Global Information Assurance Certification Paper

GIAC INTRUSION ANALYST

Certification Practical V2.9

SANS Baltimore, Maryland 2001

James A Conz, Jr

(this page intentionally left blank)

ASSIGNMENT 1 – NETWORK DETECTS

Network Detect 1: RPC Portmap Requests

TRACE A:

```
[**] RPC portmap request rstatd [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/29-15:09:48.928810 202.130.248.188:875 -> XXX.XXX.XXX.12:111
UDP TTL:49 TOS:0x0 ID:58868 IpLen:20 DgmLen:84
Len: 64
[Xref => http://www.whitehats.com/info/IDS10]
```

TRACE B:

```
15:09:48.928810 202.130.248.188.875 > XXX.XXX.XXX.12.111:  [udp sum ok]
udp 56 (ttl 49, id 58868, len 84)
0x0000   4500 0054 e5f4 0000 3111 63e3 ca82 f8bc        E..T....1.c.....
0x0010   XXXX XX0C 036b 006f 0040 9be4 6407 ae72        .....k.o.@..d..r
0x0020   0000 0000 0000 0002 0001 86a0 0000 0002        ................
0x0030   0000 0003 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0001 86b8 0000 0001 0000 0011        ................
0x0050   0000 0000                                      ....
15:09:49.030339 XXX.XXX.XXX.12.111 > 202.130.248.188.875:  [udp sum ok]
udp 28 (ttl 64, id 3052, len 56)
0x0000   4500 0038 0bec 0000 4011 2f08 XXXX XX0C        E..8....@./.....
0x0010   ca82 f8bc 006f 036b 0024 a98f 6407 ae72        .....o.k.$..d..r
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 0000                            ........
```

1. *Source of the trace:*

   These traces were collected on a client network. Note that the traces have been sanitized (including the hex output).

2. *Detect was generated by:*

   This detect was generated by the Snort Intrusion Detection system running in packet logger mode. The data collected was later processed through snort again in NIDS mode, generating the alert in Trace A.

3. *Probability the source address was spoofed:*

   In that the attacker is attempting to gather information about the remote system (in this case, for a possible future exploit), and would thus require some sort of response from the remote system, it is unlikely that the source IP address is spoofed.

4. *Description of attack:*

This attack is an information gathering effort – the RPC portmapper can provide information about the various RPC services that are available on a given host. Many of these services have been the source of vulnerabilities. This type of attack is currently a candidate for inclusion in the CVE list, referenced as CAN-1999-0632.

5. *Attack mechanism:*

   This attack works by querying the remote system to see if the RPC Portmapper is running. A UDP packet is sent to port 111 on the remote system. If the portmapper is running, and the packet is able to reach its destination, a response is sent back to the attacker indicating success.

   The RPC queries can be generated using the program "rpcinfo" that ships with most flavours of UNIX. The example below illustrates a successful query. Note the program number 100000 – this is the portmapper service itself:

```
echelon$ rpcinfo -u 192.168.99.11 100000
program 100000 version 2 ready and waiting
echelon$
```

   An unsuccessful query will occur when the portmapper is not enabled or reachable, or the program number is not available. In the following example, it is not available:

```
echelon$ /usr/local/sbin/rpcinfo -u 192.168.99.77 100000
rpcinfo: RPC: Port mapper failure - RPC: Timed out
program 100000 is not available
echelon$
```

   In this detect, the target host does indeed respond, indicating that the queried program (bolded text 0001 86a0 in trace is equal to decimal 100000, or portmapper) is available. The next step would be to query portmapper for a list of registered RPC programs. The attacker can then search the list for an exploitable vulnerability to complete the compromise.

6. *Correlations:*

   This type of scan is common on the Internet (see http://www.whitehats.com/info/IDS10).

   However, the source host 202.130.248.188, was reported by the Computer and Network Security Officer at the University of Auckland, New Zealand on May 23, 2001 as being the source of a port scan searching for DNS server (port 53 scan)[†].

   This host belongs to a Chinese network block, and is referenced in DNS by the hostname `idc.sta.net.cn`. It is also, at the time of this writing, running a web server: http://idc.sta.net.cn. This site is protected by username and password, and is in Chinese.

7. *Evidence of active targeting:*

   Given the fact that this host has been the source of many port scans, it seems likely that the scan was part of a much larger, automated sweep.

8. *Severity:*

   The severity of the incident is calculated using the following formula:

[†] See the original message at http://www.incidents.org/archives/intrusions/msg00397.html

```
( Criticality + Lethality ) – ( System Countermeasures + Network Countermeasures )
```

All four values are rated on a scale of one (1) to five (5), with five being the highest. The resulting value is the overall severity level.

CRITICALITY:	*3*

The target host was a non-critical Linux server being used as a network diagnostic tool.

LETHALITY:	*2*

The Lethality level of this attack is a 2 because it an information gathering effort. However, the server did respond positively, indicating that the portmapper service was indeed running. Follow on attacks would potentially have a much greater level of severity.

SYSTEM COUNTERMEASURES:	*3*

System countermeasures were inadequate to prevent against this type of attack. Though the operating system was relatively recent, it was configured with vulnerable services, and readily gave out a list of its registered RPC programs.

NETWORK COUNTERMEASURES:	*2*

Network countermeasures were ineffective, primarily because the device in question was located outside of the network's firewall. Router packet filters, if present, did not prevent the attack from taking place.

OVERALL SEVERITY:	***0***

9. *Defensive recommendation:*

Relocate the target behind the network's firewall. This may not be possible because of the type of tasks the device is performing (e.g., network monitoring). At a minimum, the host should be hardened as much as possible by disabling all unnecessary services (most evidently RPC portmapper at 111/udp and 111/tcp) and installing host-based security tools such as PortSentry and TCP Wrappers. Implement router ACLs to prevent unnecessary inbound traffic from reaching the host.

10. *Multiple choice test question:*

What tool, common to most UNIX distributions, can be used to query the portmapper service?

a. dig

b. pmap

c. rpcinfo

d. rpcq

Answer: c

Network Detect 2: SuperScan

TRACE A:

```
[**] ICMP superscan echo from windows [**]
06/29-04:37:43.867657 217.84.158.142 -> XXX.XXX.XXX.2
ICMP TTL:115 TOS:0x0 ID:29764 IpLen:20 DgmLen:36
Type:8 Code:0 ID:256 Seq:12340 ECHO

[**] ICMP superscan echo from windows [**]
06/29-04:37:43.881004 217.84.158.142 -> XXX.XXX.XXX.3
ICMP TTL:115 TOS:0x0 ID:29766 IpLen:20 DgmLen:36
Type:8 Code:0 ID:256 Seq:12852 ECHO

[**] ICMP superscan echo from windows [**]
06/29-04:37:43.898794 217.84.158.142 -> XXX.XXX.XXX.4
ICMP TTL:115 TOS:0x0 ID:29769 IpLen:20 DgmLen:36
Type:8 Code:0 ID:256 Seq:13620 ECHO

[**] INFO FTP anonymous FTP [**]
06/29-05:42:27.735825 217.84.158.142:3068 -> XXX.XXX.XXX.2:21
TCP TTL:51 TOS:0x0 ID:37351 IpLen:20 DgmLen:56 DF
***AP*** Seq: 0x814B0D9C Ack: 0xD1EAE8A6 Win: 0xFFCD TcpLen: 20
```

TRACE B:

```
04:37:43.867657 217.84.158.142 > XXX.XXX.XXX.2: icmp: echo request (ttl
115, id 29764, len 36)
0x0000   4500 0024 7444 0000 7301 0770 d954 9e8e        E..$tD..s..p.T..
0x0010   XXXX XX02 0800 c6cb 0100 3034 0000 0000        ..........04....
0x0020   0000 0000 0dcb c6cb 036e 7331 0767             .........ns1.g
04:37:43.881004 217.84.158.142 > XXX.XXX.XXX.3: icmp: echo request (ttl
115, id 29766, len 36)
0x0000   4500 0024 7446 0000 7301 076d d954 9e8e        E..$tF..s..m.T..
0x0010   XXXX XX03 0800 c4cb 0100 3234 0000 0000        ..........24....
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
04:37:43.898794 217.84.158.142 > XXX.XXX.XXX.4: icmp: echo request (ttl
115, id 29769, len 36)
0x0000   4500 0024 7449 0000 7301 0769 d954 9e8e        E..$tI..s..i.T..
0x0010   XXXX XX04 0800 c1cb 0100 3534 0000 0000        ..........54....
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
05:42:27.735825 217.84.158.142.3068 > XXX.XXX.XXX.2.21: P [tcp sum ok]
2169179548:2169179564(16) ack 3521833126 win 65485 (DF) (ttl 51, id
37351, len 56)
0x0000   4500 0038 91e7 4000 3306 e9b3 d954 9e8e        E..8..@.3....T..
0x0010   XXXX XX02 0bfc 0015 814b 0d9c d1ea e8a6        .........K......
0x0020   5018 ffcd 0663 0000 5553 4552 2061 6e6f        P....c..USER.ano
0x0030   6e79 6d6f 7573 0d0a                            nymous..
```

1. *Source of the trace:*

   These traces were collected on my company's network. Note that the traces have been sanitized (including the hex output).

2. *Detect was generated by:*

   This detect was generated by the Snort Intrusion Detection system running in NIDS mode. Trace A shows the snort alerts, and Trace B shows the raw tcpdump output.

3. *Probability the source address was spoofed:*

   This appears to be a reconnaissance effort. Since the attacker would need to receive the response packets to gain any information, it is very unlikely that the source IP addresses have been spoofed.

4. *Description of attack:*

   As the snort signature indicates, the first three packets may have been generated by the Windows port scanning program SuperScan. Three consecutive IP addresses are ping'ed in rapid succession, and then a little more than an hour later, the attacker returns and anonymously FTP's to the first address (XXX.XXX.XXX.2).

5. *Attack mechanism:*

   A very probable explanation of this trace is that the attacker had been running a sweep using the SuperScan[‡] software. Once targets were identified, the attacker returned and checked for anonymous FTP.

   A closer look at the source IP address of the attacker reveals that it is a dial-up IP address from the German telecommunications company, Deutsche Telekom. This information was arrived at in the following manner:

   - An nslookup was performed on the source IP address:

```
echelon$ nslookup 217.84.158.142
Server:  xxx.xxx.com
Address:  XXX.XXX.XXX.1

Name:    pD9549E8E.dip.t-dialin.net
Address:  217.84.158.142
```

   - A WHOIS query is issued via the Network Solutions home page to determine the owner of the t-dialin.net domain. The following information is returned:

```
Registrant:
Deutsche Telekom Online Service GmbH (T-DIALIN2-DOM)
   Waldstrasse 3
   Weiterstadt, D-64331
   DE

<snip>[§]
```

[‡] For more information on SuperScan, see http://www.foundstone.com

[§] Contact information and nameservers are also listed with Network Solutions, but omitted from this document for brevity.

6. *Correlations:*

   A search of www.incidents.org, www.google.com, and www.altavista.com did not reveal any correlating events from this source IP address. However, this IP address does belong to a large dial-up provider, and it is very likely that the IP address is only temporarily assigned to dial-up users.

7. *Evidence of active targeting:*

   The initial scan was probably part of a sweep. The Anonymous FTP attempt, however, was evidence of active targeting. Because of the time differential between the ICMP ECHO_REQUEST packets and the anonymous FTP attempt, it appears that the attacker manually returned to the target to check for the existence of anonymous FTP. They may have been looking for writable FTP servers, or vulnerable FTP services.

8. *Severity:*

   The severity of the incident is calculated using the following formula:

   ```
   ( Criticality + Lethality ) – ( System Countermeasures + Network Countermeasures )
   ```

   All four values are rated on a scale of one (1) to five (5), with five being the highest. The resulting value is the overall severity level.

CRITICALITY:	*5*

   The target host was a firewall and a core application server located behind it, and thus the criticality level is very high.

LETHALITY:	*2*

   The Lethality level of this attack is a 2 because it is an information gathering effort.

SYSTEM COUNTERMEASURES:	*5*

   System countermeasures were in place to ensure this type of attack would not be successful. The server in question is running the latest operating system security patches, and has been hardened through configuration minimization, best practices, and host-based security software such as TCP wrappers.

NETWORK COUNTERMEASURES:	*4*

   Network countermeasures include router ACLs (though they did not prevent this reconnaissance) and an application proxy firewall. Although very strong, they are configured to permit both protocols involved in this reconnaissance effort. If the system countermeasures were not configured to prevent the use of anonymous FTP, the network countermeasures would not have prevented misuse of the FTP system. The application proxy may have prevent compromise of the FTP server through buffer overrun, but “normal” FTP traffic would have been permitted.

OVERALL SEVERITY:	***-2***

9. *Defensive recommendation:*

   Although the reconnaissance was successful, existing countermeasures would likely prevent any exploitation of the information gained.

That said, it's always possible to increase security – so if FTP services are not required, block using ACLs at the router and remove any rules permitting such access from the firewall. Additionally, contact the ISP that hosts the attacking machine, and notify them of the infraction. If they fail to respond, consider applying ingress router ACLs that drop all packets from part of all of their IP allocation. Note that this may affect legitimate use, so take this course of action with caution.

10. Multiple choice test question:

All of the following are true about ICMP ECHO_REQUEST and ECHO_REPLY packets except:

a. They have sequence numbers

b. They have an identifier

c. They can carry data

d. They are ICMP type 0 and 8, respectively

e. They are ICMP type 8 and 0, respectively

Answer: d

Network Detect 3: IIS Unicode Attack!

TRACE A:

```
[**] spp_http_decode: IIS Unicode attack detected [**]
06/29-10:22:29.668933 217.127.10.182:21724 -> XXX.XXX.XXX.2:80
TCP TTL:111 TOS:0x0 ID:39053 IpLen:20 DgmLen:130 DF
***AP*** Seq: 0x17571F Ack: 0xC65E0638 Win: 0x2238 TcpLen: 20

[**] WEB-MISC http directory traversal [**]
06/29-10:22:29.668933 217.127.10.182:21724 -> XXX.XXX.XXX.2:80
TCP TTL:111 TOS:0x0 ID:39053 IpLen:20 DgmLen:130 DF
***AP*** Seq: 0x17571F Ack: 0xC65E0638 Win: 0x2238 TcpLen: 20

[**] spp_http_decode: IIS Unicode attack detected [**]
06/29-10:22:29.688547 217.127.10.182:21725 -> XXX.XXX.XXX.2:80
TCP TTL:111 TOS:0x0 ID:39565 IpLen:20 DgmLen:130 DF
***AP*** Seq: 0x175723 Ack: 0x115B8BCD Win: 0x2238 TcpLen: 20

[**] WEB-MISC http directory traversal [**]
06/29-10:22:29.688547 217.127.10.182:21725 -> XXX.XXX.XXX.2:80
TCP TTL:111 TOS:0x0 ID:39565 IpLen:20 DgmLen:130 DF
***AP*** Seq: 0x175723 Ack: 0x115B8BCD Win: 0x2238 TcpLen: 20
```

TRACE B:

```
10:22:29.668933 217.127.10.182.21724 > XXX.XXX.XXX.2.80: P [tcp sum ok]
1529631:1529721(90) ack 3328050744 win 8760 (DF) (ttl 111, id 39053, len
130)
0x0000   4500 0082 988d 4000 6f06 3a71 d97f 0ab6        E.....@.o.:q....
0x0010   XXXX XX02 54dc 0050 0017 571f c65e 0638        ....T..P..W..^.8
0x0020   5018 2238 e5e8 0000 4745 5420 2f73 6372        P."8....GET./scr
0x0030   6970 7473 2f2e 2e25 6331 2531 632e 2e2f        ipts/..%c1%1c../
0x0040   7769 6e6e 742f 7379 7374 656d 3332 2f63        winnt/system32/c
0x0050   6d64 2e65 7865 3f2f 632b 6469 722b 633a        md.exe?/c+dir+c:
0x0060   5c20 4854 5450 2f31 2e30 0d0a 486f 7374        \.HTTP/1.0..Host
0x0070   3a20 XXXX 2eXX XX2e 3231 2e32 0d0a 0d0a
:.XXX.XXX.XXX.2....
0x0080   0d0a                                           ..

10:22:29.668933 217.127.10.182.21724 > XXX.XXX.XXX.2.80: P [bad tcp
cksum b2b4!] 0:90(90) ack 1 win 8760 (DF) (ttl 111, id 39053, len 130)
0x0000   4500 0082 988d 4000 6f06 3a71 d97f 0ab6        E.....@.o.:q....
0x0010   XXXX XX02 54dc 0050 0017 571f c65e 0638        ....T..P..W..^.8
0x0020   5018 2238 e5e8 0000 4745 5420 2f73 6372        P."8....GET./scr
0x0030   6970 7473 2f2e 2ec1 1c2e 2e2f 7769 6e6e        ipts/....../winn
0x0040   742f 7379 7374 656d 3332 2f63 6d64 2e65        t/system32/cmd.e
0x0050   7865 3f2f 632b 6469 722b 633a 5c20 4854        xe?/c+dir+c:\.HT
```

```
0x0060   5450 2f31 2e30 0d0a 486f 7374 3a20 36XX        TP/1.0..Host:.XX
0x0070   XXXX XXXX XXXX 2e32 0d0a 0d0a 0d0a 0d0a        XXXXXXXX........
0x0080   0d0a                                           ..

10:22:29.688547 217.127.10.182.21725 > XXX.XXX.XXX.2.80: P [tcp sum ok]
1529635:1529725(90) ack 291212237 win 8760 (DF) (ttl 111, id 39565, len
130)
0x0000   4500 0082 9a8d 4000 6f06 3871 d97f 0ab6        E.....@.o.8q....
0x0010   XXXX XX02 54dd 0050 0017 5723 115b 8bcd        ....T..P..W#.[..
0x0020   5018 2238 024b 0000 4745 5420 2f73 6372        P."8.K..GET./scr
0x0030   6970 7473 2f2e 2e25 6330 2539 762e 2e2f        ipts/..%c0%9v../
0x0040   7769 6e6e 742f 7379 7374 656d 3332 2f63        winnt/system32/c
0x0050   6d64 2e65 7865 3f2f 632b 6469 722b 633a        md.exe?/c+dir+c:
0x0060   5c20 4854 5450 2f31 2e30 0d0a 486f 7374        \.HTTP/1.0..Host
0x0070   XXXX XXXX XXXX XX2e XXXX 2e32 0d0a 0d0a
:.XXX.XXX.XXX.2....
0x0080   0d0a                                           ..

10:22:29.688547 217.127.10.182.21725 > XXX.XXX.XXX.2.80: P [bad tcp
cksum 8b55!] 0:90(90) ack 1 win 8760 (DF) (ttl 111, id 39565, len 130)
0x0000   4500 0082 9a8d 4000 6f06 3871 d97f 0ab6        E.....@.o.8q....
0x0010   XXXX XX02 54dd 0050 0017 5723 115b 8bcd        ....T..P..W#.[..
0x0020   5018 2238 024b 0000 4745 5420 2f73 6372        P."8.K..GET./scr
0x0030   6970 7473 2f2e 2ec0 2539 762e 2e2f 7769        ipts/...%9v../wi
0x0040   6e6e 742f 7379 7374 656d 3332 2f63 6d64        nnt/system32/cmd
0x0050   2e65 7865 3f2f 632b 6469 722b 633a 5c20        .exe?/c+dir+c:\.
0x0060   4854 5450 2f31 2e30 0d0a 486f 7374 3a20        HTTP/1.0..Host:.
0x0070   XXXX 2eXX XX2e XXXX 2e32 0d0a 0d0a 0d0a
XXX.XXX.XXX.2....
0x0080   0d0a                                           ..
```

1. *Source of the trace:*

   These traces were collected on my company's network. Note that the traces have been sanitized (including the hex output).

2. *Detect was generated by:*

   This detect was generated by the Snort Intrusion Detection system running in NIDS mode. Trace A shows the snort alerts, and Trace B shows the raw tcpdump output.

3. *Probability the source address was spoofed:*

   While it is not inconceivable that TCP session hijacking couple be employed in this type of attack, it is unlikely. This particular implementation of this attack returns information to the attacker's browser, letting them know the attack has been successful (dir output of the root system drive). For this reason alone, it is highly unlikely that the source IP address has been spoofed in this case.

4. *Description of attack:*

The attack exploits a serious vulnerability in unpatched versions of Microsoft IIS 4.0 and IIS 5.0, whereby an attacker can read files outside of the IIS server document root, and, in many circumstances, execute arbitrary code. This vulnerability is referenced by CVE-2000-0884 and BUGTRAQ:20001017.

5. *Attack mechanism:*

   This attack is accomplished through the exploitation of a bug in Microsoft's IIS 4.0 and IIS 5.0 whereby URL requests are not properly normalized before being executed. Unicode characters are used to "break out" of the document root, and traverse the directory structure. The following GET request was issued by the attacker (several variations were also issued, mainly varying the Unicode characters):

   ```
   http://www.xxxxxxxxx.com/scripts/..%c1%1c../winnt/system32/
   cmd.exe?/c+dir+c:\
   ```

   Several other variations were also issued, mainly varying the Unicode characters. This was likely due to the lack of success the attacker experienced. This was in no small part due to the fact that the target host was not a Microsoft machine at all, and thus not vulnerable to this type of attack.

   Had this attack succeeded, it would have executed the Microsoft command shell (cmd.exe) with the arguments "/c dir c:\." The output of the command is piped through the Web server and back to the attacker's browser.

   Obviously, there are far more serious implications to this than a mere directory listing. The commands are executed with under the user account that IIS runs under, which is typically IUSR_hostname. Although this account generally has only limited privileges, they are, in most cases, more than enough to seriously compromise the security of the target host. This includes theft of data (e.g., credit card numbers), as well as possible destructive action.

   A more detailed explanation of this attack can be found at http://www.securiteam.com/windowsntfocus/Web_Server_Folder_Traversal_vulnerability_Patch_available_exploit_.html

   Note the presence of packets with bad TCP checksums – this is most probably evidence of a bad connection between the attacker (probably dial-up link) and the remote host. However, it could also be an indicator of poorly written exploit code and packet crafting.

6. *Correlations:*

   These attacks have been as widespread at the software they target.

   This attack was sourced from an IP address belonging to a block assigned to France Telecom. A search of www.incidents.org, www.google.com, and www.altavista.com did not reveal any correlating attacks for this source IP address.

7. *Evidence of active targeting:*

   Active targeting is certainly in evidence in this case as the attacker tried several times to compromise the target using the IIS Unicode vulnerability. The fact that the target host was not, in fact, a Microsoft server did not seem to deter the attacker (though it did deter the attack).

8. *Severity:*

   The severity of the incident is calculated using the following formula:

   ```
   ( Criticality + Lethality ) – ( System Countermeasures + Network Countermeasures )
   ```

All four values are rated on a scale of one (1) to five (5), with five being the highest. The resulting value is the overall severity level.

CRITICALITY:	*4*

The target host was a WWW server located behind a firewall. This same host also provides other important services.

LETHALITY:	*1*

This attack has a Lethality of 1 since the target host is not vulnerable to the intrusion vector the attacker is targeting (it is a UNIX WWW server, and this attack relies on a vulnerability in Microsoft code).

SYSTEM COUNTERMEASURES:	*5*

In addition to being the wrong type of system to mount this attack on, other system countermeasures were also in place. The server in question is running the latest operating system security patches, and has been hardened through configuration minimization, best practices, and host-based security software such as TCP wrappers.

NETWORK COUNTERMEASURES:	*3*

This type of attack has traditionally evaded most network countermeasures since it takes advantage of services typically opened to the Internet (HTTP). Additionally, because the URL GET requests are not malformed (they typically comply with HTTP/1.1), application proxy firewalls will probably not block this type of attack.

OVERALL SEVERITY:	***-3***

9. *Defensive recommendation:*

The existing defensive posture, in this case, is very adequate to prevent this type of attack from being successful. On an on-going basis, ensure that all Internet exposed servers have the most recent security fixes applied.

If potentially vulnerable servers (IIS) are introduced to the network environment at a future date, ensure that the latest security fixes are applied. Additional information regarding fixes can be found at www.microsoft.com.

10. *Multiple choice test question:*

Thwarting Unicode attacks is best accomplished using which of the following techniques:

a. Applying relevant hot-fixes

b. Changing the default password for the IUSR_host account

c. Installing an application proxy firewall

d. Running SSL

Answer: a

Network Detect 4: The call is coming from inside the house!

TRACE A:

```
May 21 14:33:20 192.168.1.1:10978 -> XXX.XXX.XXX.12:5191 SYN ******S*
May 21 14:33:20 192.168.1.1:10979 -> XXX.XXX.XXX.12:5192 SYN ******S*
May 21 14:33:20 192.168.1.1:10980 -> XXX.XXX.XXX.12:5193 SYN ******S*
May 21 14:33:20 192.168.1.1:11001 -> XXX.XXX.XXX.12:113 SYN ******S*
May 21 14:33:20 192.168.1.1:10983 -> XXX.XXX.XXX.12:1025 SYN ******S*
May 21 14:33:20 192.168.1.1:10984 -> XXX.XXX.XXX.12:67 UDP
May 21 14:33:20 192.168.1.1:10985 -> XXX.XXX.XXX.12:19 SYN ******S*
May 21 14:33:20 192.168.1.1:10986 -> XXX.XXX.XXX.12:19 UDP
May 21 14:33:20 192.168.1.1:10987 -> XXX.XXX.XXX.12:4144 SYN ******S*
<snip>
May 21 14:33:22 192.168.1.1:26271 -> XXX.XXX.XXX.12:80 NMAPID **U*P*SF
May 21 14:33:22 192.168.1.1:26275 -> XXX.XXX.XXX.12:5190 XMAS **U*P**F

[**] SCAN nmap fingerprint attempt [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:22.516894 192.168.1.1:26271 -> XXX.XXX.XXX.12:80
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
**U*P*SF Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40 UrgPtr: 0x0
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS05]

[**] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:22.516932 192.168.1.1:26272 -> XXX.XXX.XXX.12:80
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
***A**** Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS28]

[**] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:22.517010 192.168.1.1:26274 -> XXX.XXX.XXX.12:5190
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
***A**** Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS28]

May 21 14:33:24 192.168.1.1:31476 -> XXX.XXX.XXX.12:80 SYN ******S*
May 21 14:33:24 192.168.1.1:31477 -> XXX.XXX.XXX.12:80 NULL ********
May 21 14:33:24 192.168.1.1:31478 -> XXX.XXX.XXX.12:80 NMAPID **U*P*SF
May 21 14:33:24 192.168.1.1:31480 -> XXX.XXX.XXX.12:5190 SYN ******S*
May 21 14:33:24 192.168.1.1:31482 -> XXX.XXX.XXX.12:5190 XMAS **U*P**F
```

```
May 21 14:33:24 192.168.1.1:31483 -> XXX.XXX.XXX.12:512 UDP
May 21 14:33:24 192.168.1.1:31489 -> XXX.XXX.XXX.12:80 SYN ******S*

[**] SCAN nmap fingerprint attempt [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:24.514116 192.168.1.1:31478 -> XXX.XXX.XXX.12:80
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
**U*P*SF Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40 UrgPtr: 0x0
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS05]

[**] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:24.514152 192.168.1.1:31479 -> XXX.XXX.XXX.12:80
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
***A**** Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS28]

[**] SCAN nmap TCP [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:24.514232 192.168.1.1:31481 -> XXX.XXX.XXX.12:5190
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
***A**** Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
[Xref => http://www.whitehats.com/info/IDS28]

May 21 14:33:26 192.168.1.1:25487 -> XXX.XXX.XXX.12:79 SYN *2****S*
May 21 14:33:26 192.168.1.1:25475 -> XXX.XXX.XXX.12:79 NULL ********
May 21 14:33:26 192.168.1.1:25476 -> XXX.XXX.XXX.12:79 NMAPID **U*P*SF
May 21 14:33:26 192.168.1.1:25478 -> XXX.XXX.XXX.12:5190 SYN ******S*
May 21 14:33:26 192.168.1.1:25480 -> XXX.XXX.XXX.12:5190 XMAS **U*P**F
May 21 14:33:26 192.168.1.1:25481 -> XXX.XXX.XXX.12:512 UDP
<snip>
```

TRACE B:

```
14:33:24.514188 0:50:4:a1:f3:97 0:50:8b:a:c0:57 0800 74:
192.168.1.1.31480 > 12.46.112.83.5190: S [tcp sum ok] 1212:1212(0) win
51200 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 128, id
444, len 60)

14:33:24.514232 0:50:4:a1:f3:97 0:50:8b:a:c0:57 0800 74:
192.168.1.1.31481 > 12.46.112.83.5190: . [tcp sum ok] ack 0 win 51200
<wscale 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 128, id 444, len
60)

14:33:24.514272 0:50:4:a1:f3:97 0:50:8b:a:c0:57 0800 74:
192.168.1.1.31482 > 12.46.112.83.5190: FP [tcp sum ok] 1212:1212(0) win
```

```
51200 urg 0 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol> (ttl 128,
id 444, len 60)

14:33:24.514431 0:50:8b:a:c0:57 0:3:6b:b5:bd:60 0800 60:
12.46.112.83.5190 > 192.168.1.1.31480: R [tcp sum ok] 0:0(0) ack 1213
win 0 (DF) (ttl 128, id 26447, len 40)

14:33:24.514497 0:50:8b:a:c0:57 0:3:6b:b5:bd:60 0800 60:
12.46.112.83.5190 > 192.168.1.1.31481: R [tcp sum ok] 0:0(0) win 0 (DF)
(ttl 128, id 26448, len 40)
```

1. *Source of the trace:*

   These traces were collected on a client's network. Note that the traces have been sanitized (including any hex output).

2. *Detect was generated by:*

   This detect was generated by the Snort Intrusion Detection system running in packet logger mode. The resulting binary tcpdump format output was then processed through snort in NIDS mode. SnortSnarf[**] was then used to collate the data into a more readable format. Excerpts from the SnortSnarf output are listed as Trace A.

3. *Probability the source address was spoofed:*

   At first glance, the answer would seem to be a resounding yes, since the source IP address is 192.168.1.1 – a RFC1918 private address that should not be routable on the Internet. A portscan with a spoofed IP address has little direct reconnaissance value to an attacker, since they will not receive the response packets and won't learn anything.

   A few possible scenarios present themselves:

   - A novice script kiddie has launched the scan using a spoofed IP address, not realizing that it doesn't really work like that;
   - A more astute attacker has launched the scan with a spoofed IP address as a decoy to distract attention from his/her real target;
   - The attacker has somehow managed to get the ISP to route his traffic using the RFC1918 address. This could be the result of egregiously misconfigured routers at the ISP;
   - The attack is coming from within the organization's own network, most probably on the external network between the firewall and the Internet router.

   A closer look at the trace makes the last option seem the most likely. Note the TTL value of the packets (in bold) that generated regular snort alerts (rather than the portscan output). They are all 128, a common starting value. Since every router that received the packet would have decremented the TTL by 1, it seems likely that these packets are being sourced on the same network at the destination.

[**] SnortSnarf is available at http://www.silicondefense.com

Trace B corroborates this theory. In the first packet, notice the source MAC address (in bold). The first three octets of the MAC address usually indicate the manufacturer of the network card that generated the packet on the local wire.[††] In this case, the first three octets are "00-50-04". According to the list at IEEE (http://standards.ieee.org/regauth/oui/oui.txt), this packet was generated by 3COM hardware.

The next bolded packet shows the target host responding with a TCP RST packet, but its destination MAC address is different! The target host is responding to its default gateway, the Internet router (a Cisco – "00-03-6b").

To summarize, using the MAC addresses, we are able to determine with reasonable certainty that the packets did not arrive via the Internet router, and thus were very probably generated locally.

4. *Description of attack:*

This is a scan performed using the publicly available port scanning software nmap by Fyodor.[‡‡] The nmap tool is a port scanner capable of performing a wide variety of scanning methods. This OS fingerprinting portion of this attack has been tentatively defined by CVE candidate CAN-1999-0454 and http://www.whitehats.com/info/IDS05.

5. *Attack mechanism:*

This particular detect appears to be a TCP SYN scan, with OS fingerprinting enabled. The presence of packets that are out of specification (multiple, conflicting flags turned on such as SYN and FIN in the same packet) strongly suggests OS fingerprinting. nmap's OS fingerprinting method crafts packets with various flags and options set and looks for subtle (sometimes not-so-subtle) variations in the responses. Note the following excerpt from Trace A:

```
[**] SCAN nmap fingerprint attempt [**]
[Classification: Attempted Information Leak] [Priority: 3]
05/21-14:33:24.514116 192.168.1.1:31478 -> XXX.XXX.XXX.12:80
TCP TTL:128 TOS:0x0 ID:444 IpLen:20 DgmLen:60
**U*P*SF Seq: 0x4BC Ack: 0x0 Win: 0xC800 TcpLen: 40 UrgPtr: 0x0
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
```

This packet is a TCP packet sent to the finger port on the target host with the SYN, FIN, URG, and PSH flags all set, along with several TCP options: windows size (WS), NOP, maximum segment size (MSS), time stamp (TS), and end of options (EOL). The way the operating system responds to this packet, including the way it treats the TCP options, will be used by nmap to determine the operating system type. A more complete treatment of this subject can be found at http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[††] It should be noted that MAC addresses can also be spoofed, and that some machines will override the hard-coded MAC address of the networking hardware by default (e.g., Sun). Some older networking protocols such as DECnet also modify the MAC address.

[‡‡] nmap is available at http://www.insecure.org.

The other part of this attack is the TCP SYN scan, also known as a stealth scan or a half-open scan. This type of scan is executed by crafting TCP SYN packets that are sent to the destination host. If the destination port is listening, it should send back a SYN|ACK packet, in which case the connection is torn down with a RST packet and nmap logs the port as open. If the destination port is not listening, the remote host should send back a RST packet in response to the initial SYN. The idea is that this type of scanning is "quieter" than a full TCP connect scan (where the three way hand-shake is completed), and that it may evade some forms of perimeter defense or intrusion detection. The reality today is that any IDS or firewall worth its weight will detect and log this type of activity.

6. *Correlations:*

   The use of nmap is widespread on the Internet. It is a favourite tool of both the "hacking community" and security professionals. Detailed information regarding nmap can be found at http://www.insecure.org/nmap/nmap_doc.html.

7. *Evidence of active targeting:*

   This was a reconnaissance effort that was launched locally, and thus it would seem that there was indeed evidence of active targeting. In fact, it is very conceivable that the attack was launched from a compromised system located outside of the network's firewall, which was the target of the attack.

8. *Severity:*

   The severity of the incident is calculated using the following formula:

   ```
   ( Criticality + Lethality ) – ( System Countermeasures + Network Countermeasures )
   ```

   All four values are rated on a scale of one (1) to five (5), with five being the highest. The resulting value is the overall severity level.

CRITICALITY:	*5*

The target host is the organization's primary Internet firewall.

LETHALITY:	*2*

Lethality of this attack is low since both the TCP scan and the OS fingerprinting portions of the attack are information-gathering efforts.

SYSTEM COUNTERMEASURES:	*5*

The system being targeting is a firewall (hardened, of course) running the latest patches and hot-fixes.

NETWORK COUNTERMEASURES:	*2*

Since the firewall was the actual target of this attack, and the attack appears to have originated from the local network, network countermeasures were ineffective at preventing this assault. If they had been effective, direct access to the local external network would not have been possible, and the scan would have had to take place from the Internet, where router ACLs would have provided an additional layer of protection (in fact, blocking the attack since they would drop packets sourced with RFC1918 addresses).

OVERALL SEVERITY: ***0***

9. *Defensive recommendation:*

   Immediately locate the server that originated the attack. If this attack was indeed generated locally, it is imperative to ascertain whether the box is a compromised host or an unauthorized interloper and follow appropriate incident response procedures to remedy the situation.

10. *Multiple choice test question:*

    What IP header field is decremented by routers?

    a. MSS

    b. Fragment Offset

    c. Time Stamp

    d. TTL

    Answer: d

Network Detect 5: In Search Of… Silly Gs

TRACE A:

```
Feb 24 14:16:01 XXX.XXX.XXX.1 10715: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.0(5232), 1 packet

Feb 24 14:16:01 XXX.XXX.XXX.1 10716: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.1(5232), 1 packet

Feb 24 14:16:01 XXX.XXX.XXX.1 10717: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.3(5232), 1 packet

Feb 24 14:16:01 XXX.XXX.XXX.1 10718: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.5(5232), 1 packet

Feb 24 14:16:01 XXX.XXX.XXX.1 10719: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.6(5232), 1 packet

<snip>

Feb 24 14:16:02 XXX.XXX.XXX.1 10764: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.60(5232), 1 packet

Feb 24 14:16:02 XXX.XXX.XXX.1 10765: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.61(5232), 1 packet

Feb 24 14:16:02 XXX.XXX.XXX.1 10766: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.62(5232), 1 packet

Feb 24 14:16:02 XXX.XXX.XXX.1 10767: %SEC-6-IPACCESSLOGP: list 120
denied tcp 216.87.208.233(40771) -> XXX.XXX.XXX.63(5232), 1 packet
```

1. *Source of the trace:*

   These traces were collected on my company's network. Note that the traces have been sanitized.

2. *Detect was generated by:*

   This detect was generated by a Cisco router with access control lists and logging enabled. Subsequent processing of the text-based log files was performed using WebTrend's Firewall Suite and grep.

3. *Probability the source address was spoofed:*

   It is unlikely that the source address has been spoofed in this case, since in order for the attack to work the attacker must receive a response. It is conceivable, but improbable, that the attacker is spoofing an IP address to "frame" the true owner of the IP address.

4. *Description of attack:*

This is a scan looking for Silicon Graphics IRIX machines. It is described in http://www.kb.cert.org/vuls/id/28027, and tentatively referenced by CVE candidate CAN-2000-0893.

5. *Attack mechanism:*

   This scan is a primitive and rudimentary OS fingerprinting effort in search for Silicon Graphics IRIX machines. IRIX runs a service called the Distributed GL Daemon (dgld) that listens on 5232/tcp. This attack works by sending a TCP SYN packet to a host (more typically, a range of hosts, as is the case in this detect) on port 5232. If a response it received, it is likely that the responding host is a Silicon Graphics IRIX machine.

   This trace was generated by a Cisco router with access control lists. The access control lists log all traffic that they block, so these packets did not reach their intended recipient. However, note the gaps in the destination IP sequence (not the intentionally omitted portion indicated by the <snip>). The entire trace appears to proceed through the entire subnet range (XXX.XXX.XXX.0 -> XXX.XXX.XXX.63), but skips XXX.XXX.XXX.2 and XXX.XXX.XXX.4. These two IP addresses were either not scanned, or not blocked by the router (and thus not logged). Since the trace scans every other IP address in the range, a likely explanation is that the router is not blocking 5232/tcp to the two skipped IP addresses. This is significant only in that if these two IP addresses belong to IRIX hosts, they may have been fingerprinted.

   Also interesting in this trace is the fact that the source port remains the same through the entire detect. This is very unnatural behaviour; a new ephemeral port should be allocated for each new TCP connection. Since each of the TCP SYN packets is sent to a new destination IP address, the source port should change. This is evidence of packet crafting.

   The scan was performed very quickly – according to the time stamps, only 1 second elapsed from start to finish.

6. *Correlations:*

   A search of the www.incidents.org and www.neohapsis.com archives revealed several correlating events. One of the earliest reported detects of this type dates back to April 12, 2000 and is attributed to Korea University (see http://www.incidents.org/archives/y2k/041200.htm).

   Several other similar scan types were reported, but the most interesting correlating event was reported by John Springer, who detected and reported the same scan, from the same IP address, on the same date (http://www.incidents.org/archives/y2k/030101.htm).

7. *Evidence of active targeting:*

   The correlating events strongly suggest that this was a broad sweep. Exactly how broad can not be determined as actual destination IP addresses were not provided in the correlating events, but it seems clear that the target network in this detect was not singled out for attack.

8. *Severity:*

   The severity of the incident is calculated using the following formula:

   ```
   ( Criticality + Lethality ) – ( System Countermeasures + Network Countermeasures )
   ```

   All four values are rated on a scale of one (1) to five (5), with five being the highest. The resulting value is the overall severity level.

CRITICALITY:	*5*

The target is a network sweep that includes the firewall and Internet router. For this reason, criticality is set very high.

LETHALITY:	*1*

The lethality of this attack is low because it is merely a crude attempt to fingerprint a specific operating system type.

SYSTEM COUNTERMEASURES:	*5*

All exposed systems are hardened, running security-enhancing software (of varying types, depending on the machines function), and patched with the latest operating system patches. None of the servers matched the attacks target operating system type.

NETWORK COUNTERMEASURES:	*4*

The router that created the logs block almost all of the packets generated in the attack, however, two were allowed to pass.

OVERALL SEVERITY:	***-3***

9. *Defensive recommendation:*

Network and system countermeasures are very adequate in protecting against this type of attack. However, the ISP hosting the machine that generated the scan should be notified of the infraction. If they fail to respond, consider applying ingress router ACLs that drop all packets from part of all of their IP allocation. Note that this may affect legitimate use, so take this course of action with caution.

10. *Multiple choice test question:*

Which of the following is **not** evidence of packet craft?

a. Identical IP ID fields in packet fragments

b. Overlapping fragment offsets

c. Repeating TCP source ports to different destination ports

d. SYN and FIN flags both set in the same packet

Answer: a

ASSIGNMENT 2 – DESCRIBE THE STATE OF INTRUSION DETECTION

Intrusion Prevention: The Next Generation of Security Software?

Introduction

The constant stream of new vulnerabilities discovered in major software products deployed all over the Internet presents a continuous and never ending challenge to IT staff and security professionals alike. Firewalls and network-based intrusion detection systems help mitigate the risk, but have their flaws. Products are emerging on the market to address these shortcomings. At times, they have been referred to as "application behaviour control" or "intrusion prevention" products. Are these new products necessary? If so, are they effective? Where do they fit in with existing security tools?

The Problem with Firewalls and Intrusion Detection Systems

The shortcomings of firewalls have long been a subject of debate. The proponents of so-called stateful inspection firewalls (circuit-level gateways) praise the high levels of performance circuit-level technology affords, while the advocates of the application level gateway argue the sacrifice of speed for security is a worthy one. However, people in both camps will generally readily admit that the firewall is not a panacea for their security concerns, but it is rather the first step in establishing a secure network infrastructure.

The next step to securing a network is the deployment of a network-based intrusion detection system (NIDS). These systems are designed to listen to traffic on the network and look for patterns that match known hostile behaviour. When such behaviour is detected, an alert is generated notifying the security administrators of the infraction. More recently, NIDS have become more aggressive, and can actually take action against an attacker. Some of the more common methods employed include "killing" the connection by spoofing a TCP RST packet, or proactively "hardening" a firewall by ordering it to block the offending IP address.

Even with the ability to kill connections and harden firewalls, the fact remains that the NIDS is not functioning in real-time (despite the assertions of the marketing people). The NIDS is *responding* after the fact – in many cases, the damage may already be done before the NIDS has had a chance to take action. This fact is highlighted in the very name of the technology: intrusion *detection*.

In addition to a lack of true real-time response capability, NIDS can only detect known attacks – or attacks that match some sort of known pattern than can be defined to the NIDS as hostile behaviour. Since new attacks are being developed every day, and they are often in wide circulation among the "hacker" community before they are defined in the IDS, there is a significant window of opportunity where a NIDS may completely miss the attack. This is a serious problem.

There is some active research in the field of "anomaly detection" whereby the NIDS uses statistical deviations to determine when a certain type of traffic may be threatening, but this is likely to be problematic for some time to come for a few reasons. The first, and perhaps most important, is that anomaly detection is very likely to lead to a great many false positives. Although this is not unusual in NIDS, it is one of the primary reasons legitimate alerts are overlooked.

Host-based intrusion detection systems suffer from some of the same limitations. In fact, some host-based "intrusion detection systems" are really nothing more than log file monitoring utilities. In these cases, the host-based IDS relies on the operating system or other add-on components to generate log file messages. It then compares the log file messages against its set of signatures, and generates an alert when a match is found. Thus, not only is the IDS completely reliant on the operating system to detect the behaviour, but also can only detect attacks that have been previously defined to it.

A Solution?

A relatively new approach centers on controlling the behaviour of an application by limiting its access to operating system calls. The concept is simple: software is installed that shims itself between the application programming interface (API) for the operating system and the actual kernel itself. The software then intercepts API calls and processes them against a rule set to determine if the call should be permitted or denied. In a way, this type of solution is a type of "kernel firewall" – legitimate application activity can be defined and permitted, and all else blocked.

Consider the following example: a typical web server buffer overrun exploit will often seem like legitimate traffic to a firewall. After all, it is traffic destined for a WWW server on port 80/tcp and the firewall is configured to permit this traffic. Assuming the firewall is not performing any data scanning at the application layer, the traffic will almost certainly be permitted to pass. If the attack is new and not yet defined to the NIDS, it may pass without triggering an alarm.[§§] At this point, it is now up to the application and its host operating system to defend against the attack. Since the application is vulnerable to the buffer overrun in the first place, it's clearly not going to be much help in preventing the attack. The buffer overrun succeeds; code is injected into the stack and executed. In this example, let's say the code reads the SAM file, and e-mails it to the attacker.

Now imagine for a second that our "kernel firewall" was installed, and configured with a rule set defining behaviour that is normal for an IIS server. When the injected code attempts to read the SAM file, the kernel firewall intercepts the system call and checks it against its rule set. As there is no rule permitting such an operation, the system call is denied, and an alert is generated to notify the administrator of the attempted infraction.

The Players

More than one than player currently competes in this market space. Entercept, Okena, and WatchGuard are among the companies that have released products that "firewall" the kernel.

Entercept's products include a general, rule-based engine that intercepts system and API calls and compares them against a known set of attacks (signatures) as well as a set of rules governing access to system resources such as the file system or registry. A special add-on is tailored for use with Microsoft's Internet Information Server (IIS), a frequent source of vulnerabilities.

Entercept's HTTP engine looks for known attacks using signatures, very much like existing network-based intrusion detection systems. However, one of Entercept's primary advantages is that it receives HTTP traffic *after* the data is decrypted. Normal network-based IDS cannot detect attacks that may be launched over an SSL connection since the data is encrypted on the wire.

[§§] Many network-based IDS are configured to detect strings that are common to most buffer overruns (e.g., the NOP slide). However, this is non-deterministic, and it is quite possible that a buffer overrun could evade such a signature.

The general Entercept engine increases network security by protecting system resources:

- File and Registry protection – the Entercept software will prevent system files from being modified. This helps protect against Trojan attacks where system binaries are replaced with modified files. Additionally, Entercept has the ability to limit the process that can modify a file. For example, a web server's document directory can be protected so that the files can only be modified by a previously defined service.
- Process/Service protection – attempts to kill (or stop) services such as the Web server itself are blocked by the Entercept software.
- User Account protection – to ensure that privileges cannot be escalated, Entercept allows rules to prevent an attacker from increasing the privileges of a user account (such as the account a WWW server is running under).

Since there are some signatures involved, the Entercept software has a facility to automatically check for and install updates to the signature base.

Okena's StormWatch is very similar architecturally to the Entercept product in that it is centered on a rule-processing engine that intercepts calls to the operating system. Okena refers to these as "interceptors," each of which watches a specific part of the operating system:

- The *network traffic interceptor* handles network based security events, detecting such attacks as port scans;
- The *registry interceptor* controls access by applications to the registry;
- The *file system interceptor* mediates access to system files.

Events that are intercepted are processed by the "Rule/Event Correlation Engine" which bases its decisions not only on the rule base, but also on previous actions performed by the application. Additionally, one benefit StormWatch has over Entercept is the ability to correlate events and take action based on the correlated data. For example events generated by a network read by an address, followed by a file system write can be correlated by the StormWatch engine. The two events occurring together can trigger an action, such as an alert or a block of the offending IP address.

WatchGuard's ServerLock again is architecturally very similar to the other two products mentioned, intercepting calls to the file system and registry, and processing them against a rule set.

What About the Operating System?

Although the notion of "intrusion prevention" or "behaviour control" security tools is being sold as a new approach to enterprise security, the fact of the matter is that it's not really a new concept at all. Some of the earliest security controls were implemented as host-based controls – passwords, user accounts, and file permissions are all mechanisms operating systems have used for decades to regulate access to resources. In fact, the management of system resources is one of the primary functions of the operating system:

> in every computer system is the software that controls processing, manages resources, and communicates with external devices like disks and printers. …[we] will use the broader term *operating system*.[***]

It is impossible to effectively *manage* system resources without *securing* them, since by its very definition the term *manage* means "to direct or control the use of", according to the *American Heritage Dictionary*.

Modern operating systems provide a wide array of mechanisms that can be used to secure resources, from individual user accounts with discrete privilege profiles to complex access controls lists (ACLs) that can be applied to system resources. Yet, we are still plagued by exploitable security vulnerabilities that make products like Entercept, Okena, and WatchGuard necessary. There are many contributing factors:

- The size of modern operating systems and applications has grown exponentially. Doug Comer's XINU operating system was less than 10,000 lines of code – it is axiomatic that there is security in simplicity. It's far easier to debug 10,000 lines of code than it is 1,000,000, and there are far fewer places for vulnerabilities to hide.
- The proliferation of the "personal computer." The personal computer did not need security features since it was designed to be used by one person. Early PC operating systems such as DOS and CPM contained little or no security features at all. To this day, Microsoft's Windows 9x/ME operating systems lack even the most basic security features such as file access permissions and discretionary user accounting.
- Today's popular operating systems are products of evolution rather than holistically engineered programs with clearly defined design parameters. Microsoft's operating systems, including Windows 2000, as well as modern UNIX variants, are all designed with requirements to support legacy applications. So called "trusted systems" are systems designed from the ground up with security as a primary objective – however, these systems are often proprietary making it hard to find applications to run on them, or making them too complex and difficult to use for the average user.
- The race to market drives software makers to produce products that are not properly tested, often resulting in egregious security vulnerabilities discovered only after they are widely deployed. This is powerfully demonstrated by continued stream of buffer overrun vulnerabilities that are almost routinely discovered against products such as Microsoft's IIS. With basic good programming practices, these types of bugs should never occur.
- Operating system security features are often disabled or misconfigured. Security and ease of use are often inversely proportional, thus operating systems such as Windows 2000 typically install with a very lax default configuration. It is up to the user to secure the operating system and "harden" it against attack. More often than not, this is not done. Often due to a lack of security awareness, or a lack of the necessary skills or knowledge, systems are deployed without any of their security features enabled.

[***] Comer, 1.

While it is clearly in the charter of the operating system to provide the security services, the concept of "outsourcing" the security services to a third party product is certainly not a new one. IBM mainframes, most notably the MVS operating system, have traditionally used third party (CA) add-on products such as CA-ACF/2 and CA-Top Secret to secure system resources. Providing "hooks" in the operating system for security software may not be such a bad idea – end users could then select a third party security application based on their security requirements. The "intrusion prevention" applications are a step in that direction.

Conclusion

Enterprise security has come full circle, back to the notion of implementing security controls at the host itself rather than at the network perimeter. The best security approach includes both network-based and host-based security solutions, and due to factors such as buggy code and misconfiguration of operating system security controls, applications such as Entercept, Okena's StormWatch, and WatchGuard's ServerLock are gaining in popularity. These solutions may be a stop-gap measure, providing features the operating systems themselves can or should provide, or they may be a move toward extricating the security functions from the operating system and placing them in the hands of third-party add-on applications. Either way, at the moment they provide a critical extra layer of protection in the quest to remain secure in a world where time-to-market has replaced quality assurance.

ASSIGNMENT 3 – ANALYZE THIS

Introduction

The following document contains the results of security audit performed for Foo University. It intended to be a practical tool to assist University technical staff in increasing the security of their network by clearly identifying key areas of vulnerability and threat, and providing recommendations to improve security in these areas.

Foo University has provided a limited amount of information for analysis. Though the size and volume of the intrusion detection logs were copious, they do not necessarily provide a clear picture of the University's network architecture. The following items would have allowed a much more thorough and effective security audit to be performed:

- Network diagrams depicting the architecture of the Foo University network infrastructure and, specifically, the location of the Snort sensors providing the IDS log data;
- Copies of the Snort configuration and rule files used to generate the IDS log data;
- Access to Foo University technical staff for the purposes of interviewing and clarification.

The following log files were used in the creation of this report:

DATE	FILENAMES
01 July 2001	alert.010701, scans.010701, oos_Jul.1.2001
02 July 2001	alert.010702, scans.010702, oos_Jul.2.2001
03 July 2001	alert.010703, scans.010703, oos_Jul.3.2001
04 July 2001	alert.010704, scans.010704, oos_Jul.4.2001
05 July 2001	alert.010705, scans.010705, oos_Jul.5.2001

Overview

Some significant security issues must be address to improve the security posture of the Foo University network. It is understood that as a University there is a fine balance between usability and the freedom often demanded by academics, and maintaining a minimally acceptable security stance. Since access to Foo University personnel was not possible during the course of this audit, it was impossible to ascertain the security requirements (or lack thereof) of the organization.

When reviewing this document please be sure to consider all of the issues raised and recommendations made within the context of Foo University's organizational requirements. It is strongly recommended, however, that the level of security awareness and the security posture as a whole at Foo University be raised. The current environment represents a risk not only for the University's users but also for others in the Internet community who might be affected by security incidents occurring on the University's network.

Network Detects

The following is a list of the network detects reported in the log files listed above. The list is sorted by the number of detects each signature generated, with a brief description as to the purpose and intent of the signature. Note that since the actual Snort rule files that were used to generate the log files were not made available during the course of this audit, it was, at times, necessary to make an educated guess as to the intent and construction of the signature generating the detect. This was done through analysis of the actual log file entries and cross-referencing with existing material on sites such as www.sans.org, www.incidents.org, and www.securiteam.com.

SIGNATURE	ALERTS
POSSIBLE TROJAN SERVER ACTIVITY	*15408*

This signature appears to look for traffic destined to or sourced from 27374/tcp, a port commonly used by Trojan Horse programs such as SubSeven, Bad Blood, and Def Con 8.

UDP SRC AND DST OUTSIDE NETWORK	*10892*

UDP traffic where neither the source nor the destination IP address match the local network (HOMENET, presumably) cause this signature to fire. This can be an indication that source routing is in use, or that internal hosts are spoofing source addresses

WATCHLIST 000220 IL-ISDNNET-990517	*1808*

Traffic sourced from the IP network 212.179.0.0 (or a subset thereof) causes this signature to fire. According to RIPE, this network belongs to ISDN Net, Ltd. in Israel.

EXTERNAL RPC CALL	*1610*

Inbound calls to RPC (111/tcp) from external hosts are indicated by this signature. RPC has historically been the source of many vulnerabilities. There are many CVE entries related to this vulnerability. Among them are CVE-1999-0003, CVE-1999-0008, CVE-1999-0208, and CVE-1999-0212.

CONNECT TO 515 FROM OUTSIDE	*965*

Connections from external networks to the LPR port, 515/tcp. Serious vulnerabilities in certain versions LPR software (most notably, LPRng) make this signature of significant concern. [CVE-1999-0032, CVE-1999-0335]

SMB NAME WILDCARD	*475*

This signature detects attempts to enumerate shares on a system running SMB file sharing (usually Microsoft Windows platforms). This can be a precursor to an attack, as it can potentially reveal a great deal about the target system, such as usernames and poorly protected shared resources [CVE-1999-0225, CVE-1999-0391]

QUESO FINGERPRINTING	*307*

Different IP stacks will respond in different ways when they receive packets that are out-of-specification (e.g., unusual combinations of flags or options, such as setting both the SYN and FIN bits on the same packet). The response is used by programs such as Queso and nmap to "fingerprint" the remote operating system. Once the OS is identified, an attacker can narrow the scope of their attack [CAN-1999-0454]

SYN-FIN SCAN!	*273*

Sending packets with both the SYN and FIN bits set is an attempt to fingerprint remote systems by sending them a packet that is out-of-specification. As in a Queso scan, the response is used to help ascertain the operating system type.

SIGNATURE	ALERTS
WINGATE 1080 ATTEMPT	*270*

WinGate is a firewall/connection-sharing product by DeerField, Inc. that has had many vulnerabilities. This signature fires when it detects packets destined for port 1080/tcp, commonly used by the WinGate software. It is also conceivable that this may be a false positive caused by legitimate SOCKS traffic (which also uses port 1080/tcp) [CVE-1999-0290, CVE-1999-0291, CVE-1999-0441, CVE-1999-0494, CAN-1999-0657, CAN-2000-1048].

PORT 55850 TCP – POSSIBLE MYSERVER ACTIVITY – REF. 010313-1	*165*

Traffic from 55850/tcp triggers this signature. This traffic may be generated by or destined for a Trinoo type Trojan called MyServer.

ATTEMPTED SUN RPC HIGH PORT ACCESS	*91*

Sun RPC programs running on high ports (usually above 32000) have had a history of serious vulnerabilities. This signature appears to detect traffic destined for those port numbers, but also appears to look into the packet payload to determine that the RPC access attempt did not success (see later signature "SUNRPC highport access!") [CVE-1999-0003, CVE-1999-0008, CVE-1999-0208, and CVE-1999-0212].

NMAP TCP PING!	*96*

Fyodor's nmap uses TCP packets with the ACK bit set to detect hosts that are up and running. The packets are sent to the target host or network; hosts that are up and running should respond with a TCP RST, telling nmap that they're up and running. This allows an attacker to get past perimeter devices that may block standard ICMP pings.

SUNRPC HIGHPORT ACCESS!	*91*

Like the previous Sun RPC signature, this too looks for traffic destined for the high ports commonly used by RPC programs. However, this signature appears to look into the payload packet for signs of a successful access [CVE-1999-0003, CVE-1999-0008, CVE-1999-0208, and CVE-1999-0212].

WATCHLIST 000222 NET-NCFC	*90*

Traffic sourced from the IP network 159.226.0.0 /16 triggers this signature. This network belongs to the Computer Network Center at the Chinese Academy of Sciences.

TCP SRC AND DST OUTSIDE NETWORK	*72*

TCP traffic where neither the source nor the destination IP address match the local network cause this signature to fire. This can be an indication that source routing is in use, or that internal hosts are spoofing source addresses

NULL SCAN!	*69*

The Null scan turns off all flags in the TCP header (including the ACK flag). Operating systems that properly implement the TCP/IP standard should drop the packet with no response when sent to an open (listening) port, and send back a RST when sent to a port that is not listening. Microsoft and a few others have ignored the standard and always send back RSTs, so this scanning method is not effective against all hosts. Several of these Null scans, in this case, were in search of hosts running the peer-to-peer file sharing application Gnutella.

HIGH PORT 65535 TCP – POSSIBLE RED WORM – TRAFFIC	*58*

This signature looks for TCP traffic on the highest port, 65535. It is apparently attempting to detect the "Red Worm" a.k.a. Adore.

SIGNATURE	ALERTS
RUSSIA DYNAMO – SANS FLASH 28-JUL-00	*20*

Judging from the traces themselves, this signature appears to be looking for traffic destined to 1214/tcp – a port that is generally used by a peer-to-peer application called Kazaa. The SANS Flash referenced in the description refers to a notice concerning Trojans sending data to Russia.

HIGH PORT 65535 UDP – POSSIBLE RED WORM – TRAFFIC	*14*

This signature looks forUDP traffic on the highest port, 65535. It is apparently attempting to detect the "Red Worm" a.k.a. Adore.

BACK ORIFICE	*3*

These detects were triggered by an attempt to connect to 31337/tcp or 31337/udp (not possible to tell from the log file provided), the port commonly used by the Back Orifice Trojan.

TCP SMTP SOURCE PORT TRAFFIC	*3*

Presumably (again, limited information can be gained from the log files provided), this signature fires when a connection attempt is made sourced on TCP port 25 (i.e., a TCP SYN packet is sent to a host sourced from port 25).

CONNECT TO 515 FROM INSIDE	*2*

Connections from internal networks to the LPR port, 515/tcp. This is potentially serious as it could be an attempt to located vulnerable hosts running LRP [CVE-1999-0032, CVE-1999-0335].

STATDX UDP ATTACK	*1*

This signature fires when it detects traffic destined for the STATD RPC program. Severe, root-compromise level vulnerabilities exist in some versions of this software. A successful attack can result in a "fully owned" box for the attacker [CVE-1999-0018]

ICMP SRC AND DST OUTSIDE NETWORK	*1*

ICMP traffic where neither the source nor the destination IP address match the local network (HOMENET, presumably) cause this signature to fire. This can be an indication that source routing is in use, or that internal hosts are spoofing source addresses.

The Top Talkers (Alerts)

The following table lists the top ten "talkers" – meaning the IP addresses that generated the most alerts (not including port scans, which will be listed separately):

IP ADDRESS	ALERTS
169.254.161.0	5800
63.250.213.124	2331
63.250.213.26	1788
212.179.34.114	551
211.23.6.234	433
165.132.31.137	432
164.164.87.134	333
199.84.54.32	311
24.159.128.162	306
211.180.236.194	274

169.254.161.0

The most alerts were generated by the IP address 169.254.161.0, which is belongs to a Class B range of IP addresses called LINKLOCAL reserved by IANA / ICANN. Windows machines (and others perhaps) take an IP address from this range when they are configured to use DHCP and unable to get a lease from a DHCP server. The signature associated with all of these alerts was the **UDP SRC and DST outside network**. Since *all* of these alerts are destined for port 137, which is used by Microsoft's WINS name service, it is probably safe to assume that these alerts were generated by misconfigured Microsoft machines.

63.250.213.124 and 63.250.213.26

The next two IP addresses, 63.250.213.124 and 63.250.213.26, both belong to a network range allocated to Yahoo! Broadcast Services, which also helps explain their destination addresses, all of which belong to the multicast range (224.0.0.0 – 239.255.255.255). This also explains the Snort alert: **UDP SRC and DST outside network**. The snort sensors were apparently not configured to recognize multicast addresses as local.

212.179.34.114

With 551 entries, 212.179.34.114 is the fourth largest source of alerts. All of these alerts were generated by the **Watchlist 000220 IL-ISDNNET-990517** rule. With the exception of one (1) alert, all others were generated by traffic destined for 1214/tcp, a port used by the peer-to-peer file sharing software Kazaa. The other alert was generated by traffic destined for 6346/tcp, a port typically used by another peer-to-peer file sharing program called Gnutella. They do not appear to be malicious, but rather the actual applications themselves. Note that this cannot be definitively determined with the data provided for this security audit.

211.23.6.234

These detects are *significantly* more disturbing than the previous. According to the log files, this host performed a scan of part of the MY.NET.0.0 network looking for hosts running RPC Portmapper (111/tcp). At the end of the scan, the attacker apparently tried to launch an attack using the STATD vulnerability against an identified target:

```
07/01-08:47:19.012259  [**] External RPC call [**] 211.23.6.234:4599 -
> MY.NET.137.170:111

07/01-08:47:19.094212  [**] External RPC call [**] 211.23.6.234:4605 -
> MY.NET.137.176:111

07/01-09:00:37.454441  [**] STATDX UDP attack [**] 211.23.6.234:835 ->
MY.NET.6.15:32776
```

Although the target was not scanned in the log files that were used in this audit, it is very conceivable that the attacker may have foot-printed the target in a previous scan. It is recommended that the host MY.NET.6.15 be checked immediately for signs of compromise.

165.132.31.137

These alerts were generated by yet another scan, this time looking for servers running vulnerable LPR services on port 515/tcp. A review for the scan files reveals that this is a simplistic TCP SYN scan, executed very quickly (within one minute). There were no correlating entries in the OOS files. The originating host is located in Korea at a university.

164.164.87.134 and 199.84.54.32

Another RPC scan, this time the attacker has cleverly randomized the destination IP address so that the hosts are not sequentially scanned. Nevertheless, still a very noisy scan. No evidence of OS fingerprinting and no evidence of compromise is present in the analyzed log files. The first IP address, 164.164.87.134 is registered to Software Technology Park in Bangalore, India, and the second to Babilliard Synapse Inc. in Quebec, Canada.

24.159.128.162

Some *very* disturbing output, indeed – this host performed a scan of the MY.NET.111.0 and MY.NET.112.0 subnets in search of Trojans, most likely SubSeven, and apparently found many infected hosts! The following output snippet indicates responses being sent back to the scanning host:

```
07/02-19:49:42.769754  [**] Possible trojan server activity [**]
MY.NET.111.75:27374 -> 24.159.128.162:3624

<snip>

07/02-19:51:50.257592  [**] Possible trojan server activity [**]
MY.NET.112.100:27374 -> 24.159.128.162:3894
```

The origin of the scan is from a network registered to Charter Communications.

211.180.236.194

Interesting detects generated from this source IP address. The first five alerts were generated by traffic destined for MY.NET.6.15 (previously seen to be the possible victim of a STATD attack, and thus potentially compromised).

```
07/03-13:12:10.697188  [**] External RPC call [**] 211.180.236.194:111
-> MY.NET.6.15:111
```

```
07/03-13:12:11.599415  [**] External RPC call [**]
211.180.236.194:2385 -> MY.NET.6.15:111

07/03-13:12:11.605117  [**] External RPC call [**] 211.180.236.194:941
-> MY.NET.6.15:111

07/03-13:12:12.151780  [**] External RPC call [**]
211.180.236.194:2385 -> MY.NET.6.15:111

07/03-13:12:12.409276  [**] External RPC call [**] 211.180.236.194:941
-> MY.NET.6.15:111
```

The first packet is 111/tcp -> 111/tcp, and may be part of the port scan. The four subsequent packets are sources on different ports, two from 2385 and two from 941. This may a Portmapper call enumerating RPC services, or something more sinister.

Approximately 30 minutes later, a full-on SYN-FIN scan begins, sourced on and destined for port 111/tcp. IP IDs, SYN and ACK sequence numbers are identical across the scan as evidenced in the corresponding OOS log file:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/03-13:21:11.894885 211.180.236.194:111 -> MY.NET.132.22:111
TCP TTL:25 TOS:0x0 ID:39426
**SF**** Seq: 0x572E1953   Ack: 0x39977805   Win: 0x404
00 00 00 00 00 00                                ......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/03-13:21:11.939228 211.180.236.194:111 -> MY.NET.132.24:111
TCP TTL:25 TOS:0x0 ID:39426
**SF**** Seq: 0x572E1953   Ack: 0x39977805   Win: 0x404
00 00 00 00 00 00                                ......

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/03-13:21:11.974906 211.180.236.194:111 -> MY.NET.132.26:111
TCP TTL:25 TOS:0x0 ID:39426
**SF**** Seq: 0x572E1953   Ack: 0x39977805   Win: 0x404
00 00 00 00 00 00
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The source IP address of 211.180.236.194 is registered to the Korean NIC. The host is an unhardened Linux box running kernel version 2.2.12 (it even has telnet enabled).

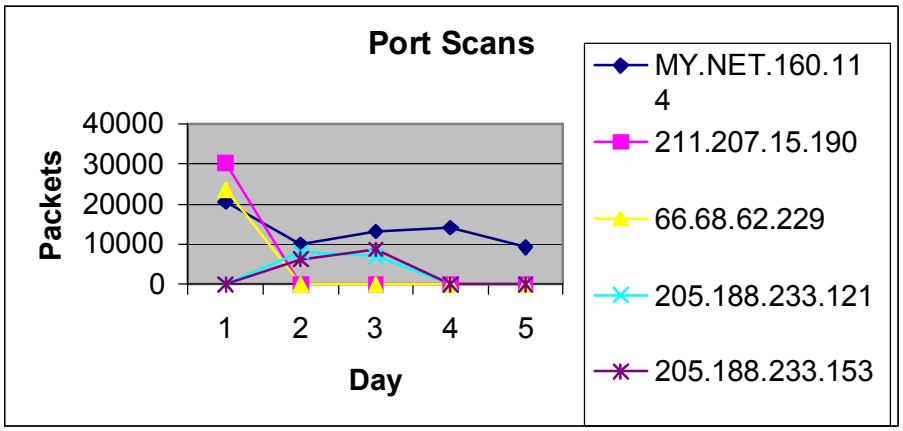
Top Port Scanners

The following table lists the top five scanners – meaning the IP addresses that generated the most port scan log entries.

IP Address	Scan Log Entries
MY.NET.160.114	66991
211.207.15.190	30156
66.68.62.229	23501
205.188.233.121	15059
205.188.233.153	14921

The internal host MY.NET.160.114 should be investigated – as the highest single source of port scans, it may be compromised.

The link graph below indicates the distribution of the port scan detects for the top five scanners across the five days of log files. Of particular interest is the fact that the two AOL IP addresses (205.188.233.X) both performed their scans on the same days – although it may also be obvious since they are from the same IP network, and were performing the same type of scan, it is now very clear that it is the same attacker.



Top Sources of Out-of-Spec Packets

The following table lists the top five sources of out-of-specification packets.

IP Address	OOS Entries
211.180.236.194	557
210.77.146.33	390
199.183.23.194	175
24.66.152.186	108
216.5.180.10	41

Attacker Information

The following table includes registration information regarding ten IP addresses that warrant particular attention because of the their potentially hostile activity on the Foo University network. The first five are the top external generators of non-portscan preprocessor alerts, and the second five are the top external generators of portscan entries. Note that they are not necessarily in order of importance or severity.

IP Address	Registration Information	Comments
211.23.6.234	Chiao Chu Co., Ltd. No. 10, Lane 154 Iang So St. Taoyuan, Taiwan 211.23.6.232 /29 Source: whois.twnic.net	This IP address generated the most overtly hostile alerts. In addition to RPC scanning, there is also some evidence that a compromise may have taken place (the STATDX UDP attack signature was triggered). At the time of this writing, this IP address is an unprotected Red Hat Linux box running Red Hat 7.0.
165.132.31.137	Yonsei University 134, Shinchon-dong, Seodaemnu-gu Seoul, 120-749 Korean Source: www.arin.net	This IP address was the second largest clearly hostile generator of alerts, performing fast scans of 515/tcp presumably looking for vulnerable LPR services to exploit.
164.164.87.134	Software Technology Park-Block III, KSSIDC Complex Keonics Electronics City Bangalore 562 158 India Source: www.arin.net	The source of randomized RPC scans, and third largest generator of non-portscan preprocessor alerts.
199.84.54.32	Babilliard Synapse Inc. 22 Beloeil Gatineau, Quebec; J8T7G3 Canada Source: www.arin.net	The source of randomized RPC scans, and fourth largest generator of non-portscan preprocessor alerts.
24.159.128.162	Charter Communications 12444 Powerscourt Drive St. Louis, MO 63131 USA Source: www.arin.net	This host performed a Trojan scan in search of SubSeven, and apparently found some infected hosts. Definitely one to keep an eye on.
211.207.15.190	Hanaro Telecom Inc. (DSL) 1445-3 Seocho-Dong Seocho-Gu 137-728 Seoul, Korea +82-2-106 Source: whois.nic.or.kr	This IP address generated a massive amount of portscan entries, sweeping a huge chunk of the MY.NET.0.0 /16 subnet for FTP servers (SYN scan to 21/tcp).

66.68.62.229	Roadrunner Southwest 13241 Woodland Park Road Herndon, VA 20171 USA Source: www.arin.net	A focused port scan against MY.NET.219.42 was launched from this host. Since the scan was targeted, activity from this host should be watched closely.
205.188.233.121 205.188.233.153	America Online, Inc. 22080 Pacific Blvd Sterling, VA 20166 USA Source: www.arin.net	AOL users scanning a large range of hosts for 6970/udp – probably looking for the GateCrasher Trojan.
148.223.228.15	UniNet S.A. de C.V Periferico Sur #31900 Jardines del Pedregal Mexico, D.F. 01900 Mexico Source: www.arin.net	This IP is registered to a Mexican ISP. It SYN scanned a large range of IP addresses in the MY.NET.0.0 /16 block looking for DNS servers (53/tcp).
61.222.34.170	Data Communication Business Group Chunghwa Telecom Co., Ltd. 21, Section 1, Hsin-Yi Road Tapei 100 Taiwan Source: whois.twnic.net	A cursory examination of the IP address revealed that it is a Red Hat Linux 7.0 host, running Apache 1.3.12. The Apache test pages have not been removed, so they provided the hostname: mail.eurasian.com.tw, which was confirmed by forward DNS lookup. This host, like the previous, performed SYN scans of a large range of addresses looking for DNS servers (53/tcp)..

Correlations

Many correlations exist with previous audit carried out for Foo University by other GIAC professionals. Some notable correlations are listed below:

- Robert Sorenson's practical had several correlations to data analyzed in this document. On the more interesting, is the STATDX attack to MY.NET.6.15, but from a different source IP address. Robert's detect was generated by 206.210.80.6, while the detect reported in this document was sourced from 211.23.6.234. This is more evidence that the host MY.NET.6.15 is either compromised, or running a service that appears to vulnerable to the STATD exploit (at least to the attackers).
- Many other practicals reported high numbers of alerts generated by the **Watchlist 000220 IL-ISDNNET-990517** signature.
- Roderick Campbell noted a large number of **Connect to 515 from Outside** alerts, noting that because of the vulnerabilities associated with LPR (which runs on port 515/tcp) this port should be blocked.

Defensive Recommendations

As has been demonstrated in just five days worth of intrusion detection data, there are some very serious security issues that Foo University needs to address. The following steps should be taken as soon as possible to increase the security of the Foo University network:

- ❒ The network perimeter must be clearly defined, with all points of egress and ingress into the network identified and secured with a perimeter security device such as a firewall, or, at an absolute minimum, a packet filtering router;
- ❒ Firewalls and/or routers should be configured to pass the minimally required traffic inbound, and should restrict outbound traffic to required services (e.g., HTTP, FTP, NNTP). Restricting traffic to only the required services will help prevent insecure applications from being used on the network. An application proxy gateway may further mitigate this by ensuring the data passing over a particular port is what it's supposed to be, for example non-HTTP protocols will not be able to pass the perimeter device using 80/tcp.
- ❒ Institute a comprehensive virus remediation program to address the issue of Trojan and other virus infection. Specifically, ensure that the infected hosts in the subnets MY.NET.111.0 and MY.NET.112.0 are cleaned.
- ❒ Ensure Internet-accessible hosts (which again, should be an absolute minimal subset of Foo University's machines) have the latest security hot-fixes and patches, paying particular attention to the traditional intrusion vectors: DNS and WWW servers. Follow industry best practices for hardening the exposed hosts against attack from the Internet, by disabling unnecessary services, removing unneeded software, and installing third party security tools to enhance the operating systems defensive mechanisms.
- ❒ Immediately verify that the host MY.NET.6.15, discussed in the "top talkers" section, is free from compromise. There is some evidence to suggest that the box may have fallen victim to a RPC vulnerability.
- ❒ Immediately verify that the host MY.NET.160.114, discussed in the "top talkers" section, is free from compromise. This host was the source of numerous port scans.
- ❒ Routinely review logs from intrusion detection sensors as well the firewalls.

Analysis Process

In order to ensure that the analysis process yielded practical results that would be useful to the technical staff at Foo University as a tool to help improve the security of their network, I chose to analyze five days worth of the vast amount of intrusion detection data that was provided for the practical. This is, of course, a snapshot in time but it provides a great deal of information regarding security infrastructure problems that need to be addressed.

There were several key tools used in the process of analysis the intrusion detection data:

- ❒ SnortSnarf v52301.1, available at http://www.silicondefense.com. This tool was extremely helpful in analyzing the Snort output logs.
- ❒ snort_stat.pl, by Yen-Ming Chen <chenym@alumni.cmu.edu>. This tool was not natively able to read the munged format that the snort logs were provided in, and required some modification in order to get the desired results.

- UNIX utilities such as perl, grep, awk, sed, sort, and uniq were all very important in the analysis of this data.

The first step in analysis the data was to concatenate the five days of alert log files, and change the MY.NET to 77.77 (all log files were previously grep'ed to make sure that 77.77 did not really exist) so it will be readable by the SnortSnarf scripts. The commands used to do this were present in many practicals:

```
for $afile in `ls alert.*`
do
      cat $afile | sed 's/MY.NET/77.77/g' >> master.alerts
done
```

The same process was followed for the scan and OOS log files. To generate the "top" lists, including the "Top Talkers," "Top Scan Sources" and "Top OOS Sources", a few different perl scripts were used in combination with some command line UNIX utilities. To grab the source IP address from the master.alert log file, and calculate the top talkers, the following scripts/commands were used (note that these perl scripts could also have been constructed in a single line):

```
sourceipalerts.pl:
while (<>) {
        if ($_ =~ m/\[\*\*\]\s([\d\.]+)/ox) {
                print $1 . "\n" ;
        }
}
perl sourceipalerts.pl < master.alerts | sort | uniq -c | \
      sort -n -r > toptalkers.txt
```

For the top scan sources:

```
sourceipscans.pl:
while (<>) {
        if ($_ =~ m/^\w{3}\s+\d+\s\d+\:\d+\:\d+\s([\d\.]+)/ox) {
                print $1 . "\n" ;
        }
}
perl sourceipscans.pl < master.scans | sort | uniq -c | \
      sort -n -r > topscanners.txt
```

For the top OOS sources:

```
sourceipoos.pl:
while (<>) {
      if ($_ =~ m/\.\d{6}\s([\d\.]+)/ox) {
              print $1 . "\n" ;
        }
}
perl sourceipscans.pl < master.oos | sort | uniq -c | \
      sort -n -r > topoossrc.txt
```

REFERENCES

American Heritage Dictionary of the English Language, Fourth Edition. Houghton Mifflin Company, 2000: New York, NY.

Bayerkohler, Marc. SANS Intrusion Detection Practical.

Bidwell, Teri. GIAC Network Intrusion Detection GCIA Practical.

Campbell, Roderick. GIAC Intrusion Analyst Certification Practical Submittal. URL: http://www.sans.org/y2k/practical/Roderick_Campbell_GCIA.doc (30 August 2001).

Common Vulnerabilities and Exposures. URL: http://cve.mitre.org (30 August 2001).

Comer, Douglas. Operating System Design: The XINU Approach. Prentice-Hall, Inc., 1984: Englewood Cliffs, NJ.

DeShon, Markus. Practical for SANS DC 2000.

Garcia, Andy. "Intercepting Intrusions with Entercept." 26 March 2001. URL: http://www.sans.org/infosecFAQ/intrusion/entercept.htm (15 August 2001).

Goodwin, PJ. GIAC Level 2 Practical Assignment for Capitol SANS.

Hollander, Yona. "The Future of Web Server Security." URL: http://www.entercept.com/products/entercept/whitepapers/wpfuture.asp (15 August 2001).

Incidents.org. URL: http://www.incidents.org (15 August 2001).

OKENA. "StormWatch: A Technical White Paper." URL: http://www.okena.com (15 August 2001).

SANS Institute. URL: http://www.sans.org (15 August 2001).

Schupp, Steve. "Limitations of Network Intrusion Detection." 1 December 2000. URL: http://www.sans.org/infosecFAQ/intrusion/net_id.htm (15 August 2001).

Sorensen, Robert. Practical Assignment for SANS Security New Orleans 2001. URL: http://www.sans.org/y2k/practical/Robert_Sorensen_GCIA.htm (30 August 2001).

Stevens, W. Richard. TCP/IP Illustrated: Volume I. Reading: Addison-Wellesly, 1994.

WatchGuard Technologies, Inc. "Protecting Corporate Information Assets: The Architecture of ServerLock." (April 2001). URL: http://www.watchguard.com/docs/sl_whitepaper.pdf (15 August 2001).

Zeltser, Lenny. Practical Assignment for SANS Security DC 2000.

APPENDIX A: MODIFIED SNORT_STAT PERL SCRIPT

```
#
# $Author: yenming $
# Yen-Ming Chen, <chenym@ALUMNI.CMU.EDU>
# $Date: 2001/07/31 14:51:29 $
#
# Angelos Karageorgiou, <angelos@unix.gr>
# contributed the DNS resolve and cache
#
# Andrew R. Baker <andrewb@uab.edu>
# 2000.03.06 - modifications to read snort alert file
#            - added html output option
#
# Paul Bobby, <paul.bobby@lmco.com>
# 03/13/2000 added scan for portscan detection in logs
#
# Ned Patterson, <jpatter@alum.mit.edu>
# 4/26/2000 - correctly parse "last message repeated" syslog messages
#           - variable column widths for text output
#
# Ryan Jian-Da Li, <jdli@freebsd.csie.nctu.edu.tw>
# 6/07/2000 - fix the problem of portscan() (add my %s5)
#           - fix the problem of signature matching
#             for the case ' IDS154 - PING CyberKit 2.2 Windows'
#           - enhance portscan(), add port counts
#
# James Conz
# 7/15/2001 - Modified for use in SANS GCIA practical v2.9
#

use Getopt::Std;                # use Getopt for options
use Socket;                       # use socket for resolving domain name from
IP
use vars qw($opt_r $opt_f $opt_a $opt_h $opt_p $opt_n $opt_t);
%HOSTS = ();                    # Hash for IP <-> domain name mapping

getopts('rfht:') || die "Could not getopts"; # get options in command line
$saddr_len = 15;
$daddr_len = 15;
$timeout = 3;                   # for name resolver
$th = $opt_t || 0;              # default threshold

# process whatever comes in
while (<>) {
  my $alert = {};
  chomp;
  # if the line is blank, go to the next one
  next if $_ eq "";
  # is this line an alert message
  if ( $_ =~ /^\[\*\*\]/ ) {
    $line = <>;
    chomp($line);
    unless ( $line eq "" ) {
      # strip off the [**] from either end.
      s/(\s)*\[\*\*\](\s)*//og;
      s/\s*\[[0-9:]+\]\s*//o;
```

```
      if ($_ =~ /^spp_anomsensor\:[\D]+\:\s([\d\.]+)/ox) {
        $alert->{PLUGIN} = "anomsensor"; $alert->{TYPE} = "plugin";
        $alert->{SIG} = $alert->{PLUGIN};
      } elsif ($_ =~ /^spp_portscan\:\sEnd\sof\sportscan\sfrom\s([\d\.]+)/ox)
{
        $alert->{PLUGIN} = "portscan"; $alert->{TYPE} = "plugin";
        $alert->{SADDR} = $1; $alert->{SIG} = $alert->{PLUGIN};
        process_data($alert); $lastwassnort = 1; next;
      } elsif ( $_ =~ /[^:]/ox) {
        $alert->{SIG} = $_; $alert->{TYPE} = "alert";
        $alert->{PLUGIN} = "none";
      }
      if                    (                   $line                   =~
m/^\[Classification\:([^\]]*)\]\s\[Priority\:\s(\d+)\]/ox) {
        $alert->{CLASS} = $1; $alert->{CONTENT} = $2; $alert->{PRIORITY} =
$3;
        $line=<>;
      }
      if ( $line =~ m/^(\d+)\/(\d+)\-(\d+)\:(\d+)\:(\d+)\.(\d+)\s
           ([\d\.]+)[\:]*([\d]*)\s[\-\>]+\s([\d\.]+)[\:]*([\d]*)/ox) {
        $alert->{MON} = $1;   $alert->{DAY} = $2;   $alert->{HOUR} = $3;
        $alert->{MIN} = $4;   $alert->{SEC} = $5;   $alert->{SADDR} = $7;
        $alert->{SPORT} = $8; $alert->{DADDR} = $9; $alert->{DPORT} = $10;
        $alert->{HOST} = "localhost";
        process_data($alert); $lastwassnort = 1; next;
      }
    } else {
      print STDERR "Warning, file may be incomplete\n";
      next;
    }
  }
  # The following code modified by James Conz for the purposes
  # of the SANS GCIA Practical v2.9, Summer 2001. This code has
  # been changed to read the stripped down snort file format used
  # in the practical. It's not necessarily elegant, but it works ;-)
  #
  # JC
  #
  if (($_ =~ m/^(\d{2})\/(\d{2})\-(\d{2})\:(\d{2})\:(\d{2})\.\d{6}
      \s+\[\*\*\]\s([^\[\*\*\]]+)\[\*\*\]\s+
      ([\d\.]+)[\:]?([\d]*)\s[\-\>]+\s([\d\.]+)[\:]?([\d]*)/ox)
      && ($_ !~ m/spp_portscan/ox)) {
    $alert->{MON} = $1;    $alert->{DAY} = $2;    $alert->{HOUR} = $3;
    $alert->{MIN} = $4;    $alert->{SEC} = $5;    $alert->{HOST} =
"localhost";
    $alert->{SIG} = $6;    $alert->{CLASS} = "";  $alert->{PRIORITY} = "";
    $alert->{SADDR} = $7; $alert->{SPORT} = $8;   $alert->{DADDR} = $9;
    $alert->{DPORT} = $10; $alert->{TYPE} = "sys";
    $alert->{PLUGIN} = "none";
    process_data($alert); $lastwassnort = 1; next;
  } elsif ($_ =~ m/^(\d{2})\/(\d{2})\-(\d{2})\:(\d{2})\:(\d{2})\.\d{6}
          \s+\[\*\*\]\sspp_portscan\:\sEnd\sof\sportscan\sfrom\s
          ([\d\.]+)/ox) {
    $alert->{MON} = $1;   $alert->{DAY} = $2; $alert->{HOUR} = $3;
    $alert->{MIN} = $4;   $alert->{SEC} = $5; $alert->{HOST} = $6;
    $alert->{SADDR} = $7; $alert->{TYPE} = "plugin";
    $alert->{PLUGIN} = "portscan";
    process_data($alert); $lastwassnort = 1; next;
  } elsif ($_ =~ m/^(\w{3})\s+(\d+)\s(\d+)\:(\d+)\:(\d+)\s([\w+\.]*)\s
```

```
[\w+\/\[\d+\]]*\:\sspp_anomsensor\:\sAnomaly\sthreshold\sexceeded\:
          \s([\d\.]+)\:\s([\d\.]+)\:([\d]+)\s[\-\>]+\s([\d\.]+)\:([\d]+)/ox)
{
    $alert->{MON} = $1;    $alert->{DAY} = $2;   $alert->{HOUR} = $3;
    $alert->{MIN} = $4;    $alert->{SEC} = $5;   $alert->{HOST} = $6;
    $alert->{SADDR} = $8;  $alert->{SPORT} = $9; $alert->{DADDR} = $10;
    $alert->{DPORT} = $11; $alert->{THR} = $7;
    $alert->{TYPE} = "plugin"; $alert->{PLUGIN} = "anomsensor";
    process_data($alert); $lastwassnort = 1; next;
  }
  # If a snort message has been repeated several times
  elsif ($lastwassnort && $_ =~ m/last message repeated (\d+) times/) {
    # put the data in the matrix again for each repeat
    $repeats = $1;
    while ($repeats) {
      push @result, $result[-1];
      $repeats--;
    }
    next;
  } else {
    $lastwassnort = 0;
    next;
  }                             # Message not related to snort
}

# begin statistics
# I should've used $#result + 1 as $total in the first version! :(
$total = $#result + 1;

for $i ( 0 .. $#result ) {
  # for the same pair of attacker and victim with same sig
  # to see the attack pattern
  # used in same_attack()
  $s0{"$result[$i]->[9]:$result[$i]->[7]:$result[$i]->[6]"}++;
  # for the same pair of attacker and victim
  # to see how many ways are being tried
  # used in same_host_dest()
  $s1{"$result[$i]->[7]:$result[$i]->[9]"}++;
  # from same host use same method to attack
  # to see how many attacks launched from one host
  # used in same_host_sig()
  $s2{"$result[$i]->[6]:$result[$i]->[7]"}++;
  # to same victim with same method
  # to see how many attacks received by one host
  # used in same_dest_sig_stat()
  $s3{"$result[$i]->[6]:$result[$i]->[9]"}++;
  # same signature
  # to see the popularity of one attack method
  # used in attack_distribution()
  $s4{"$result[$i]->[6]"}++;
  # source ip
  $s5{"$result[$i]->[7]"}++;
  # destination ip
  $s6{"$result[$i]->[9]"}++;
}

# begin report
```

```
print_head();
print_summary();
print_menu();
same_attack();
same_host_dest();
same_host_sig();
same_dest_sig_stat();
attack_distribution();
if ($opt_p) {
  portscan();
}
if ($opt_n) {
  anomsensor();
}
print_footer();

# print the header (e.g. for mail)
sub print_head {
  if ($opt_h) {
    print "<html>\n<head>\n";
    print "<title>Snort Statistics</title>";
    print "</head>\n<body>\n";
    print "<h1>Snort Statistics</h1>\n";
  } else {
    print "Subject: snort daily report\n\n";
  }
}

# print the time of begin and end of the log
sub print_summary {
  if ($opt_h) {
    print "<table>\n";
    print "<tr><th>The log begins at:</th>\n";
    print  "<td>$result[0]->[0]  $result[0]->[1]  $result[0]->[2]:$result[0]-
>[3]:$result[0]->[4]</td></tr>\n";
    print "<tr><th>The log ends at:</th>\n";
    print          "<td>$result[$#result]->[0]          $result[$#result]->[1]
$result[$#result]->[2]:$result[$#result]->[3]:$result[$#result]-
>[4]</td></tr>\n";
    print "<tr><th>Total events:</th><td> $total</td></tr>\n";
    print     "<tr><th>Signatures     recorded:</th><td>     ".     keys(%s4)
."</td></tr>\n";
    print "<tr><th>Source IP recorded:</th><td> ". keys(%s5) ."</td></tr>\n";
    print    "<tr><th>Destination    IP    recorded:</th><td>    ".    keys(%s6)
."</td></tr>\n";
    print    "<tr><th>Portscan    detected:</th><td>    ",    eval    '$#posres
+1',"</td></tr>\n" if $opt_p;
    print    "<tr><th>Anomaly    detected:</th><td>    ",    eval    '$#anores
+1',"</td></tr>\n" if $opt_n;
    print "</table>\n";
    print "<hr>\n";
  } else {
    print "The log begins from: $result[0]->[0] $result[0]->[1] $result[0]-
>[2]:$result[0]->[3]:$result[0]->[4]\n";
    print "The log ends     at: $result[$#result]->[0] $result[$#result]->[1]
$result[$#result]->[2]:$result[$#result]->[3]:$result[$#result]->[4]\n";
    print "Total events: $total\n";
    print "Signatures recorded: ". keys(%s4) ."\n";
    print "Source IP recorded: ". keys(%s5) ."\n";
```

```
    print "Destination IP recorded: ". keys(%s6) ."\n";
    print "Portscan recorded: ", eval '$#posres +1',"\n" if $opt_p;
    print "Anomaly recorded: ", eval '$#anores +1',"\n" if $opt_n;
  }
}

# print menu for HTML page
sub print_menu {
  if ($opt_h) {
    print "<ul><a name=\"top\"></a>\n";
    print "<li><a href=\"#same_hdm\">Number of attacks from same host to same
destination with same method</a>\n";
    print "<li><a href=\"#same_hd\">Percentage and number of attacks from a
host to a destination</a>\n";
    print "<li><a href=\"#same_hm\">Percentage and number of attacks from one
host to any with same method</a>\n";
    print "<li><a href=\"#same_d\">Percentage and number of attacks to one
certain host</a>\n";
    print "<li><a href=\"#same_m\">Distribution of attack methods</a>\n";
    print    "<li><a    href=\"#portscan\">Portscans    performed    to/from
HOME_NET</a>\n" if $opt_p;
    print "<li><a href=\"#spade\">Anomaly detected by SPADE</a>\n" if $opt_n;
    print "</ul><HR>\n";
  }
}

# to see the frequency of the attack from a certain pair of
# host and destination
sub same_attack {
  if ($opt_h) {
    print "<h3><a name=\"same_hdm\">Number of attack from same host to same
destination using same method</a></h3>\n";
    print "<table>\n";
    print                                "<tr><th>#                                of
attacks</th><th>from</th><th>to</th><th>with</th></tr>";
    foreach $k (sort { $s0{$b} <=> $s0{$a} } keys %s0) {
      @_ = split ":",$k;
      print "<tr><td>$s0{$k}</td><td>$_[1]</td><td>$_[0]</td>
             <td>$_[2]</td></tr>\n" if $s0{$k} > $th;
    }
    print "</table><a href=\"#top\">Top</a><hr>\n";
  } else {
    section_header("The number of attacks from same host to same
destination using same method\n", "asdm");
    foreach $k (sort { $s0{$b} <=> $s0{$a} } keys %s0) {
      @_ = split ":",$k;
      printf("   %-2d     %-${saddr_len}s   %-${daddr_len}s   %-20s\n",
             $s0{$k},$_[1],$_[0],$_[2]) if $s0{$k} > $th;
    }
  }
}

# to see the percentage and number of attacks from a host to a destination
sub same_host_dest {
  if ($opt_h) {
    print "<h3><a name=\"same_hd\">Percentage and number of attacks from a
host to a destination</a></h3>\n";
    print "<table>\n";
```

```
    print                          "<tr><th>%</th><th>#                          of
attacks</th><th>from</th><th>to</th></tr>\n";
    foreach $k (sort { $s1{$b} <=> $s1{$a} } keys %s1) {
      @_ = split ":",$k;
      printf("<tr><td>%-2.2f</td><td>%-2d</td><td>%-20s</td><td>%-20s</td>
             <td>\n",$s1{$k}/$total*100,$s1{$k},$_[0],$_[1])  if  $s1{$k}  >
$th;
    }
    print "</table><a href=\"#top\">Top</a><hr>\n";
  } else {
    section_header("Percentage and number of attacks from a host to a
destination\n", "pasd");
    foreach $k (sort { $s1{$b} <=> $s1{$a} } keys %s1) {
      @_ = split ":",$k;
      printf("%5.2f    %-2d      %-${saddr_len}s   %-${daddr_len}s\n",
            $s1{$k}/$total*100, $s1{$k},$_[0],$_[1]) if $s1{$k} > $th;
    }
  }
}

# to see how many attacks launched from one host
sub same_host_sig {
  if ($opt_h) {
    print "<h3><a name=\"same_hm\">Percentage and number of attacks from one
host to any with same method</a></h3>\n";
    print "<table>\n";
    print                          "<tr><th>%</th><th>#                          of
attacks</th><th>from</th><th>type</th></tr>\n";
    foreach $k (sort { $s2{$b} <=> $s2{$a} } keys %s2) {
      @_ = split ":",$k;
      printf("<tr><td>%-2.2f</td><td>%-4d</td><td>%-20s</td><td>%-28s</td>
            </tr>\n",$s2{$k}/$total*100,$s2{$k},$_[1],$_[0])  if  $s2{$k}  >
$th;
    }
    print "</table><a href=\"#top\">Top</a><hr>\n";
  } else {
    section_header("Percentage and number of attacks from one host to any
with same method\n", "pasm");
    foreach $k (sort { $s2{$b} <=> $s2{$a} } keys %s2) {
      @_ = split ":",$k;
      printf("%5.2f    %-4d    %-${saddr_len}s   %-28s\n",
            $s2{$k}/$total*100, $s2{$k},$_[1],$_[0]) if $s2{$k} > $th;
    }
  }
}

# to see how many attacks received by one host (destination correlated)
sub same_dest_sig_stat {
  if ($opt_h) {
    print "<h3><a  name=\"same_d\">Percentage  and  number  of  attacks  to  one
certain host</a></h3>\n";
    print "<table>\n";
    print                          "<tr><th>%</th><th>#                          of
attacks</th><th>to</th><th>type</th></tr>\n";
    foreach $k (sort { $s3{$b} <=> $s3{$a} } keys %s3) {
      @_ = split ":",$k;
      printf("<tr><td>%-2.2f</td><td>%-4d</td><td>%-25s</td><td>%-
28s</td><td>\n",$s3{$k}/$total*100,$s3{$k},$_[1],$_[0]) if $s3{$k} > $th;
    }
```

```
    print "</table><a href=\"#top\">Top</a><hr>\n";
  } else {
    section_header("Percentage and number of attacks to one certain host \n",
"padm");
    foreach $k (sort { $s3{$b} <=> $s3{$a} } keys %s3) {
      @_ = split ":",$k;
      printf("%5.2f    %-4d    %-${daddr_len}s  %-28s\n",$s3{$k}/$total*100 ,
             $s3{$k},$_[1],$_[0]) if $s3{$k} > $th;
    }
  }
}

# to see the popularity of one attack method
sub attack_distribution {
  if ($opt_h) {
    print "<h3><a name=\"same_m\">Distribution of attack methods</a></h3>\n";
    print "<table>\n";
    print "<tr><th>%</th><th># of attacks</th><th>methods</th></tr>\n";
    foreach $k (sort { $s4{$b} <=> $s4{$a} } keys %s4) {
      @p1 = split ":",$k;
      if ($s4{$k} > $th) {
        printf("<tr><td>%-2.2f</td><td><B>%-4d</B></td><td><B>%-32s</B></td>
                </tr>\n", $s4{$k}/$total*100,$s4{$k},$p1[0]);
        foreach $k2 (sort { $s0{$b} <=> $s0{$a} } keys %s0) {
          @p2 = split ":",$k2;
          printf("<tr><td></td><td>%-4d</td><td>%-32s</td></tr>\n",  $s0{$k2},
join(" -> ", $p2[1],$p2[0])) if $p1[0] eq $p2[2];
        }
      }
    }
    print "</table><a href=\"#top\">Top</a><hr>\n";
  } else {
    section_header("The distribution of attack methods\n", "pam");
    foreach $k (sort { $s4{$b} <=> $s4{$a} } keys %s4) {
      @p1 = split ":",$k;
      if ($s4{$k} > $th) {
        printf("%5.2f                    %-4d                    %-32s\n",
$s4{$k}/$total*100,$s4{$k},$p1[0]);
        foreach $k2 (sort { $s0{$b} <=> $s0{$a} } keys %s0) {
          @p2 = split ":",$k2;
          printf("\t\t %-4d  %-${saddr_len}s -> %-${daddr_len}s\n", $s0{$k2},
$p2[1],$p2[0]) if $p1[0] eq $p2[2];
        }
      }
    }
  }
}

# portscan (if enable -p switch)
# Please use '-A fast' to generate the log, so portscan() can process it.
# contributed by: Paul Bobby, <paul.bobby@lmco.com>
#                 Jian-Da Li, <jdli@freebsd.csie.nctu.edu.tw>
sub portscan {
  my (%s7, %s8);
  # to see how many times a host performs portscan
  # used in portscan()
  for $i (0 .. $#posres) {
    $s7{"$posres[$i]->[0]"}++;
  }
```

```
  if ($opt_h) {
    print     "<h3><a     name=\"portscan\">Portscans     performed     to/from
HOME_NET</a></h3>\n";
    print "<table>\n";
    print "<tr><th>Scan Attempts</th><th>Source Address</th></tr>\n";
    foreach $k (sort { $s7{$b} <=> $s7{$a} } keys %s7) {
      print "<tr><td>$s7{$k}</td><td>$k</td></tr>\n" if $s7{$k} > $th;
    }
    print "</table><a href=\"#top\">Top</a><HR>\n";
  } else {
    section_header("Portscans performed to/from HOME_NET\n", "as");
    foreach $k (sort { $s7{$b} <=> $s7{$a} } keys %s7) {
      printf(" %-4d    %-${saddr_len}s\n", $s7{$k},$k) if $s7{$k} > $th;
    }
  }
}

# anomsensor (if enable -n switch)
# This function process data generated by spp_anomsensor plug-in (SPADE)
# By Yen-Ming Chen <chenym@alumni.cmu.edu>
sub anomsensor {
  my (%s7);
  # to see how many times a host performs portscan
  # used in anomsensor()
  for $i (0 .. $#anores) {
    $s7{"$anores[$i]->[1],$anores[$i]->[3],$anores[$i]->[4]"}++;
  }
  if ($opt_h) {
    print "<h3><a name=\"spade\">Anomaly detected by SPADE</a></h3>\n";
    print "<table>\n";
    print  "<tr><th>Scan  Attempts</th><th>Source  Address</th><th>Destination
Address</th><th>Destination Ports</th></tr>\n";
    foreach $k (sort { $s7{$b} <=> $s7{$a} } keys %s7) {
      @_ = split(/,/,$k);
      print
"<tr><td>$s7{$k}</td><td>$_[0]</td><td>$_[1]</td><td>$_[2]</td></tr>\n"     if
$s7{$k} > $th;
    }
    print "</table><a href=\"#top\">Top</a><HR>\n";
  } else {
    section_header("Anomaly detected by SPADE\n", "asdo");
    foreach $k (sort { $s7{$b} <=> $s7{$a} } keys %s7) {
      @_ = split(/,/,$k);
      printf("       %-4d        %-${saddr_len}s   %-${daddr_len}s\t%-6d\n",
$s7{$k},$_[0],$_[1],$_[2]) if $s7{$k} > $th;
    }
  }
}

# print the footer (needed for html)
sub print_footer {
  if ($opt_h) {
    print                   "Generated                   by                   <a
href=\"http://xanadu.incident.org/snort/\">snort_stat.pl</a>\n";
    print "</body>\n</html>\n";
  }
}

#
```

```
# resolve host name and cache it
# contributed by: Angelos Karageorgiou, <angelos@stocktrade.gr>
# edited by: $Author: yenming $
#
sub resolve {
  local ($mname, $miaddr, $mhost = shift);
  $miaddr = inet_aton($mhost);
  if (!$HOSTS{$mhost}) {
    $mname ="";
    eval {
      local $SIG{ALRM} = sub {die "alarm\n" }; # NB \n required
      alarm $timeout;
      $mname = gethostbyaddr($miaddr, AF_INET);
      alarm 0;
    };
    die if $@ && $@ ne "alarm\n"; # propagate errors
    if ($mname =~ /^$/) {
      $mname = $mhost;
    }
    $HOSTS{$mhost} = $mname;
  }
  return $HOSTS{$mhost};
}

# Use a title and a short code to write the section headers
# This is used in place of a FORMAT as this allows variable column widths
# contributed by: Ned Patterson, <jpatter@alum.mit.edu>
#
sub section_header {
  my $linelength;
  $title = shift;
  $_ = shift;
  print("\n\n$title");
  # constant for method length for now
  $linelength = (/p/?7:0) + (/a/?20:0) + (/s/?$saddr_len:0) +
    (/d/?$daddr_len+3:0) + (/m/?20:0);
  print( '=' x $linelength, "\n");
  print(" " x 7, " #  of\n")           if (/pa.*/);
  print("  # of\n attacks  ")          if (s/^a([sdm]*)/$1/);
  print("  %    ")                    if (s/^p([asdm]*)/$1/);
  print("attacks   ")                 if (s/^a([sdm]*)/$1/);
  printf("%-${saddr_len}s   ", "from") if (s/^s([dm]*)/$1/);
  printf("%-${daddr_len}s   ", "to"  ) if (s/^d(m*)/$1/);
  printf("%-5s  ", "ports"  )          if (s/^o(m*)/$1/);
  print("method")                     if (/^m/);
  print("\n");

  print( '=' x $linelength, "\n");
}

# Put data $alert into matrix for further process
# INPUT: $alert
sub process_data() {
  $self = shift;
  # if the resolve switch is on
  if ($opt_r) {
    $self->{SADDR} = resolve($self->{SADDR});
    unless ($opt_f) {
      if ( length($self->{SADDR}) > $saddr_len ) {
```

```
        $saddr_len = length($self->{SADDR});
      }
    }
    $self->{DADDR} = resolve($self->{DADDR});
    unless ($opt_f) {
      if ( length($self->{DADDR}) > $daddr_len ) {
        $daddr_len = length($self->{DADDR});
      }
    }
  }
  # put those data into a big matrix
  if ($self->{PLUGIN} eq "anomsensor") {
    push @anores , [$self->{THR},$self->{SADDR},$self->{SPORT},
                    $self->{DADDR},$self->{DPORT}];
    $opt_n = 1;
  } elsif ($self->{PLUGIN} eq "portscan") {
    push @posres , [$self->{SADDR}];
    $opt_p = 1;
  } elsif ($self->{TYPE} eq "sys" || $self->{TYPE} eq "alert") {
    push @result ,[$self->{MON},$self->{DAY},$self->{HOUR},$self->{MIN},
                   $self->{SEC},$self->{HOST},$self->{SIG},$self->{SADDR},
                   $self->{SPORT},$self->{DADDR},$self->{DPORT}];
    $lastwassnort = 1;
  }
  1;
}
```