Assignment 1 – Network Detects

The following detects were encountered while reviewing the 11/11-12 weekend SHADOW wrap-up and SnortSnarf reports. The IDS is still in development, having been in operation now for about a month. It runs on a pair of BSDI 4.1 systems with SHADOW 1.6 and Snort 1.6.3-patch2. Currently, the SHADOW script *fetchem.pl* processes the tcpdump data files through Snort using the 10102k.rule set. The resulting snort output is then processed by SnortSnarf v102700.1 to create a web-based summary of Snort alert and portscan logs.

Detect #1

The 1am SHADOW wrap-up report on 11/11/00 listed the following host because it had tried to contact 9 different internal hosts:

```
num dests  source ip      source name

9          195.70.47.74   something.very.very.screwed.up.at.macroda.hu
```

The following is a subset (26 packets omitted) of all packets this host had sent or received during a 24 hour time period:

```
01:16:35.223382 195.70.47.74.33786 > OUR.NET.3.253.2195: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 11794)
01:17:29.618985 195.70.47.74.36488 > OUR.NET.22.26.1870: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 19182)
01:20:13.422772 195.70.47.74.44845 > OUR.NET.183.19.1465: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 40663)
01:21:20.153278 195.70.47.74.48045 > OUR.NET.252.145.1465: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 49756)
01:30:21.196537 195.70.47.74.9295 > OUR.NET.115.197.1673: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 28468)
01:30:23.827116 195.70.47.74.9295 > OUR.NET.115.197.1673: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 29081)
01:42:06.136831 195.70.47.74.44581 > OUR.NET.14.50.1673: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 57974)
01:42:07.837302 195.70.47.74.44581 > OUR.NET.14.50.1673: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 58221)
01:48:27.771469 195.70.47.74.63856 > OUR.NET.229.136.2499: R 0:0(0) ack 674719802 win 0 [tos 0x60] (ttl 242, id 17916)
```

```
01:48:29.987828 195.70.47.74.63856 > OUR.NET.229.136.2499: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 18233)
01:48:32.181124 195.70.47.74.63856 > OUR.NET.229.136.2499: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 18551)
01:56:38.369848 195.70.47.74.22948 > OUR.NET.83.98.1153: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 16742)
01:56:39.625948 195.70.47.74.22948 > OUR.NET.83.98.1153: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 16938)
01:56:57.223610 195.70.47.74.23614 > OUR.NET.196.148.1178: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 19257)
01:57:00.048868 195.70.47.74.23614 > OUR.NET.196.148.1178: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 19623)
02:04:58.824242 195.70.47.74.48013 > OUR.NET.136.45.2239: R 0:0(0) ack 674719802 win 0 [tos 0x60] (ttl 242, id 58239)
02:04:58.826085 195.70.47.74.48013 > OUR.NET.136.45.2239: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 58240)
02:07:26.656123 195.70.47.74.55346 > OUR.NET.41.156.1178: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 12078)
02:18:04.726792 195.70.47.74.21384 > OUR.NET.16.210.1536: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 5233)
02:18:06.018814 195.70.47.74.21384 > OUR.NET.16.210.1536: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 5449)
02:18:07.307237 195.70.47.74.21384 > OUR.NET.16.210.1536: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 5670)
02:18:25.379733 195.70.47.74.22822 > OUR.NET.65.141.1913: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 8813)
02:19:02.049542 195.70.47.74.24528 > OUR.NET.44.203.1536: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 13991)
02:21:39.230598 195.70.47.74.32539 > OUR.NET.89.0.1913: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 34414)
02:34:15.208473 195.70.47.74.4700 > OUR.NET.122.100.1626: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 43606)
02:35:24.390129 195.70.47.74.8022 > OUR.NET.2.8.1271: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 52989)
02:36:53.206268 195.70.47.74.12091 > OUR.NET.209.241.1626: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 64777)
02:36:54.732905 195.70.47.74.12091 > OUR.NET.209.241.1626: R 0:0(0) ack 674719802 win 0 [tos 0x60] (ttl 242, id 64978)
02:40:16.617439 195.70.47.74.22839 > OUR.NET.176.131.1271: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 26279)
02:40:18.299274 195.70.47.74.22839 > OUR.NET.176.131.1271: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 26508)
02:40:20.030834 195.70.47.74.22839 > OUR.NET.176.131.1271: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 26742)
02:40:21.749165 195.70.47.74.22839 > OUR.NET.176.131.1271: R 0:0(0) ack 674719802 win 0 [tos 0x40] (ttl 242, id 26967)
```

1. Source of trace:

Company DMZ network.

2. Detect was generated by:

The detect that appears in the SHADOW hourly wrap-up was created by the SHADOW *find_scan.pl* script. This script is called by the *fetchem.pl* script to examine the captured data for any external hosts that have exceeded a threshold for the number of different internal hosts that are accessed. The file *filters/Site1/filter.getall* contains a tcpdump filter that excludes web traffic and defines which internal network(s) to ignore. The threshold value used by *find_scan.pl* is defined in the file *sites/Site.ph*.

(N.B.: All referenced file pathnames are relative to the base directory that the SHADOW system was installed in.)

The trace output was produced using the SHADOW script *one_day_pat.pl* as follows:

```
./one_day_pat.pl -n -d 20001111 -l Site1 -p 'host 195.70.47.74'
```

This script processes the day's captured data through tcpdump using the supplied pattern. Any matching packets will be output in a format dependant on the type of packet that tcpdump has decoded. For this trace all packets are TCP and follow the general format:

timestamp srcaddr.srcport > dstaddr.dstport: flags data-seqno ack window urgent options

The first line of the trace is repeated here and then disassembled with additional comments for each component of the decoded tcpdump packet.

```
01:16:35.223382 195.70.47.74.33786 > OUR.NET.3.253.2195: R 0:0(0) ack 674719802 win 0 [tos 0x40]
```

timestamp	`01:16:35.223382`	
srcaddr	`195.70.47.74`	The source IP address in dot-quad notation.
srcport	`33786`	The source port number.
dstaddr	`OUR.NET.3.253`	The destination IP address in dot-quad notation (partially obfuscated).
dstport	`2195`	The destination port number.
flags	`R`	Possible flags are **S**YN, **F**IN, **P**SH or **R**ST
data-seqno	`0:0(0)`	*first-seqno:last-seqno (number of data bytes)*
ack	`ack 674719802`	The sequence number, incremented by one, of the last successfully received data byte.
window	`win 0`	
urgent		N/A (not present in the trace above).
options	`[tos 0x40] (ttl 242, id 11794)`	TCP Type Of Service, TTL and packet ID.

3. Probability the source address was spoofed:

Low. This detect is probably the third-party effect of a Denial of Service attack on the source host.

4. Description of attack:

The source host is sending RST packets, each with the same ack value, 674719802. Although the destination hosts in the trace are not in use, it is the pattern used by the script *one_day_pat.pl* that verifies the packets were sent without prior stimulus. This is accomplished by matching the address 195.70.47.74 as either the source or the destination host.

While the reception of RST packets with no prior stimulus could indicate a stealth scan attack, the specific ack 674719802 has been attributed to the Denial of Service tool synk4.c, which is a SYN flood tool and is classified by CVE-1999-0116. So, the source host here is actually under attack by a third-party that is spoofing the destination hosts listed in the trace above.

A SYN flood attacks the three-way handshake that is used to establish a TCP connection. A normal three-way handshake occurs as follows:

```
Client                  Server
------                  ------
 SYN--------------------->
   <---------------------SYN-ACK
 ACK--------------------->
```

A SYN flood attack works by sending the initial SYN with a *spoofed* source address. This directs the victim host to reply to the wrong host with the SYN-ACK. The victim host will wait some specified period of time before canceling the pending connection. If enough SYN packets are received and the victim host has a TCP/IP stack with limited queue depth it may be unable to accept genuine connection requests.

A more detailed treatment of SYN flood attacks can be found at the following locations:

http://www.cert.org/advisories/CA-1996-21.html

http://www.niksula.cs.hut.fi/~dforsber/synflood/result.html.

5. Attack mechanism:

In http://lists.insecure.org/incidents/2000/Apr/0026.html Dave Dittrich provides details on how the synk4.c tool operates. It sends crafted SYN packets with an initial sequence number of 674719801 to the host under attack. It is also capable of spoofing the source address with randomly generated numbers.

In examining the source code for synk4.c obtained from http://packetstorm.securify.com/Exploit_Code_Archive/synk4.c, there are additional details on how it operates. First, the attacker sets the range of destination ports via low and high parameters on the command line. Second, the tool randomly generates the source port within a range of 1001-2500. Finally, the tool executes a nested loop pair during operation. The outer loop executes a total of 32,767 times with a new source port randomly generated for each iteration. The inner loop iterates through the destination port range, generating a different spoofed source address each time.

From these we can draw a set of criteria to fit an observed trace of network traffic to this tool, they are:

- A given source host sends RST packets with an ack of 674719802.
- The packets should be received with no prior stimulus assuming the local network is not the source of the attack.
- When sorted by timestamp the source port will increment between indeterminate low and high values one or more times.
- The destination host may be a single IP address but will most likely be random.
- The destination port will be random values in the range of 1001-2500.

However, since the source code for this tool is generally available it can be altered to render one or more of these criteria invalid. In fact, the trace above has some anomalies that deviate from the criteria developed for this tool. First, note that some of the RST packets appear to repeat on the order of 2, 3 or 4 times. While it is possible that the same IP address could be generated randomly it's doubtful considering that 4 separate calls to the random function are made to create the address. A more plausible answer for this could be the simple addition of a third innermost loop in the execution logic that would repeat sending the same packet by a random value between 1 and 4.

There is another aspect as well, the TOS option. The source for synk4.c indicates that this is set to 0 for every packet. Yet in the trace above the TOS is most often set to 0x40. Oddly, some of the repeated packets show a TOS value of 0x60. Both values relate to precedence options that should not be used.

This anomaly coupled with the odd hostname warranted sending the contacts for this source host a short message to help research this detect further. No answer has been received to date. For now, these TOS values remain an unanswered puzzle.

6. Correlations:

Figure 7 in the following document details a trace that is attributed to a reset scan. However, that trace fits the criteria in section 5 above perfectly and may be in fact the third-party affect of DoS attack using synk4.c.

http://www.ds-osac.org/whatsnew/cyberlib.cfm?KEY=3566

The next set of links all show traces with variations to the criteria in section 5. Again, trivial modifications to the source code would explain these variations.

http://secinf.net/info/ids/intv2-8.htm
http://www.sans.org/y2k/081000.htm
http://www.sans.org/y2k/063000.htm

7. Evidence of active targeting:

Not directly. An attacker has actively targeted the source host and we are seeing the third-party affect of this.

8. Severity:

Using the formula *Severity = (Criticality + Lethality) – (System + Net countermeasures)* the severity of this detect is:

Criticality:	0; The destination addresses are not in use.
Lethality:	0; These are naturally occurring RST packets.
System:	0; No system involved.
Net:	5; Firewalls and border routers do not return ICMP non-existent host errors.

(0 + 0) – (0 + 5) = -5

9. Defensive recommendation:

The steps detailed in http://www.sans.org/y2k/egress.htm provide a solid basis for minimizing a site's exposure to attacks that use spoofed addresses.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer.

Which of the following values does the synk4.c DoS tool use for its initial sequence number?

A. 0x674719802
B. 0x674711610
C. 0x28376839
D. 0x952F5C2

Answer: C.

Detect #2

The following Snort alerts were found during a review of the hourly IDS reports.

```
[**] IIS-asp-dot [**]
11/11-02:15:36.727950 63.211.232.60:2519-> OUR.NET.136.21:80
TCP TTL:116 TOS:0x0 ID:47730 DF
***AP*** Seq: 0x7B58C37 Ack: 0x95310B0 Win: 0x1F39

[**] IIS-asp-dot [**]
11/11-02:16:01.159934 63.211.232.60:2520-> OUR.NET.136.21:80
TCP TTL:116 TOS:0x0 ID:48754 DF
***AP*** Seq: 0x7B58C1E Ack: 0x952F5C2 Win: 0x2238
```

The relevant packets associated with each alert were extracted and decoded by Snort as follows:

```
11/11-02:15:36.727950 63.211.232.60:2519 -> OUR.NET.136.21:80
TCP TTL:116 TOS:0x0 ID:47730  DF
***AP*** Seq: 0x7B58C37   Ack: 0x95310B0   Win: 0x1F39
47 45 54 20 2F 64 6E 62 69 77 73 68 6F 6D 65 2E  GET /dnbiwshome.
61 73 70 2E 20 48 00 00 00 00 00 00 00 00 00 00  asp. H..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

11/11-02:15:36.731452 OUR.NET.136.21:80 -> 63.211.232.60:2519
TCP TTL:125 TOS:0x0 ID:29927  DF
***AP*** Seq: 0x95310B0   Ack: 0x7B58DA6   Win: 0x1D14
48 54 54 50 2F 31 2E 31 20 34 30 34 20 4F 62 6A  HTTP/1.1 404 Obj
65 63 74 20 4E 6F 00 00 00 00 00 00 00 00 00 00  ect No..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

11/11-02:16:01.159934 63.211.232.60:2520 -> OUR.NET.136.21:80
TCP TTL:116 TOS:0x0 ID:48754  DF
***AP*** Seq: 0x7B58C1E   Ack: 0x952F5C2   Win: 0x2238
47 45 54 20 2F 64 6E 62 69 77 73 68 6F 6D 65 2E  GET /dnbiwshome.
```

```
61 73 70 25 32 65 00 00 00 00 00 00 00 00 00 00  asp%2e..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

11/11-02:16:01.162718 OUR.NET.136.21:80 -> 63.211.232.60:2520
TCP TTL:125 TOS:0x0 ID:56551  DF
***AP*** Seq: 0x952F5C2   Ack: 0x7B58D8F   Win: 0x1E4D
48 54 54 50 2F 31 2E 31 20 34 30 34 20 4F 62 6A  HTTP/1.1 404 Obj
65 63 74 20 4E 6F 00 00 00 00 00 00 00 00 00 00  ect No..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

1. Source of trace:

Company DMZ network.

2. Detect was generated by:

The Snort alerts were detected using the 10102k.rule set available from http://www.snort.org.

A Snort alert follows the general format of:

[**] alert-message [**]
timestamp srcaddr:srcport -> dstaddr:dstport
protocol options
flags seqno ackno window

The first alert entry is repeated here and then disassembled with additional comments for each component.

```
[**] IIS-asp-dot [**]
11/11-02:15:36.727950 63.211.232.60:2519-> OUR.NET.136.21:80
TCP TTL:116 TOS:0x0 ID:47730 DF
***AP*** Seq: 0x7B58C37 Ack: 0x95310B0 Win: 0x1F39
```

alert-message	`IIS-asp-dot`	The message defined in the Snort rule that triggered this alert.
timestamp	`11/11-02:15:36.727950`	
srcaddr	`63.211.232.60`	The source IP address in dot-quad notation.
srcport	`2519`	The source port number.
dstaddr	`OUR.NET.136.21`	The destination IP address in dot-quad notation (partially obfuscated).
dstport	`80`	The destination port number.
protocol	`TCP`	
options	`TTL:116 TOS:0x0 ID:47730 DF`	Time to Live, Type of Service, packet ID, Do Not Fragment flag.
flags	`***AP***`	Possible flags are 2, 1, S, F, R, P, A, U (N.B.: ordering changes for Snort versions > 1.6.)
seqno	`Seq: 0x7B58C37`	
ackno	`Ack: 0x95310B0`	The sequence number, incremented by one, of the last successfully received data byte.
window	`Win: 0x1F39`	

The command used to extract the decoded packets is:

```
./gunzip -c tcp.2000111102.gz | snort -d -N -p -q -v -r - 'host 63.211.232.60'
```

The protocol header in the decoded packets follows the same general format of a Snort alert minus the *alert-message*. The application layer data in the packet follows the packet header. The format for that data is displayed as hexadecimal and ASCII respectively provided the appropriate command line option is given.

3. Probability the source address was spoofed:

Zero. An established TCP connection is present and data was exchanged between the source and destination hosts.

4. Description of attack:

These are actually two related attacks. The first, which was assigned CAN-1999-0154, refers to an attacker attempting to download the source code of an IIS ASP script by appending a *".”* to the URL. The highlighted section first decoded packet shows the attacker attempting to do just this via the HTTP command GET /dnbiwshome.asp.

The second attack was assigned CAN-1999-0253, which refers to an alternate method of retrieving ASP source by appending *“%2e”* to the ASP URL. This attack is present above in the highlighted section of the third decoded packet which shows the attacker using the HTTP command GET /dnbiwshome.asp%2e

A deeper scan of the entire weekend for this source address was run but, no further activity was found.

(N.B.: CAN-1999-0154 was not found at http://cve.mitre.org but it is referenced in the comments for CAN-1999-0253.)

5. Attack mechanism:

The first attack depends on a bug in IIS 3.0 that will return the source code of an ASP script if the virtual directory has both read and execute access present and a *".”* is appended to the ASP URL.

The second attack also affects IIS 3.0 but only after the iis-fix hotfix has been applied to IIS to fix the original exposure. In this case the attacker can now retrieve the ASP source by appending *“%2e”* to the ASP URL.

6. Correlations:

http://xforce.iss.net/static/336.php
Xforce description of the original exposure.

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0253
CVE candidate details for the %2e variant of the original exposure.

7. Evidence of active targeting:

Yes. The attacker attempted to retrieve the default ASP files for this segment of the company's web space.

8. Severity:

Using the formula *Severity = (Criticality + Lethality) – (System + Net countermeasures)* the severity of this detect is:

Criticality: 4; OUR.NET.136.21 is a company web server.
Lethality: 2; Downloading ASP source code could lead to a secondary compromise.
System: 5; The web server's OS and HTTP service are fully patched and have no known vulnerabilities.
Net: 3; The firewalls are packet-filtering systems, not proxies but, the IDS does examine HTTP content.

(4 + 2) – (5 + 3) = -2

9. Defensive recommendation:

For this attack specifically; no action need be taken as all web servers are running IIS 4.0 with all necessary service packs and hotfixes applied. The source address in question has been added to the SHADOW filters and Snort rules so that any future activity can be tracked.

In general, apply appropriate hotfixes if IIS 3.0 has to remain in production. Otherwise, upgrade to IIS 4.0 and also apply all necessary hotfixes to prevent exposure to other known vulnerabilities.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer.

Which of the following URLs will download ASP source code when performed against an unpatched IIS 3.0 web server?

A. http://anywebhost/foobar.asp
B. http://anywebhost/foobar.asp#2e
C. http://anywebhost/foobar.asp&2e
D. http://anywebhost/foobar.asp%2e

Answer: D.

Detect #3

The following Snort alerts were found during a review of the hourly IDS reports.

```
[**] IDS135 - CVE-1999-0265 - MISC-ICMPRedirectHost [**]
11/12-07:07:47.424161 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46551
REDIRECT

[**] IDS135 - CVE-1999-0265 - MISC-ICMPRedirectHost [**]
11/12-07:07:47.643088 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46563
REDIRECT
```

The relevant packets associated each alert were extracted and decoded by Snort as follows:

```
11/12-07:07:47.424161 212.247.190.1 -> OUR.NET.146.207
```

```
ICMP TTL:242 TOS:0xC0 ID:46551
REDIRECT
D4 F7 BE 14 45 00 00 2C F4 BD 40 00 70 06 50 A9  ....E..,..@.p.P.
XX XX 92 CF D4 F7 BE 14 00 50 52 63 15 86 F0 74  .........PRc...t
37 C4 57 64 60 12 00 00 00 00 00 00 00 00 00 00  7.Wd`...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

11/12-07:07:47.643088 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46563
REDIRECT
D4 F7 BE 14 45 00 00 BC F6 BD 40 00 70 06 4E 19  ....E.....@.p.N.
XX XX 92 CF D4 F7 BE 14 00 50 52 63 15 86 F6 29  .........PRc...)
37 C4 59 2E 50 18 00 00 00 00 00 00 00 00 00 00  7.Y.P...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

1. Source of trace:

Company DMZ network.

2. Detect was generated by:

The Snort alerts were detected using the 10102k.rule set.

Please refer to Detect #2, section 2 for details on the format of network traffic decoded by Snort.

3. Probability the source address was spoofed:

Possible. ICMP host redirect messages are error messages that don't require a response.

4. Description of attack:

ICMP host redirect messages are sent to the source host telling it to send future packets for a given destination host to an alternate gateway. As per CVE-1999-0265, these redirects may crash or lock up a host.

5. Attack mechanism:

In this case the ICMP host redirects appear to be genuine. The contents of an ICMP redirect message are (from RFC792):

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Gateway Internet Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Internet Header + 64 bits of Original Data Datagram      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Since an ICMP redirect message is supposed to be sent from a gateway in response to a packet that could have be sent via a more optimal route we can assume that source host in this case, 212.247.190.1 is an intermediate router with routing knowledge for the destination host. The first four highlighted bytes in the trace above, D4 F7 BE 14, translate to 212.247.190.20. So, the router is telling the source, OUR.NET.146.207 that packets for some destination host should be sent via 212.247.190.20.

In order to find out what that destination host is we need to decode the included IP header. Again, the source and destination values in the IP header are highlighted above, XX XX 92 CF D4 F7 BE 14, which decode to OUR.NET.146.207 for the source and 212.247.190.20 as the destination.

In summary, the router 212.247.190.1 has instructed the source host OUR.NET.146.207 to send traffic destined for 212.247.190.20 to

212.247.190.20. This rather puzzling redirect is better explained in the correlations section that follows.

6. Correlations:

The first step was to find out what traffic has occurred between OUR.NET.146.207 and 212.247.190.20. For that, the following command was run:

```
./one_day_pat.pl -n -d 20001207 -l Site1 -p 'host 212.247.190.20'
```

The following output confirms that 212.247.190.20 has completed a three-way handshake to one of the company's web severs, OUR.NET.146.207.

```
07:07:47.284934 212.247.190.20.21091 > OUR.NET.146.207.80: S 935614307:935614307(0) win 16384  (DF) [tos 0x10]
07:07:47.286911 OUR.NET.146.207.80 > 212.247.190.20.21091: S 361164916:361164916(0) ack 935614308 win 8760  (DF)
07:07:47.446312 212.247.190.20.21091 > OUR.NET.146.207.80: . ack 361164917 win 17520 (DF) [tos 0x10]
...
```

The rest of the trace confirms that this was a routine web session. Also, the timestamps in the three-way handshake confirm that the ICMP redirects were sent in response to the establishment of this connection. However, we need to dig a little deeper to figure out why they were sent by performing a traceroute to both external hosts to shed some additional light.

What traceroute does is map the intermediate gateways that packets take to reach a destination host. Keep in mind that this may not be the same route that the captured traffic originally took since alternate paths can result due to the design of the IP protocol. Especially when several hours or, in this case, days have elapsed between when the traceroute is performed and when the data was captured.

Here are the results from running traceroute to each host. Note that the output has been trimmed to exclude hops related to company routing and ISP connectivity.

```
Tracing route to 212.247.190.20 over a maximum of 30 hops
```

```
...
  8   170 ms   180 ms   180 ms  198.67.142.230
  9   171 ms   180 ms   180 ms  198.67.142.226
 10   170 ms   180 ms   181 ms  198.67.142.132
 11   171 ms   180 ms   180 ms  gbg3-core.pos5-0.swip.net [130.244.193.97]
 12   180 ms   180 ms   181 ms  bck3-core.srp6-0.swip.net [130.244.198.68]
 13   190 ms   191 ms   180 ms  kst1-core.pos5-0.swip.net [130.244.193.85]
 14   171 ms   190 ms   180 ms  kst23.fasteth0-0.swip.net [130.244.198.200]
 15   200 ms   210 ms   191 ms  transit01-gw.swip.net [130.244.41.50]
 16   210 ms   221 ms   240 ms  mail.transit.se [212.247.164.195]
 17     *        *        *     Request timed out.
 18   271 ms   270 ms   310 ms  212.247.190.20

Trace complete.

Tracing route to 212.247.190.1 over a maximum of 30 hops
...
  8   170 ms   181 ms   180 ms  198.67.142.230
  9   161 ms   180 ms   170 ms  198.67.142.226
 10   170 ms   180 ms   181 ms  198.67.142.132
 11   171 ms   180 ms   180 ms  gbg3-core.pos5-0.swip.net [130.244.193.97]
 12   171 ms   180 ms   180 ms  bck3-core.srp6-0.swip.net [130.244.198.68]
 13   191 ms   190 ms   190 ms  kst1-core.pos5-0.swip.net [130.244.193.85]
 14   191 ms   180 ms   190 ms  kst23.fasteth0-0.swip.net [130.244.198.200]
 15   180 ms   201 ms   190 ms  130.244.41.54
 16   190 ms   201 ms   220 ms  212.247.190.1

Trace complete.
```

(N.B.: The NT variant of traceroute was used since it uses ICMP echo messages instead of UDP packets that the UNIX version of traceroute uses. When these traces were first performed the host 212.247.190.20 could not be reached using the UNIX version but was successfully reached with the NT version. This is most likely due to router ACLs and/or firewall filtering rules.)

What these results show is that traffic for each host takes a different path starting at hop 15. Note that while the two hosts at hop 15 are different they are both part of the same 'B' class network, 130.244, which is administered by swip.net which is a Swedish ISP. At hop 16 the traffic reaches the 212.247.190 'C' class network, which is administered by transit.se, a Swedish cable TV company.

(N.B.: Network administration information was found by using the *whois* functionality provided by www.geektools.com.)

At this point it is reasonable to conclude that these redirects were the result of a temporary network problem somewhere in either the swip.net or transit.se networks that resulted in the traffic between OUR.NET.146.207 and 212.247.190.20 taking an alternate route. The host 212.247.190.1 appears to be a border router for a backup connection between transit.se and swip.net. It's likely that 212.247.190.1 has routing knowledge of 212.247.190.20, which could be the cause of the redirect messages.

7. Evidence of active targeting:

None. Analysis indicates that these were genuine ICMP host redirect messages.

8. Severity:

Using the formula *Severity = (Criticality + Lethality) – (System + Net countermeasures)* the severity of this detect is:

Criticality: 4; OUR.NET.146.207 is a company web server.
Lethality: 1; ICMP redirect messages cannot reach the web server.
System: 5; The web server's OS and HTTP service are fully patched and have no known ICMP redirect vulnerabilities.
Net: 4; The web server is behind a firewall that blocks incoming ICMP redirect messages.

(4 + 1) – (5 + 4) = -4

9. Defensive recommendation:

There are two recommendations for ICMP host based redirect messages. First, although RFC1122 states that a host *should* discard a redirect message if the gateway indicated in the redirect message is not on the same local network as the host, this can't be relied on. Effective measures should be taken to block incoming ICMP redirects at either the border router or the firewall for increased protection. Second, hosts that need to receive ICMP redirect messages need to be appropriately patched if the TCP/IP stack has been

shown to be vulnerable as per CVE-1999-0265.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer.

Which statement is not true with respect to ICMP redirect messages?

A. An ICMP redirect message includes a copy of the original IP header plus 8 bytes of data.
B. A host with IP forwarding enabled cannot generate ICMP redirect messages.
C. An ICMP redirect message instructs the source host which gateway to send traffic to for a given host.
D. An ICMP type of 5 and code of 3 is an ICMP redirect message.

Answer: B

Detect #4

The following Snort alert was found during a review of the hourly IDS reports.

```
[**] IDS29 - SCAN-Possible Queso Fingerprint attempt [**]
11/12-08:19:13.672861 64.37.121.145:42043-> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0 DF
12****S* Seq: 0xDCCB2937 Ack: 0x0 Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 74668505 0 NOP WS: 0
```

The relevant packets associated this alert were extracted and decoded by Snort as follows:

```
11/12-08:19:13.672861 64.37.121.145:42043 -> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0  DF
12****S* Seq: 0xDCCB2937   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 74668505 0 NOP WS: 0
0x0000: 08 00 20 B0 50 21 00 50 BD F3 44 00 08 00 45 00  .. .P!.P..D...E.
0x0010: 00 3C 00 00 40 00 36 06 0E A0 40 25 79 91 CC FE  .<..@.6...@%y...
0x0020: AF 67 A4 3B 00 19 DC CB 29 37 00 00 00 00 A0 C2  .g.;....)7......
0x0030: 16 D0 F1 B6 00 00 02 04 05 B4 04 02 08 0A 04 73  ...............s
0x0040: 59 D9 00 00 00 00 01 03 03 00                    Y.........
```

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

11/12-08:19:13.673749 OUR.EXT.NET.103:25 -> 64.37.121.145:42043
TCP TTL:63 TOS:0x0 ID:3219  DF
***A**S* Seq: 0x336A1B24   Ack: 0xDCCB2938   Win: 0x2238
TCP Options => MSS: 1460 NOP WS: 0 NOP NOP TS: 55413283 74668505
0x0000: 00 50 BD F3 44 00 08 00 20 B0 50 21 08 00 45 00  .P..D... .P!..E.
0x0010: 00 3C 0C 93 40 00 3F 06 F9 0C CC FE AF 67 40 25  .<..@.?......g@%
0x0020: 79 91 00 19 A4 3B 33 6A 1B 24 DC CB 29 38 A0 12  y....;3j.$..)8..
0x0030: 22 38 0E 00 00 00 02 04 05 B4 01 03 03 00 01 01  "8..............
0x0040: 08 0A 03 4D 8A 23 04 73 59 D9                    ...M.#.sY.

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

11/12-08:19:13.708864 64.37.121.145:42043 -> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0  DF
***A**** Seq: 0xDCCB2938   Ack: 0x336A1B25   Win: 0x16D0
TCP Options => NOP NOP TS: 74668509 55413283
0x0000: 08 00 20 B0 50 21 00 50 BD F3 44 00 08 00 45 00  .. .P!.P..D...E.
0x0010: 00 34 00 00 40 00 36 06 0E A8 40 25 79 91 CC FE  .4..@.6...@%y...
0x0020: AF 67 A4 3B 00 19 DC CB 29 38 33 6A 1B 25 80 10  .g.;....)83j.%..
0x0030: 16 D0 45 28 00 00 01 01 08 0A 04 73 59 DD 03 4D  ..E(.......sY..M
0x0040: 8A 23                                            .#

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

11/12-08:19:13.791091 OUR.EXT.NET.103:25 -> 64.37.121.145:42043
TCP TTL:63 TOS:0x0 ID:3227  DF
***AP*** Seq: 0x336A1B25   Ack: 0xDCCB2938   Win: 0x2238
TCP Options => NOP NOP TS: 55413283 74668509
0x0000: 00 50 BD F3 44 00 08 00 20 B0 50 21 08 00 45 00  .P..D... .P!..E.
0x0010: 00 89 0C 9B 40 00 3F 06 F8 B7 CC FE AF 67 40 25  ....@.?......g@%
0x0020: 79 91 00 19 A4 3B 33 6A 1B 25 DC CB 29 38 80 18  y....;3j.%..)8..
0x0030: 22 38 EF 0D 00 00 01 01 08 0A 03 4D 8A 23 04 73  "8.........M.#.s
0x0040: 59 DD 32 32 30 20 73 6D 74 70 32 2E              Y.220 smtp2.

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

11/12-08:19:13.826334 64.37.121.145:42043 -> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0  DF
***A**** Seq: 0xDCCB2938   Ack: 0x336A1B7A   Win: 0x16D0
TCP Options => NOP NOP TS: 74668521 55413283
0x0000: 08 00 20 B0 50 21 00 50 BD F3 44 00 08 00 45 00  .. .P!.P..D...E.
0x0010: 00 34 00 00 40 00 36 06 0E A8 40 25 79 91 CC FE  .4..@.6...@%y...
```

```
0x0020: AF 67 A4 3B 00 19 DC CB 29 38 33 6A 1B 7A 80 10  .g.;....)83j.z..
0x0030: 16 D0 44 C7 00 00 01 01 08 0A 04 73 59 E9 03 4D  ..D........sY..M
0x0040: 8A 23                                            .#

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

11/12-08:19:13.826830 64.37.121.145:42043 -> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0  DF
***AP*** Seq: 0xDCCB2938   Ack: 0x336A1B7A   Win: 0x16D0
TCP Options => NOP NOP TS: 74668521 55413283
0x0000: 08 00 20 B0 50 21 00 50 BD F3 44 00 08 00 45 00  .. .P!.P..D...E.
0x0010: 00 53 00 00 40 00 36 06 0E 89 40 25 79 91 CC FE  .S..@.6...@%y...
0x0020: AF 67 A4 3B 00 19 DC CB 29 38 33 6A 1B 7A 80 18  .g.;....)83j.z..
0x0030: 16 D0 A0 44 00 00 01 01 08 0A 04 73 59 E9 03 4D  ...D.......sY..M
0x0040: 8A 23 48 45 4C 4F 20 73 65 72 76 65              .#HELO serve

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

11/12-08:19:13.827617 OUR.EXT.NET.103:25 -> 64.37.121.145:42043
TCP TTL:63 TOS:0x0 ID:3228  DF
***AP*** Seq: 0x336A1B7A   Ack: 0xDCCB2957   Win: 0x2238
TCP Options => NOP NOP TS: 55413283 74668521
0x0000: 00 50 BD F3 44 00 08 00 20 B0 50 21 08 00 45 00  .P..D... .P!..E.
0x0010: 00 83 0C 9C 40 00 3F 06 F8 BC CC FE AF 67 40 25  ....@.?......g@%
0x0020: 79 91 00 19 A4 3B 33 6A 1B 7A DC CB 29 57 80 18  y....;3j.z..)W..
0x0030: 22 38 77 BC 00 00 01 01 08 0A 03 4D 8A 23 04 73  "8w........M.#.s
0x0040: 59 E9 32 35 30 20 73 6D 74 70 32 2E              Y.250 smtp2.

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

11/12-08:19:13.863534 64.37.121.145:42043 -> OUR.EXT.NET.103:25
TCP TTL:54 TOS:0x0 ID:0  DF
***AP*** Seq: 0xDCCB2957   Ack: 0x336A1BC9   Win: 0x16D0
TCP Options => NOP NOP TS: 74668524 55413283
0x0000: 08 00 20 B0 50 21 00 50 BD F3 44 00 08 00 45 00  .. .P!.P..D...E.
0x0010: 00 70 00 00 40 00 36 06 0E 6C 40 25 79 91 CC FE  .p..@.6..l@%y...
0x0020: AF 67 A4 3B 00 19 DC CB 29 57 33 6A 1B C9 80 18  .g.;....)W3j....
0x0030: 16 D0 53 1B 00 00 01 01 08 0A 04 73 59 EC 03 4D  ..S........sY..M
0x0040: 8A 23 4D 41 49 4C 20 46 52 4F 4D 3A              .#MAIL FROM:

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

(N.B.: the data portion of the decoded packets above have been trimmed to obscure the information included in them.)

1. Source of trace:

Company DMZ network.

2. Detect was generated by:

The Snort alert was detected using the 10102k.rule set.

Refer to Detect #2, section 2 for details on the format of Snort decoded packets.

3. Probability the source address was spoofed:

Zero. The packet the alert triggered on was the initial SYN packet of a three-way handshake that was successfully completed.

4. Description of attack:

As highlighted in the trace above, 12****S*, indicates that an initial SYN packet with reserved bits 1 & 2 set was sent to the destination host. This particular flag pattern is associated with the queso fingerprinting tool and has been classified under CAN-1999-0454.

5. Attack mechanism:

Although both reserved bits are set in the initial SYN packet, this turns out to be a false-positive. After examining the rest of the

session and performing the correlations detailed in the next section it became clear that this was, in fact, a valid SMTP session.

6. Correlations:

The message logs on the mail gateway were first examined to confirm that a message had really been received from the source host, one was received and subsequently transferred to the internal recipient.

After reading a paper at http://www.securityfocus.com/frames/?focus=ids&content=/focus/ids/articles/portscan.html on an analysis of nmap and queso, a second examination of all traffic coming from the source host was performed to verify that none of the other crafted packets that queso sends were present.

There was also one reference on SYN packets with the reserved bits set at http://kernelnotes.org/kpatch20.html. The documentation for the patch states that after this patch has been applied these bits are cleared before responding to a SYN packet. However, no mention was made regarding how the system behaved prior to the patch being applied. If the bits are left set when the host subsequently generates a SYN packet for an outgoing connection this detect would be fully explained providing the source host associated with this trace is running Linux with an unpatched kernel.

7. Evidence of active targeting:

None. This was a routine SMTP message transfer.

8. Severity:

Using the formula *Severity = (Criticality + Lethality) – (System + Net countermeasures)* the severity of this detect is:

Criticality: 4; OUR.EXT.NET.103 is a company e-mail gateway.
Lethality: 0; none, routine message transfer.

System: 5; the gateway's OS and SMTP daemon are fully patched and up-to-date.
Net: 4; the gateway is behind a firewall that blocks non-SMTP associated traffic.

(4 + 0) – (5 + 4) = -5

9. Defensive recommendation:

If there is a suitable firewall in front of the hosts it may be able to prevent packets with "impossible" TCP flag combinations from reaching the host. An example of this for Firewall-1 is detailed at http://www.phoneboy.com/fw1/faq/0198.html. For hosts located directly on a DMZ network there are limited defenses depending on the OS in use to prevent an attacker from being able to remotely fingerprint the host. For UNIX oriented systems there is an article on this subject located at http://www.pgci.ca/fingerprint.html that provides information to reduce this type of exposure. However, please note that there are many other behaviors besides these crafted packets that an attacker can examine to determine the OS in use.

Another paper, http://www.securitywizards.com/papers/probes.html, helps with the identification and handling of network probes in order to respond to this sort of activity more effectively.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer.

What technique does the queso OS fingerprinting tool use to determine the OS running on the target host?

A. Telnet banner.
B. FTP banner.
C. 7 impossible TCP flag combinations.
D. None of the above.

Answer: D.

Assignment 2 - Evaluate an Attack

The tool evaluated in this section was compiled from source code obtained from http://packetstorm.securify.com/9906-exploits/pizzathief32.c.

Attack Methodology

The pizzathief32 tool is an implementation of an FTP PASV exploit. This exploit has been assigned CVE-1999-0351 and has been classified as a denial of service and/or method for an attacker to gain unauthorized data access. This exploit is popularly known as the *"Pizza Thief"* because of the analogy that Jeffrey R. Gerber used to describe its operation in an Infowar.com advisory located at http://www.infowar.com/iwftp/iw_sec/iw_sec.shtml.

The File Transfer Protocol (RFC959) is unusual compared to most protocols because it uses two TCP connections to transfer files. The client initiates the *control connection* by connecting to an FTP server's well-known port (21/tcp). The client uses the control connection to issue various FTP commands (these are protocol level commands, not application level, as are commonly used by various FTP client applications) to process the FTP session. The *data connection* is a transient connection that is created for each file transferred (and for other reasons that are not relevant to this discussion).

Of the several methods for initiating the data connection, two are most commonly used; active and passive. The server establishes an active connection after the client issues a PORT command. Two pieces of information are sent to the server as part of the PORT command, the client's IP address and the port number that the client is listening on. The server then initiates the active connection by connecting to the client at the assigned IP address and port number. A passive connection works in an opposite manner. The client requests an IP address and port number from the server by issuing a PASV command. The server responds with its IP address and port number that it will listen on. The client receives the server's response then initiates the connection in order to transfer the file.

Passive connections are the objective of this exploit. If an attacker can connect to the port ahead of the client then the attacker will be able to receive the data transferred over that port (assuming it is a server-to-client transfer). The reason this is possible is because

RFC959 does not allow for an FTP server to demand that the IP address that connects to the port is the same as the IP address used to issue the PASV command. In effect, the process that controls the FTP session and the process that performs the file transfer are assumed by the FTP model to be logically separate which means that they can be located on physically separate machines as well.

How the pizzathief32 tool performs this exploit is detailed in-line with the annotated trace in the following section.

Annotated Network Trace

The pizzathief32 tool is a Windows command line executable with the following available options:

pizzathief32 [-v -t*N* -s] *servername*

servername	the ftp server to run the exploit against
-v	verbose mode
-*tN*	timeout in *N* milliseconds
-s	dump output to stdout

During the test, the following command line was used:

```
pizza32 -v thor
```

The output this tool produced during the run was:

```
G:\>pizza32 -v thor
requesting username
requesting password
we are logged in now
227 Entering Passive Mode (192,168,0,2,5,37).
trying ports 1318 thru 1381
227 Entering Passive Mode (192,168,0,2,5,38).
trying ports 1319 thru 1382
227 Entering Passive Mode (192,168,0,2,5,39).
```

```
trying ports 1320 thru 1383
```

(N.B.: The tool was run on a Windows 98 desktop against an FTP service running on a Windows NT 4.0 server with SP6 applied. Several test runs were made with a third machine as the victim FTP client in an effort to trace the tool while intercepting an actual connection, which was unsuccessful, probably because all three machines are on the same network segment. Time did not permit building a suitable environment with the victim running on a slow serial link or writing a custom victim client.)

When the tool is started it completes a three-way handshake to the FTP server's well-known port (21). After the connection has completed the FTP server issues an opening banner to the client:

```
11/18-20:23:25.113879 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:32020  DF
******S* Seq: 0x3269E8E   Ack: 0x0   Win: 0x2000
TCP Options => MSS: 1474 NOP NOP SackOK

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

11/18-20:23:25.114245 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:51516  DF
***A**S* Seq: 0xA1B0DA   Ack: 0x3269E8F   Win: 0x228C
TCP Options => MSS: 1474

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

11/18-20:23:25.114585 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:32276  DF
***A**** Seq: 0x3269E8F   Ack: 0xA1B0DB   Win: 0x228C

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

11/18-20:23:25.115516 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:51772  DF
***AP*** Seq: 0xA1B0DB   Ack: 0x3269E8F   Win: 0x228C
32 32 30 20 74 68 6F 72 20 4D 69 63 72 6F 73 6F  220 thor Microso
66 74 20 46 54 50 20 53 65 72 76 69 63 65 20 28  ft FTP Service (
56 65 72 73 69 6F 6E 20 34 2E 00 00 00 00 00     Version 4......

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

Next, the tool attempts an anonymous login to the FTP server with the username *"ftp"* and *"pizzaman@illuminati.gov"* for the

password. The server responds that anonymous access is allowed and finally, that the login was successful.

```
11/18-20:23:25.118840 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:32532  DF
***AP*** Seq: 0x3269E8F   Ack: 0xA1B10A   Win: 0x225D
75 73 65 72 20 66 74 70 0A                    user ftp.

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

11/18-20:23:25.119420 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:52028  DF
***AP*** Seq: 0xA1B10A   Ack: 0x3269E98   Win: 0x2283
33 33 31 20 41 6E 6F 6E 79 6D 6F 75 73 20 61 63  331 Anonymous ac
63 65 73 73 20 61 6C 6C 6F 77 65 64 2C 20 73 65  cess allowed, se
6E 64 20 69 64 65 6E 74 69 74 00 00 00 00 00 00  nd identit......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00                          ........

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

11/18-20:23:25.136360 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:32788  DF
***AP*** Seq: 0x3269E98   Ack: 0xA1B152   Win: 0x2215
70 61 73 73 20 70 69 7A 7A 61 6D 61 6E 40 69 6C  pass pizzaman@il
6C 75 6D 69 6E 61 74 69 2E 67 6F 76 0A           luminati.gov.

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

11/18-20:23:25.139477 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:52284  DF
***AP*** Seq: 0xA1B152   Ack: 0x3269EB5   Win: 0x2266
32 33 30 20 41 6E 6F 6E 79 6D 6F 75 73 20 75 73  230 Anonymous us
65 72 20 6C 6F 67 67 65 64 20 69 6E 2E 0D 0A     er logged in...

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

At this point the tool has established an FTP session with the server. It issues a PASV command and the server responds to the client with its IP address and the port number it will listen on. Note that the server's IP address is 192.168.0.2, which can be clearly seen in the highlighted data portion of the middle packet below.

What isn't clear is the port number because tcpdump did not capture enough of the packet. The two values following the IP address;

5,3, are misleading due to the missing '7' (see the tool's output above). Since port numbers are 16 bit values which are shown in network byte order, we multiply the first value by 256 then add the second value to come up with the port number; (5 * 256) + 37 = 1317.

The tool sends a second PASV command as well. Although this seems odd, what is actually happening is that the first PASV was issued before the tool entered the main execution loop. So, the response from the second PASV won't be acted on until the next time through the loop.

(N.B.: The ACK packet to the second PASV is shown here out-of-sequence for clarity.)

```
11/18-20:23:25.153362 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:33044  DF
***AP*** Seq: 0x3269EB5   Ack: 0xA1B171   Win: 0x21F6
70 61 73 76 0A                                    pasv.

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

11/18-20:23:25.155141 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:52540  DF
***AP*** Seq: 0xA1B171   Ack: 0x3269EBA   Win: 0x2261
32 32 37 20 45 6E 74 65 72 69 6E 67 20 50 61 73  227 Entering Pas
73 69 76 65 20 4D 6F 64 65 20 28 31 39 32 2C 31  sive Mode (192,1
36 38 2C 30 2C 32 2C 35 2C 33 00 00 00 00 00     68,0,2,5,3.....

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

11/18-20:23:25.170735 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:33300  DF
***AP*** Seq: 0x3269EBA   Ack: 0xA1B1A0   Win: 0x21C7
70 61 73 76 0A                                    pasv.

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

11/18-20:23:25.172476 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:52796  DF
***AP*** Seq: 0xA1B1A0   Ack: 0x3269EBF   Win: 0x225C
32 32 37 20 45 6E 74 65 72 69 6E 67 20 50 61 73  227 Entering Pas
73 69 76 65 20 4D 6F 64 65 20 28 31 39 32 2C 31  sive Mode (192,1
36 38 2C 30 2C 32 2C 35 2C 33 00 00 00 00 00     68,0,2,5,3.....
```

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

11/18-20:23:25.381428 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:50196  DF
***A**** Seq: 0x3269EBF   Ack: 0xA1B1CF   Win: 0x2198

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

Now let's revisit the tool's output for a moment and focus on two lines of output that are repeated here. Keep in mind that the tool has issued two PASV commands.

```
227 Entering Passive Mode (192,168,0,2,5,37).
trying ports 1318 thru 1381
```

Here we see that the tool is going to scan a range of ports from 1318 through 1381. The main loop calls a subroutine that scans a range of 64 ports starting with the next port that was returned from the PASV command, which makes sense. But, because the second PASV is issued before the port scan is completed, the next higher port, 1318 in this case, is returned from the second PASV command. This is a bug and limits the tool's effectiveness because it will always complete a three-way connection to the first port each time it calls the scan loop.

To confirm this let's take a look at the packets for port 1318.

```
11/18-20:23:25.172626 192.168.0.5:2110 -> 192.168.0.2:1318
TCP TTL:128 TOS:0x0 ID:33556  DF
******S* Seq: 0x3269ECA   Ack: 0x0   Win: 0x2000
TCP Options => MSS: 1474 NOP NOP SackOK

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

11/18-20:23:25.172929 192.168.0.2:1318 -> 192.168.0.5:2110
TCP TTL:128 TOS:0x0 ID:53052  DF
***A**S* Seq: 0xA1B0EF   Ack: 0x3269ECB   Win: 0x228C
TCP Options => MSS: 1474

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

11/18-20:23:25.173275 192.168.0.5:2110 -> 192.168.0.2:1318
TCP TTL:128 TOS:0x0 ID:33812  DF
***A**** Seq: 0x3269ECB   Ack: 0xA1B0F0   Win: 0x228C
```

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

A session was established as expected. The other 63 ports in the scan will only complete a three-way handshake if another client is also using a passive FTP session. Otherwise, their SYN packets will receive an RST packet because the server is not listening on the port. The following, for brevity, shows the packets associated with the 2nd and 64th ports scanned by the tool. Note the immediate RST packets in response to the tool's SYN packets.

```
11/18-20:23:25.188948 192.168.0.5:2111 -> 192.168.0.2:1319
TCP TTL:128 TOS:0x0 ID:34068  DF
******S* Seq: 0x3269EDA   Ack: 0x0   Win: 0x2000
TCP Options => MSS: 1474 NOP NOP SackOK

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

11/18-20:23:25.189173 192.168.0.2:1319 -> 192.168.0.5:2111
TCP TTL:128 TOS:0x0 ID:53308
***A*R** Seq: 0x0   Ack: 0x3269EDB   Win: 0x0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
********************** [ Packets Deleted ] ***********************
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/18-20:23:25.359688 192.168.0.5:2173 -> 192.168.0.2:1381
TCP TTL:128 TOS:0x0 ID:49940  DF
******S* Seq: 0x3269F85   Ack: 0x0   Win: 0x2000
TCP Options => MSS: 1474 NOP NOP SackOK

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

11/18-20:23:25.359903 192.168.0.2:1381 -> 192.168.0.5:2173
TCP TTL:128 TOS:0x0 ID:3645
***A*R** Seq: 0x0   Ack: 0x3269F86   Win: 0x0

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

The next thing that happens after all of the ports have been opened is the tool goes to sleep for 5 seconds before dropping into a loop to process each open port. The loop first checks if a port has a connection established. If there is connection the tool dumps whatever data it is able to read from the port, either to the screen if the –s command line option is present, or to a file. The last step is to close the

port.

Okay, let's rewind back to port 1318. The connection is still established after all 64 ports have been scanned. We see the 5-second delay after the connection to port 1318 was established. Next, the tool closes the connection by sending a FIN packet, which is an orderly shutdown. The server responds to the FIN with an ACK packet of it's own.

So, what's with the RST packet right after that? Well, it's fairly safe to assume that the Microsoft FTP server is in a blocked read state after the connection has been established. When the tool closes the connection an error results because the close is unexpected from the server's perspective. The RST packet is probably the result of the server aborting the file transfer request.

```
11/18-20:23:30.359916 192.168.0.5:2110 -> 192.168.0.2:1318
TCP TTL:128 TOS:0x0 ID:35093  DF
***A***F Seq: 0x3269ECB   Ack: 0xA1B0F0   Win: 0x228C

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

11/18-20:23:30.360208 192.168.0.2:1318 -> 192.168.0.5:2110
TCP TTL:128 TOS:0x0 ID:53565  DF
***A**** Seq: 0xA1B0F0   Ack: 0x3269ECC   Win: 0x228C

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

11/18-20:23:30.365143 192.168.0.2:1318 -> 192.168.0.5:2110
TCP TTL:128 TOS:0x0 ID:53821  DF
*****R** Seq: 0xA1B0F0   Ack: 0x3269ECC   Win: 0x0

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

At this point the tool exits the port scan subroutine and returns to the main execution loop. The loop restarts, processing the response from the second PASV sent so long ago. It then sends another PASV to the server as it prepares for another iteration of the port scan subroutine.

```
11/18-20:23:30.364555 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:35349  DF
***AP*** Seq: 0x3269EBF   Ack: 0xA1B1CF   Win: 0x2198
70 61 73 76 0A                                  pasv.

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

```
11/18-20:23:30.367417 192.168.0.2:21 -> 192.168.0.5:2109
TCP TTL:128 TOS:0x0 ID:54333  DF
***AP*** Seq: 0xA1B1CF   Ack: 0x3269EC4   Win: 0x2257
32 32 37 20 45 6E 74 65 72 69 6E 67 20 50 61 73  227 Entering Pas
73 69 76 65 20 4D 6F 64 65 20 28 31 39 32 2C 31  sive Mode (192,1
36 38 2C 30 2C 32 2C 35 2C 33 00 00 00 00 00     68,0,2,5,3.....

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

11/18-20:23:30.484205 192.168.0.5:2109 -> 192.168.0.2:21
TCP TTL:128 TOS:0x0 ID:52245  DF
***A**** Seq: 0x3269EC4   Ack: 0xA1B1FE   Win: 0x2169

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

The tool will continue scanning indefinitely because the main loop is infinite. The tool will only terminate if any of the system calls it makes returns an error when called or if a keyboard interrupt occurs.

Now, how does this look when viewed from the perspective of a SHADOW hourly wrap-up report? Hmm, looks like an honest-to-goodness port scan from this altitude…

(N.B.: The tool doesn't create any crafted packets. Those are real-live Win98 sequence numbers.. ;-)

```
192.168.0.5 > 192.168.0.2
19:23:25.172626 zerk.radigan.org.2110 > thor.radigan.org.1318: S 52862666:52862666(0) win 8192  (DF)
19:23:25.188948 zerk.radigan.org.2111 > thor.radigan.org.1319: S 52862682:52862682(0) win 8192  (DF)
19:23:25.189419 zerk.radigan.org.2112 > thor.radigan.org.1320: S 52862682:52862682(0) win 8192  (DF)
19:23:25.283156 zerk.radigan.org.2113 > thor.radigan.org.1321: S 52862776:52862776(0) win 8192  (DF)
19:23:25.283509 zerk.radigan.org.2114 > thor.radigan.org.1322: S 52862776:52862776(0) win 8192  (DF)
19:23:25.283984 zerk.radigan.org.2115 > thor.radigan.org.1323: S 52862777:52862777(0) win 8192  (DF)
19:23:25.284472 zerk.radigan.org.2116 > thor.radigan.org.1324: S 52862777:52862777(0) win 8192  (DF)
19:23:25.284946 zerk.radigan.org.2117 > thor.radigan.org.1325: S 52862778:52862778(0) win 8192  (DF)
19:23:25.285482 zerk.radigan.org.2118 > thor.radigan.org.1326: S 52862778:52862778(0) win 8192  (DF)
19:23:25.285969 zerk.radigan.org.2119 > thor.radigan.org.1327: S 52862779:52862779(0) win 8192  (DF)
19:23:25.286444 zerk.radigan.org.2120 > thor.radigan.org.1328: S 52862779:52862779(0) win 8192  (DF)
19:23:25.302857 zerk.radigan.org.2121 > thor.radigan.org.1329: S 52862796:52862796(0) win 8192  (DF)
19:23:25.303228 zerk.radigan.org.2122 > thor.radigan.org.1330: S 52862796:52862796(0) win 8192  (DF)
19:23:25.303709 zerk.radigan.org.2123 > thor.radigan.org.1331: S 52862797:52862797(0) win 8192  (DF)
19:23:25.304175 zerk.radigan.org.2124 > thor.radigan.org.1332: S 52862797:52862797(0) win 8192  (DF)
19:23:25.304646 zerk.radigan.org.2125 > thor.radigan.org.1333: S 52862798:52862798(0) win 8192  (DF)
```

```
19:23:25.305117 zerk.radigan.org.2126 > thor.radigan.org.1334: S 52862798:52862798(0) win 8192  (DF)
19:23:25.305596 zerk.radigan.org.2127 > thor.radigan.org.1335: S 52862799:52862799(0) win 8192  (DF)
19:23:25.306069 zerk.radigan.org.2128 > thor.radigan.org.1336: S 52862799:52862799(0) win 8192  (DF)
19:23:25.306552 zerk.radigan.org.2129 > thor.radigan.org.1337: S 52862800:52862800(0) win 8192  (DF)
19:23:25.307058 zerk.radigan.org.2130 > thor.radigan.org.1338: S 52862800:52862800(0) win 8192  (DF)
19:23:25.307521 zerk.radigan.org.2131 > thor.radigan.org.1339: S 52862801:52862801(0) win 8192  (DF)
19:23:25.307952 zerk.radigan.org.2132 > thor.radigan.org.1340: S 52862801:52862801(0) win 8192  (DF)
19:23:25.308443 zerk.radigan.org.2133 > thor.radigan.org.1341: S 52862801:52862801(0) win 8192  (DF)
19:23:25.308917 zerk.radigan.org.2134 > thor.radigan.org.1342: S 52862802:52862802(0) win 8192  (DF)
19:23:25.309456 zerk.radigan.org.2135 > thor.radigan.org.1343: S 52862802:52862802(0) win 8192  (DF)
19:23:25.309958 zerk.radigan.org.2136 > thor.radigan.org.1344: S 52862803:52862803(0) win 8192  (DF)
19:23:25.310447 zerk.radigan.org.2137 > thor.radigan.org.1345: S 52862803:52862803(0) win 8192  (DF)
19:23:25.310930 zerk.radigan.org.2138 > thor.radigan.org.1346: S 52862804:52862804(0) win 8192  (DF)
19:23:25.311429 zerk.radigan.org.2139 > thor.radigan.org.1347: S 52862804:52862804(0) win 8192  (DF)
19:23:25.311905 zerk.radigan.org.2140 > thor.radigan.org.1348: S 52862805:52862805(0) win 8192  (DF)
19:23:25.312385 zerk.radigan.org.2141 > thor.radigan.org.1349: S 52862805:52862805(0) win 8192  (DF)
19:23:25.312868 zerk.radigan.org.2142 > thor.radigan.org.1350: S 52862806:52862806(0) win 8192  (DF)
19:23:25.313327 zerk.radigan.org.2143 > thor.radigan.org.1351: S 52862806:52862806(0) win 8192  (DF)
19:23:25.314285 zerk.radigan.org.2145 > thor.radigan.org.1353: S 52862807:52862807(0) win 8192  (DF)
19:23:25.314765 zerk.radigan.org.2146 > thor.radigan.org.1354: S 52862808:52862808(0) win 8192  (DF)
19:23:25.330808 zerk.radigan.org.2147 > thor.radigan.org.1355: S 52862824:52862824(0) win 8192  (DF)
19:23:25.331251 zerk.radigan.org.2148 > thor.radigan.org.1356: S 52862824:52862824(0) win 8192  (DF)
19:23:25.331753 zerk.radigan.org.2149 > thor.radigan.org.1357: S 52862825:52862825(0) win 8192  (DF)
19:23:25.332241 zerk.radigan.org.2150 > thor.radigan.org.1358: S 52862825:52862825(0) win 8192  (DF)
19:23:25.332734 zerk.radigan.org.2151 > thor.radigan.org.1359: S 52862826:52862826(0) win 8192  (DF)
19:23:25.333292 zerk.radigan.org.2152 > thor.radigan.org.1360: S 52862826:52862826(0) win 8192  (DF)
19:23:25.333798 zerk.radigan.org.2153 > thor.radigan.org.1361: S 52862827:52862827(0) win 8192  (DF)
19:23:25.334276 zerk.radigan.org.2154 > thor.radigan.org.1362: S 52862827:52862827(0) win 8192  (DF)
19:23:25.334766 zerk.radigan.org.2155 > thor.radigan.org.1363: S 52862828:52862828(0) win 8192  (DF)
19:23:25.335266 zerk.radigan.org.2156 > thor.radigan.org.1364: S 52862828:52862828(0) win 8192  (DF)
19:23:25.335746 zerk.radigan.org.2157 > thor.radigan.org.1365: S 52862829:52862829(0) win 8192  (DF)
19:23:25.336232 zerk.radigan.org.2158 > thor.radigan.org.1366: S 52862829:52862829(0) win 8192  (DF)
19:23:25.336730 zerk.radigan.org.2159 > thor.radigan.org.1367: S 52862830:52862830(0) win 8192  (DF)
19:23:25.337217 zerk.radigan.org.2160 > thor.radigan.org.1368: S 52862830:52862830(0) win 8192  (DF)
19:23:25.337704 zerk.radigan.org.2161 > thor.radigan.org.1369: S 52862831:52862831(0) win 8192  (DF)
19:23:25.338193 zerk.radigan.org.2162 > thor.radigan.org.1370: S 52862831:52862831(0) win 8192  (DF)
19:23:25.338681 zerk.radigan.org.2163 > thor.radigan.org.1371: S 52862832:52862832(0) win 8192  (DF)
19:23:25.339181 zerk.radigan.org.2164 > thor.radigan.org.1372: S 52862832:52862832(0) win 8192  (DF)
19:23:25.339834 zerk.radigan.org.2165 > thor.radigan.org.1373: S 52862833:52862833(0) win 8192  (DF)
19:23:25.340402 zerk.radigan.org.2166 > thor.radigan.org.1374: S 52862833:52862833(0) win 8192  (DF)
19:23:25.341010 zerk.radigan.org.2167 > thor.radigan.org.1375: S 52862834:52862834(0) win 8192  (DF)
19:23:25.341150 zerk.radigan.org.2168 > thor.radigan.org.1376: S 52862834:52862834(0) win 8192  (DF)
19:23:25.357789 zerk.radigan.org.2169 > thor.radigan.org.1377: S 52862851:52862851(0) win 8192  (DF)
19:23:25.358213 zerk.radigan.org.2170 > thor.radigan.org.1378: S 52862851:52862851(0) win 8192  (DF)
19:23:25.358711 zerk.radigan.org.2171 > thor.radigan.org.1379: S 52862852:52862852(0) win 8192  (DF)
```

```
19:23:25.359194 zerk.radigan.org.2172 > thor.radigan.org.1380: S 52862852:52862852(0) win 8192  (DF)
19:23:25.359688 zerk.radigan.org.2173 > thor.radigan.org.1381: S 52862853:52862853(0) win 8192  (DF)
```

Assignment 3 - "Analyze This" Scenario

This analysis is based on about 19 days of Snort alert and scans logs covering a period of just over a month. The following sections summarize those alerts and/or scans that require further research using captured network traffic, logs and host audits in order to properly categorize these findings as real issues or as normal activity. Once an in-depth analysis has been completed then a realistic set of action items can be drafted.

Host & Port Mapping

A significant amount of host and port mapping activity has been aimed at MY.NET. While the total number of port scan entries in the log files totals over 247,000, the following list only shows those hosts that have scanned 1,000+ hosts or ports:

Total	Source Address	Scan Type
42652	195.114.226.41	SYN
31779	24.180.134.156	SYN
27125	210.125.174.11	UDP
25469	35.10.82.111	SYN
22140	206.186.79.9	SYN
20155	24.17.189.83	SYN
19965	212.141.100.97	SYN
14813	63.248.55.245	UDP
4663	129.186.93.133	SYN
3300	194.165.230.250	SYN
3234	210.55.227.138	SYN
2777	MY.NET.1.3	UDP

2542	MY.NET.1.13	UDP
2294	MY.NET.1.5	UDP
2279	MY.NET.1.4	UDP
2392	210.61.144.125	SYN/FIN
1944	168.187.26.157	SYN
1781	209.123.198.156	UDP
1094	216.99.200.242	SYN

The total number of scans by these hosts is 232,398 or about 93% of the total scanning activity.

Watchlist 000222 NET-NCFC

There were 18,800+ entries associated with this alert, more than any other alert examined in this analysis. Without access to the Snort ruleset to evaluate the alert definition I can't be sure if this is triggered on source addresses within the 159.226.0.0 network alone or in addition to SYN packets. In any event, this address block is assigned to the following:

The Computer Network Center Chinese Academy of Sciences (NET-NCFC)
P.O. Box 2704-10,
Institute of Computing Technology Chinese Academy of Sciences
Beijing 100080, China

The following notable activity was observed between MY.NET and 159.226.0.0:

- Have or are trying to gain telnet access to several hosts in MY.NET
- Conducting SMTP transfers with MY.NET.100.230, MY.NET.253.41, MY.NET.253.42, MY.NET.253.43, MY.NET.6.7 showing up in the alert logs most often.
- Outbound telnet from MY.NET.163.32 to 159.226.45.3.
- Outbound telnet from MY.NET.60.11 to 159.226.41.166.

- Outbound POP3 from MY.NET.163.32 to 159.226.45.3.
- Possible remote admin tool *"RAT"* running on MY.NET.162.199.

Watchlist 000220 IL-ISDNNET-990517

Similar to the block of alerts detailed in the previous section, this alert generated over 5,200 entries. As with the rule for the previous alert, the same applies here. The 212.179.7.0 network block appears to be assigned to an ISP located in Israel:

ISDN Net Ltd.
BEZEQ INTERNATIONAL
40 Hashacham Street
Petach-Tikva 49170 Israel

The following notable activity was observed between MY.NET and 212.179.7.0:

- Open relay use of MY.NET.253.41, MY.NET.253.42, MY.NET.253.43.
- HTTPS activity to MY.NET.5.29.
- Outbound telnet from MY.NET.98.168 to 212.179.58.2.
- Possible Napster activity on MY.NET.157.200, MY.NET.181.87, MY.NET.221.94 and MY.NET.206.154.
- Miscellaneous activity that needs further investigation to classify properly.

RPC and Portmapper Activity

Approximately 2,000 alerts have been generated in relation to the following activity. Although some can be attributed to reconnaissance, others point to hosts in MY.NET that might be compromised or facing a concerted effort to be compromised.

- 205.188.179.33:4000 -> MY.NET.217.42:32771 (although the number of days that these alerts span this could indicate allowed

activity or even simple stimulus; ICR server? – hard to say for sure.)

- 205.188.153.109:4000 -> MY.NET.219.26:32771 and 205.188.153.98:4000 -> MY.NET.222.66:32771, etc… (ditto.)
- Various hosts to MY.NET.6.15:111 could be possible RPC vulnerability scans.

SNMP Public Community String

Over 800 alerts have been attributed to this vulnerability for hosts in MY.NET and MY.NET.101.192. This is considered on of the top 10 Internet Security Risks as documented in http://www.sans.org/topten.htm.

WinGate Proxy Activity

Most notable in this group was a rather exhaustive scan of MY.NET for WinGate proxies performed by 168.187.26.157 which part of the 168.187.0.0 network that is registered to:

Kuwait Ministry of Communations (NET-MOC-KW)
PO Box No 31811111
KW

Miscellaneous Reconnaissance

This section relates to general recon activities; null scans, SYN-FIN scans, NMAP pings, OS fingerprinting and so on. All told, there were over 3,400 alerts that fell into this category.

One of the most active scans was an exhaustive SYN-FIN scan of 2,300+ addresses in MY.NET for active FTP servers by 210.61.144.125 which is part of the 210.61.144.0 network which is assigned to:

Abnet Information Co., Ltd
6F. No.22-5
Ning Hsia Rd Taipei, Taiwan

Another large SYN-FIN scan of MY.NET (600+ addresses) was performed by 213.25.136.60, which is part of the 213.25.136.32 - 213.25.136.63 network range assigned to:

Infores Bilgoraj
Bilgoraj, Poland

Miscellaneous Alerts

The following specific alerts should also be investigated:

- Happy 99 Virus detected going to MY.NET.6.25 and MY.NET.179.80
- Possible wu-ftpd exploit going to MY.NET.150.24 and MY.NET.202.202
- Site exec – Possible wu-ftpd exploit going to MY.NET.150.24, MY.NET.202.190 and MY.NET.202.202

A few alerts related to small fragments was also detected going to 8 different MY.NET hosts.

Assignment 4 - Analysis Process

The following sections detail the tools and resources used to accomplish the analysis in the preceding section.

The data set consisted of 18 Snort portscan log files, 20 Snort alert log files and 18 OOS log files. In each set there was one duplicate file, which was excluded from the data used for the analysis. Each file was first edited to delete the non-log information at the beginning of each file. The duplicate files were run against their mate using the *diff* tool to ensure that the two files were, in fact, exact duplicates.

After the files were cleaned a shell script was written to create a daily SnortSnarf view of the log files. The script created a subdirectory for each day of logs. For those days that were missing either an alert log or portscan log were provided an empty file. The daily SnortSnarf web reports provided an overview to get a feel for the dataset before using the tools described below to drill down.

Host & Port Scanning

A single concatenated portscan file was created that contained a total of over 237,000 entries. Since these entries would not fit into a single Excel spreadsheet I took a different approach to derive the table shown above. I first modified the scripts that Lenny Zeltser created for his practical to create a file containing just the host address and scan label from the portscan file. The resulting file was then processed using the following command:

```
sort scans.all | uniq –c >scans.lst
```

What this command does is *sort* the file then pipe the output into *uniq* which will output one line of the file and dump any duplicate lines. The *–c* command option instructs *uniq* to prefix each line with the total number of duplicate lines found.

The resulting file was then edited to remove any entries with a count less than 1,000. The final result was manually sorted and pasted into this document.

Alert Logs

I modified Lenny's script to create a comma-separated alert file for use by Excel. To make the data easier to manage I first created multiple worksheets in the workbook for the following classes of alerts; *Alerts, Portscans, Reconn, WinGate, SNMP, RPC, ISDNNET and NET_NCFC.* You'll probably have a different dataset to work with so you might need to add a sheet or two and drop one of the ones I used. Roughly speaking, grab a cup, sort by alertname and then start scrolling through the 60k lines of alerts. You'll see which ones need to be separated into separate sheets.

After segregating the data in this fashion it becomes a matter of sorting each worksheet in several ways to see what sort of patterns

develop while you scroll through the workbook. As I worked through this exercise I found myself working through each workbook in roughly this order:

- Sort by timestamp only, which gives you a pretty good 50k view of the world to start.
- Then sort by destination port, destination host and source host (top-to-bottom, in order). High activity destination hosts by the services they provide tend to surface using this sort order.
- Then sort by source host, source port and destination host. This tends to show the source hosts that are doing scans, check the timestamps to see if it's long or short term.
- Try sorting by either destination host or source host and timestamp to get a host-centric view of chronological activity.

Resources

The following web resources were used for research during the writing of this practical.

http://www.sans.org
http://www.whitehats.com
http://cve.mitre.org
http://www.geektools.com
http://www.google.com