 03 63 6F 6D 00 01 00 00   .....xyz.com....
00050:  01                                                .
```

We can see then, that the signature for a compressed zone transfer request is a value of 0x0100 in the next-to-last word of a DNS query. Here is an example rule for Snort:

```
alert tcp any any -> any 53 (msg: "Possible ZXFR DoS attack"; flags:PA; content:
"|03|com|00 01 00|"; nocase;)
```

This rule will match TCP packets from any source IP address and source IP port, to any destination IP address on destination port 53, has both the PSH and ACK flags set, has content that matches 0x03 followed by "com" (case insensitive) followed by a 1 byte 0x00 string termination byte and then the 0x0100 pattern for the ZXFR transfer type. When the above rule was added to a standard ruleset (10102kany.rules, http://www.snort.org/Files/10102k.rules), Snort generated the following alert on the captured packets:

```
[**] Possible ZXFR DoS attack [**]
11/20-10:54:26.374801 10.11.65.10:40139 -> 10.11.65.8:53
TCP TTL:255 TOS:0x0 ID:12276  DF
***AP*** Seq: 0xBA9DDE2B   Ack: 0xFAE7E50D   Win: 0x2238
00 19 61 D7 00 00 00 01 00 00 00 00 00 00 03 74  ..a............t
77 63 03 63 6F 6D 00 01 00 00 01                 wc.com.....
```

Since the name of the zone that might be requested is of unknown length, the only way to match the ZXFR signature is to search for the "com" top level domain suffix followed by the ZXFR signature pattern. Of course, additional rules would have to be added to alert on ZXFR requests for .net, .org, .edu, .gov, .mil and any other top level domains that ICANN comes up with. This is clearly not a totally satisfactory solution since almost any string can be used as the name of the zone that is being requested in a crafted packet. Unfortunately, the limitations of the Snort ruleset do no allow for matching the 4th and 3rd bytes from the end of the packet.

Assignment 3 – "Analyze This" Scenario

Executive Summary

Our organization was asked to provide a bid to provide security services for GIAC Enterprises, a dot.com startup that sells electronic fortune cookie sayings. We were provided with data from a Snort system with a fairly standard rulebase that covers approximately one month. There are gaps in the data due to power failures, disk drives filling up and other system problems. We have analyzed the data provided and correlated our findings with those of previous analyses of earlier data, being especially alert for signs of compromised systems or network problems in generating this report.

Key Findings

1. The GIAC Enterprises network continues to be the target of numerous probes and attacks.
   a. More that 250,000 suspicious events were logged by the Snort IDS between 8/15 and 9/14
   b. More than 38,400 security alerts were logged by the Snort IDS between 8/11 and 9/14
2. Compromised systems still exist on your network, and the number of infected systems is increasing.
3. Current network and system defenses are inadequate to adequately protect the GIAC Enterprise network.
4. There appears to be a problem with the GIAC Enterprise network that is causing the generation of an abnormally large number of packets with malformed TCP flag settings.

Significant results

This report focuses on possibly compromised systems and potential network problems within the GIAC Enterprise network. Previous analyses have been performed on older data and it was felt that it would serve no useful purpose to fill up this report with basically duplicate data regarding the dangers of each of the different types of attacks detected. That being said, it is still useful to get an overall view of what kinds of attacks are being perpetrated against the GIAC Enterprise network. Table 1 is a summary of alerts generated from the GIAC Enterprise network during from 8/11 00:33:44 to 9/14 23:21:39 (as reported by SnortSnarf v111500.1):

Signature	# Alerts	# Sources	# Destinations
TCP **S***** scan	187304	75	29242
UDP scan	58282	52	520
Watchlist 000222 NET-NCFC	17839	44	19
WinGate 1080 Attempt	6063	335	2151

SYN-FIN scan!	5457	6	3005
Watchlist 000220 IL-ISDNNET-990517	5260	19	21
TCP **SF**** scan	3065	6	3005
Attempted Sun RPC high port access	1990	8	11
SNMP public access	922	16	1
SMB Name Wildcard	336	16	14
TCP ******** scan	164	63	73
NMAP TCP ping!	132	10	42
SUNRPC highport access!	64	5	3
Probable NMAP fingerprint attempt	63	6	28
Queso fingerprint	54	11	23
External RPC call	40	6	3
Tiny Fragments - Possible Hostile Activity	12	5	8
TCP SMTP Source Port traffic	8	2	2
site exec - Possible wu-ftpd exploit - GIAC000623	6	1	4
Possible wu-ftpd exploit - GIAC000623	2	1	2

Table1.

Table 1 does not list approximately 1350 alerts generated by packets with other illegal TCP flag combinations (223 combinations). We will discuss these packets later in this report (part 3).

1. TCP **S***** scan – A sequence of TCP packets with the SYN flag set (start of 3-way handshake) arriving at a frequency rate high enough to trigger the Snort scan detection routines. There were 75 hosts identified as initiating these scans, none of which was identified as a machine inside the GIAC Enterprise network. Table 2 shows selected external machines performing TCP scans:

	Source	Destination	Duration	Description
1	195.114.226.41 apollo-dh0040.multiweb.net	Entire MY.NET Class B network (24,067 hosts)	Earliest: 00:46:11 on 8/15 Latest: 02:35:53 on 8/15	Scan for FTP servers (port 21)
2	24.180.134.156 cc349491-a.hwrd1.md.home.com	Hosts on MY.NET.208 subnet (91 hosts)	Earliest: 04:48:03 on 9/11 Latest: 05:19:13 on 9/11	NMAP port scan
3	35.10.82.111 mcc-4.user.msu.edu	Entire MY.NET Class B network (25,469 hosts)	Earliest: 04:35:20 on 8/16 Latest: 05:16:28 on 8/16	Scan for SubSeven v2.1 infected systems (port 27374)
4	206.186.79.9 ns.arex.com	Entire MY.NET Class B network (22,156 hosts)	Earliest: 22:35:21 on 9/9 Latest: 02:13:08 on 9/10	Scan for DNS servers (port 53), follows up with UDP/53 to 16 hosts.
5	24.17.189.83 c679190-a.mckiny1.tx.home.com	Entire MY.NET Class B network (20,163 hosts)	Earliest: 03:48:41 on 9/8 Latest: 06:27:48 on 9/8	Scan for FTP servers (port 21), follows up with possible wu-ftpd exploits

6	212.141.100.97 gw2a61-1-d97.wind.it	Entire MY.NET Class B network (19,968 hosts)	Earliest: 06:13:57 on 9/2 Latest: 09:00:16 on 9/2	Scan for FTP servers (port 21)
7	129.186.93.133 skinner.cs.iastate.edu	Partial MY.NET Class B network (4,663 hosts)	Earliest: 21:24:04 on 9/6 Latest: 22:29:25 on 9/6	Scan for Telnet servers (port 23)
8	216.99.200.242 securedesign.net	3 hosts: MY.NET.97.209, MY.NET.97.216, MY.NET.98.188	Earliest: 11:42:13 on 9/4 Latest: 11:42:17 on 9/4	Port scans. MY.NET.97.209, TCP MY.NET.97.216, TCP MY.NET.98.188, TCP + UDP

Table 2.

Discussion: The range of hosts being scanned ranges from the entire class B address space of MY.NET (examples 1,3,4,5,6) to individual subnets (2) all the way down to individual servers (8). Most of these scans are simple reconnaissance, searching for hosts with easily exploitable services (FTP, Telnet) or for already compromised system (SubSeven, other Trojans/backdoors). Of particular interest are scans like (4) where a scan for servers that respond to TCP/53 is followed up by UDP/53 requests to 16 servers:

```
Sep  9 22:35:28 206.186.79.9:2713 -> MY.NET.1.4:53 UDP
Sep  9 23:06:05 206.186.79.9:2906 -> MY.NET.71.15:53 UDP
Sep  9 23:22:25 206.186.79.9:3046 -> MY.NET.109.41:53 UDP
Sep  9 23:31:46 206.186.79.9:3088 -> MY.NET.130.134:53 UDP
Sep  9 23:36:16 206.186.79.9:3112 -> MY.NET.140.198:53 UDP
Sep 10 00:02:58 206.186.79.9:3278 -> MY.NET.204.218:53 UDP
Sep 10 00:27:12 206.186.79.9:3352 -> MY.NET.1.4:53 UDP
Sep 10 00:27:12 206.186.79.9:3356 -> MY.NET.1.5:53 UDP
Sep 10 01:10:40 206.186.79.9:3707 -> MY.NET.100.165:53 UDP
Sep 10 01:13:02 206.186.79.9:3720 -> MY.NET.106.128:53 UDP
Sep 10 01:14:09 206.186.79.9:3727 -> MY.NET.109.41:53 UDP
Sep 10 01:23:22 206.186.79.9:3782 -> MY.NET.130.122:53 UDP
Sep 10 01:23:23 206.186.79.9:3783 -> MY.NET.130.134:53 UDP
Sep 10 01:29:10 206.186.79.9:3825 -> MY.NET.144.25:53 UDP
Sep 10 01:44:58 206.186.79.9:3894 -> MY.NET.181.88:53 UDP
```

This may indicate which server within the GIAC Enterprise network are were responding to TCP/53. The scan for FTP servers in (5) is followed up by possible wu-ftpd exploits on the following servers:

MY.NET.202.202
MY.NET.202.190
MY.NET.99.104
MY.NET.150.24

Similarly, this may give us a clue as to which machines are responding to FTP connection requests. These two examples of a scan followed up by a possible exploit attempt illustrate the necessity to restrict the current free access from the Internet to the GIAC Enterprise network with a firewall or other filtering device.

Finding: Review the configuration of the systems listed above as potential DNS and FTP servers. If these systems should not be running these services, a thorough security review of these systems is in order.

Finding: The GIAC Enterprise network needs to have the protection of a filtering device (router, firewall, etc.) to protect the assets of the company from the malicious activity originating from the Internet.

2. UDP scan -- A sequence of UDP packets arriving at a frequency rate high enough to trigger the Snort scan detection routines. There were 52 hosts identified as initiating these scans, 4 of which were inside the GIAC Enterprise network – MY.NET.1.3, MY.NET.1.4, MY.NET.1.5 and MY.NET.1.13. Table 3 shows a summary of the internal machines performing UDP scans:

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total))
MY.NET.1.3	2787	2787	348	348
MY.NET.1.13	2542	2542	118	118
MY.NET.1.5	2294	2294	330	330
MY.NET.1.4	2279	2279	332	332

Table 3.

Note that the number of destinations targeted by MY.NET.1.3, MY.NET.1.4 and MY.NET.1.5 are approximately equal. A perusal of the captured detects reveals the following two patterns:

```
Sep 3 09:04:01 MY.NET.1.3:53-> MY.NET.5.202:1025 UDP
Sep 3 09:04:03 MY.NET.1.4:53-> MY.NET.5.202:1026 UDP
Sep 3 09:04:05 MY.NET.1.5:53-> MY.NET.5.202:1027 UDP
Sep 3 09:04:07 MY.NET.1.3:53-> MY.NET.5.202:1028 UDP
Sep 3 09:04:09 MY.NET.1.4:53-> MY.NET.5.202:1029 UDP
Sep 3 09:04:11 MY.NET.1.5:53-> MY.NET.5.202:1030 UDP
```

This example illustrates what appears to be a either a series of responses to DNS queries from MY.NET.5.202 (note the incrementing destination ports) or perhaps a coordinated UDP port scan utilizing MY.NET.1.3, MY.NET.1.4 and MY.NET.1.5. Here is a summary of a 5 minute slice of alerts generated on 9/3:

Timestamp	Source IP:port	Destination IP:port	Count
9/3/00 9:03	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.111:45460 UDP	55
9/3/00 9:03	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.148:2563 UDP	12
9/3/00 9:03	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.198:1032 UDP	37
9/3/00 9:03	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.100.147:123 UDP	14
9/3/00 9:03	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.100.178:123 UDP	16
9/3/00 9:03	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.100.125:123 UDP	17
9/3/00 9:03	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.100.169:123 UDP	16
9/3/00 9:03	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.1.1:123 UDP	17
9/3/00 9:04	MY.NET.1.3:123, MY.NET.1.4:123, MY.NET.1.5:123	MY.NET.100.187:123 UDP	5
9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.214:1283 UDP	8
9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.164:31780 UDP	46
9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.164:53 UDP	38
9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.121:123 UDP	15

9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.164:123 UDP	14
9/3/00 9:05	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.10.198:1491 UDP	11
9/3/00 9:06	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.121:55890 UDP	34
9/3/00 9:06	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.187:1029 UDP	28
9/3/00 9:08	MY.NET.1.13:7003	MY.NET.100.121:7001 UDP	9
9/3/00 9:08	MY.NET.1.3:53, MY.NET.1.4:53, MY.NET.1.5:53	MY.NET.100.165:6821 UDP	23

Table 4.

Note that in most, but not all of the cases listed above the destination port listed is the start of a linearly increasing sequence of port numbers. The UDP traffic from these three hosts from port 123 to port 123 could either be legitimate NTP responses or a camouflaged host scan of the network.

In order to determine if the traffic being generated on source port 53 from these three machines is legitimate DNS query responses, we performed a time-based summary of all instances of packets that fit the

```
(MY.NET.1.3:53, MY.NET.1.4:53, or MY.NET.1.5:53) -> MY.NET.xxx.yyy:zzzz
```

and

```
 (MY.NET.1.3:123, MY.NET.1.4:123, or MY.NET.1.5:123) -> MY.NET.xxx.yyy:123
```

patterns of traffic. If these are DNS servers for the GIAC Enterprise network we expect to get a reasonably even distribution of traffic for each day. What we found is illustrated in Table 5:

Date	Packets	Number	Hosts
8/15	26	2	MY.NET.101.89, MY.NET.101.99
8/16	25	1	MY.NET.101.89
8/17	31	1	MY.NET.101.89
8/18	9	1	MY.NET.101.89
8/28	64	4	MY.NET.101.89, MY.NET.101.140, MY.NET.101.141, MY.NET.101.142
9/2	11	1	MY.NET.101.89
9/3	9,647	428	MY.NET.101.89 + 427 others
9/5	8	1	MY.NET.101.89
9/6	16	1	MY.NET.101.89
9/7	15	1	MY.NET.101.89
9/8	10	1	MY.NET.101.89
9/11	16	1	MY.NET.101.89
9/14	20	1	MY.NET.101.89

Table 5.

Three things are immediately apparent from this analysis: 1) the burst of activity on 9/3 is obviously abnormal, and thus suspicious, 2) the host MY.NET.101.89 always had packets sent to it on each of the other days when packets originated from MY.NET.1.3, MY.NET.1.4, or MY.NET.1.5, and 3) these packets are probably not DNS query responses. When we examine other analyses performed on earlier datasets, we find that
MY.NET.1.3, MY.NET.1.8, MY.NET.99.51 and MY.NET.253.114 were listed as possibly compromised hosts (http://www.sans.org/y2k/practical/Tammy_Fletcher.doc),
MY.NET.1.3, MY.NET.253.12, MY.NET.1.4 and MY.NET.100.164 were listed as possibly compromised hosts (http://www.sans.org/y2k/practical/Guy_Bruneau.doc), and
MY.NET.1.3, MY.NET.1.4 and MY.NET.101.92 were listed as possibly compromised (http://www.sans.org/y2k/practical/William_Lorimer.doc).

Finding: This analysis indicates that MY.NET.1.3, MY.NET.1.4 and MY.NET.1.5 are compromised and it is a strong probability that MY.NET.101.89 has also been compromised. These systems should be removed from the GIAC Enterprise network until a security audit can be performed on them.

Finding: Insufficient information was provided to determine the exact mechanism controlling the security event on 9/3. We recommend that additional NIDS resources be deployed within the GIAC Enterprises network to more closely monitor and capture suspicious network traffic until such time as the current infestation is cleaned up.

The other host identified in Tables 3 and 4 that needs to be scrutinized is MY.NET.1.13. Table 6 lists the destination ports that MY.NET.1.13 sent UDP packets to (118 different destination hosts):

Count	Destination Port
26	111
2231	7001
69	7002
48	7003
21	7008
61	7021
68	7028

Table 6.

Here is an excerpt from the IANA list of well-known port numbers (http://www.isi.edu/in-notes/iana/assignments/port-numbers):

```
#                              John Murphy
afs3-fileserver7000/tcp   file server itself
afs3-fileserver7000/udp   file server itself
afs3-callback  7001/tcp   callbacks to cache managers
afs3-callback  7001/udp   callbacks to cache managers
afs3-prserver  7002/tcp   users & groups database
afs3-prserver  7002/udp   users & groups database
afs3-vlserver  7003/tcp   volume location database
```

```
afs3-vlserver  7003/udp   volume location database
afs3-kaserver  7004/tcp   AFS/Kerberos authentication service
afs3-kaserver  7004/udp   AFS/Kerberos authentication service
afs3-volser    7005/tcp   volume managment server
afs3-volser    7005/udp   volume managment server
afs3-errors    7006/tcp   error interpretation service
afs3-errors    7006/udp   error interpretation service
afs3-bos       7007/tcp   basic overseer process
afs3-bos       7007/udp   basic overseer process
afs3-update    7008/tcp   server-to-server updater
afs3-update    7008/udp   server-to-server updater
afs3-rmtsys    7009/tcp   remote cache manager service
afs3-rmtsys    7009/udp   remote cache manager service
#
ups-onlinet     7010/tcp   onlinet uninterruptable power supplies
ups-onlinet     7010/udp   onlinet uninterruptable power supplies
#                         Brian Hammill
talon-disc     7011/tcp   Talon Discovery Port
talon-disc     7011/udp   Talon Discovery Port
talon-engine   7012/tcp   Talon Engine
talon-engine   7012/udp   Talon Engine
microtalon-dis  7013/tcp   Microtalon Discovery
microtalon-dis  7013/udp   Microtalon Discovery
microtalon-com  7014/tcp   Microtalon Communications
microtalon-com  7014/udp   Microtalon Communications
talon-webserver 7015/tcp   Talon Webserver
talon-webserver 7015/udp   Talon Webserver
#                         Jack Curtin
#               7016-7019  Unassigned
dpserve        7020/tcp   DP Serve
dpserve        7020/udp   DP Serve
dpserveadmin   7021/tcp   DP Serve Admin
dpserveadmin   7021/udp   DP Serve Admin
#                         Allan Stanley
#               7022-7069  Unassigned
```

The traffic to destination ports 7001, 7002, 7003 and 7008 appears to be consistent with client-server traffic between Andrew File System (AFS) enabled hosts. What is suspicious about this traffic is that the AFS client-server packets are logged only between 09:03:19 and 09:10:28 on 9/3 – right in the middle of the significant event identified in Table 5. It is entirely likely that MY.NET.1.13 was used as a resource during the attack. The other ports touched by MY.NET.1.13 include our old friend, portmapper – 111/UDP. Here are the portmapper packets generated by MY.NET.1.13 during the event:

```
Sep  3 09:03:19 MY.NET.1.13:40577 -> MY.NET.6.20:111 UDP
Sep  3 09:03:29 MY.NET.1.13:622 -> MY.NET.6.20:111 UDP
Sep  3 09:03:34 MY.NET.1.13:622 -> MY.NET.6.20:111 UDP
Sep  3 09:03:44 MY.NET.1.13:622 -> MY.NET.6.20:111 UDP
Sep  3 09:04:04 MY.NET.1.13:622 -> MY.NET.6.20:111 UDP
Sep  3 09:05:39 MY.NET.1.13:40579 -> MY.NET.6.31:111 UDP
Sep  3 09:05:49 MY.NET.1.13:624 -> MY.NET.6.31:111 UDP
Sep  3 09:05:54 MY.NET.1.13:624 -> MY.NET.6.31:111 UDP
Sep  3 09:06:04 MY.NET.1.13:624 -> MY.NET.6.31:111 UDP
Sep  3 09:06:24 MY.NET.1.13:624 -> MY.NET.6.31:111 UDP
Sep  3 09:06:49 MY.NET.1.13:40580 -> MY.NET.6.32:111 UDP
Sep  3 09:06:59 MY.NET.1.13:625 -> MY.NET.6.32:111 UDP
Sep  3 09:07:04 MY.NET.1.13:625 -> MY.NET.6.32:111 UDP
Sep  3 09:07:14 MY.NET.1.13:625 -> MY.NET.6.32:111 UDP
Sep  3 09:07:34 MY.NET.1.13:625 -> MY.NET.6.32:111 UDP
Sep  3 09:08:01 MY.NET.1.13:40581 -> MY.NET.6.39:111 UDP
Sep  3 09:08:11 MY.NET.1.13:626 -> MY.NET.6.39:111 UDP
Sep  3 09:08:16 MY.NET.1.13:626 -> MY.NET.6.39:111 UDP
Sep  3 09:08:26 MY.NET.1.13:626 -> MY.NET.6.39:111 UDP
Sep  3 09:08:46 MY.NET.1.13:626 -> MY.NET.6.39:111 UDP
Sep  3 09:09:11 MY.NET.1.13:40582 -> MY.NET.6.44:111 UDP
Sep  3 09:09:21 MY.NET.1.13:627 -> MY.NET.6.44:111 UDP
Sep  3 09:09:26 MY.NET.1.13:627 -> MY.NET.6.44:111 UDP
Sep  3 09:09:36 MY.NET.1.13:627 -> MY.NET.6.44:111 UDP
Sep  3 09:09:56 MY.NET.1.13:627 -> MY.NET.6.44:111 UDP
Sep  3 09:10:21 MY.NET.1.13:40583 -> MY.NET.6.20:111 UDP
```

The repeating source ports are usually indicative of retries, but to be safe, it would be prudent to take a hard look at these 5 hosts.

Finding: Host MY.NET.1.13 is compromised!

Finding: If AFS (Andrew File System) needs to be deployed within the GIAC Enterprise network, the security settings should be reviewed. If AFS is not supposed to be deployed within the GIAC Enterprise network, a thorough security audit of 118 hosts is in order: (list available upon request)

3. Analysis of "full decode" Snort alerts – the data provided in the full decode Snort alerts is restricted to only packets that have illegal combination of TCP flags. There are two large blocks of packets within this data that correspond to SYN-FIN host scans launched against the GIAC Enterprise network. Table 7 lists the 6 hosts that triggered the Snort SYN-FIN alert:

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total))
210.61.144.125 Taiwan, HINET8-144-TW	4784	7222	2392	2411
213.25.136.60 server.roztocze.com.pl	663	1326	663	663
130.149.41.70 bessy.physik.TU-Berlin.DE	4	606	1	1
18.116.0.75 FLUTTER.MIT.EDU	3	18	3	3
210.101.101.110 Korea, KORNET	2	5	1	1
24.201.209.192 modemcable192.209-201-24.mtl.mc.videotron.ca	1	2	1	1

Table 7.

The machine at 130.149.41.70, bessy.physik.TU-Berlin.DE, generated 606 alerts, all of which were illegal combinations of TCP flags. It appears that there something wrong with that machine's hardware.

In addition to the SYN-FIN alerts listed above, the TCP ******** scan, NMAP TCP ping!, Probable NMAP fingerprint attempt, and Queso fingerprint alerts listed in Table 1 are also associated with a malformed TCP flag field. Even if we attribute all of the packets associated with these "named" illegal TCP flag combinations to malicious activity, that still leaves 1363 packets with other illegal TCP flag combinations. From that total, subtract the 606 packets attributed to 130.149.41.70 in Table 7 and we're still left with 757 packets with illegal TCP flag combinations. Of the packets captured in the full decode Snort alert files, 349 packets originated from hosts in the GIAC Enterprise network. Most of these hosts have only a very few packets attributed to illegal TCP flag combinations – minimum = 1, maximum = 15, average = 2.

Finding: There may be a low-level hardware problem with the GIAC Enterprise network that is causing an abnormally high occurrence of damaged packets.

Assignment 4 -- Analysis Process

Three different formats of Snort detect files were provided for this analysis. The first task was to determine what type of information was being provided by the Snort data. The bulk of the information provided was in the form of Snort alerts recorded in "fast" mode (scan files). A second source of information was provided in the form of Snort alerts recorded in "full" mode (alert files). Finally, approximately 7500 records of Snort alerts recorded with full decode output were provided. The scan and alert files were generated with a standard Snort rulebase. The full decode alerts were all generated from TCP packets that had malformed TCP flag combinations.

The first step in the analysis is to organize the data provided. All of the scan files (SnortS*.txt) were combined into one file (sscan.txt), all of the alert files (SnortA*.txt) were combined into another file (salert.txt) and the decode files (SOOS*.txt) were combined into a third file (flags.txt). Both the scan and alert files contain output related to starting and stopping Snort that is not directly related to the detect information contain therein. In order to facilitate the analysis of the data, the following commands were executed to extract only the pertinent detect lines and then sort the data by timestamp:

Alert file: `egrep "^0" salert.txt | sort > alerts.txt`
Scan file: `egrep "^Aug|^Sep" sscan.txt | sort > scans.txt`

A summary file of the packets contained in the combined flags.txt file was created using the following:

```
egrep "MY.NET" flags.txt | sort > flag-times.txt
```

Each line of the flag-times.txt file contains the timestamp, source IP address and port and the destination IP address and port. It was found to be much easier to search this summary file than the combined decode file.

The primary data analysis tool used to organize and visualize the data was SnortSnarf, (http://www.silicondefense.com/snortsnarf/) a Perl program designed to take Snort alert files and produce HTML output suitable for analysis and troubleshooting. Unfortunately, SnortSnarf does not understand obfuscations such as “MY.NET.” The following Perl scripts were run on the alerts.txt and scans.txt files to change “MY.NET” to 255.254:

```
perl -pi -e "s/MY.NET/255.254/g" alerts.txt
perl -pi -e "s/MY.NET/255.254/g" scans.txt
```

The alerts.txt and scans.txt files were then used as input to Snortsnarf:

```
snortsnarf alerts.txt scans.txt
```

This sorting process performed earlier in the creation of the alerts.txt and scans.txt files had the additional benefit of significantly reducing the memory usage and computation time required by SnortSnarf (your mileage may vary). The resulting directory structure of HTML output generated by SnortSnarf was then published on an Apache web server and a normal web browser was then used to explore the data.

The initial approach to analyzing the data was essentially free-form; just an exploration of what was contained in the data without any formal goal. This has the benefit of acquainting the analyst with the breadth and scope of the data provided. It is also entirely too easy to become overwhelmed with the quantity of data provided. Eventually, it occurred to me to start looking for evidence of compromised systems within the data (it always helps to read the directions!). Once that goal was firmly in mind, the first-cut analysis of the data was straightforward – look for detects that have a source IP address of MY.NET.*.* (or in the case of the HTML views provided by SnortSnarf, 255.254.*.*).

When analyzing the data, I found it important to constantly ask myself, “is this reasonable?” This question takes many forms, such as: “Does the pattern of apparent DNS query responses from MY.NET.1.3, MY.NET.1.4 and MY.NET.1.5 fit what I know of normal DNS query and response patterns?” If not, as in this case, the next step is to formulate another hypothesis and ask yourself “is this reasonable?” It is quite normal to formulate several hypotheses and then end up rejecting them because the data doesn’t fit your theory.

Microsoft Excel was used extensively in the analysis of the data and Microsoft Word in the preparation of this report.

I accumulated the following list of Internet resources to assist in the analysis process:

APNIC Whois	http://www.apnic.net/
ARIN Whois	http://whois.arin.net/whois/index.html
Common Vulnerabilities and Exposures	http://cve.mitre.org/
Department of Defense Network Information Center	http://www.nic.mil/dodnic/
GIAC - CVE Entries	http://www.sans.org/y2k/CVE.htm
Global Incident Analysis Center	http://www.sans.org/giac.htm
IETF RFC Page	http://www.ietf.org/rfc.html
InterNIC Whois	http://www.internic.net/whois.html
NetworkIce - Ports	http://advice.networkice.com/advice/Exploits/Ports/

NSI - WHOIS Lookup	http://whois.networksolutions.com/cgi-bin/whois/whois
Packet Storm - Hacker Code	http://packetstorm.securify.com/
RIPE Whois Database	http://www.ripe.net/cgi-bin/whois
SANS Institute Online - Home Page	http://www.sans.org/newlook/home.htm
SecurityFocus	http://www.securityfocus.com/
SecurityPortal	http://www.securityportal.com/
Silicon Defense	http://www.silicondefense.com/
SiliconDefence - SnortSnarf	http://www.silicondefense.com/snortsnarf/main.html
Simovits Consulting - Trojan Port list	http://www.simovits.com/nyheter9902.html
Snort	http://www.snort.org/
Well Known Port Numbers	http://www.isi.edu/in-notes/iana/assignments/port-numbers
Whitehats Network Security Resource	http://www.whitehats.com/index.shtml