Global Information Assurance Certification Paper

SANS GCIA Practical Assignment

David Oborn

Contents

Net Detects

Log Formats

ZoneAlarm

ZoneAlarm is a packet filtering personal firewall. The log entries used here are one intercept per line with the following format:

date, time, source_address:source_port, destination_address:destination_port, protocol

Snort

The snort logs in this section were generated by snort 1.7 and each intercept generates 4 lines of logging. The logging format is:

The alert text defined in the rule
timestamp src-ip:src-port -> dest-ip:dest-port
TCP TTL type-of-service IP-ID IP-header-length datagram-length
TCP-flags sequence-num ack-num window-size TCP-header-length

fwtk - http-gw proxy

The firewall log entries we use here were generated by the http-gw proxy on an fwtk bastion host. Each web request results in 4 lines being logged. The first part of each line is date/time and identification:

month day time firewall-hostname proxy-name[process-id]:

The second part of each line is either a initial log of the request:

```
log host=inside-host/inside-IP protocol=HTTP cmd=get dest=dest-host path=web-page-path
```

Or a line specifying what action the proxy is taking:

```
permit host=inside-host/inside-IP use of gateway
```

Or a line reporting the MIME content type of the incoming request:

```
content-type= mime-content-type
```

Or a line logged when the connection is closed at the completion of the request:

```
exit host=inside-host/inside-IP cmds=num-cmds-processed in=bytes-received
out=bytes-sent user=unauth duration=seconds
```

Detect 1 - Trojan Scan

```
2001/02/28,17:49:34 -6:00 GMT,24.23.106.20:21748,24.xxx.yyy.zzz:1243,TCP
2001/02/28,17:49:36 -6:00 GMT,24.23.106.20:21750,24.xxx.yyy.zzz:27374,TCP
2001/02/28,17:49:36 -6:00 GMT,24.23.106.20:21751,24.xxx.yyy.zzz:9055,TCP
```

Source of trace

This data was collected by Karen Frederick and was published on GIAC at http://www.sans.org/y2k/030201-0900.htm The machine that logged the activity is on a cable modem.

Detect was generated by

The trace was generated by Zone Alarm. See the log formats section for the ZoneAlarm log format.

Probability the source address was spoofed

These are TCP connection attempts to (in 2 out of the 3) well known trojan port numbers. It is likely that the IP address is not spoofed because the attacker will want to see the results of the connnection attempt. Given the closeness of the source port numbers it looks like the attacking machine may not be doing much other TCP activity at the time of the attck.

Description of the attack

Port numbers 1243 and 27374 are closely associated with the SubSeven trojan (original and 2.1). These first 2 lines are most likely scanning for installed copies of SubSeven. If SubSeven is installed on a machine it is likely to be listening on these port numbers as they are the defaults. The third line tries to connect to port 9055. As of 5 March 2001 I can find no reference to this port number on the security related sites I have referred to (although 9055 happens to be the code for an ICMP flood in more recent versions of Stacheldraht). It seems likely that this line is also scanning for an installed trojan, either a new type or a build that is configured to use a different port number.

Attack mechanism

The SubSeven trojan comes in 3 parts: the server, which resides on the compromised machine, the client, which his used by the attacker to control the servers, and the Edit Server which is used to customise the server before it compromises the victim.

When the server has compromised a machine it attempts to contact the attacker to inform them of the success of the infestation. This is done in a pre-configured manner (set when the attacker used Edit Server). From then on it listens for commands on the port configured by the Edit Server. The attacker uses the client to connect to the server using TCP on the configured port and if sucessful can then send any one of a large number of commands to the server, controlling the compromised machine and possibly using it as a launching point for other attacks.

The activity noted in this detect is an attacker looking for installed copies of the trojan.

Correlations

Scans for SubSeven trojans are very common and have been around for some time, but scans for port 9055 appear to be new and as at 5 March 2001 I have seen only one other report, also in GIAC http://www.sans.org/y2k/030201-0900.htm from Marc Reibstein. His detect shows the same pattern of port numbers.
I found these papers useful when researching SubSeven:
Crapanzano, Jamie. "Deconstructing SubSeven, the Trojan Horse of Choice".
http://www.sans.org/infosecFAQ/malicious/subseven.htm
Wentzel, James. "What is SubSeven? Giving away control of your machine!".
http://www.sans.org/infosecFAQ/malicious/subseven2.htm

Evidence of active targeting

This detect shows only traffic going to 1 machine so it is possible that the machine is being actively targeted. It is most likely though that this is part of a larger scan of the address range of the cable subnet.

Severity

Severity 1.

(Criticality + Lethality) - (System Countermeasures + Network Countermeasures) = Severity
(2+4) - (5+0) = 1

Criticality = 2. No details of the system were posted but as it is on a cable modem it is probably a home machine. It therefore is unlikely to be a critical piece of network infrastructure.
Lethality = 4. If the attack found a working copy of SubSeven on the machine it could be all over (depending on how the attacker used the access)
System Countermeasures = 5. Zone Alarm is installed and is blocking unauthorised ports.
Network Countermeasures = 0. The cable provider is unlikely to provide any network protection.

Defensive recommendation

The system that collected the detect is running a personal firewall: Zone Alarm. As long as port access is managed intelligently this will keep the machine from being remotely controlled by a SubSeven client. Using Zone Alarm will also prevent a remote client connecting to port 9055. A personal firewall will not prevent infection by a trojan though and the usual precautions of running up-to-date virus software, managing shares and patches and not running untrusted code are also worthwhile.

Multiple choice test question

```
2001/02/28,17:49:34 -6:00 GMT,24.23.106.20:21748,24.xxx.yyy.zzz:1243,TCP
2001/02/28,17:49:36 -6:00 GMT,24.23.106.20:21750,24.xxx.yyy.zzz:27374,TCP
2001/02/28,17:49:36 -6:00 GMT,24.23.106.20:21751,24.xxx.yyy.zzz:9055,TCP
```

Given the above ZoneAlarm trace, which of the statements below is most likely to be correct:

a) This is part of a mapping scan to see which hosts on the subnet are alive.
b) 24.23.106.20 is probably looking for compromised PCs
c) 24.23.106.20 is probably a spoofed address
d) 24.23.106.20 has probably been infected by trojan software and is trying to "phone home"

Answer: (b)

Detect 2 - Host scan on ports 511 and 510

From the Alert file:

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.865104 195.114.67.218:511 -> aaa.bbb.ccc.97:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.866406 195.114.67.218:511 -> aaa.bbb.ccc.102:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.866726 195.114.67.218:511 -> aaa.bbb.ccc.103:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.868925 195.114.67.218:511 -> aaa.bbb.ccc.106:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.869448 195.114.67.218:511 -> aaa.bbb.ccc.107:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-14:54:43.872069 195.114.67.218:511 -> aaa.bbb.ccc.110:511
TCP TTL:233 TOS:0x0 ID:23561 IpLen:20 DgmLen:40
******S* Seq: 0x36E947C5 Ack: 0x7960F1D5 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.843349 195.114.67.218:510 -> aaa.bbb.ccc.102:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.843415 195.114.67.218:510 -> aaa.bbb.ccc.99:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.843617 195.114.67.218:510 -> aaa.bbb.ccc.103:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]

03/13-16:11:41.844007 195.114.67.218:510 -> aaa.bbb.ccc.100:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.847505 195.114.67.218:510 -> aaa.bbb.ccc.104:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.848101 195.114.67.218:510 -> aaa.bbb.ccc.107:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.848261 195.114.67.218:510 -> aaa.bbb.ccc.108:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

Source of trace

This trace was captured from a sensor on the outside of a corporate firewall co-managed by the author.

Detect was generated by

The detect was generated by snort 1.7 using the "new" reduced ruleset downloaded from www.snort.org in early March 2001. The specific rule that captured this traffic is a local addition that simply catches all inward TCP connection attempts that are not explicitly allowed. See the log formats section for the snort log format.

Probability the source address was spoofed

These packets appear to be probes to see if particular hosts exist and whether they listen on these ports. The attacker would in most cases need to provide their real address in order to see the reply come back. It seems unlikely that the source address is spoofed.

Description of the attack

The attacker sends SYN packets to addresses in the selected range to see if they are there and whether they are listening on the selected ports. If they are then they could be compromised hosts offering a root shell.

Attack mechanism

The packets are in 2 separate TCP scans. The first scan is for port 511 on 6 hosts and the second

is for port 510 on 7 hosts. The total number of host addresses scanned over the 2 scans is 10. The scan range contains gaps and doesn't appear to follow subnet boundaries.

The packets are all very similar within each scan and there are also similarities between the 2 scans.

The following characteristics suggest that the packets are crafted:

- Within each scan the packets keep the same source port, IP ID and sequence number, which is not the case with normal TCP SYN packets. I would expect to see all of these values change for a new connection attempt.
- The ack number in each packet is non-zero which very odd for a SYN packet. I would expect it to be 0.
- The window size is amazingly low at 40 bytes.
- There are no retry packets - an innocent connection attempt would try a few times before giving up.

Between the scans the attacker has changed the IP ID, source port, sequence number and ack number.

Correlations

This type of scan has been noted on SANS by a number of people. Activity on port 511 has been noted by Don Elsner , Keith Pachulski (who dealt with a t0rn compromised system using 511 as a rootshell port) and Bruce Lilly, and has generally been attributed to scans looking for systems running t0rn.
Activity on port 510 has been noted by many people (including David Sullivan and Blair Perry)

Evidence of active targeting

A second scan on a different port number within 20 minutes could be evidence that the attacker has seen that there were no responses to the first scan and decided to try again, and therefore suggests active targeting, but they haven't targeted a particular host. I am surprised that the attacker has not covered the whole subnet or tried different mapping techniques (there is only one real advertised host in this range so they have not made a selection based on DNS entries).

Severity

Severity -2.

(Criticality + Lethality) - (System Countermeasures + Network Countermeasures) = Severity
(5+2) - (4+5) = -2

Criticality = 5. The probed network contains an busy firewall.
Lethality = 2. This is reconnaissance and not an exploit.
System Countermeasures = 4. The firewall is hardened and monitored. Note that I have rated the system counter measures on the firewall rather than the end system because there is no end

system for most of these addresses. The one address that does exist is the firewall. Network Countermeasures = 5. The firewall screening blocks these ports.

Defensive recommendation

Add these ports as specific rules in snort in order to better track this type of probe. No extra hardening is required as the existing file integrity checking would pick up an infection of this type.

Multiple choice test question

[**] LOCAL blocked connection attempted [**]
03/13-16:11:41.848261 195.114.67.218:510 -> aaa.bbb.ccc.108:510
TCP TTL:233 TOS:0x0 ID:43651 IpLen:20 DgmLen:40
******S* Seq: 0x40BF94EF Ack: 0x48EEDB18 Win: 0x28 TcpLen: 20

In the above packet, which field(s) stand out as being a bit odd?
a) The TTL seems too big
b) The source and destination ports shouldn't be the same
c) The sequence number is the same as in a known DOS attack
d) The window is tiny and the ack number is non-zero

Answer: (d)

Detect 3 - SMTP Overflow (false alarm)

From the Alert file:

```
[**] SMTP chameleon overflow [**]
03/13-17:03:46.798285 203.96.152.32:1362 -> aaa.bbb.ccc.ddd:25
TCP TTL:63 TOS:0x0 ID:53983 IpLen:20 DgmLen:552 DF
***AP*** Seq: 0xADE428AC  Ack: 0x63B94CFB  Win: 0x4000  TcpLen: 20
```

The dump of the packet (first part only, for brevity):

```
03/13-17:03:46.798285 203.96.152.32:1362 -> aaa.bbb.ccc.ddd:25
TCP TTL:63 TOS:0x0 ID:53983  DF
*****PA* Seq: 0xADE428AC   Ack: 0x63B94CFB   Win: 0x4000
68 65 6C 70 20 69 6E 20 64 65 63 69 64 69 6E 67  help in deciding
20 77 68 61 74 20 74 6F 20 77 65 61 72 20 74 68   what to wear th
6F 75 67 68 2E 0D 0A 46 72 65 64 0D 0A 2D 2D 2D  ough...Fred..---
2D 2D 20 4F 72 69 67 69 6E 61 6C 20 4D 65 73 73  -- Original Mess
61 67 65 20 2D 2D 2D 2D 2D 0D 0A 46 72 6F 6D 3A  age -----..From:
```

[rest of packet (lots of addresses) deleted]

Source of trace

This trace was captured from a sensor on the outside of a corporate firewall co-managed by the author.

Detect was generated by

The detect was generated by snort 1.7 using the "new" reduced ruleset downloaded from www.snort.org in early March 2001.
The rule that generated this alert was:

```
alert tcp $EXTERNAL_NET any -> $SMTP 25 (msg:"SMTP chameleon overflow";
content: "HELP "; nocase; flags: A+; dsize: >500; depth: 5;
reference:arachnids,266; reference:cve,CAN-1999-0261;)
```

See the log formats section for the snort log format.

Probability the source address was spoofed

The packet is from the middle of an established TCP connection so it is very unlikely to be spoofed.

Description of the attack

The alert suggestes that this packet is an attempt to trigger a buffer overflow in a version of the Netmanager Chameleon SMTP daemon .

Attack mechanism

One of the buffer overflows in Netmanager Chameleon SMTPd is triggered by specifying the command HELP topic with the topic longer than 514 characters (see http://www.insecure.org/sploits/netmanage.chameleon.overflows.html).

It can be seen from the contents of the packet that this is not an attack. The packet is a normal fragment of a mail message but matches the alert rule because it contains the string "help " and is longer than 500 characters.

Correlations

This is not an attack so correlation is not required.

Evidence of active targeting

We are not being actively targetted.

Severity

Severity -3

(Criticality + Lethality) - (System Countermeasures + Network Countermeasures) = Severity
(5+0) - (4+4) = -3

Criticality = 5. The affected machine is a busy firewall.
Lethality = 0. This is a false alarm..
System Countermeasures = 4. The SMTP server is hardened and monitored.
Network Countermeasures = 4. A restrictive firewall is in place but it doesn't block port 25

Defensive recommendation

If we get too many of these it may be a good idea to exclude this rule from our IDS to reduce false alarms as we do not use the affected product, the attack is fairly old and the likelyhood of false alarms is high.

Multiple choice test question

```
alert tcp $EXTERNAL_NET any -> $SMTP 25 (msg:"SMTP chameleon overflow";
content: "HELP "; nocase; flags: A+; dsize: >500; depth: 5;
reference:arachnids,266; reference:cve,CAN-1999-0261;)
```

You want to know more about the exploit that the above snort rule detects. A good starting point would be to:

a) Go to www.arachnids.org
b) Use google to search for chameleon
c) Search for CAN-1999-0261 on www.cve.mitre.org
d) Look for a Canadian CERT web site because they probably found the vulnerability first.

Answer: (c)

Detect 4 - x86 NOOP signature (false alarm)

From the alert file

```
[**] EXPLOIT x86 NOOP [**]
03/13-17:03:18.488815 205.188.252.121:80 -> aaa.bbb.ccc.100:3345
TCP TTL:36 TOS:0x0 ID:57358 IpLen:20 DgmLen:576
***A**** Seq: 0x25CA3454  Ack: 0x62184F92  Win: 0x7FFF  TcpLen: 20
```

A dump of that packet

```
03/13-17:03:18.488815 0:2:B9:92:9:C1 -> 0:2:B9:EE:2B:61 type:0x800 len:0x24E
205.188.252.121:80 -> aaa.bbb.ccc.100:3345 TCP TTL:36 TOS:0x0 ID:57358
IpLen:20 DgmLen:576
***A**** Seq: 0x25CA3454  Ack: 0x62184F92  Win: 0x7FFF  TcpLen: 20
0x0000: 00 02 B9 EE 2B 61 00 02 B9 92 09 C1 08 00 45 00  ....+a........E.
0x0010: 02 40 E0 0E 00 00 24 06 B0 DE CD BC FC 79 D2 30  .@....$......y.0
0x0020: 67 64 00 50 0D 11 25 CA 34 54 62 18 4F 92 50 10  gd.P..%.4Tb.O.P.
0x0030: 7F FF 56 0C 00 00 92 92 92 92 92 92 92 92 92 92  ..V.............
```

```
0x0040: 92 92 92 93 92 92 92 92 92 92 92 91 91 91 91 91  ................
0x0050: 91 91 92 92 92 92 92 92 92 92 92 92 92 92 93 93  ................
0x0060: 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92  ................
0x0070: 92 92 92 92 92 92 92 92 92 92 93 92 92 92 92 92  ................
0x0080: 92 92 92 93 93 93 93 93 93 93 93 93 93 93 94 94  ................
0x0090: 94 94 94 94 94 93 93 93 93 93 93 93 93 93 93 94  ................
0x00A0: 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 95  ................
0x00B0: 95 95 95 95 95 95 95 95 94 95 95 95 95 95 95 95  ................
0x00C0: 95 94 94 94 94 94 94 94 94 94 94 94 93 93 93 92  ................
0x00D0: 92 92 92 92 91 91 91 91 91 91 91 91 91 91 91 91  ................
0x00E0: 91 91 91 91 91 91 91 91 90 90 90 91 91 91 91 92  ................
0x00F0: 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93  ................
0x0100: 93 93 92 92 92 92 92 92 92 92 92 92 92 92 92 92  ................
0x0110: 92 92 92 92 92 92 92 91 91 91 91 90 90 90 90 8F  ................
0x0120: 8F 8F 8F 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0130: 90 90 91 91 91 91 91 91 91 91 91 91 91 92 92 92  ................
0x0140: 92 92 92 92 92 92 91 92 91 91 91 91 91 91 91 91  ................
0x0150: 90 90 90 90 90 90 90 90 8F 8F 8F 8F 8F 8F 8F 8F  ................
0x0160: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0170: 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91  ................
0x0180: 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91  ................
0x0190: 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91  ................
0x01A0: 91 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91  ................
0x01B0: 91 91 91 91 91 91 91 91 91 91 90 90 90 90 90 90  ................
0x01C0: 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91  ................
0x01D0: 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 92  ................
0x01E0: 92 92 92 92 92 92 92 92 92 93 93 93 92 92 92 92  ................
0x01F0: 92 91 91 91 91 91 91 91 91 91 91 91 91 91 90 90  ................
0x0200: 90 90 90 90 90 90 90 90 90 90 8F 8F 8F 8F 90 90  ................
0x0210: 90 90 90 90 90 8F 8F 90 90 90 90 90 90 90 90 90  ................
0x0220: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0230: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 8F  ................
0x0240: 8F 8F 8F 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E 8E     ...............
```

[205.188.252.121 translates to www.icq.com]

Firewall Trace (all traffic with www.icq.com at the time of the IDS alert)

```
Mar 13 17:02:51 firewall http-gw[9108]: log host=my.web.cache/a.b.c.d
protocol=HTTP cmd=get dest=www.icq.com
path=/soundboards/scm/Voices/wassup.scm
Mar 13 17:02:51 firewall http-gw[9108]: permit host=my.web.cache/a.b.c.d use
of gateway
Mar 13 17:02:52 firewall http-gw[9108]: content-type= application/x-icq-scm
Mar 13 17:02:55 firewall http-gw[9116]: log host=my.web.cache/a.b.c.d
protocol=HTTP cmd=get dest=www.icq.com
path=/soundboards/scm/Voices/wassup_2.scm
Mar 13 17:02:55 firewall http-gw[9116]: permit host=my.web.cache/a.b.c.d use
of gateway
```

```
Mar 13 17:02:56 firewall http-gw[9116]: content-type= application/x-icq-scm
Mar 13 17:03:02 firewall http-gw[9128]: log host=my.web.cache/a.b.c.d
protocol=HTTP cmd=get dest=www.icq.com path=/soundboards/scm/Voices/yoda.scm
Mar 13 17:03:02 firewall http-gw[9128]: permit host=my.web.cache/a.b.c.d use
of gateway
Mar 13 17:03:03 firewall http-gw[9128]: content-type= application/x-icq-scm
Mar 13 17:03:30 firewall http-gw[9108]: exit host=my.web.cache/a.b.c.d cmds=1
in=490747 out=0 user=unauth duration=39
Mar 13 17:03:39 firewall http-gw[9116]: exit host=my.web.cache/a.b.c.d cmds=1
in=555653 out=0 user=unauth duration=44
Mar 13 17:04:27 firewall http-gw[9128]: exit host=my.web.cache/a.b.c.d cmds=1
in=1056501 out=0 user=unauth duration=85
```

Source of trace

The IDS trace was captured from a sensor on the outside of a corporate Internet firewall co-managed by the author.

Detect was generated by

The IDS trace above was generated by snort 1.7 using the "new" reduced ruleset downloaded from www.snort.org in early March 2001.

The rules that generated this alert was:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP";
content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90|"; flags: A+; reference:arachnids,181;)
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP";
content:"|9090 9090 9090 9090 9090 9090 9090 9090|";
reference:arachnids,181;)
```

See the log formats section for the snort log format and the fwtk http-gw log format.

Probability the source address was spoofed

It is very unlikely that the source address is spoofed as the packet appears to come from an established TCP session (ack bit set) and the firewall proxy has logged traffic to that host at the same time.

Description of the attack

The NOOP signature is a number of contiguous bytes that could be no-operation machine languange codes for a particular architecture. NOOPs are often used to pad out buffer overflow attacks, so this alert is indicating that it may have found an attempt to run attack code via a buffer overflow exploit on an x86 architecture machine.

Attack mechanism

Attacks that use buffer overflows are generally an attempt to run an attack program code on the target machine. The attack code is written to the stack of a legitimate program already running on the target (e.g. a daemon) by exploiting a known programming flaw in the particular daemon. Getting the attack code onto the stack and running often requires providing an unexpectedly large input and particular memory alignment (see Chris_Kuethe's practical for a good summary) and the NOOP code is often chosen for the padding.

In this case though, the firewall logs show that the files being downloaded from www.icq.com at the time of the IDS detect were ICQ sound scheme files (x-icq-scm). These files are used to hold sounds to play when ICQ events occur (see http://www.icq.com/sounds) and a search of bugtraq (www.securityfocus.com) on 26 March 2001 failed to find any buffer overflows via sounds files. It is most likely that this is a false alarm.

Correlations

The firewall logs were most useful in showing what activity was in progress with www.icq.com at the time of the IDS detect. Further correlation with other incidents was not necessary because this was a false alarm.

Evidence of active targeting

There was no active targetting because this was a false alarm.

Severity

Severity -3

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

Criticality = 2. Although the connection is proxied by the firewall, the packet was part of an HTTP download so the affected machine would most likely be the workstation doing the download (which isn't all that critical). This is a judgment call though as an attack like this could just as easily go after a vulnerability in an HTTP proxy. I would expect to see that sort of attack in HTML or something else the proxy is likely to be parsing rather than a sound file.
Lethality = 0. This is a false alarm.
System Countermeasures = 1. The PC is not likely to be particularly hardened or monitored for file changes.
Network Countermeasures = 4. A restrictive firewall is in place.

Defensive recommendation

The firewall logs and the IDS logs are not time synchronised. They are fairly close at the moment but automatic synchronisation would be much better.

Multiple choice test question

```
[**] EXPLOIT x86 NOOP [**]
03/13-17:03:18.488815 205.188.252.121:80 -> aaa.bbb.ccc.100:3345
TCP TTL:36 TOS:0x0 ID:57358 IpLen:20 DgmLen:576
***A**** Seq: 0x25CA3454  Ack: 0x62184F92  Win: 0x7FFF  TcpLen: 20
```

In this snort alert, NOOP is:
a) A machine language instruction that can crash a PC
b) Some kind of wierd hacker term
c) A machine language instruction that is often characteristic of trojan software
d) A machine language instruction that often occurs in large bunches in code that exploits buffer overflows.

Answer: (d)

Detect 5 - Reconnaissance from Korea

From the alert file:

```
[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.570449 211.207.193.221:23082 -> aaa.bbb.ccc.102:21
TCP TTL:252 TOS:0x0 ID:4279 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C98FE48  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.574894 211.207.193.221:22916 -> aaa.bbb.ccc.97:21
TCP TTL:113 TOS:0x0 ID:34727 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C94F23D  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.574958 211.207.193.221:22916 -> aaa.bbb.ccc.97:21
TCP TTL:253 TOS:0x0 ID:37043 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C94F23E  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.575024 211.207.193.221:23048 -> aaa.bbb.ccc.98:21
TCP TTL:252 TOS:0x0 ID:4274 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9601C3  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.575093 211.207.193.221:23475 -> aaa.bbb.ccc.99:21
TCP TTL:113 TOS:0x0 ID:34729 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C96D7DF  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

```
[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.575160 211.207.193.221:23475 -> aaa.bbb.ccc.99:21
TCP TTL:253 TOS:0x0 ID:37045 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C96D7E0  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576183 211.207.193.221:23077 -> aaa.bbb.ccc.101:21
TCP TTL:252 TOS:0x0 ID:4278 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9876FB  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576254 211.207.193.221:23082 -> aaa.bbb.ccc.102:21
TCP TTL:113 TOS:0x0 ID:34732 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C98FE47  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576436 211.207.193.221:23082 -> aaa.bbb.ccc.102:21
TCP TTL:253 TOS:0x0 ID:37049 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C98FE48  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576501 211.207.193.221:22903 -> aaa.bbb.ccc.96:21
TCP TTL:252 TOS:0x0 ID:4272 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9452A0  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576569 211.207.193.221:22903 -> aaa.bbb.ccc.96:21
TCP TTL:253 TOS:0x0 ID:37042 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9452A0  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576637 211.207.193.221:22903 -> aaa.bbb.ccc.96:21
TCP TTL:113 TOS:0x0 ID:34726 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C94529F  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576704 211.207.193.221:22916 -> aaa.bbb.ccc.97:21
TCP TTL:252 TOS:0x0 ID:4273 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C94F23E  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] spp_portscan: PORTSCAN DETECTED from 211.207.193.221 (THRESHOLD 4
connections exceeded in 0 seconds) [**]
03/23-16:24:54.579569
[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576773 211.207.193.221:23048 -> aaa.bbb.ccc.98:21
```

```
TCP TTL:113 TOS:0x0 ID:34728 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C9601C2  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576842 211.207.193.221:23048 -> aaa.bbb.ccc.98:21
TCP TTL:253 TOS:0x0 ID:37044 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9601C3  Ack: 0x0  Win: 0x0  TcpLen: 20
```

From the APNIC Whois database:

```
Search results for '211.207.193.221'

inetnum              211.206.0.0 - 211.211.255.255
netname              HANANET
descr                Hanaro Telecom, Inc.
country              KR
admin-c              CWP3-AP
tech-c               SCH7-AP
mnt-by               MNT-KRNIC-AP
changed              hostmaster@apnic.net 20001228
source               APNIC
```

Source of trace

The IDS trace was captured from a sensor on the outside of a corporate Internet firewall co-managed by the author.

Detect was generated by

Snort 1.7, see the Log Formats section for the snort log format details. The specific rule that captured this traffic is a local addition that simply catches all TCP traffic destined for hosts that do not exist or are not externally visible.

Probability the source address was spoofed

The source address is very unlikely to be spoofed because the attacker is using TCP and will want to see the responses.

Description of the attack

This activity appears to be doing a couple of different tasks. The SYN packets are an attempt to create a map of which hosts on our range allow incoming ftp connections. The function of the RESET+ACK packets is unclear but they may be an attempt to do some crude fingerprinting.

Attack mechanism

Three packets have been sent to each target IP address. The 3 packets are not in any particular order and also are scattered amongst packets to other hosts in the same scan. Each 3 packets to a particular IP address come from the same source port. The IP IDs jump around - they are not constantly increasing as could be expected in normal circumstances. The 3 packets are: 1 connection attempt to port 21 (ftp) and 2 RESET+ACK packets to the same port. The packets with the ack bit set have the acknowledgment number set to 0. The TTLs and IP IDs are interesting:

- The SYN packets all have a TTL of 113 and an IP ID in the 3472x range.
- The RESET+ACK packets that have an IP ID in the 427x range all have a TTL of 252
- The RESET+ACK packets with an IP ID in the 3704x range all have a TTL of 253.

These are interesting patterns and seem unlikely to ocurr naturally. The TTL for packets received from a particular host could usually be expected to stay static (over reasonably short time frames) as a particular path through the internet is likely to last for a while at least. It seems most unlikely that SYN packets would always travel a path that is twice the length of the path travelled by RESET+ACK packets from the same host at the same time. It is also unlikely that a any implementation of TCP would produce IP IDs in this manner.

These patterns lead me to conclude that these are crafted packets.

The next question is why was this attacker sending us this traffic. The direct information on our FTP status would be of interest to someone looking for places to store warez or looking for other FTP server vulnerabilities to exploit. The mapping information would be useful reconnaisance for any later attack, but it is likely that if mapping was the aim then the attacker would have tried other likely ports - for example ones that are more likely to be open.

Correlations

I couldn't find any reference to this IP address on www.sans.org but with known vulnerabilities in wu-ftpd (and no doubt others) I expect scanning for ftp servers is fairly common.

Evidence of active targeting

The choice of only one probe port and the otherwise undirected nature of the scan indicates a lack of good information about the target. This is unlikely to be an attack specifically directed at us -- it is more likely to be just someone out mapping FTP servers.

Severity

Severity -2.

(Criticality + Lethality) - (System Countermeasures + Network Countermeasures) = Severity
(5+2) - (4+5) = -2

Criticality = 5. The probed network contains an busy firewall.
Lethality = 2. This is reconnaissance and not an exploit.
System Countermeasures = 4. The firewall is hardened and monitored. Note that I have rated the system counter measures on the firewall rather than the end system because there is no end system for most of these addresses. The one address that does exist is the firewall.
Network Countermeasures = 5. The firewall screening blocks incoming FTP.

Defensive recommendation

Keep blocking incoming FTP.

Multiple choice test question

```
[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576704 211.207.193.221:22916 -> aaa.bbb.ccc.97:21
TCP TTL:252 TOS:0x0 ID:4273 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C94F23E  Ack: 0x0  Win: 0x0  TcpLen: 20

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576773 211.207.193.221:23048 -> aaa.bbb.ccc.98:21
TCP TTL:113 TOS:0x0 ID:34728 IpLen:20 DgmLen:48 DF
******S* Seq: 0x6C9601C2  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] LOCAL TCP traffic to non-bastion host [**]
03/23-16:24:54.576842 211.207.193.221:23048 -> aaa.bbb.ccc.98:21
TCP TTL:253 TOS:0x0 ID:37044 IpLen:20 DgmLen:40
***A*R** Seq: 0x6C9601C3  Ack: 0x0  Win: 0x0  TcpLen: 20
```

Looking at the TTL fields in the above packets, which of the statements below seem most reasonable:

a) There is some perfectly normal dynamic re-routing going on
b) Maybe these packets are from traceroute
c) Those TTLs are a bit odd, maybe they are crafted
d) Somewhere on their path there is a faulty router

Answer: (c)

Bind 8.2.2 INFOLEAK and TSIG - bind8x.c

Introduction

The bind8x.c program uses 2 vulnerabilities in ISC Bind to first discover information about a

name server daemon and then use that information in an attempt to obtain elevated priviledges on the server. Bind8x.c contains Linux specific shell code but could be adapted to work against other architectures.

I obtained the source of bind8x.c from http://packetstorm.securify.com/0102-exploits/bind8x.c .

The test bed used in this paper was a RedHat 6.0 machine running bind 8.2.2 as the target and with bind8x.c running on a Solaris 8 workstation. The traces were captured using ethereal on the Solaris workstation.

Traces

Summary

First an (edited) ethereal trace summary of the exchange.

```
No. Time       Source      Dest       Proto Info
1 0.000000     attacker    target     DNS   Inverse query
2 0.008756     target      attacker   DNS   Inverse query response, Format error
3 0.009155     attacker    target     DNS   Standard query unused <Name goes
past end of captured data in packet>
4 0.014304     target      attacker   DNS   Standard query response
5 0.014336     attacker    target     ICMP  Destination unreachable
6 0.019776     attacker    target     TCP   33902 > 36864 [SYN] Seq=301077146
Ack=0 Win=24820 Len=0
7 0.021989     target attacker        TCP   36864 > 33902 [RST, ACK] Seq=0
Ack=301077147 Win=0 Len=0
```

Packet 1 is an exploitation of the infoleak vulnerability (CERT VU#325431).
Packet 2 is the reply containing information which will be used for inserting the shell code in the TSIG exploit.
Packet 3 is the TSIG exploit containing the shellcode (CERT vulnerability VU#196945).
Packet 4 is the response from the target just before it runs the shell code (or, in my case, just before the name server dies with a segmentation violation)
Packet 5 is presumably caused because bind8x.c doesn't bother listening for the reply to its query so the attacker machine generates this "port unreachable" reply.
Packet 6 is bind8x.c trying to connect to its shellcode - which if all is well at this point will be listening on port 36864
Packet 7 is the target telling the attacker that there is no process listening on that port - in other words the attempt to gain a root shell has failed. In my case it fails because some of the data in the exploit is compiler option (and probably Linux version) dependant. The same code reportedly works on RedHat 6.1 with the same version of bind [1].

The Infoleak query (packet 1)

The first packet sent to the target is a DNS inverse query which is crafted so as to confuse the name server - it has no questions (which is fairly odd thing to see in a query) and 1 answer. The

answer record says it is 255 bytes long, which is way past the end of the packet.

```
Domain Name System (query)
    Transaction ID: 0xbeef
    Flags: 0x0980 (Inverse query)
        0... .... .... .... = Query
        .000 1... .... .... = Inverse query
        .... ..0. .... .... = Message is not truncated
        .... ...1 .... .... = Do query recursively
    Questions: 0
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    Answers
        <Root>: type A, class inet, addr 0.0.0.0
            Name: <Root>
            Type: Host address
            Class: inet
            Time to live: 1 second
            Data length: 255
            Addr: 0.0.0.0
```

The infoleak reply (packet 2)

We see this reply back from the target machine (ethereal output).

```
User Datagram Protocol
    Source port: domain (53)
    Destination port: 36516 (36516)
    Length: 727
    Checksum: 0xd389

Domain Name System (response)
    Transaction ID: 0xbeef
    Flags: 0x8981 (Inverse query response, Format error)
        1... .... .... .... = Response
        .000 1... .... .... = Inverse query
        .... .0.. .... .... = Server isn't an authority for domain
        .... ..0. .... .... = Message is not truncated
        .... ...1 .... .... = Do query recursively
        .... .... 1... .... = Server can do recursive queries
        .... .... .... 0001 = Format error
    Questions: 0
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
```

It looks fairly innocuous - on the face of it it just says that the previous packet had a format error. The returned packet however contains the contents of a portion of the bind server's process space.

If we look at the code in bind8x.c that processes this response we see that there is more information in the response than is obvious from the DNS decode. The variable buff contains the response packet.

```
long xtract_offset(char* buff, int len)
{
 long ret;

 /* Here be dragons. */
 /* (But seriously, the values here depend on compilation options
         *  used for BIND.
         */
 ret = *((long*)&buff[0x214]);
 argevdisp1 = 0x080d7cd0;
 argevdisp2 = *((long*)&buff[0x264]);
 printf("[d] argevdisp1 = %08x, argevdisp2 = %08x\n",
  argevdisp1, argevdisp2);

 // dumpbuf(buff, len);

 return(ret);
}
```

This code is pulling out the values at offset 532 and 612 to plug into the shellcode.

Sending the TSIG exploit (packet 3)

Having gathered most of the information it needs to run the shellcode (some is already hardcoded), bind8x.c then sends out the packet containing the shellcode (hidden in 7 questions) with the TSIG record in the additional records section.

```
Domain Name System (query)
    Transaction ID: 0xdead
    Flags: 0x0180 (Standard query)
        0... .... .... .... = Query
        .000 0... .... .... = Standard query
        .... ..0. .... .... = Message is not truncated
        .... ...1 .... .... = Do query recursively
    Questions: 7
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
```

```
    Queries
        <Name goes past end of captured data in packet>: type unused, class
unknown
            Name: <Name goes past end of captured data in packet>
            Type: unused
            Class: unknown

        [ 5 duplicate query records snipped ... ]


        <Name goes past end of captured data in packet>: type unused, class
unknown
            Name: <Name goes past end of captured data in packet>
            Type: unused
            Class: unknown
    Additional records
```

The packet triggers error handling within the name server because it contains a TSIG record but no valid key [2]. The error handler makes an assumption about the size of the buffers it is working with and ends up writing the shellcode in a place where it can be executed.

The TSIG exploit reply (packet 4)

The name server sends this packet in reply to the TSIG exploit packet. By this time the shellcode has already been installed and is about to run.

```
Domain Name System (response)
    Transaction ID: 0xdead
    Flags: 0x8180 (Standard query response, No error)
        1... .... .... .... = Response
        .000 0... .... .... = Standard query
        .... .0.. .... .... = Server isn't an authority for domain
        .... ..0. .... .... = Message is not truncated
        .... ...1 .... .... = Do query recursively
        .... .... 1... .... = Server can do recursive queries
        .... .... .... 0000 = No error
    Questions: 7
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        <Name goes past end of captured data in packet>: type unused, class
unknown
            Name: <Name goes past end of captured data in packet>
            Type: unused
            Class: unknown
```

```
        [ 5 duplicate query records snipped ... ]

        <Name goes past end of captured data in packet>: type unused, class
unknown
            Name: <Name goes past end of captured data in packet>
            Type: unused
            Class: unknown
    Additional records
        <Root>: type unknown, class unknown
            Name: <Root>
            Type: Unknown RR type (250)
            Class: unknown
            Time to live: 0 time
            Data length: 17
            Data
```

The final stages

The ICMP port unreachable packet (packet 5) is irrelevant because it is just a side effect of bind8x.c not bothering to listen for a reply to its attack.

Packet 6 shows where the conversation would go if it had worked on my test-bed. Bind8x.c tries to connect to port 36864 on which the shellcode should be listening by now. The shellcode performs the following steps[3]:

- create a socket
- bind it to port 36864
- listen on it
- when a connection arrives, accept it
- start /bin/sh and connect it up to the network connection

These steps will result in the attacker having a root shell if named is running as root.

Lessons

What have I learnt from playing with this code and analysing the traces?

- It reinforced the value of defense in depth. If the site implements a policy of "that which is not explicitly allowed, is denied" then the followup inward TCP connection to port 36864 would be unlikely to succeed and the attacker would not know whether the attack had succeeded.
- That shellcode attacks are fairly "brittle." If the target's compiler options or operating system are a bit different from expected then the exploit may not work. Denying the attacker good recconaisance may help gain an edge for the defenders.
- That daemons shouldn't run as root is it is possible to avoid it (and it certainly is possible with named, see the recommendations section of [2]).

References

1. Miller, Toby. "Hacker Tools and Their Signatures, Part One: bind8x.c". Revised 11 April 2001. http://www.securityfocus.com/focus/ids/articles/bind8.html
2. Squires, Alicia. "Recent BIND Vulnerabilities With an Emphasis on the "tsig bug"". 16 February 2001. http://www.sans.org/infosecFAQ/DNS/BIND.htm
3. http://packetstorm.securify.com/0102-exploits/bind8x.c .
4. Carnegie Mellon Software Engineering Institute. "CERT Advisory CA-2001-02 Multiple Vulnerabilities in BIND". Revised 4 April 2001. http://www.cert.org/advisories/CA-2001-02.html
5. Internet Software Consortium. "BIND Vulnerabilities". http://www.isc.org/products/BIND/bind-security.html

Analyse This

Restriction

Between the time you captured this data and when I started work on it the snort ruleset changed dramatically (a new "cleaned up" version was released on 1 March 2001). The earlier ruleset that you used is nolonger available so I have been unable to look at the exact definitions of the rules you used. In many cases this hasn't mattered but in a few areas I have had to reccommend capturing more data when access to the ruleset may have been sufficient.

Top Issues

Sun RPC Activity

The following systems have been on the receiving end of suspicious looking activity on port 32771:

MY.NET.99.51	MY.NET.206.222	MY.NET.221.130
MY.NET.99.51	MY.NET.17.44	MY.NET.218.238

The source port of some of the activity to MY.NET.221.130 was 3456 which could be the "terror trojan".

The port 32771 activity to MY.NET.202.94 was probably a securedesign.net self scan.

MY.NET.6.15 saw quite a lot of possible RPC activity from outside your network and also received a statd attack as shown below. There is no suspicious activity from MY.NET.6.15 in the alert file after the statd attack to suggest that it has been compromised, but you may wish to check. You may also wish to check whether it is necessary to allow portmapper access to this

machine from the outside.

```
2001-01-06.n:01/06-05:04:21.793408  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-05:04:21.829933  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-05:04:21.830004  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-05:04:21.888825  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-05:04:21.888876  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-05:04:21.919235  [**] External RPC call [**]
206.210.80.6:1414 -> MY.NET.6.15:111
2001-01-06.n:01/06-06:39:35.583605  [**] STATDX UDP attack [**]
206.210.80.6:1074 -> MY.NET.6.15:32776
```

Internal Scanner

On 18 January 2001 MY.NET.70.38 conducted 2 scans of portions of your internal network - a "stealth" TCP scan and a scan looking for print daemons (for which there is a known exploit). This host could be in use for authorized network scanning or it could be compromised. I would tend to suspect it is compromised as it earlier conducted a single packet probe of an external address:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
01/04-12:33:02.700945 MY.NET.70.38:52576 -> 203.202.20.66:88
TCP TTL:42 TOS:0x0 ID:63272
**SF*P*U Seq: 0xAEF17506   Ack: 0x0   Win: 0x1000
TCP Options => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL EOL
```

Unusual Activity on MY.NET.202.94

Check the paragraph on MY.NET.202.94 in the Top 5 Scan Destinations section.

Smurfing

Your routers appear to allow broadcast pings into MY.NET.70.0 and this facility has been used by a great many people. Allowing broadcast pings gives attackers the chance to use your network as a smurf amplifier to launch denial of service attacks against other networks. I strongly reccommend that you block broadcast pings at your routers (for Cisco routers use the "no ip directed-broadcast" interface command).

Tiny Fragments

Many alerts about "Tiny fragments - Possible Hostile Activity" have been seen on a number of hosts. It is not obvious what type of traffic is causing these alerts. I noticed the following characteristics:

- Most of the source addresses have no reverse DNS lookup
- The source addresses that did have reverse lookups were not other servers (e.g. web or DNS) and looked like dialup clients.
- Some of the source addresses were almost certainly false (e.g. 8.8.8.8, 4.4.4.4)

On 29 November MY.NET.219.122 sent tiny frags to an external address without any apparent stimulus.

To nail this down you would need to collect the contents of the packets and have a look at them. Without that data I would suspect that most of this traffic is probably attempted denial of service attacks. I would recommend looking at MY.NET.219.122 to see if it is infected with hostile software.

The following hosts were on the receiving end of the 'Tiny Fragment' traffic.

MY.NET.1.10	MY.NET.1.8	MY.NET.1.9	MY.NET.100.230
MY.NET.201.14	MY.NET.202.18	MY.NET.215.106	MY.NET.217.162
MY.NET.219.46	MY.NET.60.11	MY.NET.71.38	MY.NET.98.123

Probable ICQ Activity (RPC false alarms)

The alerts file contains warnings of a lot of supposedly SUN RPC traffic between external and internal systems on the SUN RPC high port of 32771 where the source port is ICQ (4000). The external hosts engaged in this activity appear to mostly be (from their names) AOL ICQ servers. In one case (MY.NET.213.158) there is additional traffic with a source port of 9898 (MonkeyCom -- a Japanese file transfer/TV/phone product - see http://www.sans.org/y2k/010900.htm).

My conclusion is that people are using their PCs/Macs for ICQ (and in one case MonkeyCom).

I suggest checking the machines listed below to see if any of the machines are Suns -- if they are, you will need to check for open RPC highport access and a probable compromise. If they arePCs/Macs then you may want to look at the machines anyway to check that ICQ and MonkeyCom are indeed responsible for the traffic and that they haven't been compromised.

I also suggest working on the snort rulebase to reduce the number of these false alarms. The newer cleaned-up snort rulebase is worth a look.

MY.NET.105.115	MY.NET.213.158	MY.NET.220.118	MY.NET.222.218

MY.NET.223.106	MY.NET.224.138	MY.NET.224.62	MY.NET.225.234
MY.NET.226.242	MY.NET.97.163	MY.NET.97.208	MY.NET.97.213
MY.NET.97.245	MY.NET.97.45	MY.NET.97.53	MY.NET.97.74
MY.NET.97.96	MY.NET.98.192	MY.NET.98.226	MY.NET.98.238

Napster Traffic

Host MY.NET.71.38 has generated a lot of traffic on port 7777 which is assiciated with Napster. If this is within your policy guidelines then no problem, otherwise you may wish to investigate it.

SNMP traffic with poorly chosen community strings being received

Three internal systems (MY.NET.100.143, MY.NET.100.206, MY.NET.100.99) are receiving SNMP traffic from ece156-dhcp-2.ecn.purdue.edu that uses a poorly chosen community string.
MY.NET.154.26 is receiving SNMP traffic from cesdis6.gsfc.nasa.gov that uses a poorly chosen community string.
It would be a good idea to contact the users of these systems to advise them to change their community strings.

SNMP traffic with poorly chosen community strings (internal)

The following internal SNMP traffic flows are using poorly chose community strings:
MY.NET.111.156, MY.NET.162.201 and MY.NET.70.42 to MY.NET.50.154
MY.NET.206.222 to MY.NET.14.1

The following hosts to MY.NET.101.192:

MY.NET.97.107	MY.NET.97.133	MY.NET.97.155	MY.NET.97.163
MY.NET.97.168	MY.NET.97.186	MY.NET.97.200	MY.NET.97.206
MY.NET.97.244	MY.NET.97.249	MY.NET.97.52	MY.NET.98.134
MY.NET.98.169	MY.NET.98.183		

I suggest you contact the administrators of these systems and ask then to change their community strings.

SMB Name Wildcard (internal)

MY.NET.101.160 has been regularly sending SMB Name Wildcard queries to MY.NET.101.192

and MY.NET.111.156 has been sending them to MY.NET.50.239 and MY.NET.125.41.
This could indicate some kind of misconfiguration.

Print services traffic from outside

The host 216-119-15-88.o1.jps.net. made 1273 connection attempts to the hosts listed below over 26 minutes on 20 December 2000. This seems a bit excessive.

MY.NET.99.104	MY.NET.100.209	MY.NET.130.86	MY.NET.214.166

Russian Dynamo

MY.NET.205.138 exchanged a significant amount of traffic with dynamic.dol.ru on 8 December, causing alerts about "Russia Dynamo - SANS Flash 28-jul-00" but despite the explicit reference in the rule I cannot find any reference to "Russia Dynamo" on www.sans.org, xforce.iss.net or www.securityfocus.org. I suggest that MY.NET.205.138 be checked.

wu-ftpd targetting

Three hosts were singled out during this period for attempts to exploit wu-ftpd, versions of which have known vulnerabilities (GIAC000623).
This probably indicates that these attackers have done their reconnaisance and know which of your servers run wu-ftpd.

MY.NET.130.98	MY.NET.156.127	MY.NET.97.162

Machines Acting as Servers

Web Servers

The following hosts may be providing HTTP services. It is advisable to check whether web servers are running on these hosts and if they are, whether they are authorised and reasonably secure (up-to-date web server software and patches).

MY.NET.100.165	MY.NET.140.163	MY.NET.140.220	MY.NET.140.57
MY.NET.145.18	MY.NET.145.9	MY.NET.179.77	MY.NET.181.144
MY.NET.253.114	MY.NET.253.125	MY.NET.6.7	MY.NET.60.14
MY.NET.99.85			

SSL servers

The following hosts appear to be serving SSL sites:

MY.NET.253.112	MY.NET.5.29	MY.NET.60.14

Mail Servers

One host appears to be providing non-SMTP mail services for external clients: MY.NET.6.7 (IMAP). You are advised to check that this is both authorised and patched.

DNS Servers

The following hosts appear to be acting as DNS servers for external clients. Some versions of DNS have known security vulnerabilites so you are advised to check this list to ensure that they are all authorised and up-to-date with patches. Three of these servers (MY.NET.1.3, MY.NET.1.4, MY.NET.1.5) were subjected to a UDP denial of service attack on 6 December.

MY.NET.1.10
MY.NET.1.3
MY.NET.1.4
MY.NET.1.5
MY.NET.1.8
MY.NET.1.9

Fingerprinting

The following host have probably been fingerprinted by either Queso or NMAP and therefore could be the subject of more focussed attacks.

MY.NET.1.6	MY.NET.100.230	MY.NET.105.120	MY.NET.145.189
MY.NET.145.9	MY.NET.179.78	MY.NET.180.26	MY.NET.201.126
MY.NET.201.130	MY.NET.201.222	MY.NET.201.42	MY.NET.201.62
MY.NET.201.66	MY.NET.201.74	MY.NET.201.78	MY.NET.202.170
MY.NET.202.26	MY.NET.202.46	MY.NET.203.66	MY.NET.204.26

MY.NET.204.38	MY.NET.205.214	MY.NET.206.122	MY.NET.207.230
MY.NET.208.114	MY.NET.209.78	MY.NET.210.6	MY.NET.211.202
MY.NET.212.214	MY.NET.213.158	MY.NET.214.78	MY.NET.214.82
MY.NET.215.74	MY.NET.217.146	MY.NET.217.158	MY.NET.217.242
MY.NET.217.90	MY.NET.218.202	MY.NET.218.226	MY.NET.219.114
MY.NET.219.150	MY.NET.219.210	MY.NET.220.102	MY.NET.221.138
MY.NET.221.78	MY.NET.222.218	MY.NET.223.2	MY.NET.223.226
MY.NET.224.242	MY.NET.224.78	MY.NET.225.238	MY.NET.225.94
MY.NET.227.194	MY.NET.253.41	MY.NET.253.42	MY.NET.253.43
MY.NET.253.53	MY.NET.53.108	MY.NET.53.184	MY.NET.53.29
MY.NET.60.11	MY.NET.60.16	MY.NET.60.38	MY.NET.60.8
MY.NET.70.119	MY.NET.97.111	MY.NET.97.215	MY.NET.97.234
MY.NET.97.243	MY.NET.97.54	MY.NET.98.147	MY.NET.98.154
MY.NET.98.156	MY.NET.98.185	MY.NET.98.211	MY.NET.98.60
MY.NET.98.98			

Top 5 Scan Destinations

MY.NET.223.86

Big scans on 5 Dec 2000 from 24.29.40.11 (18744 packets) and 24.4.196.167 (29528 packets).

MY.NET.201.78

A big scan on 6 Dec 2000 from 24.180.134.156 (26370 packets)

MY.NET.202.94

24 Dec: This host saw a lot of UDP port 9000 traffic (possibly CIPE VPN?) to hosts in 207.46.204.0
30 Dec: TCP port scanned by 216.99.200.242
31 Dec: TCP scanning from this host of a large number of hosts on particular port numbers (1 per host), as well as being UDP port scanned by 216.99.200.242
1 and 2 Jan saw the same TCP scanning activity as on 31 Dec followed by port 9000 UDP traffic to the same network as on 24 Dec interspersed with small bursts of TCP traffic.

What was causing these traffic patterns? A benign explanation could be a remote user doing some work from home over the holiday period, but it looks suspicious to me.

MY.NET.98.182

TCP port scanned by 66.20.207.21 on 28 Dec 2000 (9262 packets).

MY.NET.203.98

TCP port scanned by 24.180.134.156 on 6 Dec 2000 (7134 packets).

Watchlist Activity

Watchlist 000222 - Chinese Academy of Sciences Network

The following types of traffic have been detected from hosts on this network. Please check that this activity fits the roles assigned to these servers and that these clients are permitted this access.

SMTP

Hosts on this network are exchanging mail (SMTP) with the following hosts inside your network.

MY.NET.100.230	MY.NET.110.150	MY.NET.145.76	MY.NET.145.9
MY.NET.253.114	MY.NET.253.41	MY.NET.253.42	MY.NET.253.43
MY.NET.253.51	MY.NET.253.52	MY.NET.253.53	MY.NET.6.34
MY.NET.6.35	MY.NET.6.47	MY.NET.75.3	MY.NET.6.7

IMAP

One particular host on this network (159.226.121.37) appears to be using your host MY.NET.6.7 as an IMAP mail server.

SSL

Hosts on this network are using MY.NET.5.29 as an SSL web server.

FTP

Host 159.226.47.14 has twice attempted to use FTP on MY.NET.145.18 (on Dec 4 and Dec 21)

but activity appeared to be limited to the control port (21) with no data transferred.

Watchlist 000220 - ISDN Net (Israeli ISP)

There were about 105000 alerts due to this watchlist, about 25% apparently napster on 6699. I suggest revisiting this watchlist to see if it can be made more specific in some way.

Analysis Method

I used perl, grep, sort, uniq and some awk to summarise the data and then mined the summary information for pointers on where to look in the detail.

I find it easier to work with log files that are named for the date that the data were collected so I wrote scripts to rename the log files. This one is for the alerts log.

```
::::::::::::::
date-logs.pl
::::::::::::::
#! /usr/local/bin/perl

use Time::Local;

my $mnum = 1;
map { $months{$_} = $mnum++; } qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov
Dec);

foreach $file (@ARGV) {
        open ( LOG, $file) || die "cannot open $file: $!";
        while (<LOG>) {
                chomp;
                # grab the report date from the subject line
                if (/^Subject: Snort \S+ Report at \S+ (\S+)\s+(\d+) \S+
(\d+)/) {
                        my ($mon, $day, $year) = ($1, $2, $3);
                        $nmon = $months{$mon};
                        # make it yesterdays date (the report closes just
after midnight)
                        ($x, $x, $x, $day, $nmon, $year, $x) =
                                localtime (
                                    timelocal (0,0,3,$day, $nmon-1, $year) -
60*60*24);
                        $nmon++;  # adjust from Jan=0 to Jan=1
                        $year += 1900;
                        if ($day =~ /^\d$/) {  # one digit
                                $day = "0$day";
                        }
```

```
                        if ($nmon =~ /^\d$/) {  # one digit
                                $nmon = "0$nmon";
                        }
                        $newname = "$year-$nmon-$day";
                }
        }
        close LOG;
        # output shell commands
        print "mv $file $newname\n";
}
```

I wanted a summary that put a number of identical alerts for a host into 1 line with a packet count. The script below takes a 1 line per alert file and produces output like that below. This script is somewhat customised for the supplied data.

Format: date, targetted-internal-machine, rule text, packet count, source hosts, destination ports
The source hosts and destination ports fields are reduced to counts if there are more than 2 hosts or ports.

```
2001-01-13,MY.NET.202.46,Queso fingerprint,20 pkts,4 hosts,6346
2000-12-16,MY.NET.202.64,connect to 515 from outside,1
pkts,209.217.166.69,515
2000-12-16,MY.NET.202.66,connect to 515 from outside,1
pkts,209.217.166.69,515
2000-12-16,MY.NET.202.69,connect to 515 from outside,1
pkts,209.217.166.69,515
2000-12-16,MY.NET.202.86,connect to 515 from outside,1
pkts,209.217.166.69,515
2000-12-30,MY.NET.202.94,Attempted Sun RPC high port access,1
pkts,securedesign.net,32771
2000-12-30,MY.NET.202.94,Back Orifice,1 pkts,securedesign.net,31337
2000-12-30,MY.NET.202.94,SUNRPC highport access!,6
pkts,securedesign.net,32771
2000-12-31,MY.NET.202.94,Attempted Sun RPC high port access,3
pkts,securedesign.net,32771
2000-12-31,MY.NET.202.94,Back Orifice,2 pkts,securedesign.net,31337
2000-12-31,MY.NET.202.94,Watchlist 000220 IL-ISDNNET-990517-clnt-
8164.bezeqint.net,15 pkts,clnt-8164.bezeqint.net,8 ports

::::::::::::::
sum-alerts.pl
::::::::::::::
#! /usr/local/bin/perl -w

use strict;

# Summarise an alert file
my %alert_list;
```

```
my @unknown;
my $portscan;

if (scalar (@ARGV) > 1) {
        die "just 1 alert file please";
}

open (ALERTS, $ARGV[0]) || die "cannot open $ARGV[0]: $!";
while (<ALERTS>) {
        if (! /^\d/) {
                next;
        }
        if (/SYN-FIN scan/) { # filter out the noisy scans
                next;
        }

        chop;
        # remove carriage returns
        if (/\r$/) {
                chop;
        }
        if (/\[\*\*\] (.*) \[\*\*\] (.*) -> (.*)$/) {
                # pick out the to/from part of the alert
                my ($alert, $src_h, $dst_h) = ($1, $2, $3);
                my ($src_p, $dst_p, $alert_h, $other_h);
                my ($alert_rec);

                # extract the port numbers if there are any
                if ($src_h =~ /:/) {
                        ($src_h, $src_p) = split (/:/, $src_h);
                }

                if ($dst_h =~ /:/) {
                        ($dst_h, $dst_p) = split (/:/, $dst_h);
                }

                # make traffic from MY.NET stand out more
                if ($src_h =~ /MY\.NET/) {
                        $alert_h = $src_h;
                        $alert = "0 SENT-$alert";
                        $other_h = $dst_h;
                } else {
                        $alert_h = $dst_h;
                        $other_h = $src_h;
                }

                # consider each different watchlist host as a separate alert
```

```
                if ($alert =~ /Watchlist/) {
                        $alert = "$alert-$other_h";
                }

                if (!$alert_list{$alert_h}->{$alert}) {
                        $alert_list{$alert_h}->{$alert} = {};
                }
                $alert_rec = $alert_list{$alert_h}->{$alert};

                if ($dst_p) {
                        $alert_rec->{"ports"}->{$dst_p}++;
                }

                $alert_rec->{"hosts"}->{$other_h}++;

                $alert_rec->{"num"}++;

        } elsif ( /spp_portscan:/) {
                # don't bother with port scans in this summary
                $portscan++;
        } else {
                push (@unknown, $_);
        }
}
close ALERTS;

if (@unknown) {
        print "!!!!!!!!!! Unknown lines:\n", join ("\n", @unknown), "\n";
}

# print out the results
my ($host, $alert);
foreach $host (sort byip keys %alert_list) {
        foreach $alert (sort keys (%{$alert_list{$host}})) {
                my ($alert_rec) = $alert_list{$host}->{$alert};
                print "$ARGV[0],$host,$alert,$alert_rec->{\"num\"} pkts,";

                my $num_hosts = scalar (keys (%{$alert_rec->{"hosts"}}));
                if ($num_hosts < 3) {
                        print join (":", keys (%{$alert_rec->{"hosts"}})),
","";
                } else {
                        print "$num_hosts hosts,";
                }

                my $num_ports = scalar (keys (%{$alert_rec->{"ports"}}));
                if ($num_ports == 0) {
```

```
                        print "\n";
                } elsif ($num_ports < 3) {
                        print join (":", keys (%{$alert_rec->{"ports"}})),
"\n";
                } else {
                        print "$num_ports ports\n";
                }
        }
        print "\n"; # separate hosts
}


# sort IP addresses nicely
sub byip {
        my ($a1, $a2, $a3, $a4) = split (/\./, $a);
        my ($b1, $b2, $b3, $b4) = split (/\./, $b);

        if ($a1 ne $b1) { return ($a1 <=> $b1); }
        if ($a2 ne $b2) { return ($a2 <=> $b2); }
        if ($a3 ne $b3) { return ($a3 <=> $b3); }
        if ($a4 ne $b4) { return ($a4 <=> $b4); }

        return ($a <=> $b);
}
```

I wanted to print the summary, so I wrote this to organise it more sensibly on the page:

```
#!/usr/local/bin/perl

# print out the summary with line breaks between hosts, keeping
# hosts on 1 page if poss.

my ($line, $host, $prevhost);

$pagelen = 60;
$prevhost = "";
$lines_left = 60;

while (<>) {
        $line = $_;
        $host = (split (/,/, $line))[1];
        if ($prevhost eq "") {
                $prevhost = $host;
        }
        if ($host eq $prevhost) {
                $buffer .= $line;
                $bufflines++;
```

```
        } else {
                if ($bufflines <= $lines_left) {
                        print $buffer;
                        $lines_left -= $bufflines;
                } else {
                        print "\014";  # new page
                        print $buffer;
                        $lines_left = $pagelen - ($bufflines % $pagelen);
                }
                # put a line after each host
                if ($lines_left != $pagelen) {
                        print "\n";
                        $lines_left--;
                }
                $buffer = $line;
                $bufflines = 1;
                $prevhost = $host;
        }
}
```

After generating the summary I used grep and perl to extract useful information. For example: the IP addresses of all the machines that have been fingerprinted.

```
perl -anF, -e 'if ($F[2] =~ /fingerprint/i) { print "$F[2]\n";}' <
report.sorted  | sort | uniq| more
```

References

Bayerkohler, Marc. SANS Intrusion Detection Practical.
http://www.sans.org/y2k/practical/Marc_Bayerkohler_GCIA.html