University Security Audit for August 16th – 20th, 2002

GIAC Certified Intrusion Analyst (GCIA) Practical Assignment

Version 4.0 (revised July 22, 2004)

David Manning

8/26/2004

Executive Summary

The goal of this audit was to analyze the University's security policies. I have analyzed all data provided and come up with various recommendations that when implemented will provide a more secure, easier manageable, and lower bandwidth producing network.

The data provided shows a numerous P2P file transfers consistently taking place on the University network. This sort of traffic could be the transfer of copyrighted movies, music, and applications. While the University cannot be held immediately liable to these copyright violations due to the DMCA Safe Harbor clause it would be a proactive move to stop this activity. Also due to the popularity of P2P, many recent Trojans and viruses have started using this vector to infect new hosts. I recommend that P2P traffic be explicitly forbidden in every professor and students Appropriate Use Policy (AUP). The network administrators can implement a block that will stop most of this traffic but P2P is very adaptable and forbidding this in the AUP would be the best way to stop it.

By looking at the network traffic being monitored I can conclude that firewall policy is in severe need of an audit. To properly implement this audit an inventory of all necessary servers and devices needs to be completed. Of course a University is a very adaptable environment and academic necessity requires some level of openness but by coordinating with professors, helpdesk staff, and network administrators this inventory can be completed.

The steps outlined above are major projects that will take a number of resources. The end results of this effort though will be a network that can be run much more efficiently using less network bandwidth.

Analyzed Files

Five consecutive days of raw logs from 8/16/2002 – 8/20/2002 were downloaded for analysis from isc.sans.org/logs/raw/. A variety of tools and procedures were used during the analysis and are all explained in the Analysis Process section below. The actual commands run and scripts are attached in the Appendix.

Alert Frequencies & Packet Statistics

All five logs were compressed into one large pcap file using the pcapmerge command. This merged file (8.16-20.2002) was then analyzed by Snort v2.2 - build 30 using the most current Snort & Bleeding Edge rules at the time (August 20th). All Snort rules were set to enabled and were ran except for the porn.rules, icmp-info.rules, chat.rules, multimedia.rules, & p2p.rules which were commented out. All Bleeding Edge rules were run with the exception of the "Yahoo Mail" rules (sid 2000041 – 2000045, 2000341). With this configuration 4073 alerts were generated. Below are the results from the Snort analysis.

Alert Count	Alert Name
1489	(http_inspect) BARE BYTE UNICODE ENCODING [**]
1137	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY [**]
766	(http_inspect) IIS UNICODE CODEPOINT ENCODING [**]
153	BAD-TRAFFIC bad frag bits [**]
130	(http_inspect) DOUBLE DECODING ATTACK [**]
96	BAD-TRAFFIC tcp port 0 traffic [**]
91	(http_inspect) WEBROOT DIRECTORY TRAVERSAL [**]
52	(http_inspect) NON-RFC HTTP DELIMITER [**]
40	(http_inspect) APACHE WHITESPACE (TAB) [**]
32	SHELLCODE x86 inc ebx NOOP [**]
20	SHELLCODE x86 NOOP [**]
14	BACKDOOR Q access [**]
12	BLEEDING-EDGE SCAN NMAP -sA [**]
12	BAD-TRAFFIC ip reserved bit set [**]
9	BLEEDING-EDGE SCAN NMAP -sT or TCP incoming connection [**]
6	MISC Tiny Fragments [**]
4	(snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
3	SHELLCODE x86 setuid 0 [**]
3	SHELLCODE x86 setgid 0 [**]
2	SCAN FIN [**]
1	MISC source port 53 to <1024 [**]
1	(http_inspect) OVERSIZE CHUNK ENCODING [**]

Table 1.0 Snort Alert Frequencies Count for all files

Packet Count	Source IP	Destination IP
3045	115.74.249.65	64.154.80.51
2664	115.74.249.65	64.154.80.50
1184	115.74.249.65	64.154.80.49
757	115.74.249.65	209.11.34.130
636	115.74.249.65	216.136.224.55

Table 1.1 Top 5 Talking Pairs by Packet Count

As you can see from the table above one host at 115.74.249.65 generated a large amount of the traffic seen in these 5 log files. Of the 22857 total packets in these log files, the 115.74.249.65 host was involved as source or destination in 21901 or 96%. The reason for this large packet count will be examined further below. Because of the possibility of losing important but lower frequency pairs the table below shows the top 5 talking pairs that do not include the host above.

Packet Count	Source IP	Destination IP
33	213.144.130.166	115.74.249.202
29	148.63.139.2	115.74.33.251
27	217.159.17.213	115.74.33.251
21	194.8.218.51	115.74.249.202
18	170.47.102.29	115.74.249.214

Table 1.2 Top 5 Non-115.74.249.65 Talking Pairs by Packet Count

Also important to note is what services were actively being targeted by external addresses. The table below expresses the top destination ports on the 115.74.0.0/16 network.

Port Count	Port Number
642	80
119	6346
96	0
33	53
30	8080

Table 1.3 Top 5 Ports Targeted by External IPs

The most clearly targeted service on the 115.74.0.0/16 network was HTTP running on port 80. This is not out of the ordinary as there are many different web server vulnerabilities that a variety of worms and exploits exist for. The only active web server that we can see a HTTP response from is 115.74.249.202. This server appears to be a valid web server running Apache 1.3.12 on RedHat.

A Gnutella P2P related port comes in as the second most targeted port on the protected network. This appears to be valid Gnutella traffic but P2P file sharing is usually not a valid use of University resources.

The 3rd most active port targeted is port 0 which does not have any application listening on it but is used by some scan tools such as hping.

The 4th and 5th ports are the respective domain (DNS) and web proxy ports. We did not see any response to any of these probes for either port.

Below we can see the top suspicious external addresses detected in the original Snort capture. The top results I have chosen are from hosts that do not involve the proxy server at 115.74.249.65 or Gnutella related traffic.

Packet Count	Source IP Address
48	211.47.255.21
32	211.47.255.24
19	213.180.193.125 (arc.yandex.ru)

Table 1.4 Top 5 Suspicious External Hosts

One of the hosts does have a valid PTR record showing a Romanian country domain. The top 2 IPs are strangely crafted with the 3rd octet set as a broadcast. Both of the 211.47.255 IPs resolve to APNIC assigned addresses located in Korea.

```
pb# whois -A 211.47.255.21
% [whois.apnic.net node-1]
% Whois data copyright terms    http://www.apnic.net/db/dbcopyright.html

inetnum:      211.46.0.0 - 211.49.255.255
netname:      KRNIC-KR
descr:        KRNIC
descr:        Korea Network Information Center
country:      KR
admin-c:      HM127-AP
tech-c:       HM127-AP
remarks:      ******************************************
remarks:      KRNIC is the National Internet Registry
remarks:      in Korea under APNIC. If you would like to
remarks:      find assignment information in detail
remarks:      please refer to the KRNIC Whois DB
remarks:      http://whois.nic.or.kr/english/index.html
remarks:      ******************************************
mnt-by:       APNIC-HM
mnt-lower:    MNT-KRNIC-AP
changed:      hm-changed@apnic.net 19991118
changed:      hm-changed@apnic.net 20010606
changed:      hm-changed@apnic.net 20040623
status:       ALLOCATED PORTABLE
source:       APNIC

person:       Host Master
address:      11F, KTF B/D, 1321-11, Seocho2-Dong, Seocho-Gu,
address:      Seoul, Korea, 137-857
country:      KR
phone:        +82-2-2186-4500
fax-no:       +82-2-2186-4496
e-mail:       hostmaster@nic.or.kr
nic-hdl:      HM127-AP
mnt-by:       MNT-KRNIC-AP
changed:      hostmaster@nic.or.kr 20020507
source:       APNIC
```

Table 1.5 Whois Information for APNIC hosts

The whois information for Romanian hosts is below in Table 1.6.

```
125.193.180.213.in-addr.arpa    name = arc.yandex.
inetnum:      213.180.192.0 - 213.180.193.255
netname:      COMPTEK-NET1
descr:        CompTek International
descr:        3, Gubkina str., Moscow, 117809
country:      RU
admin-c:      YNDX1-RIPE
tech-c:       YNDX1-RIPE
status:       ASSIGNED PA
notify:      ****@yandex.net
mnt-by:       COMPTEK-MNT-RIPE
changed:     *****@comptek.ru 20020607
source:       RIPE

route:        213.180.192.0/20
descr:        CompTek network / special
origin:       AS13238
notify:      ****@comptek.ru
mnt-by:       COMPTEK-MNT-RIPE
changed:     *****@comptek.ru 20010123
source:       RIPE

role:         Yandex LLC Network Operations
address:      Yandex LLC
address:      40A Vavilova st.
address:      117333, Moscow, Russia
phone:        +7 095 9743555
fax-no:       +7 095 9743565
e-mail:      ****@yandex.net
trouble:      ------------------------------------------------------
trouble:      Points of contact for Yandex LLC Network Operations
trouble:      ------------------------------------------------------
trouble:      Routing and peering issues: ****@yandex.net
trouble:      SPAM issues:                ******@yandex.ru
trouble:      Network security issues:    ******@yandex.ru
trouble:      Mail issues:                ***********@yandex.ru
trouble:      General information:        *****@yandex.ru
trouble:      ------------------------------------------------------
admin-c:      VLI1-RIPE
tech-c:       KBG2-RIPE
notify:      ****@yandex.net
nic-hdl:      YNDX1-RIPE
mnt-by:       COMPTEK-MNT-RIPE
changed:     *****@comptek.ru 20020607
source:       RIPE
```
Table 1.6 Whois Information for 213.180.193.125

Now we are going to try and determine the OS running on each of the hosts involved. This analysis is done using p0f v1.8.3 on our merged pcap file.

Source IP Address	OS Guess
211.47.255.21	Linux 2.4.1-14
211.47.255.24	Linux 2.4.1-14
213.180.193.125	Macintosh PPC Mac OS X (10.2.1 and v?)

Table 1.7 p0f Estimation of OS on Remote Hosts

The p0f analysis shows that the two Korean hosts are both running the same range of Linux 2.4.1-14 kernels. This also leads one to thing that this could be one host spoofing multiple IPs.

Relationship of Hosts

The raw log files examined do not provide all network traffic between hosts but only the traffic that originally triggered a Snort rule. It is hard to determine exactly what traffic has passed through the network but using the data provided the network will be reconstructed to the best of my ability.

As we noticed in the tables above the host at 115.74.249.65 generated a large percentage of the snort alerts. Looking at the traffic from this source we can see that most of traffic sourced from this host (15063 of 21216 or 71%) are outbound web requests.

We can further examine the traffic from this with the command
"tcpdump -nnXr 8.16-20.2002 src host 115.74.249.65".

```
01:08:05.986507 115.74.249.65.64252 > 211.99.211.30.80: P
1530424958:1530425436(478) ack 3194147429 win 64240 (DF)
  0000: 4500 0206 1dcd 4000 7a06 5da9 734a f941  E....Í@.z.]©sJùA
  0010: d363 d31e fafc 0050 5b38 6e7e be62 d265  ÓcÓ.úü.P[8n~¾bÒe
  0020: 5018 faf0 8b10 0000 4745 5420 2f61 6469  P.úð....GET /adi
  0030: 2f73 6561 7263 682e 7369 6e61 2e63 6f6d  /search.sina.com
  0040: 2e63 6e2f 7365 6172 6368 7265 7375 6c74  .cn/searchresult
  0050: 733b 6b77 3d25 4433 253b 6f72 643d 3632  s;kw=%D3%;ord=62
  0060: 3030 3235 3931 3633 353f 2035 3931 3633  002591635? 59163
  0070: 353f 2048 5454 502f 312e 310d 0a41 6363  5? HTTP/1.1..Acc
  0080: 6570 743a 2069 6d61 6765 2f67 6966 2c20  ept: image/gif,
  0090: 696d 6167 652f 782d 7862 6974 6d61 702c  image/x-xbitmap,
  00a0: 2069 6d61 6765 2f6a 7065 672c 2069 6d61   image/jpeg, ima
  00b0: 6765 2f70 6a70 6567 2c20 6170 706c 6963  ge/pjpeg, applic
  00c0: 6174 696f 6e2f 766e 642e 6d73 2d65 7863  ation/vnd.ms-exc
  00d0: 656c 2c20 6170 706c 6963 6174 696f 6e2f  el, application/
  00e0: 6d73 776f 7264 2c20 2a2f 2a0d 0a52 6566  msword, */*..Ref
  00f0: 6572 6572 3a20 6874 7470 3a2f 2f73 6561  erer: http://sea
  0100: 7263 682e 7369 6e61 2e63 6f6d 2e63 6e2f  rch.sina.com.cn/
  0110: 6367 692d 6269 6e2f 7365 6172 6368 2f73  cgi-bin/search/s
  0120: 6561 7263 682e 6367 693f 5f73 6561 7263  earch.cgi?_searc
  0130: 686b 6579 3dd3 a2d3 ef0d 0a41 6363 6570  hkey=Ó¢Óï..Accep
  0140: 742d 4c61 6e67 7561 6765 3a20 7a68 2d63  t-Language: zh-c
  0150: 6e0d 0a41 6363 6570 742d 456e 636f 6469  n..Accept-Encodi
  0160: 6e67 3a20 677a 6970 2c20 6465 666c 6174  ng: gzip, deflat
  0170: 650d 0a55 7365 722d 4167 656e 743a 204d  e..User-Agent: M
  0180: 6f7a 696c 6c61 2f34 2e30 2028 636f 6d70  ozilla/4.0 (comp
  0190: 6174 6962 6c65 3b20 4d53 4945 2036 2e30  atible; MSIE 6.0
  01a0: 3b20 5769 6e64 6f77 7320 4e54 2035 2e31  ; Windows NT 5.1
  01b0: 290d 0a48 6f73 743a 2061 642e 636e 2e64  )..Host: ad.cn.d
                            <SNIP>
```

```
07:52:27.606507 115.74.249.65.63168 > 208.254.63.69.80: P
2377840376:2377840864(488) ack 1240991753 win 64240 (DF)
  0000: 4500 0210 2124 4000 7d06 ed86 734a f941  E...!$@.}.í.sJùA
  0010: d0fe 3f45 f6c0 0050 8dba f6f8 49f8 0809  Ðþ?EöÀ.P.ºöøIø..
  0020: 5018 faf0 d95d 0000 4745 5420 2f74 6d70  P.úðÙ]..GET /tmp
  0030: 6164 2f63 6f6e 7465 6e74 2f72 656c 6961  ad/content/relia
  0040: 7175 6f74 652f 696d 6167 6573 2f53 6176  quote/images/Sav
  0050: 6555 7074 6f37 3025 7632 2e67 6966 2048  eUpto70%v2.gif H
  0060: 5454 502f 312e 310d 0a41 6363 6570 743a  TTP/1.1..Accept:
  0070: 202a 2f2a 0d0a 5265 6665 7265 723a 2068   */*..Referer: h
  0080: 7474 703a 2f2f 6164 2e74 7261 6666 6963  ttp://ad.traffic
  0090: 6d70 2e63 6f6d 2f74 6d70 6164 2f63 6f6e  mp.com/tmpad/con
  00a0: 7465 6e74 2f72 656c 6961 7175 6f74 652f  tent/reliaquote/
  00b0: 4e45 572e 6874 6d6c 0d0a 4163 6365 7074  NEW.html..Accept
  00c0: 2d4c 616e 6775 6167 653a 2065 6e2d 7573  -Language: en-us
  00d0: 0d0a 4163 6365 7074 2d45 6e63 6f64 696e  ..Accept-Encodin
  00e0: 673a 2067 7a69 702c 2064 6566 6c61 7465  g: gzip, deflate
  00f0: 0d0a 5573 6572 2d41 6765 6e74 3a20 4d6f  ..User-Agent: Mo
  0100: 7a69 6c6c 612f 342e 3020 2863 6f6d 7061  zilla/4.0 (compa
  0110: 7469 626c 653b 204d 5349 4520 352e 353b  tible; MSIE 5.5;
  0120: 2057 696e 646f 7773 204e 5420 352e 3029   Windows NT 5.0)
                        <SNTP>
```
Table 1.8 Hex Packet Data of Outbound Web Requests from 115.74.249.65

This packet data shows a MS Internet Explorer 6.0 on Windows XP User-Agent and a MS Internet Explorer 5.5 on Windows 2000 User-Agent both originating our high traffic IP. Looking through other packets we can see other versions of both MS Internet Explorer and Microsoft Windows originating from this source IP. We can conclude that the host at 115.74.249.65 is a proxy server for all internal hosts within the 115.74.0.0/16 network. This proxy server appears to do port NATing for the 115.74.0.0/16 network but like most proxies it keeps the HTTP User-Agent of the originating workstation. By issuing the command below and seeing that no results are returned we can confirm that all externally bound web traffic passes through the 115.74.249.65 proxy server.

```
tcpdump -nnr 8.16-20.2002 src net 115.74.0.0/16 and dst
port 80 and not host 115.74.249.65
```

Determing that 115.74.249.65 is an outbound webproxy takes care of 15063 of 21216 of the packets originating from it but leaves us with quite a few that do not meet the HTTP web proxy definition. By further examining traffic with the following command we can see a variety of different destination ports originate from the proxy server.

```
pb# tcpdump -nnqr 8.16-20.2002 src host 115.74.249.65 and not dst
port 80
<snip>
11:12:53.306507 115.74.249.65.62298 > 64.4.12.183.1863: tcp 247 (DF)
11:14:32.326507 115.74.249.65.62298 > 64.4.12.183.1863: tcp 171 (DF)
16:08:12.956507 115.74.249.65.61455 > 64.4.12.193.1863: tcp 140 (DF)
11:48:41.896507 115.74.249.65.63021 > 24.186.100.122.6347: tcp 54
(DF)
11:48:43.916507 115.74.249.65.63053 > 24.222.138.80.9540: tcp 54 (DF)
11:48:45.916507 115.74.249.65.63076 > 142.59.90.225.7475: tcp 54 (DF)
11:48:48.876507 115.74.249.65.63083 > 152.19.208.65.6879: tcp 54 (DF)
11:48:50.906507 115.74.249.65.63098 > 24.148.7.103.6518: tcp 54 (DF)
11:48:51.866507 115.74.249.65.63099 > 24.186.100.122.6347: tcp 54
(DF)
<snip>
06:43:39.136507 115.74.249.65.62490 > 68.8.21.246.5285: tcp 54 (DF)
06:43:40.056507 115.74.249.65.62492 > 131.128.137.54.7508: tcp 54
(DF)
06:43:42.046507 115.74.249.65.62507 > 24.186.100.122.6347: tcp 54
(DF)
06:43:42.986507 115.74.249.65.62492 > 131.128.137.54.7508: tcp 54
(DF)
06:43:43.126507 115.74.249.65.62511 > 68.8.21.246.5285: tcp 54 (DF)
06:43:47.096507 115.74.249.65.62518 > 207.192.203.33.6490: tcp 54
(DF)
06:43:48.136507 115.74.249.65.62520 > 66.27.109.189.8473: tcp 54 (DF)
```
Table 1.9 Outbound Requests Not Destination Port 80 from Web Proxy

The top 5 non port 80 ports returned are 6347, 1863, 6348, 6349, & 5293. MSN Messenger is primarily used on 1863 so now we know other application protocols are being proxied through this host. The ports 6347 through 6349 are all Gnutella client ports and looking at the data we can see that many other high destination ports seen here are also related to Gnutella traffic.

```
pb# tcpdump -nnXr 8.16-20.2002 port 5293
09:05:34.516507 115.74.249.65.61584 > 130.91.233.91.5293: P
2552044823:2552044877(54) ack 4276429179 win 17520 (DF)
  0000: 4500 005e 24b0 4000 7c06 9139 734a f941  E..^$°@.|..9sJùA
  0010: 825b e95b f090 14ad 981d 1d17 fee5 217b  .[é[ð..-....þå!{
  0020: 5018 4470 522f 0000 474e 5554 454c 4c41  P.DpR/..GNUTELLA
  0030: 2043 4f4e 4e45 4354 2f30 2e36 0d0a 5573   CONNECT/0.6..Us
  0040: 6572 2d41 6765 6e74 3a20 476e 7563 6c65  er-Agent: Gnucle
  0050: 7573 2031 2e36 2e30 2e30 0d0a 0d0a       us 1.6.0.0....
```
Table 1.10 Gnutella Traffic from Proxy on non-Gnutella ports

The destination port of 5293 turns out to be more Gnutella related traffic. We can conclude that both the Gnutella and MSN Messenger applications are allowed to also use this proxy.

Now we just need to determine the rest of the network layout. Looking through the merged dump file provides two unique MAC address of 00:00:0c:04:b2:33 and 00:03:e3:d9:26:c0.

```
01:11:53.546507 0:0:c:4:b2:33 0:3:e3:d9:26:c0 0800 549:
115.74.249.65.64331 > 211.99.211.30.80: P 1832340388:1832340883(495)
ack 2462627999 win 17024 [tos 0x10]
```

```
01:54:00.016507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 60: 163.23.216.66.84
> 115.74.215.161.80: . ack 0 win 1400
```

Both of these MAC addresses belong to Cisco Systems, Inc [1]]. Using the resource originally discovered by Rob Perdue [2] during one of his detects I have further drilled down into the MAC address to make a more educated guess about the devices. In the report referenced above, Rob Perdue did further searches and found a MAC address of a PIX that matched the first 6 characters of the 00:03:e3:d9:26:c0 MAC address. I did some further search on my own and found reference to the first 8 characters of the 00:00:0c:04:b2:33 MAC referenced as a router [4]. Due to the traffic flow of internal traffic coming from 00:00:0c:04:b2:33 with a destination of 00:03:e3:d9:26:c0 we can make a guess that 00:03:e3:d9:26:c0 is the MAC address of the internal interface of the PIX and that 00:00:0c:04:b2:33 is the MAC address of an internal router.

The original Snort sensor that captured these packets can be assumed to be monitoring off tap or a hub placed between these two devices. The diagram below gives my best estimation of the sensor placement.

```
       { Internet }
            |
|-----------------------|
|       Cisco PIX       |
|   00:03:e3:d9:26:c0   |
|-----------------------|
            |
            |         |----------------------|
        [TAP/HUB]---|      Snort Sensor      |
            |         | (Sniffing Interface) |
            |         |----------------------|
            |
|-----------------------|
|      Cisco Router     |
|   00:00:0c:04:b2:33   |
|-----------------------|
            |
      |---------------|
      | 115.74.0.0/16 |
      |---------------|
```

Table 1.11 Suggested Network Diagram

Defensive Recommendations

Examining all the alerts does not show any obviously compromised system but it does point out some weaknesses in your egress and ingress filtering. Based on the diagram in Table 1.11 we are assuming that that we are inside a PIX firewall and that is the last firewall filtering that will take place before the packets reach the end hosts. The current firewall policy appears to allow many different ports inside that should be blocked by on the ingress filtering. A university does have somewhat of a unique environment due to the academic necessity for openness but there are some rules that can be put in place to cut down on the traffic seen.

The web server at 115.74.249.202 is running a version of Apache that was released over two years ago [14]. There have been many security updates and releases since then and this server should be immediately updated.

Even though there is only one valid web server running on this network there are numerous inbound HTTP scans. This highlights that the firewall policy is passing any destination port 80 into the network and not filtering for the destination IP. This is just one example that illustrates that the policy should be reject everything and only allow in what services we want to be internet reachable. This of course requires an inventory of all necessary services on the internal network and all necessary services & access that some courses may require. While is a daunting task it will prove to be fruitful when a manageable & understood network emerges.

The proxy server at 115.74.249.65 can provide some level of security and appropriate use control but at the current time does not appear to be doing so. We are seeing a large number of Gnutella application packets connecting through the proxy that could be set up to be dropped. If egress filtering blocks everything that is not necessary and everything is forced to go through the proxy we can shut down almost all Gnutella activity. Many viruses & spy ware programs are being distributed in this manner and excluding it from the network will not only cut down on network traffic but also cut down on users calling into the Helpdesk for virus and spy ware related problems.

Also we need to confirm that there is no external access to the proxy server at 115.74.249.65. If an external host could access this proxy server they could then have a gateway into our internal network. They could also then be able to attack other networks and make it appear like it was our host performing the attack.

They were 4073 alerts generated in these 5 days of Snort logs. Due to the initial raw data only being log data from a Snort machine and not a full packet capture some alerts that required a state to be established were never triggered. This is discussed in more detail in Detect #2. The alerts that we do have show many false positives that can be filtered out to help the every day analyst find the signal in the noise.

If there is only one valid web server running on the network and it is running Apache then all IIS web server related rules can be disabled. If we can upgrade the Apache server to the most current release we can disable many of the http_inspect rules also. Most of the SHELLCODE events seen have been determined to be false-positives generating on streaming media. By changing the SHELLCODE_PORTS variable in the snort.conf file we can filter these out.

Detailed Analysis of Detects

In Table 1.0 the top events seen in the merged log file were displayed in ascending order. Now we drill deeper into a few select detects to determine exactly what happened.

Detect #1 - SHELLCODE x86 setuid 0/SHELLCODE x86 setgid 0

Description of Detect

This attack is usually seen at the end of a non-trivial buffer overflow attack as 'setuid 0' or 'setgid 0' command is set to escalate the privileges of the application being attacked. This exploit could be manually run against a vulnerable host or rolled into a worm that could operate completely autonomously. In a successful exploitation this command would be run to grab root privileges before another command, such as open a shell and bind it to a port, would be run.

Both the setuid and setgid alerts were generated from two different hosts in the same /24 network. Doing an ARIN lookup for both IPs returns the same whois information.

```
OrgName:    Yahoo! Broadcast Services, Inc.
OrgID:      YAHO
Address:    701 First Avenue
City:       Sunnyvale
StateProv:  CA
PostalCode: 94089
Country:    US

NetRange:   63.250.192.0 - 63.250.223.255
CIDR:       63.250.192.0/19
NetName:    NETBLK2-YAHOOBS
NetHandle:  NET-63-250-192-0-1
Parent:     NET-63-0-0-0-0
NetType:    Direct Allocation
```

Table 2.1 Whois Information for setuid/setgid Attacking Host

In this case it appears a false positive generated by the Microsoft streaming protocol (TCP 1755) coincidentally matching the "b017 cd80" string that the Snort signature is matching on. In the example of the setgid being set to 0 the string that the Snort signature is matching on is "b0b5 cd80".

Reason for Analysis

If a host did successfully have shell code 'setuid 0' or 'setgid to 0' would mean that the system is close to being completely compromised. The attacker would be one command away from opening up a shell and binding it to a port or executing any other malicious command on the target host. Because this is such a high priority alert it is of critical importance to determine if it is valid.

Detect was Generated by

This detect was generated by the Snort v2.2 (build 30) intrustion detection system. The following 2 rules triggered on two similar 4 byte strings detected in the network traffic.

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any
(msg:"SHELLCODE x86 setgid 0"; content:"|B0 B5 CD 80|";
reference:arachnids,284; classtype:system-call-detect;
sid:649; rev:8;)

alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any
(msg:"SHELLCODE x86 setuid 0"; content:"|B0 17 CD 80|";
reference:arachnids,436; classtype:system-call-detect;
sid:650; rev:8;)
```

Table 2.2 Snort Rules 649 & 650 (setgid 0 & setuid 0)

The rule above alerts on any external host from a defined shell code port, everything except port 80 in this case, directing the “B0 B5 CD 80” or “B0 17 CD 80” string to our internal network. Because there is no offset this signature looks to match either of those strings at the start of the packet data. The first part of the content “B0 17” is the assembly language move call for the system move call [Rich Helton]. The second part of the content, “CD 80” is the instruction to call the interrupt handler that will execute the function [Rich Helton].

```
[**] [1:650:8] SHELLCODE x86 setuid 0 [**]
[Classification: A system call was detected] [Priority: 2]
09/17-13:10:17.056507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
63.250.205.41:1755 -> 115.74.249.65:63116 TCP TTL:116 TOS:0x0
ID:44961 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x1F8D5D6D  Ack: 0x1C06D8CC  Win: 0x4090  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS436]

<snip>

[**] [1:649:8] SHELLCODE x86 setgid 0 [**]
[Classification: A system call was detected] [Priority: 2]
09/16-19:52:11.296507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
63.250.205.11:1755 -> 115.74.249.65:64689 TCP TTL:115 TOS:0x0
ID:17639 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x26BD7672  Ack: 0xCA9AD173  Win: 0x40B0  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS284]
```

Table 2.3 Snort Alerts Excerpt – setuid(0) and setgid(0)

Probability the Source Address was Spoofed

In this case we do not have a complete packet dump so we can not confirm that the initial TCP SYN 3-way handshake was completed. If we did have a complete packet capture we could verify if a 3-way handshake was completed and rule out the possibility that the source address was spoofed.

In this case we can look at the packet data and see that this is not a 1 packet delivery exploit and therefore the source address would have to have traffic ACKed before it could deliver the rest of the payload.

Attack Mechanism

In this detect, the attack was attempted from a Yahoo! Streaming media Broadcasting server against a local IP on our network. A successful attack generated by this alert would be preceeded by a crafted buffer overflow and postceeded by a command to open up a remote shell on a unique port on the destination host.

In this case the detect appears to be a false positive cause by 4 bytes being present at the start of some streaming media content.

Because of the proxy server at 115.174.249.65 we can not tell what end host on our network this packet was destined for. While attacks against proxy servers do exist this detect appears to be a false positive destined for an internal IP.

Correlations

The setuid 0 and setgid 0 Snort alerts frequent false positive on downloaded data. This is why the default snort.conf file provided has $SHELLCODE_PORTS defined as anything not equal to port 80. Frequent triggers on normal jpeg, gif, and other binary downloads are extremely common without this variable set as it is currently. I did not find any exact correlations on these log files but I did find similar detect done by Rich Helton [10].

Currently there are 74 CVE entries dealing with problems with the setuid call [11]. There are also various CVE entries dealing with vulnerabilities in streaming media programs but I did not find any that would affect a Linux/UNIX host running an application that received the ms-streaming protocol.

Evidence of Active Targeting

We do not have a complete packet capture for this network but looking at the traffic presented there is no prior or post contact from the host in question. This does not look to be a case of active targeting.

Severity

Severity = (2 + 5) – (1 + 3)

Criticality – The system targeted is a workstation running on the internal network. It has much less importance than a functional server.

Lethality – A successful attack of this manner would lead to root access on the target host. This of course is as lethal as any attack can be because if gives the attacker full control over the host.

System Countermeasures – As this host is participating in receive a media stream we can assume that it does not have a strict host based firewall running on it or any other protection.

Network Countermeasures – This attack did have to pass through a proxy server on the network that could have been set up to block certain parameters. Because we cannot see on the other side of the proxy or examine any of the proxy's logs we cannot come to any conclusion.

This equation will give us a total severity of 3 on a maximum scale of 8.

Detect #2 – Web Attacks directed against Internal Apache Server

Description of Detect

On our internal network there is a web server at 115.74.249.202 running Apache 1.3.12 with FrontPage on RedHat Linux. There were a variety of attacks detected against this web server but to narrow down the false positives this detect will only deal with attacks that particularly target the Apache web server.

By excluding IIS vulnerabilities from the traffic directed towards the 115.74.249.202 server we are left with a variety of different attacks.

```
pb# tethereal -r 8.16-20.2002 -R 'ip.addr == 115.74.249.202' | egrep -vi
'(default.ida|dll|vti|cmd.exe|PROPFIND)'
23 6831.600000 24.48.78.100 -> 115.74.249.202 HTTP GET /cgi-
local/formmail.cgi?recipient=st3mm@aol.com&subject=http://www.XXXXXXXX/cgi-
local/
formmail.cgi&body=JupZ&email=yym@aol.com HTTP/1.1
25 8044.980000 213.144.130.166 -> 115.74.249.202 HTTP HEAD
/main/catalog/ethprod.pdf HTTP/1.1
104 22833.180000 213.11.160.2 -> 115.74.249.202 TCP http > http [ACK] Seq=0
Ack=0 Win=1400 Len=0
1487 38351.340000 216.199.89.139 -> 115.74.249.202 HTTP OPTIONS / HTTP/1.1
2677 69999.850000 219.164.183.47 -> 115.74.249.202 HTTP GET
/main/../images/head1.jpg HTTP/1.1
7225 144570.480000 198.51.119.130 -> 115.74.249.202 TCP 11679 > http [RST,
ACK] Seq=0 Ack=0 Win=31968 Len=1332
8268 180135.820000 115.74.249.202 -> 195.29.191.46 HTTP HTTP/1.1 403
Forbidden (text/html)
17886 312784.980000 194.7.125.10 -> 115.74.249.202 TCP 58097 > webcache
[SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
18082 314433.120000 208.61.206.232 -> 115.74.249.202 FTP Request: PASS
```

Table 2.4 Non-IIS Attacks against Internal Web Server

As you can see from the tethereal output above 9 unique attacks can be detected directed for the web server. Most of these attempts are simple reconnaissance probes that the most critical of which is the attempt to access formmail.cgi.

The formmail.cgi attack targets a web server running a vulnerable version of the formmail.cgi and attempts to have the script successfully send an email to the address in the URL above. If this initial test from 24.48.78.100 (Adelphia Cable ISP) seen in the URL above is successful it is likely that the attacker will return to this site to send a large number of spoofed & spam laden emails from this server.

With the rise of spam in recent years many network administrators are taking a pro-active approach to spam defense and having a flood of spam emails originate from the 115.74.0.0 network and internal domain could result in the the network being blacklisted at various antispam sites [7].

Reason for Analysis

This detect was analyzed because it is the only host on the network that was detected sending return traffic on port to a remote host. If you take out the proxy server at 115.74.249.65 this host was actually the only host on the 115.74.0.0/16 network that generated any response to an external host. Of course this is inferred by looking at our snort binary pcap file and not a full packet capture from this network. There very well could be other hosts on the network that did no generate any snort alerts during the initial packet capture.

Detect was Generated by

The detect was generated by the Snort v2.2 (build 30) intrustion detection system and further analyzed using tethereal.

The table below shows the relevant Snort alerts that involved the 115.74.249.202 host.

Alert Count	Alert Name
37	(http_inspect) WEBROOT DIRECTORY TRAVERSAL [**]
29	(http_inspect) NON-RFC HTTP DELIMITER [**]
9	BLEEDING-EDGE SCAN NMAP -sT or TCP incoming connection [**]
1	(snort_decoder) WARNING: TCP Data Offset is less than 5! [**]
1	SCAN FIN [**]
1	(http_inspect) IIS UNICODE CODEPOINT ENCODING [**]

Table 2.5 Snort Alerts Involving 115.74.249.202

The alert that would have generated by the formmail attack seen in Table 1.10 never was never triggered as you can see in Table 1.11. By examining the rule context below we can determine why this occurred.

In this rules “flow” condition the traffic has to be established for the alert to check for the “/formmail” string. However in this case because the raw file was not a complete packet capture but only a binary logging from a previous Snort run this alert was never generated.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
CGI formmail access"; flow:to_server,established;
uricontent:"/formmail"; nocase; reference:arachnids,226;
reference:bugtraq,1187; reference:bugtraq,2079; reference:cve,1999-
0172; reference:cve,2000-0411; reference:nessus,10076;
reference:nessus,10782; classtype:web-application-activity; sid:884;
rev:14;)
```

Table 2.6 Formmail CGI Rule from Snort v2.2

Probability the Source Address was Spoofed

Because all proper HTTP traffic requires state to be established before doing a HTTP GET we can assume that the 3-way handshake was completed before all attacks. Some HTTP clients do send the initial GET request on the 3rd part of the handshake but this would still require at the SYN-ACK packet to be received on the originating host which would confirm that the source address was not spoofed.

Attack Mechanism

The formmail script is primarily intended as a way for web users to easily and quickly send an email to a set user usually at the web site domain. However there have been versions that did not validate the destination email address allowing anybody to anonymously send emails using this unintended email gateway. Many spammers are constantly on the look out for anonymous email gateways to pass their emails through with the primary motivation being a small amount of money for every successful email sent out.

In this detect, the attack would have succeeded if the web site at 115.74.249.202 was running a vulnerable version of the Formmail.cgi script. As this is a fairly old vulnerability and we did not see any repeated attempts after the initial probes we can assume that this cgi is not currently running on the web server in question. However this is something that should be confirmed by either examining a complete packet capture and looking for a HTTP 404 code or by examining the Apache logs on the server in question.

Correlations

Similar detects have been done by Michael Holstein [10], Heather M. Larrieu [12], and Loic Juillard [13]. Their analysis was very similar to mine with no major differences.

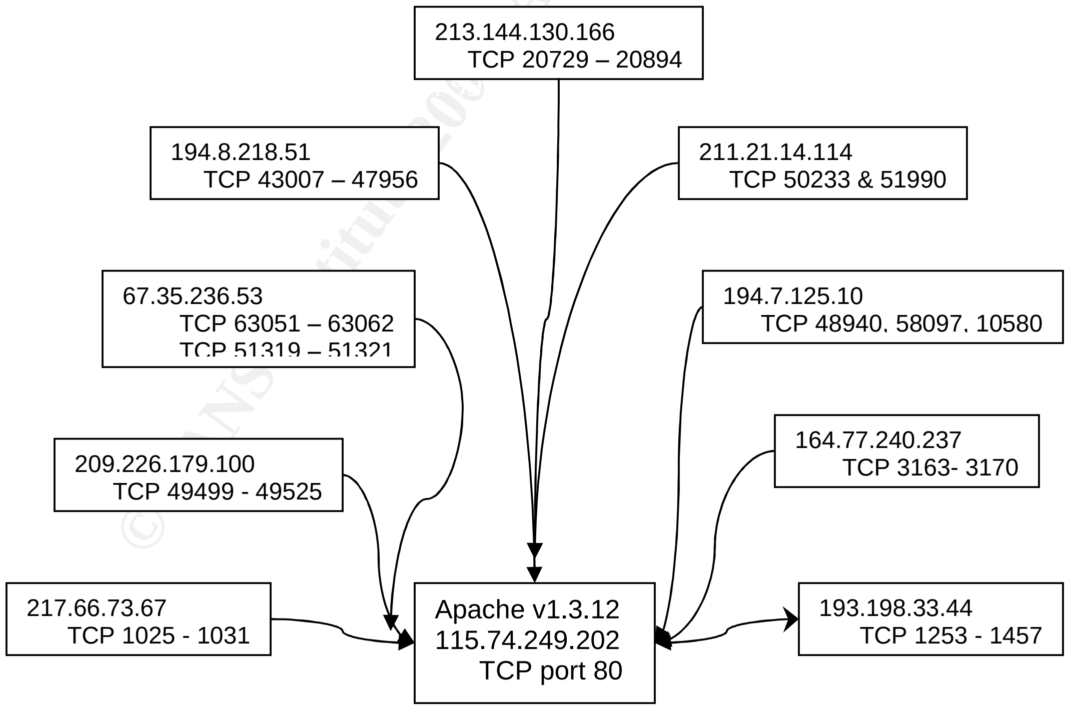
Both CVE-1999-0172 & CAN-2001-0357 reference the vulnerabilities inherent in these older Formmail scripts.

There also is a 2002 paper written by Ronald F. Guilmette and Justin Mason, maintainer of SpamAssassin, detailing some vulnerabilities in FormMail 1.9 [14].

Evidence of Active Targeting

The only traffic on this network seen from the source host of 24.48.78.100 is multiple formmail.cgi script attempts to 115.74.249.202. The attacker could have scanned the network previously for all hosts running Apache servers on port 80 and then returned later to test for the existence of a vulnerable formmail.cgi script.

Included below is a link graph showing the top scans and the only response from the web server at 115.74.249.202. This shows how the alerts for a typical web server have a many to one relationship.



213.144.130.166 TCP 20729 – 20894
194.8.218.51 TCP 43007 – 47956
211.21.14.114 TCP 50233 & 51990
67.35.236.53 TCP 63051 – 63062 TCP 51319 – 51321
194.7.125.10 TCP 48940, 58097, 10580
209.226.179.100 TCP 49499 - 49525
164.77.240.237 TCP 3163- 3170
217.66.73.67 TCP 1025 - 1031
Apache v1.3.12 115.74.249.202 TCP port 80
193.198.33.44 TCP 1253 - 1457

Table 2.7 Link Graph for Apache web server

Severity

Severity = (5 + 3) – (1 + 2) = 5

Criticality – The system targeted is a production web server running on our local network. Looking at some of the pages served by the host it appears that it is hosting a variety of critical documents.

Lethality – While there are known DoS [CVE-2000-0255] attacks against FormMail CGI programs, this was not the method of attack in this case. The actual anonymous sending of the email does not appear to be a very lethal attack. But if this was allowed there is a small possibility that the whole network block could be blacklisted on one of the many SPAM real-time black lists denying the network access to send any emails to SMTP servers that subscribe to that blacklist.

System Countermeasures – We do not have very much information about the web server in question so we are unable to determine exactly what countermeasures are present on the system. The Apache server running on the host is a fairly old version that was released on the 25th of February 2000 [15]. We can probably jump to the conclusion that a web server that has not been updated in over 2 years (detect seen on 9/16/2002) is not actively monitored and is not running any host based firewall or HIDS.

Network Countermeasures – Due to the limited knowledge we have about the network we cannot determine what network countermeasures are in place. We do know that this traffic has already passed through a Cisco PIX and we see why no reason it would not pass through the next hop Cisco Router. Because this was valid HTTP traffic there are no explicit firewall rules that could have blocked this. This is not taking into account the recent advent of in-line network application layer firewalls/intrusion prevention systems that could detect and drop this attack before it reached the web server.

Detect #3 – Nmap ACK scan

Description of Detect

This is a reconnaissance attack to probe what ports are open on a host but instead of the setting the packet flags to SYN they are set to ACK. This is an "advanced method is usually used to map out firewall rule sets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets." [16]. If the firewall is not stateful and this packet passes through then a reset being received from the target host will show this. This scan is somewhat unusual because it can never say if a port is open or not. It simply marks them as "filtered" if no response is received and "unfiltered" if a RST is sent back. That does make it useful for mapping out firewall rules [16] but it makes it a scan take requires other reconnaissance before an attack can be attempted.

Reason for Analysis

This detect was chosen for analysis because of its relatively new nature and current non-inclusion in the default Snort rule set. Many Snort users are now adding the Bleeding-Edge rules to their Snort configuration files but there is not much documentation on exactly what these rules are triggering on.

Detect was Generated by

This detect was generated by the Snort v2.2 (build 30) intrusion detection system Bleeding-Edge rule 2000538.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BLEEDING-EDGE SCAN
NMAP -sA"; dsize:0; flags:A,12; fragbits:!D; window:1024;
reference:arachnids,162; classtype:attempted-recon; sid:2000538;
rev:1;)
```

Table 2.8 Snort Bleeding-Edge Scan NMAP –sA rule

This signature is set to alert on any host in the $EXTERNAL_NET on any port that hits our network on any port. The most relevant fields from the rule above are the “flags: A,12” and “window:1024”. The flags field means it is looking for a packet with just the ACK bit set no matter what reserved bits 1 & 2 are set as. There are three unique signatures in the Bleeding-Edge rules and the only difference in each one is the windows size. This means that the Nmap –sA scan sets one of these three window sizes every time it scans. Because is there is no “flow” rule it is looking for this packet regardless of state.

Probability Source Address was Spoofed

This scan does not require any connection to be established so this source address could possibly be spoofed. The attacker could be sniffing on that network from a different IP and still see the response or could be performing a large scan and using decoys (Nmap –D flag) to make it harder for us to track down the real source. With this scan there is a high probability that this address could be spoofed.

Attack Mechanism

```
pb# tcpdump -nnr 8.16-20.2002 host 115.74.249.65 | grep "win 1024"
09:18:33.596507 12.111.47.194.80 > 115.74.249.65.3844: . ack 0 win 1024
09:18:33.626507 141.155.200.194.80 > 115.74.249.65.3844: . ack 0 win 1024
07:44:15.926507 65.64.34.124.80 > 115.74.249.65.3844: . ack 0 win 1024
07:44:16.006507 65.122.47.124.80 > 115.74.249.65.3844: . ack 0 win 1024
16:35:51.536507 65.64.34.124.80 > 115.74.249.65.3844: . ack 1 win 1024
16:35:51.656507 65.122.47.124.80 > 115.74.249.65.3844: . ack 1 win 1024
08:13:10.496507 12.111.47.194.80 > 115.74.249.65.4100: . ack 0 win 1024
08:13:10.536507 141.155.200.194.80 > 115.74.249.65.4100: . ack 0 win 1024
```
Table 2.9 Tcpdump of all packets that match the Snort NMAP –sA rule

In this detect, we saw 4 unique hosts each hit the web proxy at 115.74.249.65 two times each. This is the only contact that each of these hosts had with the any host on the network. Also the last octet of each of these hosts is 194 or 124. This leads credence to the theory that these hosts were originally spoofed. However we have to consider the fact that the original log file we were examining here was not a complete packet capture from the network but a "binary alert" file that contained all traffic that the original version of Snort alerted on.

Correlations

Even though this is a Bleeding-Edge signature there has been previous work of similar nature done by Antony Gummery [17] and Todd Williams [18]. There also many documents out there such as the Nmap man page [16] and various hacking papers [19] that document exactly what an ACK scan can accomplish against a stateless firewall.

Evidence of Active Targeting

All traffic seen here was directed at the proxy server above. The original Snort sensor detected no other traffic from any of these hosts.

Severity

(4 + 1) – (1 + 1) = 4 out of a maximum of 8

Criticality – The target system in all of these probes is the proxy server for the entire 115.74.0.0/16 network. If this were to be compromised the attacker would be on the internal network and be able to easily monitor all http and MSN Messenger traffic.

Lethality – However this detect was a simple reconnaissance probe directed towards the proxy server that was just mapping out the firewall rules and live hosts behind the firewall.

System Countermeasures – We can not successfully determine the OS on the proxy server and therefore cannot determine what countermeasures reside on the host.

Network Countermeasures – This traffic was passed inside the firewall and we had a complete packet capture we could determine if the firewall in question passed this ACK traffic through without having a connection established. This would mean that the firewall was stateless and was not a very effective defense against this traffic. However this traffic could be a part of a greater connection but we cannot tell with the data presented.

Analysis Process

All analysis was done using Snort v2.2.0 (August 19th rules), Tcpdump v3.4.0, p0f v1.8.3, GNU tethereal 0.9.14, and Oinkmaster v1.0 running on an OpenBSD 3.4 server. All of the data examined was merged into one pcap file with the following command

```
pcapmerge -r 2002.8.16 -r 2002.8.17 -r 2002.8.18 -r
2002.8.19 -r 2002.8.20 -w 8.16-20.2002
```

All snort rules were kept up to date using the oinkmaster.pl script run as a cron job once a day. Also some custom scripts were written to examine the data and all are attached in the appendix for further review and use.

A perl script was written to go through the data files, determine the home network, and run the data files through snort. This script was big help in automating the process of turning the raw data files into human usable alerts.

When examining as much data as contained in these raw logs it is important to be able to see the signal through the noise. I used a variety of command line tools and scripts to do this. One example is a command line pipe used to examine the name of all alerts generated so I can easily find alerts of greater importance.

```
grep "\[\*\*" <snort_alert_file> | cut –d’ ‘ –f3- | sort |
uniq –c | sort -nr
```

The command above would return every single alert generated by snort in descending count list. This easily helped me see the top alerts but also more importantly it let me see alerts that had a lower count but might warrant further investigation due to their nature.

I also appended the “ `awk ‘{print $1”,”$2}’` “ command to many of my command line pipes so I could then import the files into MS Excel as a CSV (Comma Seperated Values) sheet.

All these tools gave me good grasp on the data I was dealing with and helped me delve deeply into the packets seen.

References

[1] IEEE Standars. “IEEE OUI and Company_id Assignments”. URL: http://standards.ieee.org/regauth/oui/oui.txt. (14 Aug 2004).

[2] Perdue, Rob. “GIAC GCIA Version 3.5 Practical Detect”. 13 Aug. 2004. URL: (NOT MIRRORED YET – See Appendix) (16 Aug 2004).

[3] Cisco Systems, Inc. “Cisco PIX Firewall Release Notes, Version 6.3(2).” CISCO PIX FIREWALL SOFTWARE. URL: http://www.cisco.com/en/US/products/sw/secursw/ps2120/prod_release_note09186a00801a6d21.html (16 Aug. 2004).

[4] Cisco Systems, Inc. “Using Terminals”. Cisco Connection Documentation URL: http://www.yars.free.net/CiscoCD/cc/td/doc/product/software/ssr91/rpc_r/58417.htm#xtocid270337 (16 Aug. 2004).

[5] Whitehats, Inc. “IDS283 "SHELLCODE-X86-SETUID0"”. arachNIDS - The Intrusion Event Database. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids283&view=protocol (15 Aug. 2004).

[6] Whitehats, Inc. “IDS284 "SHELLCODE-X86-SETGID0"”. arachNIDS - The Intrusion Event Database. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids284&view=protocol (15 Aug. 2004).

[7] Sourcefire Vulnerability Research Team. “Snort Signature Database” URL: http://www.snort.org/snort-db/sid.html?id=650 (15 Aug. 2004).

[8] Sourcefire Vulnerability Research Team. “Snort Signature Database” URL: http://www.snort.org/snort-db/sid.html?id=649 (15 Aug. 2004).

[9] OpenRBL. “Openrbl:zones:DNSBL Zones:” URL: http://us.openrbl.org/zones.htm (16 Aug. 2004).

[10] Helton, Rich “GIAC GCIA Version 3.4 Practical Detect 2 Rich Helton”. 26 Jan. 2004.
URL: http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00137.html (16 Aug. 2004).

[10] Holstein, Michael. “SANS GCIA Practical Assignment”. June 04, 2002. URL: http://www.giac.org/practical/Michael_Holstein_GCIA.doc (18 Aug. 2004) 10 – 14.

[11] CVE. “Common Vulnerabilties and Exposure: setuid” URL: http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=setuid (18 Aug. 2004)

[12] Larrieu, Heather. “Intrusion Detection in Depth”. URL: http://www.giac.org/practical/GCIA/Heather_Larrieu_GCIA.doc (18 Aug. 2004) 26 – 31.

[13] Juillard, Loic. “[Spam relay scanning]” 16 Aug 2003. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/08/msg00151.html (18 Aug 2004).

[14] Guilmette, Ronald F. & Mason , Justin. “Anonymous Mail Forwarding Vulnerabilities in FormMail 1.9” 23 Jan. 2002. URL: http://www.monkeys.com/anti-spam/formmail-advisory.pdf (18 Aug 2004).

[15] Apache Week. “Apache Week: Issue 188, 25th February 2000”. 25 Feb. 2000. URL: http://www.apacheweek.com/issues/00-02-25. (18 Aug 2004).

[16] Fyodor. “Nmap network security scanner man page”. URL: http://www.insecure.org/nmap/data/nmap_manpage.html. (Aug 20, 2004).

[17] Gummery, Anthony. “LOGS: GIAC GCIA Version 3.3 Practical Detect(#2)”. 22 Mar. 2003.
URL: http://www.dshield.org/pipermail/intrusions/2003-March/007239.php. (Aug 20, 2004).

[18] Williams, Todd. “LOGS: GIAC GCIA Version 3.3 Practical”. 30 Dec. 2003. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00176.html. (Aug 20, 2004).

[19] Detach “Dealing with Firewalls”. HACKING UNIX. 7 Jan 2002. URL: http://duho.hackaholic.org/pub/hacking_unix/hacking_unix-part4.txt. (Aug 20, 2004).

Appendix

Command Line Commands Used

```
#Returns the alert title for every snort alert generated
grep "\[\*\*" <snort_alert_file>

#Snort command line options that were run for every file examined
snort -b -A full -c /etc/snort/snort.conf -devk none -r
/root/sans/raw/<log_file_name> -l /root/sans/raw/d<log_file_name> -h
115.74.0.0/16

#Command to produce the top talking pairs from a pcap file
tcpdump -qnnr 8.16-20.2002 | cut -d' ' -f2,4 | awk -F. '{ print $1"."$2"."$3"."$4"
"$5"."$6"."$7"."$8}' | awk '{print $1","$3}' | sort | uniq -c | sort -nr | awk '{print
$1","$2}' > top_talking_pairs

#Command to produce the top destination ports
tcpdump -qnnr 8.16-20.2002 dst net 115.74.0.0/16 | cut -d' ' -f2,4 | awk -F. '{ print
$5"."$6"."$7"."$8" "$9}' | awk '{print $3}' | cut -d':' -f1 | sort | uniq -c | sort -nr >
top_dst_home_ports

#!/usr/bin/perl -w
#####################################################################
###########
# snortll perl script
# this script takes in a list of dumpfiles, makes a directory for them,
# determines the home network, and then runs snort against the file while
# logging into the directory made previously.
# All log data is written to /var/snortall.log for later examination
# Written by David Manning
#####################################################################
###########

open (LOG,">>/var/snortall.log") || die "Cannot open $LOG: $!";;

open (DUMPFILES,"dumpfiles") || die "Cannot open $DUMPFILES: $!";;

@dumpfiles = <dumpfiles>;

foreach $file (@dumpfiles) {
        chomp ($file);
        $homenet="";
        `mkdir d$file`;
        print LOG "Making directory for $file\n";
        @hosts = `tcpdump -qtnnr $file | cut -f1,2 -d'.' | sort | uniq`;
```

```
	foreach $ip (@hosts) {
		if (!(`tcpdump -nnr $file -c 1 not net $ip`)) {
			$homenet = "$ip" + "/16";
			last;
		}
	}
	print LOG "$file has a network of $homenet\n";
	print LOG "Running Snort for $file..";
	`snort -b -A full -c /etc/snort/snort.conf -devk none -r /root/sans/script/$file -l
/root/sans/script/d$file -h $homenet/16` ;
	print LOG "..Done\n";
}
```

Rob Perdue's Practical from References (Not yet Mirrored)

```
From: rsp7@njit.edu <rsp7@njit.edu>
Reply-To: "Intrusions List (GCIA Practicals)" <intrusions@lists.sans.org>
To: intrusions@lists.sans.org
Date: Fri, 13 Aug 2004 11:56:30 -0400 (EDT)
Subject: [Intrusions] LOGS: GIAC GCIA Version 3.5 Practical Detect By Rob Perdue
```

Hello all,

Below is one of my detects required for the GCIA certification. Your comments, suggestions and questions are greatly appreciated.

Thanks!

Network Detect 1: Chat MSN Message

Snort Alert:

```
[**] [1:540:11] CHAT MSN message [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
07/18-08:48:35.294488 46.5.180.250:61149 -> 64.4.12.155:1863
TCP TTL:125 TOS:0x0 ID:38562 IpLen:20 DgmLen:182 DF
***AP*** Seq: 0x94BC4902  Ack: 0x57D2776D  Win: 0xF906  TcpLen: 20
```

1. Source of Trace:

This log was taken from the GIAC Certification Practical logs posted at http://isc.sans.org/logs/raw/2002.6.18.

Few items of note regarding the log file.

The log files are the result of a Snort instance running in binary logging mode. This means that only the packets that violate the ruleset will appear in the log.

All of the IP addresses of the protected network space have been "munged". Additionally, the checksums have been modified to prevent clever people from discovering the original IP addresses. You will find that certain keywords within the packets have been replaced with "X"s. All ICMP, DNS, SMTP and Web traffic has also been removed.

IP addresses belonging to non-local hosts are the actual IP addresses.

The above items were taken from the readme file located at
http://isc.sans.org/logs/raw/README.

Ruleset and Snort version used to generate the log are unknown.

Network Layout

What makes these logs especially challenging is that the analyst is given no knowledge of the network from which these logs were created. However, within the log there is enough information to be gathered to give the analyst a good, but not exact, picture of the network at hand.

In this case Ethereal version 0.10.5a with WinPcap 3.0 will be used to help determine the layout.

After loading the log into Ethereal and sorting by time stamp, a quick eyeball of source and destination IP's revealed that all the traffic was inbound or outbound to network 46.5.0.0/16. By sorting the Ethereal output by source address it was found that 46.5.180.250 was the only source address in the log that belonged to that network. At this point http://www.arin.net/ was consulted to see if the 46.5.0.0/16 network had an owner. Using 46.5.180.250 in the Arin Whois lookup produced:

```
OrgName:    Internet Assigned Numbers Authority
OrgID:      IANA
Address:    4676 Admiralty Way, Suite 330
City:       Marina del Rey
StateProv:  CA
PostalCode: 90292-6695
Country:    US

NetRange:   46.0.0.0 - 46.255.255.255
CIDR:       46.0.0.0/8
NetName:    RESERVED-46
```

Since this network range is reserved by IANA it would appear this would be the protected network. To verify, other IP's from the log that did not belong to this network address were also run through Arin and other registries. These IP's were found to belong to various public owners.

With the protected network determined, the MAC addresses were examined to get an idea of how many devices transmit traffic through the sensor which generated the log. Returning to the only protected IP seen generating outbound traffic, 46.5.180.250, the mac address associated with this IP, 00:00:0c:04:b2:33, was used as a starting point.

Using eth.src == 00:00:0c:04:b2:33 as an Ethereal display filter, it was found that the only outbound traffic associated with this MAC address was from 46.5.180.250. While examining the packets displayed with the above filter it appeared that not only was all outbound traffic coming from the same source mac address but also all the traffic was destined to various IP's but a single mac address, 00:03:e3:d9:26:c0. With these two MAC addresses identified the display filter, eth.src != 00:00:0c:04:b2:33 and eth.src != 00:03:e3:d9:26:c0, was used to see if there were any other MAC addresses sending traffic inbound or outbound. There were not.

With the sole two MAC addresses found, that ended up in the log anyway, the next step was to see what kind of devices these were.

Even though Ethereal is nice enough to give us the manufacturers associated with these MAC addresses, they were manually referenced using the OUI list found at http://standards.ieee.org/regauth/oui/oui.txt. Using this list and Ethereal it was determined both MAC addresses belonged to Cisco devices.

```
00-03-E3   (hex)              Cisco Systems, Inc.
0003E3     (base 16)           Cisco Systems, Inc.
                         170 West Tasman Dr.
                         San Jose CA 95134
                         UNITED STATES

00-00-0C   (hex)              CISCO SYSTEMS, INC.
00000C     (base 16)           CISCO SYSTEMS, INC.
                         170 WEST TASMAN DRIVE
                         SAN JOSE CA 95134-1706
```

With the information found thus far a simple network layout can be produced:

```
PROTECTED NETWORK ---CISCO DEVICE 1---SNORT---CISCO DEVICE 2---OUTSIDE
 46.5.0.0/16                00:00:0c:04:b2:33
00:03:e3:d9:26:c0
```

The weakness in this diagram is that there is not much known about the protected network or the nature of the two Cisco devices.

To come up with a better idea of the network, I kept on digging.

Using the MAC OUI's an attempt was made to narrow down the possibilities of what these Cisco devices are.

Poking around http://www.cisco.com an article was found pertaining to a pix software release. In this document, http://www.cisco.com/en/US/products/sw/secursw/ps2120/prod_release_note09186a00801a6d21.html , there was an example "show ver" run on a pix firewall which produced the following output(only relevant data shown):

```
0:ethernet0:address is 0003.e300.1552, irq 10
       1:ethernet1:address is 0003.e300.1553, irq 7
```

The OUI's of these two pix interfaces match that of CISCO DEVICE 2 (00:03:e3:d9:26:c0). Though by no means a sure thing, with the above information and the common practice of sandwiching firewalls with NIDS, it would make sense that CISCO DEVICE 2 is the inside interface of a PIX Firewall.

The same search was performed for the MAC of CISCO DEVICE 1 but the results were inconclusive. Though best guess would be that CISCO DEVICE 1 is a router.

With a good feeling about identifying the hardware involved in the network, attention was turned to gathering more information about the protected network. A look at the traffic inbound to the protected network, using display filter ip.dst == 46.5.0.0/16 and sorted by protocol, shows that most of the inbound traffic is for HTTP, FTP, and DNS. Though some of this traffic is due to less then honorable packets, there is enough traffic to imply the presence of these services inside the protected network. The nature of this traffic suggests that the sensor is placed at an entry/exit for the outer point of a DMZ.

Looking then at the outbound traffic from the 46.5.0.0/16 network using display filter ip.src == 46.5.0.0/16, it is seen that ,as mentioned earlier, all outbound traffic from the protected network is from 46.5.180.250. Looking through this traffic there were outbound http connections, file sharing connections, some outbound IM traffic, and few ssh connections. Most of the traffic was outbound http so I focused on that for a clue. Outbound requests to port 80 were focused on using a display filter of ip.src == 46.5.180.250 and tcp.dstport == 80. In these packets there are various GET requests to various web servers. In the payload of these packets it can be seen that there are various clients making these requests. Below is an export from ethereal highlighting these findings. (Export has been edited to show only pertinent information)

Snip 1

```
Internet Protocol, Src Addr: 46.5.180.250, Dst Addr: 61.218.76.250
Transmission Control Protocol, Src Port: 63085, Dst Port: http (80), Seq:
512801141, Ack: 1905796758, Len: 425
Request Method: GET
   Accept: */*\r\n
   Referer: http://www.corega.com.tw/lan.htm\r\n
   Accept-Language: en-us,zh-hk;q=0.7,zh-tw;q=0.3\r\n
   Accept-Encoding: gzip, deflate\r\n
   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
1.0.3705)\r\n
   Host: www.corega.com.tw\r\n
   Connection: Keep-Alive\r\n
```

Snip 2

```
Internet Protocol, Src Addr: 46.5.180.250, Dst Addr: 209.225.0.6
Transmission Control Protocol, Src Port: 61962, Dst Port: http (80), Seq:
723942382, Ack: 3571025822, Len: 246
 Request Method: GET
   Accept: */*\r\n
   Accept-Language: en-us\r\n
   Accept-Encoding: gzip, deflate\r\n
   User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)\r\n
   Host: servedby.advertising.com\r\n
   Connection: Keep-Alive\r\n
```

Looking at these packets shows a GET request made from a Windows 2k machine running IE 6 and a separate GET request being made from a Windows 98 box running IE 5.5.

These findings would support that IP 46.5.180.250 is possibly a proxy server for allowing outbound traffic for internal clients on the protected network.

With this additional information the network seen in the log most likely resembles the following:

```
              Internal Protected Network (46.5.0.0/?)
                      |
                      |
              Few Routers Thrown In There
                      |
                      |
-------------- Proxy Server (46.5.180.250)
|                     |
|             Few More Routers (due to TTL values seen in outbound packets)
|                     |
|                     |
DMZ                Router 1 (00:00:0c:04:b2:33)
|                     |
|             Snort---
|                     |
-------------- Pix Firewall (00:03:e3:d9:26:c0)
                      |
                      |
        More Stuff (Possibly a NIDS and border router)
               |
               |
        Internet
```

2. Detect Was Generated By:

The raw log used in this analysis was generated by and unknown version of Snort using an unknown ruleset and configuration file.

The detects used in my analysis were generated by:

```
-*> Snort! <*-
Version 2.1.3-ODBC-MySQL-FlexRESP-WIN32 (Build 27)
By Martin Roesch (roesch@sourcefire.com , www.snort.org)
1.7-WIN32 Port By Michael Davis (mike@datanerds.net , www.datanerds.net/~mike)
1.8 - 2.1 WIN32 Port By Chris Reid (chris.reid@codecraftconsultants.com)
```

The version of Snort utilized in my analysis had all rules enabled and the stream4 preprocessor disabled.

Fist step was to run the log through Snort using the following command:

```
snort -c c:\snort\etc\snort.conf -l c:\snort\log -k none -r c:\snort\2002.6.18
```

Options-
-c - was used to specify a config file
-l - was used to let snort know where to log the goods
-k none - was used to ignore the munged checksums
-r was used to read in the log file

There were plenty of events to go around but I wanted to look for something that I had not seen discussed yet. I decided to go with the following signature:

```
[**] [1:540:11] CHAT MSN message [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
07/18-08:48:35.294488 46.5.180.250:61149 -> 64.4.12.155:1863
TCP TTL:125 TOS:0x0 ID:38562 IpLen:20 DgmLen:182 DF
***AP*** Seq: 0x94BC4902  Ack: 0x57D2776D  Win: 0xF906  TcpLen: 20
```

Though perhaps not as scary as some other signatures, this traffic shows a potentially large vulnerability in the protected network. If the source of this traffic is not addressed it is very possible that the analyst will be seeing a lot more of the scary signatures then he or she cares to.

The rule that caught this traffic is:

```
alert tcp $HOME_NET any <> $EXTERNAL_NET 1863 (msg:"CHAT MSN message";
flow:established; content:"MSG "; depth:4; content:"Content-Type|3A|"; nocase;
content:"text/plain"; distance:1; classtype:policy-violation; sid:540; rev:11;)
```

For this rule the stimulus was traffic outbound to port 1863, plain text content, and "MSG" included in the content. This stimulus fired rule "Chat MSN message."

3. Probability the source address was spoofed:

The source address in these signatures is not likely to be spoofed. The signature is firing on a series of instant messages being sent using the MSN messenger, if the addresses were spoofed there wouldn't be much of a conversation, the attacker would never see the returned messages.

4. Description of Attack:

The MSN traffic detected by Snort is an attack, this traffic is a violation of policy which may lead to a future attack.

In this case MSN Messenger was downloaded to a client inside the protected network, installed, configured and activated. Though I cannot say for sure, the fact that the chat rules were enabled in Snort leads me to believe that the security personnel responsible for the network are aware of the threat that Instant Messaging can pose. It is unknown if there is a policy in place for this type of activity, but there appears to be an interest in monitoring it.

5. Attack Mechanism:

Instant messaging is one of the most popular forms of electronic communication. It's nature is much like that of email, the client has an address list, is able to send and receive files, and receive rich content such as hyperlinks. This kind of activity opens another avenue for a possible attack to a protected network.

A few of the biggest threats that IM'ing presents to a network are as follows:

Information leaks - IM text is usually sent out in clear text. If there is an attacker listening they can read the entire conversation. This could be anything from how someone's dinner was the night before to sensitive company information.

Opening for a worm - According to an article seen on esecurityplanet at http://www.esecurityplanet.com/trends/article.php/3373251 , published on June 24th, 2004, attacks focused on IM software are on the rise.

"Actually, between 2002 and 2003 there was a 400 percent increase in IM malware, according to Symantec's figures. Since 2002, 25 instant messaging worms have been released into the wild, with about 20 of them coming out last year alone. At least five or six have hit the wild so far this year, reports Chien."

Open vulnerabilities - Just like any other piece of software IM software can open vulnerabilities on the machine it is running on.

An example of a vulnerability that can be introduced to a network through the use of this software can seen here: http://securityresponse.symantec.com/avcenter/security/Content/1943.html . This advisory discusses a buffer overflow condition in the MSN Messaging software which could allow remote code execution. Though this advisory is from 2002, it highlights the potential avenues for attack that the unregulated use of this type of software can open.

A more recent advisory was released in 2004 and can be read at http://securityresponse.symantec.com/avcenter/security/Content/9828.html . This advisory discusses a possible information leak in MSN Messaging software.

I believe that attackers are always looking for the path of least resistance. With much of the security focus and money these days turned to scanning emails for viruses and getting OS's patched products such as IM software may slip underneath the radar. If this software is being used on a network and not controlled many levels of security could be bypassed with the only protection being that which may be deployed on client PC's. Attackers are aware of this and I expect to see a continued rise in attacks focusing on IM software.

6. Correlations:

This is a new detect. I have not been able to find a previous paper discussing this particular signature.

Because this is signature is alerting on a policy violation there is not a CVE associated with this alert.

A good article on the threat IM software posses can be read at:
http://www.esecurityplanet.com/trends/article.php/3373251

And some vulnerabilities concerning MSN Messenger can be read at:
http://securityresponse.symantec.com/avcenter/security/Content/1943.html
http://securityresponse.symantec.com/avcenter/security/Content/9828.html

7. Evidence of Active Targeting:

There is no evidence of active targeting. The traffic appears to be normal conversation between two MSN clients.

8. Severity:

Severity is calculated using the following formula:
severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Each category is given a value between 1 (lowest) and 5 (highest).

Criticality = 3:

I cannot say for sure what kind of system is running the MSN Messenger software, most likely this is a Windows Workstation but it could just as well be a Windows Server. Since I cannot determine what kind of box is running this software I will take a middle of the road stance and assign a value of 3.

Lethality = 3

This traffic is not that of an attack. However, if this box were to be attacked by a worm it is possible for the entire network to be taken down. Since there have only been 5 or 6 worms targeting IM software seen in the wild last year I will only give this a value of 3. I would expect this value to rise as attacks against this software increase.

System Countermeasures = 1

There is no way to determine from the log what, if any, antivirus and firewall software is running on the PC. I will assume the worst case that there are no countermeasures running on the box and assign a value of 1.

Network Countermeasures = 1

This traffic is being allowed over the protected network so there are no network countermeasures in place. Snort is only being used to detect this traffic, which has to count for something so I will give this a value of 1.

Using the above formula this gives a severity rating of 4.

Severity = 4

9. Defensive Recommendations:

The first thing that should be done to combat the use of IM software is to develop a policy and make sure that each user of the network is aware of this policy. This policy will vary from company to company, but it is from these policies that the rest of the defensive actions will be determined.

In most cases I would guess that if the company is concerned enough about IM software to develop a policy for it, this policy will be a no tolerance policy. In this case I would make the following recommendations.

- Block known IM ports on company firewalls, especially those between the protected network and the outside world.

- Restrict user rights on workstations to prevent unauthorized installation of software.

- Leverage existing IDS devices to detect this traffic by enabling chat signatures. If there is no IDS device, then one should be purchased.

- Savvy users could use http tunneling software, such as httport, to bypass firewall rules and avoid chat detection. Analysts should be aware of this potential violation and look for tunneling traffic during analysis of daily IDS logs.

- Ensure all company computers are running antivirus software with up to date definitions.

- Make sure that all current and future employees are aware of this policy.

10. Multiple Choice Question:

It is decided by your company that the existing Snort NIDS will be used to detect the use of IM software. Currently, the NIDS are not configured to do so. What step(s) must be taken to enable the detection of IM traffic?

A) Disable the Stream4 Preproccesor in the snort.conf file.
B) Enable the p2p.rules file in the snort.conf file.
C) Enable the chat.rules file and enable the Stream4 Preproccesor in the snort.conf file.
D) Enable the chat.rules file in the snort.conf file.

Correct Answer: D