Analyzing Anomalous Traffic

Becky Pinkard

GCIA Practical Assignment Version 2.8

Table of Contents:

Detects (5)

Describe the State of Intrusion Detection

Analyze This

Appendices

DETECTS (5)

Detect 1 – Fin scan (false interpretation)

```
11:35:02.657886 client.net.37.253.56553 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 6997 4000 f206 b10b c61f 25fd
                         xxxx xx03 dce9 0050 3796 677d 000e 7017
                         5011 4470 10c4 0000 c7f1 677d e557
11:35:02.660172 client.net.37.253.57520 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 6998 4000 f206 b10a c61f 25fd
                         xxxx xx03 e0b0 0050 37cc 1bc1 000e 6807
                         5011 4470 6093 0000 6c39 1bc1 6492
11:35:02.666650 client.net.37.253.57521 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 6999 4000 f206 b109 c61f 25fd
                         xxxx xx03 e0b1 0050 37cf 09d4 000e 681a
                         5011 4470 7269 0000 35e9 09d4 6966
11:35:02.666874 client.net.37.253.17705 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 699a 4000 f206 b108 c61f 25fd
                         xxxx xx03 4529 0050 2240 7d9f 32d3 05a2
                         5011 4470 df68 0000 e50a 7d9f 2f74
<snip 87 records…>

11:44:37.080438 client.net.37.253.57522 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 bd2a 4000 f206 5d78 c61f 25fd
                         xxxx xx03 e0b2 0050 37cd 15d0 000e 680f
                         5011 4470 6679 0000 f7dc 15d0 613e
11:44:37.080720 client.net.37.253.57519 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 bd2b 4000 f206 5d77 c61f 25fd
                         xxxx xx03 e0af 0050 37ce 0fc5 000e 6818
                         5011 4470 6c7d 0000 a02b 0fc5 796c
11:44:37.080729 client.net.37.253.56945 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 bd2c 4000 f206 5d76 c61f 25fd
                         xxxx xx03 de71 0050 379e 35b4 000e 9b02
                         5011 4470 1612 0000 e509 35b4 1bc0
11:44:37.082116 client.net.37.253.56551 > my.company.net.3.80: F 0:0(0) ack 1 win 17520 (DF)
                         4500 0028 bd2d 4000 f206 5d75 c61f 25fd
                         xxxx xx03 dce7 0050 3793 7d07 000e 87cd
                         5011 4470 e388 0000 18a0 7d07 2f67
```

1. **Source of trace:**

   The source of this trace was my company’s network. I believe it to be a false interpretation because while it was pinpointed with a specific traffic filter, I still am not certain what the stimulus for the traffic was.

2. **Detect was generated by:**

   This detect was generated by a tcpdump filter applied at the external interface of our company firewall specifically in response to anomalous “icmp: time exceeded in transit” errors originating from our internal host and talking to the client.NAT.hide.253 address. The icmp messages looked like this:

   ```
   23:19:43.487411 my.company.net.3 > client.net.37.253: icmp: time exceeded in-transit
   ```

```
23:19:49.512552 my.company.net.3 > client.net.37.253: icmp: time exceeded in-transit
```

This could have been determined via a tcpdump filter with masking explicit to this client.net.37.253 host (tcpdump –i eth-s1p1c0 host client.net.37.253); however, to validate that the traffic was occurring only in conjunction with this particular client, a filter that masked out the fin bit and presented all of the fin packets was implemented: tcpdump –i eth-s1p1c0 'tcp and (tcp[13] & 0x01 != 0)'. This filter listens on the external interface of the fw (-i eth-s1p1c0) and listens for the fin flag set (0x01 != 0) in the flags bit of the tcp header (tcp[13]).

3. **Probability the source address was spoofed:**

   The source address was most definitely not spoofed as it was discovered to be an address belonging to a client of my company. The changing source ports, that appeared to be consistent with a scan that does not utilize a static source port, were actually created by multiple connections from customers clustered behind the client.net.37.253 address.

4. **Description of attack:**

   Originally believed to have been some type of fin port scan; however after a closer look, it was determined that the traffic was specific to only one host. Further investigation at arin.net showed the IP address belonging to a client of our web software. I also hesitate to classify this particular scan as a false positive, for the reason that the "ack 1" bits are anomalous to the normal fin-ack, fin-ack close sequence and we are missing the return fin-ack sequence.

5. **Attack mechanism:**

   Had this been a fin port scan - the attack normally works by sending crafted packets to a range of addresses utilizing only the 'F' or fin flag. This type of attack works for reconnaissance because the destination machine will "alert" on open ports by not responding while silently dropping the fin packets. Closed ports will send a reset in response to any packet received that does not contain a reset (RFC 793). Attackers utilize "stealthy" measures of this type in attempt to avoid being logged. This type of technique is not always foolproof in that the packets are abnormally occuring and different protocol stacks and operating systems will respond in various manners.

6. **Correlations:**

   This particular detect could not be correlated with any previous submits to GIAC or other websites. Requests to intrusion@sans.org also did not turn up more information. To date, the traffic has only been detected one time on my company network. Because this was capture from a production environment, there was no opportunity to attempt reproduction. I believe that a "blue screen" or cold boot of one of our web servers behind the load balancer could have caused this traffic.

   This traffic can almost be completely duplicated utilizing the stealth fin option in Nmap. Implementing the following Nmap syntax:

   nmap -P0 -sF -p80 my.company.net.3 (nmap –Ping0 –stealthFin –port80 dest. IP)

   will generate this traffic:

   ```
   23:19:43.479773 my.test.net.101.60496 > my.company.net.3.80: F 0:0(0) win 1024
   23:19:43.487411 10.0.0.208 > 208.12.178.101: icmp: time exceeded in-transit
   23:19:49.502991 my.test.net.101.60497 > my.company.net.3.80: F 0:0(0) win 1024
   23:19:49.512552 10.0.0.208 > 208.12.178.101: icmp: time exceeded in-transit
   ```

   As you can see, the main difference (and the reason the scan deserved further analysis) is there is no ack bit in the dump caused by Nmap. The icmp error messages are generated and the fin bit is zero – but no ack bit of value 1. Also of value is the fact that the source ports in the Nmap scan appear to be incrementing as they should whereas the source ports in the detect are much more scattered.

7. **Evidence of active targeting:**

   Active targeting would have definitely been assumed if this had been a real scanner. The packets were directed only at this particular server, which is a vital piece of our web application. Considering the traffic was seen only in conjunction with the one server, the packets directed solely to port 80 would have been a good indication that the attacker knew a web server resided at that address. I would have then assumed the intent of the traffic to qualify the belief that a web server resided at that IP or an attempt to map the operating system type (based on the crafted appearance of the packets).

8. **Calculate severity:**

   Severity = (Criticality + Lethality) – (System Countermeasures + Network Countermeasures)

   (5 + 2) – (4 + 4) = -1

   Target Criticality:
   This machine plays an integral part in serving our web application to many clients. If it were compromised, the integrity of data that we maintain would be jeopardized.

   Attack Lethality:
   If the intention of the traffic had been to port scan this host, the results of their scan would not have directly compromised this box. Since the best estimate of this detect is that it is not malicious traffic, the lethality would be nil, except for the fact that the true stimulous is still undetermined. Final answer - 2.

   System Countermeasures:
   The box was fully patched minus the latest IIS remote buffer overflow vulnerability. Reference: http://www.microsoft.com/technet/security/bulletin/ms01-023.asp

   Network Countermeasures:
   This server is situated behind a firewall and packet/content matching intrusion detection measures are in place at the border. However, because the firewall rule allows port 80 traffic to this host, the packet was able to elicit a response from our internal host.

9. **Defensive recommendation:**

   From a curiosity standpoint, I would recommend a reproduction attempt to determine the exact stimulus of this dump. I would also follow-up with the system or security administrator of the client site to share data and brainstorm. However, this would appear to be a scan at the worst and filters specifically listening for repeat offenders have not alarmed to date, so existing countermeasures are adequate and the nature of this detect therefore appears non-malignant.

10. **Multiple choice test question:**

If an open port receives an initial packet with the fin flag set, how is it supposed to process the packet - based on the RFC specifications?

A. Respond with a reset.
B. Silently discard the packet.
C. Respond with a syn-ack.
D. Drop the packet and respond with a fin-ack.

Take me to the answer!

Detect 2 – IIS Unicode attack

(sanitized and trimmed for brevity)

```
[**] spp_http_decode: IIS Unicode attack detected [**]
05/21-17:50:43.920163 216.203.179.199:1048 -> xxx.xxx.xxx.2:80 TCP
TTL:57 TOS:0x0 ID:38 IpLen:20 DgmLen:399 DF
***AP*** Seq: 0x31C0E7D4  Ack: 0x36E8CAF4  Win: 0x7D78  TcpLen: 20
47 45 54 20 2F 6D 73 61 64 63 2F 2E 2E 25 63 30  GET /msadc/..%c0
25 61 66 2E 2E 2F 2E 2E 25 63 30 25 61 66 2E 2E  %af../..%c0%af..
2F 2E 2E 25 63 30 25 61 66 2E 2E 2F 77 69 6E 6E  /..%c0%af../winn
74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65  t/system32/cmd.e
78 65 3F 2F 63 2B 64 69 72 2B 63 3A 5C 77 69 6E  xe?/c+dir+c:\win
6E 74 20 48 54 54 50 2F 31 2E 30 0D 0A 43 6F 6E  nt HTTP/1.0..Con
6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C  nection: Keep-Al
69 76 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A  ive..User-Agent:
20 4D 6F 7A 69 6C 6C 61 2F 34 2E 37 32 20 5B 65   Mozilla/4.72 [e
6E 5D 20 28 58 31 31 3B 20 55 3B 20 4C 69 6E 75  n] (X11; U; Linu
78 20 32 2E 32 2E 31 34 2D 35 2E 30 20 69 36 38  x 2.2.14-5.0 i68
36 29 0D 0A 48 6F 73 74 3A 20 xx xx xx xx xx xx  6)..Host: xxx.xx
xx xx xx xx xx 32 0D 0A 41 63 63 65 70 74 3A 20  .xxx.2..Accept:
69 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65  image/gif, image
2F 78 2D 78 62 69 74 6D 61 70 2C 20 69 6D 61 67  /x-xbitmap, imag
65 2F 6A 70 65 67 2C 20 69 6D 61 67 65 2F 70 6A  e/jpeg, image/pj
70 65 67 2C 20 69 6D 61 67 65 2F 70 6E 67 2C 20  peg, image/png,
2A 2F 2A 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F  */*..Accept-Enco
64 69 6E 67 3A 20 67 7A 69 70 0D 0A 41 63 63 65  ding: gzip..Acce
70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E 0D  pt-Language: en.
0A 41 63 63 65 70 74 2D 43 68 61 72 73 65 74 3A  .Accept-Charset:
20 69 73 6F 2D 38 38 35 39 2D 31 2C 2A 2C 75 74   iso-8859-1,*,ut
66 2D 38 0D 0A 0D 0A                             f-8....
```

1. **Source of trace:**

   This detect was caught by a Snort sensor in a test network belonging to my company.

2. **Detect was generated by:**

   The Unicode attempt was flagged by Snort, version 1.7 (http://www.snort.org) running a pared down rule set from Max Vision's arachnids site. (http://www.whitehats.com/ids/index.html)
   Snort is designed to perform content matching at the packet level. The rule set contains lines matching the content of known vulnerabilities. When Snort is engaged, these rules are compared against the traffic flow and matching patterns are flagged into special log files. The alert above was caught using a Snort function called "preprocessor http_decode". It is included in the rules configuration file. The sole purpose of this code is:

   "HTTP Decode is used to process HTTP URI strings and convert their data to non-obfuscated ASCII strings. This is done to defeat evasive web URL scanners and hostile attackers that could otherwise elude the content analysis strings used to examine HTTP traffic for suspicious activity. The preprocessor module takes HTTP port numbers (separated by spaces) to be normalized as its arguments (typically 80 and 8080)." http://www.snort.org/

   Basically, what this is saying is that the Unicode characters in the string (%c0%af) were caught by this preprocessor and changed back into their ASCII form - a slash (/).

3. **Probability the source address was spoofed:**

   It is doubtful that the source address would be spoofed for this type of detect. This hack depends on the 3-way handshake being completed in order to get to the stage where the attacker can push their http request to the server. It is possible that the attack is occurring from another compromised machine that the attacker also "owns".

4. **Description of attack:**

   Using the unicode encoded slash, the attacker hopes to escape the web directory of the server, ending up in the root directory. Operating in this manner, the attacker then has access with the NT permissions that have been assigned to the IUSR_computername account. By default, the account is a member of the 'everyone' group. Reference CVE-2000-0884 at http://cve.mitre.org.

5. **Attack mechanism:**

   This is more advanced play on a previous vulnerability. The first exploit was called 'File Permission Canonicalization' (reference Microsoft Security Bulletin MS00-0057 at http://www.microsoft.com/technet/security/bulletin/ms00-057.asp).

   This detect shows how encoding the slash (/) using unicode characters makes it possible to escape to the root directory of the web server. Once at the root directory level, the attacker may utilize default programs in the \winnt\system32 directory. For example, TFTP can be used to pull other files/programs of the attacker's choice onto the web server. From this vantage point, the possibilities are limitless.

6. **Correlations:**

   Brent Erickson turned in a nice copy of this same detect: http://www.sans.org/y2k/041901.htm. The unicode characters in his trace are highlighted in red. As you can see, the attacker is attempting to enumerate the directory listing of the c:\ drive. Interestingly enough, the attacker is using a proxy (normally used to hide tracks, so to speak) –

however in the headers below, you can see that the proxy is not quite anonymous and has included the address that it is forwarding for: 62.41.38.10.

```
[**] spp_http_decode: IIS Unicode attack detected [**]
04/12-05:44:29.537613 213.121.247.193:61522 -> x.x.x.23:80
TCP TTL:41 TOS:0x0 ID:2938 IpLen:20 DgmLen:289 DF
***AP*** Seq: 0xEF818D34  Ack: 0x844F3E92  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 15433327 0
47 45 54 20 2F 6D 73 61 64 63 2F 2E 2E 25 63 30  GET /msadc/..%c0
25 61 66 2E 2E 2F 2E 2E 25 63 30 25 61 66 2E 2E  %af../..%c0%af..
2F 2E 2E 25 63 30 25 61 66 2E 2E 2F 77 69 6E 6E  /..%c0%af../winn
74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65  t/system32/cmd.e
78 65 3F 2F 63 2B 64 69 72 2B 63 3A 5C 20 48 54  xe?/c+dir+c:\ HT
54 50 2F 31 2E 30 0D 0A 56 69 61 3A 20 31 2E 30  TP/1.0..Via: 1.0
20 50 72 6F 78 79 3A 33 31 32 38 20 28 53 71 75   Proxy:3128 (Squ
69 64 2F 32 2E 33 2E 53 54 41 42 4C 45 31 29 0D  id/2.3.STABLE1).
0A 58 2D 46 6F 72 77 61 72 64 65 64 2D 46 6F 72  .X-Forwarded-For
3A 20 36 32 2E 34 31 2E 33 38 2E 31 30 0D 0A 48  : 62.41.38.10..H
6F 73 74 3A 20 xx xx xx xx xx xx xx xx xx xx xx  ost: x.x.x.
32 33 0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F  23..Cache-Contro
6C 3A 20 6D 61 78 2D 61 67 65 3D 32 35 39 32 30  l: max-age=25920
30 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B  0..Connection: k
65 65 70 2D 61 6C 69 76 65 0D 0A 0D 0A           eep-alive...
```

7. Evidence of active targeting:

Highly probable. The targeted host does run a web server that was vulnerable to this attack. This vulnerability is particular to IIS, so odds are the attacker would have scanned for service and/or OS type, looking for machines to try a favorite exploit on.

8. Calculate severity:

Severity = (Criticality + Lethality) – (System Countermeasures + Network Countermeasures)

(5 + 5) – (5 + 3) = 2

Target Criticality:
Target is a web server running on a test network. However it is on a small test domain, so falling to this attack would create easier doorways to attack other systems on the test domain.

Attack Lethality:
This attack is very lethal – both because it is easy to implement (does not require any significant coding expertise to run) and because a large percentage of IIS servers utilize default permissions for the IUSR_computername account. The permissions are too broad by default. Reference MS00-0078 for more information: http://www.microsoft.com/technet/security/bulletin/ms00-078.asp.

System Countermeasures:
This host is one of the machines that all new patches/hot fixes are tested on, so it stays current with known vulnerabilities. Also, permissions of the IUSR_computername account have been locked down, the web directories reside on a separate partition, and permissions on the system tools have been restricted.

Network Countermeasures:
As previously mentioned, test network is monitored via Snort and protected behind a firewall. However, because port 80 (web service) access is required, the attacker was still able to reach the web server with the attack string. Active blocking is not implemented, so the attacker would be able to test other machines in this network for the vulnerability.

9. Defensive recommendation:

Continue to stay diligent with patches and hot fixes. Recommend periodic testing of exploit to ensure patches are functioning properly. Some patches may need to be reinstalled after new software is implemented; always reapply after making changes to server.

10. Multiple choice test question:

```
05/21-17:50:43.920163 216.203.179.199:1048 -> xxx.xxx.xxx.2:80 TCP
TTL:57 TOS:0x0 ID:38 IpLen:20 DgmLen:399 DF
***AP*** Seq: 0x31C0E7D4  Ack: 0x36E8CAF4  Win: 0x7D78  TcpLen: 20
47 45 54 20 2F 6D 73 61 64 63 2F 2E 2E 25 63 30  GET /msadc/..%c0
25 61 66 2E 2E 2F 2E 2E 25 63 30 25 61 66 2E 2E  %af../..%c0%af..
2F 2E 2E 25 63 30 25 61 66 2E 2E 2F 77 69 6E 6E  /..%c0%af../winn
74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65  t/system32/cmd.e
78 65 3F 2F 63 2B 64 69 72 2B 63 3A 5C 77 69 6E  xe?/c+dir+c:\win
6E 74 20 48 54 54 50 2F 31 2E 30 0D 0A 43 6F 6E  nt HTTP/1.0..Con
6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C  nection: Keep-Al
69 76 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A  ive..User-Agent:
20 4D 6F 7A 69 6C 6C 61 2F 34 2E 37 32 20 5B 65   Mozilla/4.72 [e
6E 5D 20 28 58 31 31 3B 20 55 3B 20 4C 69 6E 75  n] (X11; U; Linu
78 20 32 2E 32 2E 31 34 2D 35 2E 30 20 69 36 38  x 2.2.14-5.0 i68
36 29 0D 0A 48 6F 73 74 3A 20 xx xx xx xx xx xx  6)..Host: xxx.xx
xx xx xx xx xx 32 0D 0A 41 63 63 65 70 74 3A 20  .xxx.2..Accept:
......
```

What part of the trace above highlights this as an abnormal web request?

A. The TTL is too low.
B. The ack and push flags should not occur simultaneously.
C. The ‘HEAD’ command should be used instead of the ‘GET’ command.
D. The unicode characters %c0%af.

Take me to the answer!

Detect 3 – Outbound scan for TCPMUX (port 1) service

Check Point FW-1 log data:

```
num;date;time;orig;type;action;alert;i/f_name;
      i/f_dir;proto;src;dst;service;s_port;len;
      rule;xlatesrc;xlatedst;xlatesport;xlatedport;
      sys_msgs

0;2May2001; 7:07:36;external.net.178.254;log;accept;;eth-e1p3c0;
      inbound;udp;internal.net.101.231;195.24.163.123;tcpmux;2620;37;
      15;external.net.178.25;195.24.163.123;56576;tcpmux;

1;2May2001; 7:08:04;external.net.178.254;log;accept;;eth-s1p3c0;
      inbound;udp;internal.net.101.231;168.120.77.90;tcpmux;4006;37;
      15;external.net.178.25;168.120.77.90;57964;tcpmux;

2;3May2001; 8:17:32;external.net.178.254;log;accept;;eth-s1p3c0;
      inbound;udp;internal.net.101.231;195.24.163.123;tcpmux;2707;37;
      15;external.net.178.25;195.24.163.123;14724;tcpmux;

3;3May2001; 8:18:01;external.net.178.254;log;accept;;eth-s1p3c0;
      inbound;udp;internal.net.101.231;168.120.77.90;tcpmux;4094;37;
      15;external.net.178.25;168.120.77.90;16113;tcpmux;
```

Correlating tcpdump log data:

```
07:07:36.764191 external.net.178.25.56576 > 195.24.163.123.1: udp 9
                4500 0025 0017 0000 7f11 52f7 xxxx xx19
                c318 a37b dd00 0001 0011 5b3f ffff ffff
                7069 6e67 00
07:07:38.789780 external.net.178.25.56576 > 195.24.163.123.1: udp 12
                4500 0028 016e 0000 7f11 519d xxxx xx19
                c318 a37b dd00 0001 0014 84d6 ffff ffff
                6465 7461 696c 7300
07:07:40.821978 external.net.178.25.56576 > 195.24.163.123.1: udp 12
                4500 0028 029f 0000 7f11 506c xxxx xx19
                c318 a37b dd00 0001 0014 84d6 ffff ffff
                6465 7461 696c 7300
07:07:42.862494 external.net.178.25.56576 > 195.24.163.123.1: udp 12
                4500 0028 03f7 0000 7f11 4f14 xxxx xx19
                c318 a37b dd00 0001 0014 84d6 ffff ffff
                6465 7461 696c 7300
07:07:44.896803 external.net.178.25.56576 > 195.24.163.123.1: udp 12
                4500 0028 04fc 0000 7f11 4e0f xxxx xx19
                c318 a37b dd00 0001 0014 84d6 ffff ffff
                6465 7461 696c 7300
07:08:04.501906 external.net.178.25.57964 > 168.120.77.90.1: udp 9
                4500 0025 0d1e 0000 7f11 b6b1 xxxx xx19
                a878 4d5a e26c 0001 0011 c694 ffff ffff
                7069 6e67 00
07:08:06.513139 external.net.178.25.57964 > 168.120.77.90.1: udp 12
                4500 0028 0e17 0000 7f11 b5b5 xxxx xx19
                a878 4d5a e26c 0001 0014 f02b ffff ffff
                6465 7461 696c 7300
07:08:08.515771 external.net.178.25.57964 > 168.120.77.90.1: udp 12
                4500 0028 0e8d 0000 7f11 b53f xxxx xx19
                a878 4d5a e26c 0001 0014 f02b ffff ffff
                6465 7461 696c 7300
07:08:10.522738 external.net.178.25.57964 > 168.120.77.90.1: udp 12
                4500 0028 0f17 0000 7f11 b4b5 xxxx xx19
                a878 4d5a e26c 0001 0014 f02b ffff ffff
                6465 7461 696c 7300
07:08:12.544155 external.net.178.25.57964 > 168.120.77.90.1: udp 12
                4500 0028 0fb0 0000 7f11 b41c xxxx xx19
                a878 4d5a e26c 0001 0014 f02b ffff ffff
                6465 7461 696c 7300
```

1. **Source of trace:**

   The source of this trace was a tcpdump filter specifically listening for port 1 traffic on the /24 belonging to my company.

2. **Detect was generated by:**

   A tcpdump filter: tcpdump –x –s1514 –i eth-s1p1c0 dst port 1. This filter was put in place after icmp port unreachable messages (type 3, code 3) were discovered on the external interface of the firewall in response to the query for udp port 1.

3. **Probability the source address was spoofed:**

   The source address is not spoofed. In this detect, the source address is actually the hiding address for internet-bound traffic originating from my company. Internally, machines are assigned RFC 1918, DHCP addresses. By RFC definition, these IP addresses are non-routable on the internet (Reference page 3 of the RFC here: http://www.ietf.org/rfc/rfc1918.txt). The hiding address is a valid, routable address.

4. **Description of attack:**

   This "attack" originated by a feature of the networked game, Half-Life. You can instruct this game to perform a server search. It is during this server search that the UPD messages are sent out. If an operator has changed their server port from the default and updated their server details to the master list, you will see requests for that port also.

5. **Attack mechanism:**

   The reason for this detect was the discovery of icmp destination port unreachable messages coming to the external interface of our firewall. Tracing back those addresses uncovers the real stimulous – an employee on an internal machine trying to find a half-life game server on port 1. Http://www.soton.net/quakefiltering.html has an excellent description on how the lookup for these game servers works. Basically, the server operator can choose a port of their liking. Also, when the user issues a request for server, their packet includes the word "ping" and "details" in their server request packets. In the detect above, the word ping is in **green** and the word details is in **purple**.

6. **Correlations:**

Check out the detect at this link: http://www.sans.org/y2k/070200-2000.htm. It is entirely possible that they are also looking at gaming traffic. Looks very similar doesn't it? It sure would be nice to know what those 12 bits of UDP traffic are!

```
12:18:38.419589 212.78.177.4.41216 > XXX.XXX.0.1.33451: udp 12
[ttl 1] (id 41233)
12:18:43.279783 212.78.177.4.41216 > XXX.XXX.0.1.33452: udp 12
[ttl 1] (id 41234)
12:18:48.330052 212.78.177.4.41216 > XXX.XXX.0.1.33453: udp 12
(ttl 2, id 41235)
12:18:53.313641 212.78.177.4.41216 > XXX.XXX.0.1.33454: udp 12
(ttl 2, id 41236)
```

Could a mis-programmed game server be dishing out the wrong information? This is where a strong correlation comes in handy. I may contact David Hoelzer and see if he's interested in this detect!

7. **Evidence of active targeting:**

There is no active targeting in this example. It turned out that the UDP requests to port 1 were actually valid traffic originating from our LAN.

8. **Calculate severity:**

Severity = (Criticality + Lethality) – (System Countermeasures + Network Countermeasures)

(0 + 0) – (1 + 1) = -2

Target Criticality:
The target in this case is external and therefore the criticality is zero.

Attack Lethality:
There is no actual attack in this detect – the icmp error messages were found to be normal in accordance with the given stimulous.

System Countermeasures:
The actual system would be the user's workstation. Workstations are notoriously under-patched and riddled with freeware and warez, so this system gets a 1.

Network Countermeasures:
Even though ingress filtering and ID systems are implemented, egress filtering is not in place; therefore, the user is able to establish any kind of outbound session they desire. From an internal perspective, this is dangerous and should be corrected.

9. **Defensive recommendation:**

Persuant to management approval, the recommendation is to implement egress filtering at the firewall. Prohibiting this type of traffic from leaving the firewall will prevent suspicious-appearing reply packets, not to mention the likely increase in employee productivity.

10. **Multiple choice test question:**

An 'icmp destination port unreachable' error message is always in response to what type of traffic?

A. Requests to closed ports.
B. Requests to open ports that are busy.
C. Requests that generate 'ICMP destination net unreachable' error messages.
D. Requests to find game servers.

Take me to the answer!

Detect 4 – Thoughtful portmapper scan

```
21:04:49.441925 199.120.78.226.39662 > xxx.xxx.xxx.0.111: udp 36 (DF)
                         4500 0040 5e75 4000 f511 8ecf c778 4ee2
                         xxxx xx00 9aee 006f 002c 0353 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
21:04:49.442368 199.120.78.226.39662 > xxx.xxx.xxx.2.111: udp 36 (DF)
                         4500 0040 5e7f 4000 f511 8ec3 c778 4ee2
                         xxxx xx02 9aee 006f 002c 0351 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
21:04:49.442376 199.120.78.226.39662 > xxx.xxx.xxx.3.111: udp 36 (DF)
                         4500 0040 5e7f 4000 f511 8ec2 c778 4ee2
                         xxxx xx03 9aee 006f 002c 0350 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
21:04:49.487002 199.120.78.226.39662 > xxx.xxx.xxx.9.111: udp 36 (DF)
                         4500 0040 5e7f 4000 f511 8ebc c778 4ee2
                         xxxx xx09 9aee 006f 002c 034a 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
21:04:49.487446 199.120.78.226.39662 > xxx.xxx.xxx.11.111: udp 36 (DF)
```

```
                         4500 0040 5e7f 4000 f511 8eba c778 4ee2
                         xxxx xx0b 9aee 006f 002c 0348 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
21:04:49.487454 199.120.78.226.39662 > xxx.xxx.xxx.14.111: udp 36 (DF)
                         4500 0040 5e89 4000 f511 8ead c778 4ee2
                         xxxx xx0e 9aee 006f 002c 0345 5a97 e73c
                         0000 0000 0000 0002 0001 86a0 0000 0002
                         0000 0004 0000 0000 0000 0000 0000 0000
<Snip rest of scan – scanned entire /24>
```

1. **Source of trace:**

   The source of this trace was a tcpdump filter specifically listening for port 111 requests on the /24 belonging to my company.

2. **Detect was generated by:**

   The tcpdump filter mentioned above. This type of filter is very easy to write: tcpdump –x –s1554 –I interface port 111
   This will catch inbound and outbound requests utilizing this port. It will also catch both tcp and udp requests.

3. **Probability the source address was spoofed:**

   Odds are this address was not spoofed, unless the attacker happens to also "own" (in some fashion or another) the box that the replies will be directed to. The attacker needs to see the responses to this traffic in order to make use of any information that is derived from his/her probing.

4. **Description of attack:**

   This detect is a deceptively simple portmapper (port 111) scan. Another portmapper scan – big deal! This particular attack goes beyond your run-of-the-mill scan in that it will attempt to make a connection to port 111 **AND** query the portmapper for a list of services registered with it.

5. **Attack mechanism:**

   Portmapper scans are an old standby because blackhats are betting that a response to a query coming from the internet means that the box is a) vulnerable and b) not closely watched.

   This scan is interesting because it shows a more thoughtful attacker – thoughtful in a scary way! Not only does he/she have the understanding to create static source ports and utilize an oftentimes-neglected protocol (UDP), but we also see the gettport (rpcinfo –p) call included in the scan. Should any host be up and listening on port 111, not only will the attacker learn that portmapper is active; they will also get the complete listing of available RPC services.

   In the detect above, the underlined numbers represent the RPC transaction ID. You can see the number is unchanging through the progression of the scan even though, according to Stevens (TCP/IP Illustrated, Volume I), the XID should change with each new RPC call. The bold, black numbers indicate the RPC version number – in this example, 2 – the current RPC version. The red, underlined portion highlights the RPC program number, which in this case is portmapper. The bolded, blue portion highlights the procedure number - in this case, pmapprocdump – which is the procedure number to list everything registered with portmapper.

   Why go straight to port 111 – it's so obvious, you say? The portmapper port is a well-known (dedicated) port. While the ports for various rpc services are oftentimes consistent (for example, port 2049 for NFS), they really rely on the port number they receive when they register with portmapper.

6. **Correlations:**

   A scan similar to this detect also be found here:
   http://www.sans.org/y2k/022300-2300.htm

```
[**] RPC - portmap-request-mountd [**]
02/22-15:02:53.171471 212.25.118.45:633 -> x.x.x.x:111
UDP TTL:49 TOS:0x0 ID:5046
Len: 64
39 BE 43 FB 00 00 00 00 00 00 00 02 00 01 86 A0 9.C.............
00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 [00 01 86 A5] 00 00 00 01 .............
00 00 00 11 00 00 00 00 ........
```

   This scan is crafted using UDP and querying each host it scans. This scan differs somewhat in that it is querying the portmapper service for a specific service – in this case, mountd (see above in brackets). 000186A5 translates to service number 100005, which is normally assigned by portmapper to the mountd service. You can see a listing of typical service listings in /etc/rpc on a RedHat Linux machine.

   David Hoelzer turned in the following trace last year. Notice the similarity to this detect. In David's trace, the attacker is pinpointing rusersd (see below in brackets) and bypassing portmapper. 000186a2 translates to service number 1000002 – ruserd. http://www.sans.org/y2k/080400.htm

```
13:20:44.293457 194.40.244.193.1757 > firewall.mynet.com.sunrpc: udp 56 (ttl 51, id 4078)
0x0000 4500 0054 0fee 0000 3311 e3c7 c228 f4c1 E..T....3....(..
0x0010 XXXX 3278 06dd 006f 0040 d538 3a8b 471b ..2x...o.@.8:.G.
0x0020 0000 0000 0000 0002 0001 86a0 0000 0002 ................
0x0030 0000 0003 0000 0000 0000 0000 0000 0000 ................
0x0040 0000 0000 [0001 86a2] 0000 0002 0000 0011 ................
0x0050 0000 0000
```

7. **Evidence of active targeting:**

   Absolutely none. I would even venture to say that the attacker is simply scanning multiples of addresses - possibly by the /24 given the closeness of the times.

8. **Calculate severity:**

   Severity = (Criticality + Lethality) – (System Countermeasures + Network Countermeasures)

   (1 + 1) – (4 + 5) = -7

Target Criticality:
Criticality gets a one here. While several critical servers are contained in the /24 that was scanned, none of them utilize the portmapper service.

Attack Lethality:
Considering portmapper is not utilized on this network, the lethality of any exploit would also be 1.

System Countermeasures:
This network contains some systems with up-to-date operating systems and patches and some test machines in need of updating.

Network Countermeasures:
Active IDS and firewall measures are in place. This network is continually scanned for rogue services.

9. Defensive recommendation:

No recommendation necessary for this environment.

10. Multiple choice test question:

What is the current RPC version number?

A. 4
B. 2
C. 1
D. 8

Take me to the answer!

Detect 5 – Smurf/fraggle attempt

```
16:37:26.655264 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:26.656706 B 213.213.44.246.45778 > xxx.xxx.xxx.0.echo: udp 64
16:37:28.095464 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:28.097287 B 213.213.44.246.62501 > xxx.xxx.xxx.0.echo: udp 64
16:37:29.576492 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:29.576596 B 213.213.44.246.3086 > xxx.xxx.xxx.0.echo: udp 64
16:37:31.072414 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:31.072517 B 213.213.44.246.30284 > xxx.xxx.xxx.0.echo: udp 64
16:37:32.448292 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:32.448397 B 213.213.44.246.58777 > xxx.xxx.xxx.0.echo: udp 64
16:37:33.856749 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:33.860266 B 213.213.44.246.48814 > xxx.xxx.xxx.0.echo: udp 64
<snip>
16:37:46.820081 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:46.820185 B 213.213.44.246.26640 > xxx.xxx.xxx.0.echo: udp 64
16:37:48.259856 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:48.259962 B 213.213.44.246.65065 > xxx.xxx.xxx.0.echo: udp 64
16:37:49.716444 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:49.718801 B 213.213.44.246.51843 > xxx.xxx.xxx.0.echo: udp 64
16:37:51.138746 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:51.140180 B 213.213.44.246.7277 > xxx.xxx.xxx.0.echo: udp 64
16:37:52.580790 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:52.580895 B 213.213.44.246.5868 > xxx.xxx.xxx.0.echo: udp 64
16:37:54.052396 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
16:37:54.052500 B 213.213.44.246.59451 > xxx.xxx.xxx.0.echo: udp 64
16:37:55.460303 B 213.213.44.246 > xxx.xxx.xxx.0: icmp: echo request
```

1. Source of trace:

This detect was discovered listening to traffic on the external side of our company firewall. A /24 is maintained there.

2. Detect was generated by:

A tcpdump filter was in place listening specifically for icmp traffic. The filter consisted of: tcpdump –i eth-s1p1c0 icmp. After the "smurfy" part of the traffic was caught, a filter was implemented specifically watching all traffic with the source IP address: tcpdump –i eth-s1p1c0 host 213.213.44.246.

3. Probability the source address was spoofed:

While this address does resolve to a known factor, the odds are that it is spoofed and the source address is actually the intended victim. This particular IP address resolves to a dial-up host in Europe. Likely the victim had irritated someone in a chat or gaming room and the attacker was attempting to DoS their connection to the internet.

4. Description of attack:

This is an example of an attempted smurf/fraggle attack. By sending the icmp and UDP packets to host 0, the attacker hopes to generate a flood of echo reply traffic from all of the hosts in the /24 network to the intended victim, which in this case is the source IP address. Reference CVE-1999-0513 (Smurf) and CVE-1999-0514 (Fraggle) at http://cve.mitre.org.

5. Attack mechanism:

By sending icmp packets to the broadcast address (in this case, host 0), the attacker was attempting to elicit a scenario in which any/all hosts in the /24 network would send their echo reply back to the source address (likely in this example, the “victim”).
This is called a smurf attack. The attacker hopes to maximize the return on investment in this instance – by sending only 1 packet to the 0 broadcast address, he/she hopes to generate a maximum number of responses to the victim. In the dial-up world, this normally leads to a noticeable slowdown on the victim’s connection and the need to shutdown and recreate their internet connection.

The other traffic we are seeing here would cause the same type of backlash. However, it is performed with the UDP protocol and therefore known by a different name – a fraggle attack. Classically, the UDP packets are also directed to the echo port. The attacker will normally choose either the echo (port 7) or chargen (port 19) services, with the piece de resistance being a loop between the echo and chargen ports.

So, we have a 64-byte packet sent to the echo port on a broadcast address – the attacker presumes at least some of the machines in the subnet will respond to the source IP, echoing those 64 bytes of data multiplied by the number of machines that will respond. Why do you suppose the source ports in the trace above are all over the place? I can think of two reasons: A) The attacker has many threads running or B) by utilizing unknown and random source ports, the attacker hopes to continue generating traffic. What happens when a UDP packet hits a closed port? You got it – icmp destination port unreachable error message. My guess would be that the echo port would send this error message BACK to the source IP address and then the game would be over – remember, an icmp error message should never be generated in response to an icmp error message.

6. **Correlations:**

Here’s an example of just a fraggle attack:

http://ricardo.ecn.wfu.edu/~plug/mail_archive/9908/0005.html

```
[root@rainbow log]# /usr/sbin/tcpdump -c30 -f \!host rainbow.galax.ls.net
tcpdump: listening on eth0
23:13:26.539631 203.108.168.25.23725 > 206.228.251.255.echo: udp 64
23:13:28.886195 203.108.168.25.14534 > 206.228.251.255.echo: udp 64
23:13:29.189371 203.108.168.25.35696 > 206.228.251.255.echo: udp 64
```

Here’s a mixed smurf/fraggle example, from a Cisco router log:

http://www.incidents.org/archives/y2k/122399.htm

```
Dec 22 16:15:26: %SEC-6-IPACCESSLOGDP: list Internet denied icmp
172.20.20.1 -> 255.255.255.255 (8/0), 1 packet
Dec 22 16:16:26: %SEC-6-IPACCESSLOGDP: list Internet denied icmp
172.20.20.2 -> 255.255.255.255 (8/0), 24 packets
Dec 22 16:16:56: %SEC-6-IPACCESSLOGP: list Internet denied udp
172.20.20.3(21820) -> 255.255.255.255(19), 1 packet
Dec 22 16:26:26: %SEC-6-IPACCESSLOGDP: list Internet denied icmp
172.20.20.4 -> 255.255.255.255 (8/0), 3 packets
Dec 22 16:27:26: %SEC-6-IPACCESSLOGDP: list Internet denied icmp
172.20.20.5 -> 255.255.255.255 (8/0), 4 packets
```

Check out this link for a program called Papasmurf – it is likely what was used to create this attack. It utilizes both udp and icmp, nicely condensed into one beast by the original author of the smurf.c and fraggle.c programs. ftp://ftp.technotronic.com/denial/papasmurf.c

7. **Evidence of active targeting:**

The intended target in this detect is actually the source IP address. However, the attacker should have done a little more homework in that their target never received any replies from our network. Directed broadcasts are turned off at the perimeter and so there were no replies sent back to the source address. The attacker could have searched http://www.netscan.org to determine valid broadcast networks. (However, this is not the intended nature of the site – Netscan’s goal is to help “clean up” by announcing the top directed broadcast offenders and by allowing a system administrator to check their own network.)

8. **Calculate severity:**

Severity = (Criticality + Lethality) – (System Countermeasures + Network Countermeasures)

(0 + 0) – (N/A + 5) = -5N/A

Target Criticality:
As the intermediate host of this attack, the target criticality is zero because our perimeter routers do not forward directed broadcasts.

Attack Lethality:
If the attacker were to stumble across a network that allows responses to these directed broadcasts, lethality would be high. However, in this case, it also earns a zero.

System Countermeasures:
As our systems were intended to be part of the attacker’s tool and not the victim, their countermeasures are negligible in this scenario. The real defense lies in the perimeter of the network, thus on to network countermeasures.

Network Countermeasures:
Network countermeasures earn a 5 for this attack – directed broadcasts are not forwarded.

9. **Defensive recommendation:**

Defenses performed as they should. Work with network administrators to perform periodic assessment testing (assure compliance for any new hardware, for example).

10. **Multiple choice test question:**

What is the main difference between a smurf and fraggle attack?

A. The fraggle attack utilizes fragmented packets to overwhelm the intended victim.

B. The smurf attack overwhelms its intended victim with UDP packets.
C. Smurf utilizes ICMP and fraggle utilizes UDP.
D. Smurfs are blue and fraggles are brown.

Take me to the answer!

DESCRIBE THE STATE OF INTRUSION DETECTION: "STICK"

Stick made its debut around March 16, 2001. According to the author (Coretez Giovanni), the tool was designed for "limited stress test capability".

From Giovanni's viewpoint, the true weakness in intrusion detection systems is the human. In his white paper, Fun with Packets – Designing a Stick (see References), he accurately describes staffing arrangements for most intrusion departments. He talks about how companies maintain just enough staff to cover the bases for an average day's worth of detects. A good comparison for this model would be the way airlines sell tickets – oversell enough to hedge the bet that enough people will not make the flight, yet the flight will still be full. Since this theory is based on manpower, Giovanni's thought is that by overwhelming the analyst with enough packets matching valid signatures, the attacker will be able to obfuscate his/her real purpose.

Stick is based on rules lists that are maintained for Snort. (See Detect 2 for a description of Snort and a link for the rules list). The most basic description of how Stick operates is that it creates highly randomized, anomalous-appearing traffic and bombards the desired host or range of hosts. It creates this traffic based on the signatures found in the Snort rules list. The victim will certainly be able to determine that they are under attack, based on the influx of anomalous packets; however, the test will be finding the proverbial "needle in the haystack" or the true attack or reconnaissance traffic.

Once downloaded, Stick is installed via a shell script. A default rules list even comes with the package. The source is comprised of lex and yacc and, in the author's own words, is really "crappy code". True to form for most hack tools, the documentation is less than desirable and misspellings abound. However, the author does state in the README that this tool should not "be connected to the internet when running" and that the software is "for the anaylsis of intrusion detection". Bases covered – check.

Once the tool has been pieced together, the command line syntax is fairly straightforward, although I had difficulty getting all of the options to work correctly. I was able to create packets with specific source IP addresses or randomized source IP addresses. However, I could only get the option for a single target to work properly. Attempting to create a range of targets did generate randomized output on the Stick terminal but the targets running both tcpdump and Snort did not capture any data. The options are fairly straightforward:

From the Stick README:
sH xxx.xxx.xxx.xxx
This is a single source IP that the IP headers should use as the source.

sC xxx.xxx.xxx.0
This is a single Class C space that has a simple random last octet.

sR aaa.aaa.aaa.xxx aaa.aaa.aaa.yyy
This is a sub class C range! ex. ./stick sR 192.168.128.2 192.168.128.55

dH xxx.xxx.xxx.xxx
This is a single desination IP for the IP header.

dC xxx.xxx.xxx.0
This is a single Class C space that has a random last octet.

dR aaa.aaa.aaa.xxx aaa.aaa.aaa.yyy
This is a sub class C range! ex. ./stick sH 10.0.0.1 dR 192.168.128.2 192.168.128.55

If no source is given then a random address from 0.0.0.0-255.255.255.255 will
be used and a destination of 10.0.0.1.

At this point in the README, the author reiterates: "This is really crappy code, I don't really know C very well but I use to be an Ada programmer. Ada is pretty dead ..."

I don't feel too bad about not getting all of the options to work correctly.

To test the effectiveness of Stick, I decided to try running an Nmap scan while having Stick direct anomalous, randomized traffic at the desired target. My question was how the output would compare after being gathered by Snort. Since I was unable to get the multiple targets feature to work, I concentrated on one specific host, utilizing Stick to swamp the target with packets matching Snort vulnerability signatures while at the same time running a TCP Nmap scan of the target. On the victim host, I ran a copy of Snort in the packet dump mode to watch the traffic and see if it would hiccup while attempting to parse the influx of data. Snort was also running in logging/alerting mode in order to capture the anomalous packets to file. The data scrolling across the Stick terminal looks similar to the following:

```
sending rule 643
sending rule 456
sending rule 26
sending rule 424
sending rule 87
sending rule 1047
sending rule 1055
sending rule 303
sending rule 913
sending rule 252
sending rule 552
sending rule 676
sending rule 879
sending rule 280
sending rule 577
sending rule 809
sending rule 300
sending rule 569
```

Interestingly enough, there are 1072 alert rules in the vision.txt file that comes with the Stick installation and when the all of the terminal data (like above example) was sorted in an Excel spreadsheet, all the rules were accounted for – numbers 1 through 1072. Number of uses of each rule varied. A quick 'grep –c alert vision.txt' (count matching hits of word 'alert') confirmed this number.

Once these programs were started, I let Stick run for approximately 1 minute while I waited for the Nmap scan to complete. During this time period, over 7,500 alerts were generated and logged by Snort. Due to the nature of the test environment, network lag was negligible. Actually using the tool over the internet would introduce more latency into the picture and probably slightly affect the total number of alerts received by the target/s.

As you can imagine, at the conclusion of this test, the Snort log directory was full of over 2,440 sub-directories – a new sub-directory being created for each new individual IP address that Snort detected. This was quickly calculated by running wc –l (word count – count number of lines) against an ls –a (list – all details) of the Snort log directory. At this point, I quickly realized that Giovanni was correct in his theory of overwhelming the analyst. As an example, the readout of this listing looks something like the following, with the Nmap scan originating from the xxx.xxx.xxx.101 address:

```
drwx------   2 root    root       4096 April 30 20:08 100.25.140.38
drwx------   2 root    root       4096 April 30 20:08 142.89.168.81
drwx------   2 root    root       4096 April 30 20:08 180.177.70.71
drwx------   2 root    root       4096 April 30 20:08 204.107.237.46
drwx------   2 root    root       4096 April 30 20:08 206.198.243.31
drwx------   2 root    root       4096 April 30 20:08 28.17.3.18
drwx------   2 root    root       4096 April 30 20:08 62.160.251.64
drwx------   2 root    root       4096 April 30 20:08 112.165.148.0
drwx------   2 root    root       4096 April 30 20:08 112.196.245.62
drwx------   2 root    root       4096 April 30 20:08 120.183.173.54
drwx------   2 root    root       4096 April 30 20:08 123.48.81.4
drwx------   2 root    root       4096 April 30 20:08 128.117.240.180
drwx------   2 root    root       4096 April 30 20:08 13.55.54.14
drwx------   2 root    root       4096 April 30 20:08 138.179.185.36
drwx------   2 root    root       4096 April 30 20:08 16.83.103.8
drwx------   2 root    root       4096 April 30 20:08 208.0.9.26
drwx------   2 root    root       4096 April 30 20:08 xxx.xxx.xxx.101
drwx------   2 root    root       4096 April 30 20:08 220.19.116.80
```

<snip approx. 2,400 lines>

Being the dedicated Snort fan that I have recently become, I happened to know of a tool called SnortSnarf. This particular tool is a set of perl scripts designed to take the output from Snort and customize it into easily readable html pages. It can be downloaded from Silicon Defense: http://www.silicondefense.com/software/snortsnarf/index.htm

Running SnortSnarf against the data accumulated by Snort during the "Stick episode", I was able to produce html pages that sorted all the anomalous traffic into pretty tables with links and stats. Examples of SnortSnarf output can be seen on their website here: http://www.silicondefense.com/software/snortsnarf/example/index.html

Once these html pages were created and viewed, it was easy to discern the Stick traffic from the "true" attack traffic (the Nmap scan). SnortSnarf, due to its statistical nature, created polite columns and rows that highlighted the now obvious Stick pattern. The table below will most easily illustrate this fact.

Signature	**# Alerts**	**# Sources**	**# Destinations**
TCP **UAP*SF scan	11	11	1
TCP **U***S* scan	11	11	1
TCP **UA**SF scan	18	18	1
TCP ******** scan	437	437	1
TCP **U***** scan	449	449	1
TCP ******S* scan	1485	1	1

Keeping a few thoughts in mind:

- I was unable to work the option for running Stick against multiple hosts.
- I had cleaned out all log entries before starting the program; therefore eliminating other anomalous traffic from clouding the view.
- This is only a piece of the entire directory listing.

With these considerations in mind, it is fairly easy to point to the obvious in the table above – it seems that the packet signature containing only 1 source address would be deserving of further observation. True to suspicion, this source turns out to be the IP address of the machine that I was running the Nmap scan from.

After running these tests with Stick, I have to admit that I agree wholeheartedly with the author – the real weakness in an intrusion detection system is the human. Your IDS can collect alerts until the end of time (or it runs out of space!), but the true test is having the knowledge, tools, and patience to sort through the hay to find that needle. Add the option to direct traffic to multiple hosts and an attacker can completely overwhelm the analyst. Another idea I could have used was the Nmap feature of adding spoofed source addresses to my scan. Add both of those thoughts to the every-day alerting and logging that is already taking place…. Quite frankly I'm surprised (and glad) we aren't seeing more of this activity taking place. It would take a full-scale cooperative effort to stop a well-organized, directed attack of this nature.

Stick highlights both facets – it is an approach to IDS technology (the human as the weakest link) and it is an attack methodology (regardless of what the author states in his README!).

ANALYZE THIS - REPORT

GIAC Enterprises:

Thank you for the opportunity to assist your company with its security needs.

The following information was extracted from approximately one month's worth of intrusion detection system information that your security team provided my company.

Due to the volume of data, every effort was made to extrapolate the most sensitve items into the following two tables. Explanations pertaining to each unique item follow the tables. The items that should be reviewed at your earliest convenience are in red.

While reviewing the findings below, please don't hesitate to contact me with any questions. I look forward to your reply.

Regards,
Becky Pinkard

Table 1: Name of Snort comment and count of comment

Intrusion Detection System Comment	Count Of Comment
UDP SRC and DST outside network	364,249
Spp_portscan	18,812
Watchlist 000220 IL-ISDNNET-990517	13,231
SYN-FIN scan!	11,607
Possible RAMEN server activity	6,970
NMAP TCP ping!	2,407
SNMP public access	1,147
TCP SRC and DST outside network	972
SMB Name Wildcard	608
Queso fingerprint	425
Attempted Sun RPC high port access	409
WinGate 1080 Attempt	381
Tiny Fragments - Possible Hostile Activity	229
Null scan!	100
Watchlist 000222 NET-NCFC	84

UDP SRC and DST Outside Network

This comment appears to be related in part to a fixed, multicast address (part of the MBONE network) that your company utilizes (224.2.127.254:9875). To help reduce the number of alerts that your intrusion detection system finds, you can fine-tune your signature files to ignore this traffic.

There is also port 137 (NetBIOS SMB service) scanning activity from this host, 169.254.67.123. Check that machines are secure against nbtstat enumeration. Make sure Snort intrusion variables include the proper networks for its internal and external networks.

Recommended action: Prohibit nbtstat enumeration outside your network. Reinforce that Snort is alerting on the proper internal and external networks.

Spp_portscan

This comment pertains to any activity that generates a "scan" notification. Snort is programmed to alert on scanning activity when it detects x number of requests in y number of seconds. Fine-tuning this rule is important in narrowing down false positives.

Recommended action: Fine-tune portscanning ruleset to minimize notifications.

Watchlist 000220 IL-ISDNNET-990517

It appears that your security team already has a watch in place for traffic coming from this network: 212.179.0.0. According to internic records, this class B belongs to an ISDN network in Israel.

The traffic appears to be Napster and Gnutella (file sharing software) related. Policy should be in place to regulate the use of these types of applications. Hacking your network may not be necessary if some of the employees have their hard drives shared out on the Gnutella network, for example. A more in-depth explanation of this software can be found at these links: http://www.gnutellanews.com/information/what_is_gnutella.shtml
http://www.napster.com

Recommended action: Create/enforce policy to prohibit Gnutella, Napster.

SYN-FIN scan!

The majority of these records are comprised of the following three scans. These are reconnaissance-gathering techniques. Up-to-date patches and OS' on ftp and dns servers will ensure that these scans remain that way.

* 128.61.136.233 scanning for FTP (port 21)
* 130.234.184.112 scanning for FTP (port 21)
* 211.248.112.67 scanning for DNS (port 53)

Recommended action: Ensure ftp and dns servers are fully patched. Turn off unnecessary services.

Possible RAMEN server activity

A readily identifying characteristic of the Ramen worm is the fact that after a machine is infected, it creates a web server on port 27374 in order to provide copies of itself. It is also possible over an entire class B that some legitimate traffic utilizing port 27374 has activated this signature.

Recommended action: Check host 111.111.253.12 to determine if this machine belongs to a security analyst who is scanning your network for port 27374. Check host 111.111.201.146 for possible Ramen compromise. Update to LPRng RH 7.0 and wu-ftpd RH 6.2 on any RedHat 6.2 and 7.0 Linux machines.

NMAP TCP ping

This in and of itself is not as critical. Nmap is a scanning tool that sends a ping packet to each host in its default scanning mode.

What is interesting about this comment is that of the 2,400+ records, all but 18 come from one of your own hosts – 111.111.70.38. There are a couple of reasons for this type of traffic from an internal host.

This machine belongs to a security analyst for your company, in which case they should have permission to perform this type of scan.
-OR-
The host has been compromised and is now being used to scan the rest of your network.

Recommended action: Track down host 111.111.70.38 and verify its function.

SNMP public access

This means that there are possibly some machines on your network that still have a default SNMP community string of “public”. If this is true it will allow intruders read access to the MIB information contained in the following machines:

128.46.156.197 accessed 111.111.100.143, 111.111.100.99, and 111.111.100.206 over a period of several days. Possibly an analyst utilizing a remote machine?

There is one instance of 128.183.38.30 accessing 111.111.154.26 by public community string on port 161.

111.111.70.42 and 111.111.111.156 both access 111.111.50.154 on port 161 utilizing a public community string.

Recommended action: Check all servers being accessed by external users ASAP. Change default string to a secure string.

TCP SRC and DST outside network

This rule has alerted on some AOL and IRC traffic and miscellaneous other items. The rule seems too broad to categorize definitive traffic patterns, other than neither source nor destination IPs are contained with the internal/external variable definitions for Snort.

Recommended action: Confer with security team to tighten up rule and clarify alert.

SMB Name Wildcard

From http://www.whitehats.com - this rule “…specifies NetBIOS traffic coming from *outside* of your local network. Allowing netbios traffic over public networks is usually very

Recommended action: Deny external requests to port 137.

Queso fingerprint

“Fingerprinting” a system means that a tool will send unusual TCP packets to a remote host in an attempt to determine it’s operating system type. Despite the standards created by the RFC’s, this technique is possible due to differences in the way vendors’ implement their TCP/IP stacks.

The client sent a TCP fingerprint query whose signature matched that of the popular 'queso' scanner (see http://www.apostols.org/projektz/queso/ for more information).

It’s possible that some of the packets captured by this rule are false positives. Much of the traffic shows to be destined to port 6346 (Gnutella), which leads to the assumption that you have several users running this software on their workstations.

Recommended action: Fingerprinting a system is one of the first steps in reconnaissance. Up-to-date system patches will prevent an attacker from being able to proceed further with any information they might uncover. Use policy to clarify use of Gnutella in monitored-networks.

Attempted Sun RPC high port access

This rule triggers on access to port 32771, which is rpc.statd and also another place for portmapper to listen on SunOS machines. It is highly probable that a majority of these alerts are false positives based on a source port of 4000 (ICQ) and source IPs that resolve to assorted machines in the icq.aol.com domain.

Recommended action: To be on the safe side, check the following 4 hosts. If any of the hosts are SunOS machines utlizing this service, specifically deny external access to 32771.

111.111.223.254
111.111.105.115
111.111.224.230
111.111.223.70

WinGate 1080 Attempt

This rule alerts on WinGate because it is the name of a proxy server and according to RFC 1928 (SOCKS Protocol Version 5):

"When a TCP-based client wishes to establish a connection to an object that is reachable only via a firewall (such determination is left up to the implementation), it must open a TCP connection to the appropriate SOCKS port on the SOCKS server system. The SOCKS service is conventionally located on TCP port 1080."

There are several addresses in this traffic that could also be actual scans for SOCKS or Wingate. Also, Undernet.Org and IRC servers will scan for misconfigured proxy services before allowing connections from a user.

The worst thing about having a misconfigured proxy server is that others will search it out to bounce their own traffic through. This is an attempt to anonymize their sessions by having their traffic appear as though it were coming from your servers. You will oftentimes see traffic coming from countries that try to practice internet censorship.

Recommended action: If proxies are not utilized in your organization, download a proxy scanner and scan your own network looking for responding machines. Turn off any rogue services. Trickscan.c is an example of a wingate scanner. It can be downloaded here: http://packetstorm.securify.com/UNIX/scanners/trickscan.c

Tiny Fragments - Possible Hostile Activity

Marty Roesch is the designer of Snort and he has this to say about "tiny fragments":

"The minfrag option checks the size of IP fragments. If a fragment is smaller than a set threshold value, an alert is generated. The concept here is that no commercial network equipment that I've ever heard of fragments their traffic to less than 256 bytes, and so anything you see below that threshold value is probably *very* suspicious. FYI, nmap and fragrouter fragment to either 8 or 24 byte fragments."

Both Nmap and Fragrouter are programs designed to craft packets in such a way that they elude intrusion detection systems.

Based on the small number of alerts, your minfrag option appears to be adjusted nicely. There were three obvious hosts scanned by these types of programs:
111.111.98.117
111.111.97.231
111.111.223.42

Recommended action: Periodically test your minfrag rule to verify consistency and accuracy of alerts.

Null scan!

A null scan indicates that a packet was received on a port and did not have any flags set. For TCP, this is almost always indicative of a crafted packet. Two tools come to mind that are capable of creating these packets – Nmap and Hping2.

Recommended action: Scans are a dime a dozen. Unfortunately, you can't completely brush this type of activity under the carpet, but you can minimize labor by keeping boxes patched and following-up only on the persistant or noisy scanners with periodic in-depth checks.

Watchlist 000222 NET-NCFC

Your security team has placed the 159.226.0.0 network, belonging to the Computer Network Center Chinese Academy of Sciences, under a watch - presumably for previous suspicious traffic. Currently, there have been approximately 66 alerts for traffic destined to various SMTP servers (see list in Table 2). There is also some miscellaneous traffic to one particular host, 111.111.60.17.

Recommended action: Confer with your security team to determine cause for watch and if current activity mandates continuance. Check activity to host 111.111.60.17.

Table 2: Name of Common Service and Enumeration of Possible Hosts

20 (FTP – Data)	22 (SSH)	25 (SMTP)	80 (HTTP)
111.111.130.91	111.111.60.38	111.111.253.41	111.111.160.109
111.111.221.54	111.111.60.11	111.111.253.43	
111.111.217.118	111.111.221.54	111.111.6.47	
		111.111.253.42	
		111.111.253.43	
109 (POP2)	**110 (POP3)**	**119 (NNTP)**	**443 (HTTPS)**
111.111.209.186	111.111.220.18	111.111.218.86	111.111.5.29
111.111.218.38	111.111.6.39	111.111.225.2	111.111.253.112
111.111.165.129	111.111.6.44		111.111.98.198
111.111.60.8			

The servers above were grouped into this table because Snort generated activity alerts for the reported services. While the services are normal in the course of every-day computer use, further inspection is necessary because the rules of Snort alert for external traffic utilizing services on what it considers to be internal, or protected, machines. If these rules are superfluous, they can be removed or redesigned in the ruleset.

The services listed in the table above should be verified for each instance of occurrence. Any services not required for production should be removed. Policy must be in place to prohibit users from running non-production services from their own server or workstation.

ANALYZE THIS – PROCESS

I have to admit that I am a very visual person, so being confronted with this amount of information was somewhat overwhelming. I took a meticulous (translation – slow) approach to organizing the data into coherent files, re-naming and re-grouping. This also helped me get a feel for the type of information gathered, since the files names did not offer up much by way of description.

During the review process, I discovered that some of the files were identical and some of the files were from the year 2000. In the best interest of providing an accurate assessment for the given time period, this data was removed from the analyzation process.

The following duplicate files were discarded:

- SnortA36.txt (identical to SnortA35.txt)
- UMBCNI31.txt (identical to UMBCNI25.txt)
- UMBCNI5.txt (identical to UMBCNI3.txt)

Comparison testing was performed with TextPad 4.4.0 (http://www.textpad.com):

- *Compare:*
  C:\GCIA_files\FastAlrt\SnortA35.txt (4492808 bytes)
  with:
  C:\GCIA_files\FastAlrt\SnortA36.txt (4492808 bytes)

  The files are identical

- *Compare:*
  C:\GCIA_files\Portscans\UMBCNI25.txt (2115215 bytes)
  with:
  C:\GCIA_files\Portscans\UMBCNI31.txt (2115215 bytes)

  The files are identical

- *Compare:*
  C:\GCIA_files\Portscans\UMBCNI3.txt (4956801 bytes)
  with:
  C:\GCIA_files\Portscans\UMBCNI5.txt (4956801 bytes)

  The files are identical

I also noticed that in the OOS log files, there were 4 files from the previous year. These were discarded and not included in the tallies. The discarded OOS files were:

- February 10, 2000
- February 11, 2000
- February 12, 2000
- March 8, 2000

In the Alert log files, there were 3 files from the previous year (2000). These files were also removed. The discarded Alert files were:

- February 12, 2000
- February 21, 2000
- March 9, 2000

In the Scan files, there were also 3 files from the previous year (2000). These files were removed. The discarded Scan files were:

- February 11, 2000
- February 21, 2000
- March 9, 2000

After cleaning these Scan files up, I joined them all together and attempted to run SnortSnarf against the 1 file (size 66.7 MB or 1,044,415 lines). I have a 566 Mhz, 384 MB RAM, x86 machine at my desk and it ran out of memory in about 5 minutes. Back to the drawing board…

Second attempt was to split the file into 4 parts and run that – SnortSnarf continued to die, somewhat gracefully, with an "Out of Memory!" error.

Seeing that my hardware wasn't going to miraculously get any better and I was loath to split the file into even smaller pieces, I changed gears. I decided to pull out Lance Spitzner's Firewall 1, Access database moves and throw them at the log file. I had analyzed log files before using this method, so it was only a matter of remembering the semantics. Check out Lance's work at his site: http://www.enteract.com/~lspitz/. He has compiled some incredible information. The logger link is towards the bottom of this page. I then utilized a combination of the Access database and Excel spreadsheets to massage the data.

I also tried an "outside the envelope" process that I learned about in the SANS course. It is called Traffic Anaylsis. This was defined to me as stepping away from the packet content and looking at the traffic from different angles. For example, using a large, white-erase calendar, I mapped the days where each type of log file was accumulated. From this viewpoint, I was able to quickly determine that there were only 3 days in the accumulated data in which an alert, log, and OOS file were all generated on the same day. Forgiving the loss of data through power failure or disk capacity, why would this happen? My guess-timate is that synonymous data occurred on those 3 days – I will confess to running out of time before I could track this down to my satisfaction. Days where all 3 occur – Jan 31, Feb 24, Mar 1.

As a side note, I also noticed that analysis and sensor stations appear to be separate, similar to the SHADOW IDS concept. (Reference http://www.nswc.navy.mil/ISSEC/CID/step.htm for more details on the inner-workings of SHADOW.) This is based on the assumption that the OOS (out-of-spec) packet files were automatically generated during a period of more detailed analysis, such as is usually done at an analysis station.

Appendix A

Answers to Multiple Choice Questions:

Detect 1:

If an open port receives an initial packet with the fin flag set, how is it supposed to process the packet - based on the RFC specifications?

A. Respond with a reset.
B. Silently discard the packet.
C. Respond with a syn-ack.
D. Drop the packet and respond with a fin-ack.

Answer: B
According to RFC 793 specifications, an open port should silently discard an initial packet with the fin flag set.

Detect 2:

What part of the trace above highlights this as an abnormal web request?

A. The TTL is too low.
B. The ack and push flags should not occur simultaneously.
C. The 'HEAD' command should be used instead of the 'GET' command.
D. The unicode characters %c0%af.

Answer: D
The slash should normally show up in your tcpdump output in its ASCII format: /

Detect 3:

An 'icmp destination port unreachable' error message is always in response to what type of traffic?

A. Requests to closed ports.
B. Requests to open ports that are busy.
C. Requests that generate 'ICMP destination net unreachable' error messages.
D. Requests to find game servers.

Answer: A
According to RFC 792 specifications, an icmp destination port unreachable error message should be generated in response to any probe packets directed at a closed port.

Detect 4:

What is the current RPC version number?

A. 4
B. 2
C. 1
D. 8

Answer: B
According to RFC 1057, the current RPC version is 2.

Detect 5:

What is the main difference between a smurf and fraggle attack?

A. The fraggle attack utilizes fragmented packets to overwhelm the intended victim.
B. The smurf attack overwhelms its intended victim with UDP packets.
C. Smurf utilizes ICMP and fraggle utilizes UDP.
D. Smurfs are blue and fraggles are brown.

Answer: C
And the papasmurf.c attack utilizes both the smurf (ICMP) and fraggle (UDP) attacks.

Appendix B

References:

1. Northcutt, Stephen, et al. Intrusion Signatures and Analysis. Indianapolis: New Riders Publishing, 2001. 20-21

2. RFC 793 – page 64, TCPs response to various flags
http://www.ietf.org/rfc/rfc0793.txt

3. Correlation for port 27015 traffic and udp traffic
http://old.caida.org/Papers/AIX00/

4. Half-life ports listing
http://user.cs.tu-berlin.de/~hanspete/dagh-l/FAQ/technik.html

5. RFC 792 – page 4, Internet Control Message Protocol
http://www.ietf.org/rfc/rfc0792.txt

6. RFC 1057 – RPC: Remote Procedure Call Protocol Specification, Version 2
http://www.ietf.org/rfc/rfc1057.txt

7. Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley, 1994. 461 - 467.

8. The Portmap Utility
(watch the wrap)
http://docsun.cso.uiuc.edu/cgi-bin/nph-dweb/ab2/coll.45.4/ONCDG/@Ab2PageView
/28914

9. The Latest In Denial Of Service Attacks: "Smurfing"
Description And Information To Minimize Effects
http://www.pentics.net/denial-of-service/white-papers/smurf.cgi

10. Smurf Amplifier Registry
http://www.powertech.no/smurf/

11. Netscan.Org
http://netscan.org/

12. Better Network Security through Peer Pressure
http://www.securityportal.com/cover/coverstory19990531.html

13. Possible DoS (fraggle) Problem
http://www.www-arc.com/sara/cve/Possible_DoS_problem.html

14. Incidents Org
http://www.incidents.org

15. Max Vision’s arachNIDS database
http://www.whitehats.com/ids/index.html

16. Snort - The Lightweight Network Intrusion Detection System
http://www.snort.org/

17. Patch Available for “File Permission Canonicalization” Vulnerability
http://www.microsoft.com/technet/security/bulletin/ms00-057.asp

18. http://www.sans.org/y2k/041901.htm

19. Patch Available for “Web Server Folder Traversal” Vulnerability
http://www.microsoft.com/technet/security/bulletin/ms00-078.asp

20. RFC 1918 - Address Allocation for Private Internets
http://www.ietf.org/rfc/rfc1918.txt

21. Blocking Half-Life, Quake and Quake II Multiplayer Games Http://www.soton.net/quakefiltering.html

22. http://www.sans.org/y2k/070200-2000.htm

23. http://www.sans.org/y2k/022300-2300.htm

24. http://www.sans.org/y2k/080400.htm

25. Common Vulnerabilities and Exposures List
http://cve.mitre.org

26. http://ricardo.ecn.wfu.edu/~plug/mail_archive/9908/0005.html

27. http://www.incidents.org/archives/y2k/122399.htm

28. Papasmurf.c v5.0 by TFreak
ftp://ftp.technotronic.com/denial/papasmurf.c

29. Netscan Org
http://www.netscan.org

30. Stick can be downloaded here:
http://packetstorm.securify.com/distributed/stick.tgz

31. Fun with Packets: Designing a Stick
http://www.eurocompton.net/stick/

32. "Stick"- A Potential Denial of Service Against IDS Systems
http://xforce.iss.net/alerts/advise74.php

33. Newsgroup comments from Stick author
http://archives.linuxbe.org/arch055/0584.html

34. "Stick"- A New Denial of Service Against IDS Systems
http://www.securiteam.com/securitynews/_Stick_-_A_New_Denial_of_Service_Against_IDS_Systems.html

35. Security experts scramble to stop new hacking tool
http://news.zdnet.co.uk/story/0,,s2085062,00.html

36. Denial of service warning for network security tool
http://www.theregister.co.uk/content/8/17660.html

37. 'Stick' causes an anti-hacking panic
http://www.zdnet.com/zdnn/stories/news/0,4586,2697767,00.html

38. RFC 1928 – SOCKS Protocol Version 5
http://www.ietf.org/rfc/rfc1928.txt

39. SnortSnarf Download from Silicon Defense
http://www.silicondefense.com/software/snortsnarf/index.htm

40. TextPad 4.4.2 Home Page:
http://www.textpad.com/

41. Logger for Check Point FireWall-1
http://www.enteract.com/~lspitz/

42. Building a Network Monitoring and Analysis Capability, Step by Step
http://www.nswc.navy.mil/ISSEC/CID/step.htm

43. Great Snort presentation – (correlate UMBC with log data headers)
http://userpages.umbc.edu/~robin/Presentations/Snort/Snort_FINAL_files/frame.htm

44. Google – Don't leave home without it
http://www.google.com

Other students whose practicals were invaluable to me during the course of writing this practical – my thanks goes out to them and everyone else who has gone down this GCIA path before me for their hard work and expertise.

http://www.sans.org/y2k/practical/Lenny_Zeltser.htm

http://www.sans.org/y2k/practical/Marc_Bayerkohler_GCIA.html

http://www.sans.org/y2k/practical/Fred_Portnoy_GCIA.doc

http://www.sans.org/y2k/practical/Teri_Bidwell_GCIA.doc

http://www.sans.org/y2k/practical/JoanneTreurniet.html

http://www.sans.org/y2k/practical/Paul_Asadoorian_GIAC.doc

http://www.sans.org/y2k/practical/Robert_Clark_GCIA.doc