SANS PENETRATION TESTING

Use offense to inform defense.
Find flaws before the bad guys do.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (SEC504)"
at https://pen-testing.sans.org/events/

Short, Christopher R.
GCIH v2.1 option 2

SQL Snake and Other Port 1433 Threats
In support of the Cyber Defense Initiative

Abstract

While many attacks against systems have so far focused on web site defacements, denial of service, and other such high-profile exploits, the proliferation of databases provides a tempting target for those attackers who want another way to gain control of systems using poor configuration and highly functional but difficult to configure software.

With more and more databases available on the Internet, and even corporate intranets, it is only logical that threats will turn in this direction. One such vulnerability is called "SQL Snake". This is the name given to an exploit that targets port 1433. This particular vulnerability and exploit can be easily misunderstood: this is not simply a database vulnerability. A successful exploit will leave the attacker with administrative control of the server that the database software resides on—possibly of the domain it resides on.
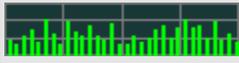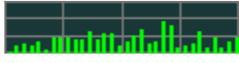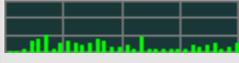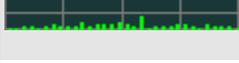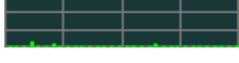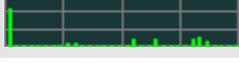
This paper will describe a known vulnerability within Microsoft SQL Server, an exploit for that vulnerability, and what can and should be done about it. It will focus less on what network administrators as individuals can do about it, but more on how they can involve their IT organization in preventing exploits such as this. Threats to an organization come from many fronts and exploit many weaknesses. Attackers will use whatever tools they have at their disposal, and defenders should do likewise.

Port Selection/Frequency of Attacks

Port 1433 is registered with IANA as assigned to Microsoft SQL Server. The listing indicates that this port uses both TCP and UDP. A "registered" port is a port which purpose has been listed by IANA for the convenience of the Internet community. Through most of the summer of 2002 this port has been among the top 10 attacked ports as listed on the incidents.org web site. The list of top 10 ports from http://isc.incidents.org/top10.html on September 14, 2002 is shown below.

Top 10 Ports

Service Name	Port Number	30 day history	Explanation
http	80		HTTP Web

			server
ms-sql-s	1433		Microsoft SQL Server
ftp	21		FTP servers typically run on this port
netbios-ssn	139		Windows File Sharing Probe
sunrpc	111		RPC. vulnurable on many Linux systems. Can get root
smtp	25		Mail server listens on this port.
???	6346		Gnutella is a peer-to-peer file sharing tool
microsoft-ds	445		
domain	53		Domain name system. Attack against old versions of BIND
printer	515		lpdng exploits in RedHat 7.0

As of this date, the number of attacks attempted against port 1433 is exceeded only by the number of attacks against port 80 (http). While these statistics don't indicate the number of *successful* attacks, this is still an alarming trend.

Other information on the Incidents.org site lists upward or downward trends in the number of attacks. On the Incidents.org homepage (http://isc.incidents.org/), the number of attacks against port 1433 is indicating no significant change in the number of attacks. It is worth noting, however, that a number of potential vulnerabilities exist on this port, and that the number of attacks against port 1433 naturally would include exploits not mentioned in this paper. On September 14, 2002, the Port report on Incidents.org for this port (http://isc.incidents.org/port_details.html?port=1433) lists 7 different CVE numbers for this port.

Description of Service

Port 1443 is used by SQL Server to accept incoming client connections. SQL Server uses the sockets network library to communicate over TCP/IP. Although IANA lists both TCP and UDP as the protocols used by SQL Server, the Microsoft documentation on the subject only speaks to using TCP with this port. (Microsoft.com/TechNet).

Protocols

According to Microsoft, this port follows the TCP/IP standard for WinSock applications. Basically, the connection sequence goes something like this:

- Client sends a SYN to port 1433 from a random source port between 1024 and 5000
- Server responds with a SYN/ACK
- Client responds with an ACK, thus establishing the connection using the standard TCP 3-way handshake
- Communication is carried out using port 1433 on the server and the randomly selected port on the client

As stated earlier, this behavior of using a "static" server port and a random client port is standard for WinSock applications. Microsoft provides the following example of this, which is the output from a **netstat –an** command. (Netstat is a command to display current TCP/IP network connections. The "a" option means to list all connections, and the "n" option lists addresses and port numbers in numeric form.)

Proto	Local Address	Foreign Address	State
TCP	157.54.178.42:1433	0.0.0.0:0	LISTENING
TCP	157.54.178.42:1433	157.54.178.31:1746	ESTABLISHED
TCP	157.54.178.42:1433	157.54.178.31:1748	ESTABLISHED
TCP	157.54.178.42:1433	157.54.178.31:1750	ESTABLISHED

(Table source: Microsoft.com/TechNet) In the above example, our mythical client has established 3 separate connections to the same SQL server, all utilizing port 1433 on the server.

The protocol used over TCP/IP for communicating with SQL Server on port 1433 is called TDS (Tabular Data Stream). This protocol reflects SQL Server's Sybase roots. TDS is a proprietary protocol originally developed by Sybase and later used by Microsoft. The official Microsoft version of TDS only runs on Windows, and public domain documentation of the protocol is incomplete. However, an open-source organization (www.freetds.org) has developed an

implementation of this protocol for Linux and Unix, and they have documented what they can. A description of the protocol, included later in this document, will be sketchy at best, considering that both Microsoft and Sybase have produced multiple implementations of the TDS protocol, and these are of dubious compatibility.

Each TDS packet begins with an 8-byte header. The header consists of the packet type, a "last packet" indicator, the packet size, and an undocumented 4-byte field. The remainder of the packet varies depending on what the packet type is. Types of packets include, but are not limited to: login, logout, column info, result set, etc.

Common Vulnerabilities

Unfortunately, it seems that the number of vulnerabilities associated with this particular service are almost too numerous to list. Any time one connects a database to a network there will be associated risks and vulnerabilities. Many of these issues involve the authentication method chosen by the system administrator.

SQL Server supports two authentication modes: it can use Windows NT/2000 authentication, or it uses its own logons (SQL Server authentication). These options may be set by the system administrator by starting Enterprise Manager and right-clicking the database server, which will bring up the following screen:

Short, Christopher R.
GCIH v2.1 option 2
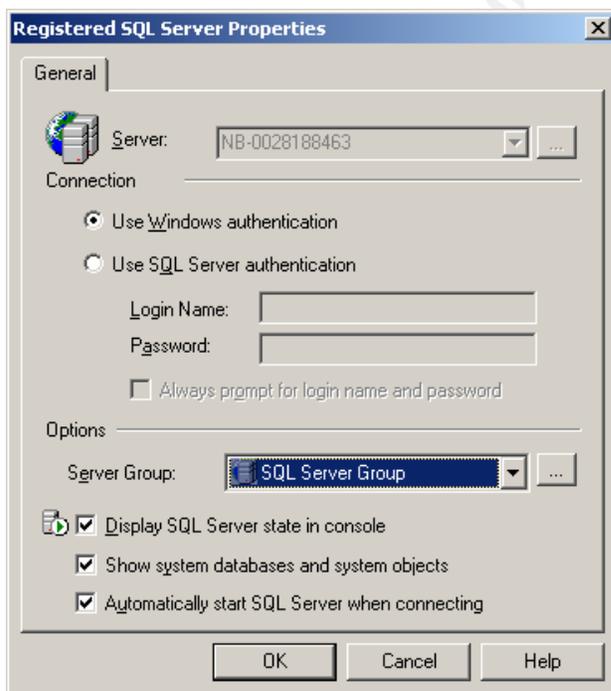
Windows NT/2000 authentication uses "trusted connections"—connections that have been validated by Windows. There are of course certain security risks associated with Windows logons, but those are not unique to SQL Server.

SQL Server authentication makes use of logon id/password combinations stored in SQL Server itself. There are 3 problems associated with this:

1. On the screen shot above, it is possible for the system administrator to specify a user ID and password that will always be used when connecting to the server. If this option is set, anyone who can connect to port 1433 will have access equivalent to whatever account is specified on this screen.

2. SQL Server's root or admin account is called "sa". By default, SQL Server versions up to 2000 ship with a blank "sa" password. SQL Server 2000 tries to discourage blank "sa" passwords, but will allow them. Previous versions do not prompt or warn about this. Many administrators never change this password, providing for easy access for anyone who can connect over the network. In particular, many administrators who use Windows authentication never bother to change the "sa" password, on the grounds that it is not used in their implementation of SQL Server. This would be the case where SQL Server is configured to use only Windows NT authentication. However, if an attacker can get the authentication method of the server changed through any of many methods, the blank "sa" password provides quick administrative access.

3. Finally, SQL Server is vulnerable to network sniffing when using SQL Server authentication. User ids and passwords are not exactly sent over the network in clear text, but it's almost as bad. There is no encryption scheme in use—passwords are subjected to a simple XOR transformation. Passwords are transmitted in UNICODE, XOR'd with a constant value. Since this value is constant across all SQL Servers, "decoding" the password is not terribly difficult, especially when one considers that the second byte of every password on the network will be 0xA5. The reason for this has to do with how Unicode works: since Unicode is a "wide" character set, the second byte is not needed for character representation, at least in Western languages. Therefore, the second byte of each character is NULL, and XORing any value with NULL (0x00) will give you the value you started with, in this case 0xA5. It is therefore not difficult to decode the password, and it is easy to find passwords being transmitted across the network, due to the use of Unicode. (Litchfield, p.5) A stored procedure for encrypting and decrypting SQL Server passwords may be found at http://www.sqlsecurity.com/uploads/decrypt_odbc_sql.txt.

Short, Christopher R.
GCIH v2.1 option 2

The above are design and administration flaws in SQL Server that attackers can exploit. However, SQL Server has a number of other vulnerabilities which can be exploited over a network. Many of these are detailed by CERT and will be referenced by their CERT numbers below.

One vulnerability is a classic buffer overflow, which is peripherally related to the network sniffing vulnerability listed above. While SQL Server uses no encryption worthy of the name regarding password authentication over a network, it does store passwords in an encrypted format in its database. When the "unencrypted" password arrives over the network, it must be encrypted in order to compare it to the stored encrypted password. SQL Server uses a .dll called pwdencrypt.dll that performs this function, but which has a buffer overflow problem. If exploited by sending a properly crafted password, a user can execute arbitrary code on the server. This code will execute with the privileges of the SQL Service account. Since Microsoft offers the option to run the SQL Service account as "Local System", this provides for some interesting exploits against systems with administrators who aren't paying attention. (CERT note VU#225555).

Another vulnerability with this service that results from a lack of proper default security is the fact that many "extended stored procedures" that ship with SQL Server 2000 are not secured relative to the actions they can perform. A good definition of extended stored procedures can be found at Swynk.com:

> Extended Stored Procedures are DLLs that can be called from within SQL code using the same conventions as Stored Procedures. As DLLs they have access to the operating system, other DLLs, OS files, etc. They are executed in the process/address space of SQL Server and thus have the potential to crash SQL Server. (Wynkoop and Hotek)

The above quote should be sufficient to point out the potential vulnerabilities without much further analysis. Suffice it to say that these extended stored procedures are designed to interact with the operating system, and are capable of making configuration changes. Furthermore, these procedures appear to have been designed with this very functionality in mind. While they are no doubt useful, code which allows a running service to perform operating system functions in its own security context must be tightly controlled.

Unfortunately, this is not the case with several powerful stored procedures. Several extended stored procedures are, by default, executable by members of the built-in Public role on SQL Server.

Several articles, including one on Swynk.com, make statements to the effect that the Public role " is the equivalent of the NT Everyone or Authenticated Users group" (Warren). This is not entirely accurate: the group "Everyone" in Windows NT/2000 includes anonymous users, while the group Authenticated Users, in

keeping with its name, does not. All database users belong to the SQL Server Public role, and this may not be revoked. However, a user must have some kind of login to the server, either by Windows NT/2000 authentication, or by a SQL Server logon, in order to access database objects granted to the Public role. Thus, the Public role is more akin to "Authenticated Users" than it is to "Everyone". (Note: this line of reasoning led to an attempt by the author to add the NT "Everyone" group to a SQL Server 2000 SP2 login list. If successful, this would have permitted anonymous queries against a given database. However, Microsoft wisely does not provide a facility for doing this: the "Everyone" group, as well as "Authenticated Users", do not appear in the "SQL Server Login Properties" dialog box in Enterprise Manager. Other groups, such as "Domain Users" and "Users' do of course appear, but it is not possible to create an anonymous logon using the GUI tools that Microsoft provides. Attempts to add the "Everyone" group by means other than Enterprise Manager were not undertaken.)

Having established that SQL Server does not easily permit totally anonymous access, there are still problems with privilege elevation or other code exploits from low-level user accounts. Generally these problems occur in one of two ways: exploitation of code that runs in a higher-privileged process than the user, or a classic buffer overflow.

The first is detailed in Microsoft's Security Bulletin MS02-043. It seems that there is a vulnerability in the following stored procedures: xp_execresultset, xp_printstatements, xp_displayparamstmt. The vulnerability exists because these stored procedures have the ability to reconnect to the database using a higher privilege level than the process that called them. In this way, an attacker can cause code to execute that he or she does not have permission to run.

Protocol used

As indicated earlier, this exploit uses Port 1433 (TCP), though an administrator can change that in SQL Server if desired. There do not seem to be particular operational reasons to do so, however. The protocol which listens on that port is Microsoft's implementation of TDS (Tabular Data Stream). While this protocol was developed by Sybase, and is still in use in Sybase products, the particular implementations of TDS are proprietary to both Microsoft and Sybase products. According to FreeTDS.org, the two versions were once identical. As is the nature of proprietary protocols, the two diverged some time ago. The two current implementations are compatible enough to create connections to each others' database products, but apparently incompatibilities will soon become apparent if production use is attempted. The version history, again from FreeTDS.org, looks like this:

TDS 4.2 was used in both Sybase databases and in the original version of SQL Server that Microsoft bought from Sybase. Version 5.0 was written by Sybase

for their products and is not used in Microsoft products. Version 7.0 is unique to SQL Server 7. It introduced support for Unicode and fields of larger than 255 characters. The current Microsoft version, version 8.0, is designed to support SQL Server 2000. (FreeTDS User's Guide)

TDS packets are variable length and come in a variety types. Because TDS is a proprietary protocol, and documentation is limited to what is released by the publishers or what can be reverse-engineered, documentation for TDS version 8 packet formats could not be found. The following should be considered accurate up to TDS version 7. TDS packets start with an 8-byte header. The first byte indicates the packet type. This will have one of the following values:

    0x01 TDS 4.2 or 7.0 query
    0x02 TDS 4.2 or 5.0 login packet
    0x04 responses from server
    0x06 cancels
    0x0F TDS 5.0 query
    0x10 TDS 7.0 login packet

The next byte is the "last packet indicator. This will have a value of 0 if there are more packets or 1 if this is the last packet.

The next two bytes indicate the packet size, and the last four are undocumented, but seem to be always zeroes.

What happens next depends on the type of packet. One type of packet, and the most important to exploiting SQL Server for the purpose of gaining control of a system, is the login packet. Proper documentation of the login packet for SQL Server versions up to 7 may be found at http://www.freetds.org/tds.html#login. If the server is configured to use SQL Server authentication instead of NTLM authentication, the login packet will contain the "encrypted" password. This has implications that go beyond the exploits that are currently the reason for all the port scans taking place on the Internet right now—implications that will be discussed under "theoretical exploits" after the current popular one is explained.

Specific Exploit

The specific exploit covered here is the abuse of the extended stored procedure "xp_cmdshell". This stored procedure, according to the Microsoft documentation, allows a user to execute "a given command string as an operating-system command shell and returns any output as rows of text." The documented syntax (from SQL Server 2000 Books Online) appears as follows:

    Syntax
    xp_cmdshell {'*command_string*'} [, **no_output**]

Arguments
'*command_string***'**
Is the command string to execute at the operating-system command shell.
command_string is **varchar(255)** or **nvarchar(4000)**, with no default.
[truncated by author]

no_output
Is an optional parameter executing the given *command_string*, and does
not return any output to the client.

Obviously this kind of power is dangerous, which is why, by default, only
members of the *sysadmin* role in SQL Server have permission to execute this
extended stored procedure. The documentation further states that, when this
stored procedure is executed by members of the *sysadmin* role, the "command
string" is executed in the process of the account that SQL Server is running in. It
also states that users who are not in the *sysadmin* role may be granted
permissions to use this procedure. In that case, the "command string" is
executed in the process of the SQL Server Agent Proxy Account, if one is
specified. Otherwise, the stored procedure will not execute.

Finally, the documentation states that, in versions of SQL Server prior to 2000, if
a user had permissions to run this extended stored procedure, that it executed in
the process of SQL Server. A configuration setting can be changed to make
earlier versions behave as SQL Server 2000 does in this regard, but an
administrator must perform the change.

There are versions of SQL Server that run on Windows 9x. Since Windows 9x
does not have the concept of security roles, any code executed by the
xp_cmdshell stored procedure runs in the context of the currently logged in user,
which is unrestricted on those systems.

While it is a good idea to restrict this sort of procedure to people who would be
authorized to run commands anyway (administrators), there are a few simple and
devastating problems. The only thing standing between an attacker and a
remote exploit is not being a member of the sysadmin role in SQL Server. An
attacker connecting remotely must therefore figure out how to gain access to
SQL Server with an account that is a member of that role. The situation which
sets up the exploit is this:

- SQL Server allows blank passwords on the 'sa' account (without warning
  in versions prior to 2000)
- The 'sa' account is an all-powerful administrative account with regards to
  SQL Server.
- The 'sa' account can not be disabled.

Therefore, all an attacker has to do is find a SQL Server with a blank 'sa' account password, execute the stored procedure xp_cmdshell, and the attacker can execute any command in the context that SQL Server runs in.

It is a judgment call as to whether this exploit is one of a poorly designed system or one that targets poor administrative work. After all, SQL Server does provide one fairly easy method of thwarting this exploit—have a password on the 'sa' account. There are other countermeasures that can be taken, as well as other potential variants of this exploit, which will be discussed later. Naturally, this kind of exploit is just screaming for automation (from an attacker's point of view), which most likely accounts for the current popularity of port 1433 as a target.

This exploit, and code to execute it (a worm), has been published by numerous sources and given numerous names. The variant which will be discussed here is called "SQL Snake". It is also known as Spida and Digispid, according to CERT. The CERT Incident Number given to this exploit/code is IN-2002-04.

The exploit works on any operating system that can run SQL Server *or* the SQL Server "run-time" package that Microsoft makes frequently available, called MSDE (Microsoft Data Engine). This "stripped down" version of SQL Server may be more vulnerable than the regular edition, as will be explained later. Operating systems that support either the runtime or the real version of SQL Server include Windows 9x, NT, 2000, and XP (all editions support some version, even it it's only the MSDE runtime).

This exploit uses TCP port 1433, which is the normal SQL Service client connection port. However, this is not an exploit against the network protocol itself, but an attack against the application that it supports. (Note: the use of port 1433 may be changed by a server administrator to another arbitrary port. In this case, the attack could be modified to use other ports as well, though the "stock" variety does not have facilities for this.)

SQL Snake is a JavaScript worm. This particular exploit performs the following steps, according to CERT:

1. assigns the guest user to the local Administrator and Domain Admins groups
2. copies itself to the victim system
3. disables the guest account
4. sets the 'sa' password to the same password as the guest account
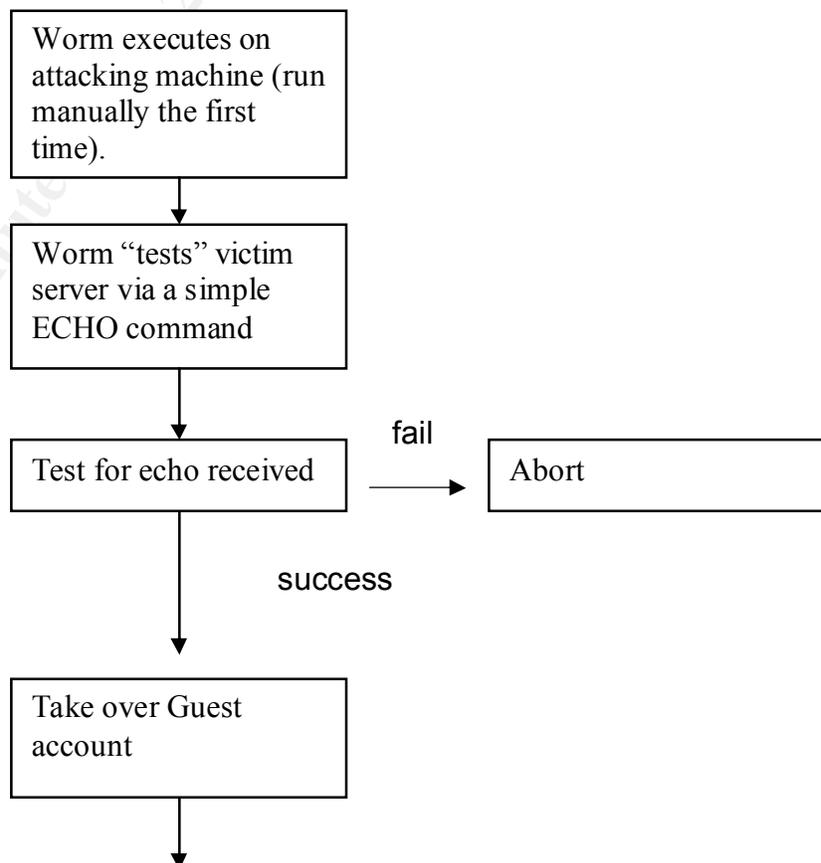5. executes the copy on the victim system

(CERT)

Short, Christopher R.
GCIH v2.1 option 2

The closest variant to SQL Snake is the "Kaiten" malicious code (CERT Incident Note IN-2001-03). This doesn't appear to be a self-spreading worm so much as a way to attack a specific system. It doesn't have its own scanner, but is rather preceded by scanning against port 1433. Kaiten is designed to receive commands from an attacker over an IRC channel that it listens on. SQL Snake is more automated, does not use IRC, and does not have facilities for processing arbitrary commands from a remote source. However, it is capable of spreading automatically.

As stated earlier, SQL Snake is designed to take advantage of the following situation:

- The attacker has access to port 1433
- SQL Server is running in Mixed (Windows NT and SQL Server) authentication mode (note: there is no such thing as "SQL Server-only" authentication mode)
- SQL Server is configured with no 'sa' password
- SQL Server is running in an account with high privileges (administrators group, local system, etc.)

The basic attack, when automated, works like this:

```
┌─────────────────────┐
│ Copy worm files to  │
│ SQL Server          │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ Execute worm on     │
│ target machine      │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ Set guest account   │
│ back the way it was │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ "Inventory" database,│
│ server configuration,│
│ and attempt to steal │
│ domain passwords—    │
│ store in file        │
│ "send.txt"           │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ Scan for other      │
│ vulnerable hosts.   │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ Email configuration │
│ data (send.txt) to  │
│ author's email      │
│ account             │
└─────────────────────┘
          │
          ▼
┌─────────────────────┐
│ Clean up scripts,   │
│ remove files created,│
│ close network       │
│ connections.        │
└─────────────────────┘
```
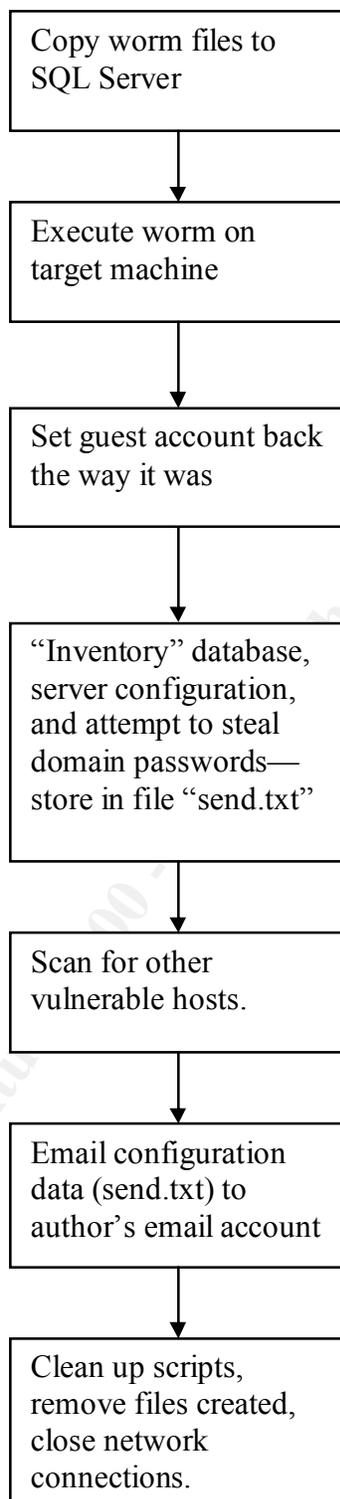
Note that the automated attack is very linear—it only really uses one major decision point in its logic. If a step doesn't work, the attack moves on to the next step. The only thing which causes a complete abort is if the target machine is protected. The attack is not guaranteed to succeed, though. Even if it locates a

vulnerable machine with a blank 'sa' password, exploiting the vulnerability requires that SQL Server have proper permissions to do what the attacker wants. Unfortunately, many administrators who fail to adequately secure their administrative accounts also fail to use the principle of least privilege.

Performing the initial attack "by hand" is rather trivial. Using the steps CERT outlined, the first thing to do is to take over the Guest account. Most administrators wisely disable this (done by default in Windows 2000), so it must be re-enabled. For the purposes of this demonstration, the SQL Query analyzer is used to enter the commands, logged on as 'sa' with no password entered. The actual exploit uses Microsoft's ADO (ActiveX Data Objects) to enter its commands. The result is the same. The Guest account is made active with the following command (output follows, with graphics removed for brevity. Commands entered are italicized, responses are in normal type.):

> *exec xp_cmdshell 'net user guest /active:yes'*
>
> The command completed successfully.
> NULL
> NULL

Next, the Guest account is assigned a password:

> *exec xp_cmdshell 'net user guest randompassword'*
>
> The command completed successfully.
> NULL
> NULL

Finally, the user Guest must be made a member of the local Administrators and Domain Administrators group Some other powerful groups would probably work work, but this is a very good combination. (Note: the output is identical to that shown above for these commands and has been omitted.)
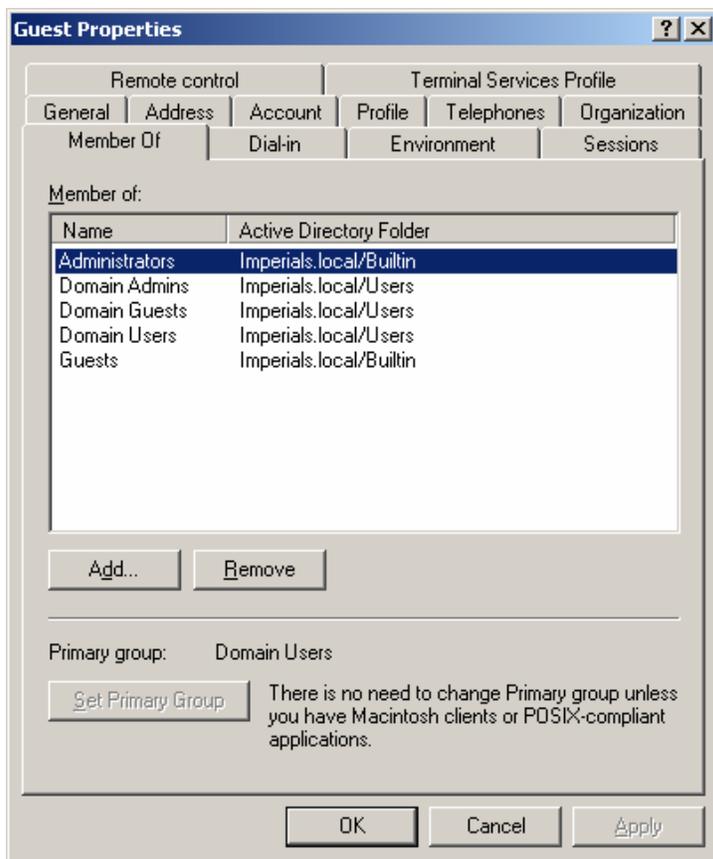
For the Administrators group:

> *exec xp_cmdshell 'net localgroup administrators guest /add'*

For the Domain Administrators group:

> *exec xp_cmdshell 'net group "Domain Admins" guest /add'*

The results of all of this can be checked so far by going into Active Directory Users and Computers (User Manger for Domains on Windows NT Server, or User Manager on NT Workstation)

From the screenshot above, our test server now has the Guest user as a member of "Administrators" and "Domain Admins". At this point, the attacker (or the script) has an account with complete control over the target server.

Up until now, this attack has been fairly generic—the steps presented above are the "setup" portion of the attack. Most worms that exploit a blank password in SQL Server will behave in a similar fashion. A user could do just about anything from the command line. As indicated earlier, the exploit code, does some very specific things.

Using the exploit code is rather simple. The attacker's machine must have JavaScript installed, and must have some fairly recent version of MDAC (Microsoft Data Access Components) installed. Almost any stock Windows machine in an office setting will meet these requirements. The file the attacker would use is called "sqlinstall.bat", and it is designed to install the worm on a specified target machine. The actual work is performed by a file called "sqlexec.js", which is a JavaScript program that is capable of executing any command against a vulnerable server.

Short, Christopher R.
GCIH v2.1 option 2

The main part of this program (from code analysis posted on Incidents.org) looks like this:

```
function usage()
{
WScript.Echo("sqlexec v1.1n" + "n" + # isn't this friendly?
"Usage : " + WScript.ScriptName + " ip user pass cmdn" +
"n" +
"Note : symbol " has replaced by ``n");
WScript.Quit();
}

if (WScript.Arguments.length < 4) #
usage(); # Take all params &
# neaten them up.
execstr = WScript.Arguments(3); #
#
for (counter = 4;counter < WScript.Arguments.length;counter++)
execstr += " " + WScript.Arguments(counter);

cn = new ActiveXObject("ADODB.Connection"); # ActiveX Data Object
cn.Provider = "sqloledb"; # through SQL OLE DB provider.
cn.Properties("Data Source").Value = WScript.Arguments(0); #
cn.Properties("User ID").Value = WScript.Arguments(1); # This is a nice,
generic
cn.Properties("Password").Value = WScript.Arguments(2); # command
wrapper,fairly
cn.Open(); # flexible.
#make a connection to the sql server
cmd = new ActiveXObject("ADODB.Command"); #
cmd.ActiveConnection = cn; #
cmd.CommandText = "xp_cmdshell '" + execstr.replace(/`/g, "'") + "'";#
The key part, via xp_cmdshell
cmd.CommandType = 1; # to run commands
rs = cmd.Execute(); #
```

This script simply takes the ip address of the target machine, a desired user name, password, and command to execute. It would be an interesting attack by itself, in fact. The rest of the code listed above simply creates an ADO connection to the server using the user ID and password supplied. It then creates a Command object of type Text, and executes the stored procedure "xp_cmdshell". This script executes all the commands that the worm uses to propagate itself, and is used to automate the manual examples shown previously.

Once loaded, the worm proceeds to scan for other vulnerable servers across a pseudorandom list of subnets. It does this using fscan.exe, which is a freely available port scanner from Foundstone. Screen output from fscan, scanning a local network for port 1433 looks like this:

```
C:\WINNT\System32\cmd.exe                                         _ □ ×
Copyright 2002 (c) by Foundstone, Inc.
http://www.foundstone.com

No IP addresses provided - using local network: 192.168.2.1-192.168.2.254
 Scan started at Fri Sep 20 23:22:23 2002

192.168.2.242    1433/tcp

 Scan finished at Fri Sep 20 23:24:32 2002
 Time taken: 3 ports in 128.781 secs (0.02 ports/sec)
```

Fscan is capable of producing output to a file, of course, which is what the worm uses. Another "third-party" program used by this worm is clemail (http://www.bysoft.se/sureshot/clemail/). This program allows email to be sent from the command line on a Windows machine. It is used to attempt to email the information gathered from the infected host to the author's email address, which has since been terminated.

A detailed description of the source code may be found at http://www.incidents.org/diary/diary.php?short=n&id=157.

Theoretical vulnerabilities

The exploits described have a common purpose: exploit a blank 'sa' password to gain access to a server, and use the database server's extended stored procedures to gain administrative access to the machine that it is running on. The commonly accepted defense is to put a password on the SQL Server 'sa' account and call it solved, since having a password on the account will block the current crop of malicious code.

There is a problem, however. The lack of a password on the 'sa' account is simply a vehicle to the vulnerability. If an attacker can gain access to the 'sa' account, with or without a password, the rest of the attack can be carried out with no modifications.

This is where the description of the TDS protocol becomes pertinent. Remember that when SQL Server authentication is enabled, the password is transmitted in the logon packet in near-cleartext, with only some relatively simple obfuscation to conceal it. If an attacker is able to sniff the network that a SQL Server resides on, then the attacker could carry out these exploits against that server using either the 'sa' account or another account with sufficient privileges.

This of course is not the only way to obtain an administrative password. A cardinal sin in the world of Microsoft web application design is coding a database

password directly into an Active Server Page. If an attacker can use any of many techniques to read the source code of a page that accesses databases, he may be able to learn a password. Some application developers are careless enough to use the 'sa' password in their applications, which is a practice that should never be permitted.

Additional vulnerability

A misconfigured system is bad enough: a system that doesn't permit proper configuration is even worse. Such may be the case with Microsoft's MSDE. This is a personal version of SQL Server. It may be installed as a run-time package onto a user's system without the knowledge of the user. MSDE comes with no administration tools. It may be administered by the SQL Server Enterprise Manager, but a user would have to acquire a license for SQL Server in order to use the administration tools that come with it. For users who otherwise have SQL Server, this is fine. However, users who acquire MSDE through redistribution may be out of luck in this area.

In the course of research, at least one third-party product was identified that could help in this area: ASP Enterprise Manager is a non-Microsoft product that can carry out some of the tasks that the Microsoft Enterprise Manager can. It is available at http://www.aspenterprisemanager.com/, but was not tested as part of this project.

Defense

Defending against this type of attack is both trivial and difficult. It is trivial for an experienced administrator to do so. Most notes on the subject mention two simple steps to take. First, assign a password to the 'sa' account. This is so often overlooked it isn't even remotely funny. Second, block port 1433 for all but computers that need access to SQL Server, and certainly from the Internet.

Certainly assigning the 'sa' password will stop this attack dead in its tracks. Also, blocking port 1433 at the border will stop the worm from infecting your systems by remote, and will stop any infected systems inside your network from scanning out to infect other systems.

However, these measures by themselves could lead to a false sense of security. First, it wouldn't be terribly difficult to take this work and make a useful Trojan out of it, which could then be distributed to unsuspecting users inside a target network. Second, blocking port 1433 at the border doesn't do anything about an attacker *inside* one's network, nor will it stop the spread of the worm inside a network once a machine is infected.

Unfortunately, detecting this worm is not terribly easy, because much of the traffic it generates looks like normal SQL Server traffic, specifically ADO

connections. The only thing that an intrusion detection system could detect reliably would be the port scanning that the worm attempts to perform. Multiple scans to port 1433 across a subnet or range of IP addresses would be cause for immediate concern, since legitimate SQL Server clients don't behave like that. A string match for the "guest" account might also work, but that could present problems on networks where that account is used for other things. An intrusion detection system that can properly decode TDS packets could also look for calls to the external stored procedures that are used in carrying out the exploit by looking for the extended stored procedure names. This would be dependent on knowing that legitimate traffic does not use those procedures, however. Some commercial scanning and IDS packages can decode TDS correctly, but these were not tested for this project.

SQL Snake does leave some files on attacked machines that can be scanned for. Specifically, according to CERT, the files transferred are:

- %SystemRoot%\System32\drivers\services.exe
- %SystemRoot%\System32\sqlexec.js
- %SystemRoot%\System32\clemail.exe
- %SystemRoot%\System32\sqlprocess.js
- %SystemRoot%\System32\sqlinstall.bat
- %SystemRoot%\System32\sqldir.js
- %SystemRoot%\System32\run.js
- %SystemRoot%\System32\timer.dll
- %SystemRoot%\System32\samdump.dll
- %SystemRoot%\System32\pwdump2.exe

If any of these files are present on a machine, it is a near-certain indication of a compromise.

One thing that should be done by any administrator, with appropriate permission if necessary, is to scan one's own network looking for vulnerable servers. Keep in mind that SQL Server comes in many flavors, and any given organization may have many more than administrators (or even users) are aware of. Developer machines, users with the current version of Microsoft Access (which includes MSDE), and users with third party software that uses MSDE may all have small and unprotected versions of SQL Server. One scanner used in the course of this project is SnakeScan from PentaSafe (www.pentasafe.com). It is available free of charge (registration required). This tool scans a local network for SQL Servers and then attempts to log into them as 'sa' with a blank password. It then provides a report listing vulnerable and not vulnerable servers. Generally it performs the initial steps used in the SQL Snake worm, omitting the malicious parts.

An alternative would be to identify SQL Servers using the same fscan tool that the SQL Snake worm uses. The tool itself is not malicious—it is merely used in that manner by this automated exploit. If the number of servers found is small, 'sa' logins with blank passwords could be attempted using Enterprise Manager or Query Analyzer. Users who do not have explicit full control over all aspects of their networks are, as always, cautioned to get written permission prior to using any tools such as these or attempting to access systems they do not normally administer. The risk of harm is low, but an IDS might register the scan as an attack, and the risk to one's career is high if someone gets embarrassed over a vulnerability and decides to take it out on the messenger.

Since an in-progress attack may very well look like legitimate traffic, depending on a site's configuration, prevention is paramount.

Port 1433 should be blocked at the gateway, as mentioned before. If UserID/Password authentication over the Internet is a must, then a VPN solution should be used. Port 1433 should never be exposed to a public network. In addition to this, there are several other countermeasures that should be put in place:

First, SQL Server authentication should be disallowed on a network wherever possible. Windows NT/2000 authentication, while not perfect, is far more secure than the simple username/password combination transmitted in near-cleartext. This may not be easy, as it goes to vendor product selection and internal programming standards, as will be described shortly.

Second, consideration should be given to protecting SQL Servers *inside* the network, to prevent attacks from Trojan programs or internal malicious users. This goes not simply to network design, but also to application design as well, which can present some serious political complications for network administrators.

Consider that applications that use SQL Server can be written in one of the following scenarios:

- A client-server application using NTLM authentication to access SQL Server.
- A client-server application using SQL Server authentication.
- Applications using some kind of "proxy", where the user's machine never makes a connection to the database server. An example would be a web application where a web server makes the connection, or a client/server application where a transaction monitor or application server is involved.

In the first scenario, the client accesses SQL Server directly, but since the authentication is based on NTLM, SQL Server security need not be turned on at the server. In this case, a policy to only allow NTLM authentication to SQL

Server would be most helpful and appropriate. This would limit the SQL Server to Intranet use, but that's appropriate in the majority of situations.

In the second instance, which is unfortunately rather common, two problems present themselves. First, the very insecure SQL Server authentication must be turned on, and second, passwords of some type must be either entered by the user or, even worse, stored in some format on the user's computer. Curiously, this problem may be a result of security options that only Microsoft offers. Other databases, such as Oracle, also use UserID/Password combinations for authentication. Because UserID/Password is a "lowest common denominator" of database authentication, this type of authentication tends to be very common in situations where a developer is trying to support multiple database types. Commercial applications in particular tend to be designed in this way. Perimeter protection may be the only feasible option in this kind of instance, depending on how many client machines need to access databases or if they can be segmented. One could try to convince management to not purchase such software on the grounds that it introduces an unacceptable security hole, but such a position is unlikely to succeed.

Finally, an application can use a proxy machine to access a database. A web server does this—the client machine doesn't make a connection, the web server does. Because of this type of architecture, the database server can be protected by a firewall internally, denying access to other machines both internally and externally. By prohibiting access by any but trusted computers, SQL Server authentication can be used with relative security. An internal firewall or protected segment can be used for this, as well as the network security features found on Windows 2000 Server.

This type of application architecture need not be limited to web servers. Transaction-monitor programs like Microsoft Transaction Server, or COM+ services in Windows 2000 can achieve the same thing. Other vendors make products for this purpose, and custom objects can also be written to achieve the same thing. Finally, the emerging world of web services provides alternatives that may be used in client/server applications. These obviously have their own security implications, but keeping users from directly connecting to database servers is never a bad thing, provided a larger hole isn't opened in the process.

One other technique that can be used to thwart attackers is to not give them the target that they're looking for. The xp_cmdshell extended stored procedure will execute code in SQL Server's process. When installing SQL Server, the easiest process to have it run in is that of "Local System". The Local System account on a Windows NT or 2000 server has full access to the machine. SQL Server does not, however, require full access to the machine in order to run properly, unless some application that legitimately uses these extended stored procedures requires it.

For this example, the same SQL Server that was abused in the exploit demonstration was reconfigured slightly. Instead of running as Local System, the server was reconfigured to run as an account which is a member of only the Domain Users group. The 'sa' password is reset to blank (with no complaint from the server), and the beginning of the exploit is attempted again:

> *exec xp_cmdshell 'net user guest /active:yes'*

When SQL Server was running as Local System, this resulted in the activation of the Guest account. This time, though, the following response was returned:

> System error 5 has occurred.
> NULL
> Access is denied.
> NULL
> NULL

Clearly, then, setting the database server to run as a non-administrative account will prevent the common scripted exploits such as SQL Snake from executing, even if the 'sa' password is blank. However, it should be kept in mind that what this is preventing is an attack on the server's operating system—the database and the data in it would still be vulnerable to regular queries or malicious updates as a result of an attacker having administrative access. If one is doing this, the SQL Server Agent service should not be forgotten either—it should get its own account and not run as Local System as well.

The reason so many SQL Servers run as Local System is that, in its installation routine, Local System is simply the easiest option for an installer to choose. It is possible to select another account, but the account must have already been created—the installer does not provide the option to create one from the installation routine, or the ability to call the proper tool to do so. It is possible to exit the install routine, or simply pause it and create the desired user accounts, but that requires thinking about it while in the middle of what should be an automated routine. Additionally, guidance is not provided at that critical moment as to which permissions are necessary for SQL Server to run. A novice administrator, or one simply in a hurry, may select the account with the highest privileges in order to avoid breaking something later. In fairness, this practice isn't limited to Microsoft products—the Unix world has had its share of problems with unnecessarily over privileged software. This is an education and policy issue for now, unless the vendor addresses this problem.

For all of these suggestions, the involvement of more than just security and systems administration staff is necessary. The solutions suggested can involve upper management, programming staff, and even end users. Communication and coordination are paramount—attackers do that all the time.

So what could the vendor do to help? In the course of this paper, several points have been identified that Microsoft could have addressed prior to releasing the current version of SQL Server.   Considering that these vulnerabilities and exploits span SQL Server versions, some would consider their failure to do so to lack any reasonable excuse.

First, blank passwords for the 'sa' account should never be permitted. While blank passwords are certain to please developers, the chaos resulting from a compromised system is not worth it. SQL Server should disallow a blank password. Admittedly, Microsoft has, in SQL Server 2000, provided a warning at install time if a blank password is used. However, when the 'sa' password was changed from a value back to blank for this demonstration, no warning was raised. At a minimum, this warning should be added to any attempt to change any password to a blank value, and should absolutely reappear when the authentication method of SQL Server is changed from Windows only to Mixed Mode. In fact, the change should prompt the administrator to enter a proper password for the 'sa' account prior to activating the change.

Second, the SQL Server installation routine should create low-privileged accounts for the SQL Server service and the Agent service to run in. Naturally an administrator should have the option to change this or to override it, but this one simple change, which is not mentioned in the world as often as the 'sa' blank password problem, would have prevented a lot of the successful attacks that have been carried out to date. Even the use of the Domain Users group in the example, while effective, does not guarantee that every function of SQL Server will work correctly—only the vendor or extensive testing can do that, though the system in question runs quite well in that configuration normally.

Third, the authentication method for TDS should be changed to use a more secure password encryption method. This would likely break compatibility with earlier versions. If it did not, then exploiting compatibility routines would probably present as many headaches as maintaining down-level authentication compatibility has for Windows 2000.

Microsoft should also provide some kind of administration tools for their MSDE database product. While they do give this away free of charge, and probably don't want it competing with Microsoft Access or with SQL Server, a full-fledged version of Enterprise Manager is not necessary. What is necessary is a way for users to be able to do something about blank 'sa' passwords in this stripped-down SQL Server without having to write their own code or depend on vendor code to do so. With the appearance of other tools to do this, perhaps this isn't as important, but other people's work is no substitute for proper design and due consideration of how products may be used or abused.

The presence of extended stored procedures and their powerful ability to interact with the operating system is the key to the current crop of exploits. While they do provide rich functionality, the price has been high. Microsoft should reconsider the inclusion of these procedures, or at the very least, provide a policy that would allow administrators to prevent SQL Server from interacting with the operating system in this manner. Such a setting should be settable only by the system administrator and should not be anything that the database administrator or any other SQL Server account can override.

Although it is probably not their overriding concern, it would help security in general a great deal if other database vendors would support authentication on Windows systems other than by UserID/Password combinations. This probably won't happen because it wouldn't strengthen their position on other platforms, but it would certainly go a long way towards providing software vendors who write for Windows a reason to use NTLM authentication.

Since Windows NTLM authentication has its own problems, it would be especially nice if the IT industry as a whole can come up with some standards for authenticating users based on something other than poorly encrypted text strings.

Additional Resources

For further reading on the vulnerabilities associated with port 1433 and associated exploits:

http://www.incidents.org/diary/diary.php?short=n&id=157

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

http://www.cert.org/incident_notes/IN-2001-13.html

http://www.eeye.com/html/Research/Advisories/AL20020522.html

http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q313418

REFERENCES

"Port Numbers and Services Database". 16 August 1995. URL:
http://www.sockets.com/services.htm#WellKnownPorts (14 September 2002)

"Top Ten Ports". 14 September 2002. URL: http://isc.incidents.org/top10.html
(14 September 2002)

"INF: TCP Ports Needed for Communication to SQL Server Through a Firewall".
1 February 2001. URL: http://support.microsoft.com/default.aspx?scid=kb;en-
us;Q287932 (13 September 2002)

Litchfield, Jeff. "Threat Profiling Microsoft SQL Server". 20 July 2002. URL:
http://www.nextgenss.com/papers/tp-SQL2000.pdf (13 September 2002)

Bruns, Brian and Lowden, James K. "Free TDS User's Guide". 8 September
2002. URL: http://www.freetds.org/userguide/ (19 September 2002).

Wynkoop, Stephen and Hotek, Michael. "SQL Server FAQ", URL:
http://www.swynk.com/faq/sql/sqlfaq_development.asp#InterXP (15 September
2002)

"Spida Worm Analysis". 22 May 2002. URL:
http://www.eeye.com/html/Research/Advisories/AL20020522.html (18 November
2002)

Bakos, George and Giang, Goufei. "SQLSnake Code Analysis". 21 May 2002.
URL: http://www.incidents.org/diary/diary.php?short=n&id=157 (5 September
2002).

Warren, Andy. "SQL Permissions: The Public Role". URL:
http://www.swynk.com/friends/warren/sqlpermissionspublicrole.asp (19
September 2002).

"xp_cmdshell" "SQL Server 2000 Books Online", Microsoft.

"Exploitation of Vulnerabilities in Microsoft SQL Server" 23 May 2002.   URL:
http://www.cert.org/incident_notes/IN-2002-04.html (18 September 2002)

""Kaiten" Malicious Code Installed by Exploiting Null Default Passwords in
Microsoft SQL Server " 27 November 2001. URL:
http://www.cert.org/incident_notes/IN-2001-13.html (18 September 2002)

Short, Christopher R.
GCIH v2.1 option 2

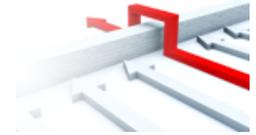"Appropriate Uses of MSDE", 1 October 2002. URL:
http://www.microsoft.com/sql/howtobuy/msdeuse.asp (11 November 2002)

Upcoming SANS Penetration Testing

SANS PENETRATION TESTING

Click Here to
{Get **Registered!**}

SANS Reboot - NOVA 2020	**Arlington, VA**	**Aug 10, 2020 - Aug 15, 2020**	**Live Event**
Instructor-Led Training | Aug 10 ET	**, VA**	**Aug 10, 2020 - Aug 15, 2020**	**CyberCon**
SANS Reboot - NOVA 2020 - Live Online	**Arlington, VA**	**Aug 10, 2020 - Aug 15, 2020**	**CyberCon**
Live Online - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	**, United Arab Emirates**	**Aug 11, 2020 - Aug 29, 2020**	**vLive**
Instructor-Led Training | Aug 17 ET	**, DC**	**Aug 17, 2020 - Aug 22, 2020**	**CyberCon**
Instructor-Led Training | Aug 17 MT	**, IL**	**Aug 17, 2020 - Aug 22, 2020**	**CyberCon**
SANS Summer Hack Europe 2020	**, United Arab Emirates**	**Aug 17, 2020 - Aug 28, 2020**	**CyberCon**
Cyber Defence APAC Live Online 2020	**, Singapore**	**Aug 17, 2020 - Aug 22, 2020**	**CyberCon**
SANS Essentials Live Online 2020	**, Australia**	**Aug 17, 2020 - Aug 22, 2020**	**CyberCon**
SANS Japan Bilingual Live Online	**, Japan**	**Aug 31, 2020 - Sep 05, 2020**	**CyberCon**
SANS London September 2020	**London, United Kingdom**	**Sep 07, 2020 - Sep 12, 2020**	**Live Event**
SANS London September 2020 - Live Online	**London, United Kingdom**	**Sep 07, 2020 - Sep 12, 2020**	**CyberCon**
SANS Philippines 2020	**Manila, Philippines**	**Sep 07, 2020 - Sep 19, 2020**	**Live Event**
SANS Baltimore Fall 2020 - Live Online	**Baltimore, MD**	**Sep 08, 2020 - Sep 13, 2020**	**CyberCon**
SANS Baltimore Fall 2020	**Baltimore, MD**	**Sep 08, 2020 - Sep 13, 2020**	**Live Event**
SANS Munich September 2020	**Munich, Germany**	**Sep 14, 2020 - Sep 19, 2020**	**Live Event**
SANS Munich September 2020 - Live Online	**Munich, Germany**	**Sep 14, 2020 - Sep 19, 2020**	**CyberCon**
SANS Network Security 2020 - Live Online	**Las Vegas, NV**	**Sep 20, 2020 - Sep 25, 2020**	**CyberCon**
SANS Network Security 2020	**Las Vegas, NV**	**Sep 20, 2020 - Sep 25, 2020**	**Live Event**
SANS Australia Spring 2020	**, Australia**	**Sep 21, 2020 - Oct 03, 2020**	**Live Event**
SANS Australia Spring 2020 - Live Online	**, Australia**	**Sep 21, 2020 - Oct 03, 2020**	**CyberCon**
SANS Northern VA - Reston Fall 2020	**Reston, VA**	**Sep 28, 2020 - Oct 03, 2020**	**Live Event**
SANS San Antonio Fall 2020	**San Antonio, TX**	**Sep 28, 2020 - Oct 03, 2020**	**Live Event**
SANS Northern VA - Reston Fall 2020 - Live Online	**Reston, VA**	**Sep 28, 2020 - Oct 03, 2020**	**CyberCon**
SANS San Antonio Fall 2020 - Live Online	**San Antonio, TX**	**Sep 28, 2020 - Oct 03, 2020**	**CyberCon**
Oil & Gas Cybersecurity Summit & Training 2020	**Virtual - US Central,**	**Oct 02, 2020 - Oct 10, 2020**	**CyberCon**
SANS Tokyo Autumn 2020	**Tokyo, Japan**	**Oct 05, 2020 - Oct 17, 2020**	**CyberCon**
SANS Amsterdam October 2020 - Live Online	**Amsterdam, Netherlands**	**Oct 05, 2020 - Oct 10, 2020**	**CyberCon**
SANS Amsterdam October 2020	**Amsterdam, Netherlands**	**Oct 05, 2020 - Oct 10, 2020**	**Live Event**
SANS Prague October 2020 - Live Online	**Prague, Czech Republic**	**Oct 12, 2020 - Oct 17, 2020**	**CyberCon**
SANS Orlando 2020 - Live Online	**Orlando, FL**	**Oct 12, 2020 - Oct 17, 2020**	**CyberCon**