Port Swigger SQLi Bible

<https://portswigger.net/web-security/sql-injection/cheat-sheet>

Detection:

* The single quote character ' and look for errors or other anomalies.
* \*\*\*\*\* often databases will use a “ instead of ‘ (double quote instead of single) be sure to check both!
* -- is the SQL comment indicator so ‘-- is another common way to detect SQL. This can cause the rest of the query to be commented out and we may see a difference in the servers response. Use # for MySQL and test all types of comments for each database type.
* Some SQL-specific syntax that evaluates to the base (original) value of the entry point, and to a different value, and look for systematic differences in the application responses.
* Boolean conditions such as OR 1=1 and OR 1=2, and look for differences in the application's responses.
* Payloads designed to trigger time delays when executed within a SQL query, and look for differences in the time taken to respond. (blind SQLi)
* OAST payloads designed to trigger an out-of-band network interaction when executed within a SQL query, and monitor any resulting interactions.

Where SQLi occurs in an SQL query:

Most SQL injection vulnerabilities occur within the WHERE clause of a SELECT query. Most experienced testers are familiar with this type of SQL injection.

However, SQL injection vulnerabilities can occur at any location within the query, and within different query types. Some other common locations where SQL injection arises are:

* In UPDATE statements, within the updated values or the WHERE clause.
* In INSERT statements, within the inserted values.
* In SELECT statements, within the table or column name.
* In SELECT statements, within the ORDER BY clause.

What can be done when we find SQLi?

* [Retrieving hidden data](https://portswigger.net/web-security/sql-injection#retrieving-hidden-data), where you can modify a SQL query to return additional results.
* [Subverting application logic](https://portswigger.net/web-security/sql-injection#subverting-application-logic), where you can change a query to interfere with the application's logic.
* [UNION attacks](https://portswigger.net/web-security/sql-injection/union-attacks), where you can retrieve data from different database tables.
* [Blind SQL injection](https://portswigger.net/web-security/sql-injection/blind), where the results of a query you control are not returned in the application's responses.

Examples of SQLi and impacts:

SQLi can be used to find hidden data. SQLi often occurs in the url where you see something like parameter=something or in logins. This is an example of a possible SQLi location and a possible SQL query:

Imagine a shopping application that displays products in different categories. When the user clicks on the **Gifts** category, their browser requests the URL:

https://insecure-website.com/products?category=Gifts

This causes the application to make a SQL query to retrieve details of the relevant products from the database:

SELECT \* FROM products WHERE category = 'Gifts' AND released = 1

This SQL query asks the database to return:

* all details (\*)
* from the products table
* where the category is Gifts
* and released is 1.

The restriction released = 1 is being used to hide products that are not released. We could assume for unreleased products, released = 0. By using the ‘-- we could likely bypass this released = 1 by commenting it out and possibly see unreleased products. We could also try below to get a similar result:

https://insecure-website.com/products?category=Gifts'+OR+1=1--

This results in the SQL query:

SELECT \* FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

The modified query returns all items where either the category is Gifts, or 1 is equal to 1. As 1=1 is always true, the query returns all items. When injecting “OR 1=1” be careful as it can cause data loss if it reaches an UPDATE or DELETE statement.

SQLi can be used in some cases to login to other accounts for example when asked for user + pass we could do administrator’-- to comment out the password field and letting us login as admin.

There are multiple types of SQL databases that are commonly used: Microsoft, MySQL, Oracle, PostgreSQL. The syntax for each is slightly different so this can affect how to comment or query in general. Once we find SQLi we should try to determine the version of the database to make our attack vectors targeted to this specific type of database. Look at the “Database Version” section of the cheat sheet for attack vectors.

**UNION** attacks: Used to retrieve data from the SQL database by including another query (usually in conjunction with SELECT)

For example, if an application executes the following query containing the user input Gifts:

SELECT name, description FROM products WHERE category = 'Gifts'

An attacker can submit the input:

' UNION SELECT username, password FROM users--

This causes the application to return all usernames and passwords along with the names and descriptions of products.

SYNTAX:

The UNION keyword enables you to execute one or more additional SELECT queries and append the results to the original query. For example:

SELECT a, b FROM table1 UNION SELECT c, d FROM table2

This SQL query returns a single result set with two columns, containing values from columns a and b in table1 and columns c and d in table2.

REQUIREMENTS:

For a UNION query to work, two key requirements must be met:

* The individual queries must return the same number of columns.
* The data types in each column must be compatible between the individual queries.

To carry out a SQL injection UNION attack, make sure that your attack meets these two requirements. This normally involves finding out:

* How many columns are being returned from the original query.
* Which columns returned from the original query are of a suitable data type to hold the results from the injected query.

DETERMINING # OF COLUMNS:

When you perform a SQL injection UNION attack, there are two effective methods to determine how many columns are being returned from the original query.

One method involves injecting a series of ORDER BY clauses and incrementing the specified column index until an error occurs. For example, if the injection point is a quoted string within the WHERE clause of the original query, you would submit:

' ORDER BY 1--

' ORDER BY 2--

' ORDER BY 3--

etc.

This series of payloads modifies the original query to order the results by different columns in the result set. The column in an ORDER BY clause can be specified by its index, so you don't need to know the names of any columns. When the specified column index exceeds the number of actual columns in the result set, the database returns an error, such as:

The ORDER BY position number 3 is out of range of the number of items in the select list.

The application might actually return the database error in its HTTP response, but it may also issue a generic error response. In other cases, it may simply return no results at all. Either way, as long as you can detect some difference in the response, you can infer how many columns are being returned from the query.

The second method involves submitting a series of UNION SELECT payloads specifying a different number of null values:

' UNION SELECT NULL--

' UNION SELECT NULL,NULL--

' UNION SELECT NULL,NULL,NULL--

etc.

If the number of nulls does not match the number of columns, the database returns an error, such as:

All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists.

We use NULL as the values returned from the injected SELECT query because the data types in each column must be compatible between the original and the injected queries. NULL is convertible to every common data type, so it maximizes the chance that the payload will succeed when the column count is correct.

As with the ORDER BY technique, the application might actually return the database error in its HTTP response, but may return a generic error or simply return no results. When the number of nulls matches the number of columns, the database returns an additional row in the result set, containing null values in each column. The effect on the HTTP response depends on the application's code. If you are lucky, you will see some additional content within the response, such as an extra row on an HTML table. Otherwise, the null values might trigger a different error, such as a NullPointerException. In the worst case, the response might look the same as a response caused by an incorrect number of nulls. This would make this method ineffective.

**--------------------------------------------------------------------------------------------------------------------------------------------**

**Using UNION:**

In cases where the application responds with the results of a SQL query, an attacker can use a SQL injection vulnerability to retrieve data from other tables within the database. You can use the UNION keyword to execute an additional SELECT query and append the results to the original query.

For example, if an application executes the following query containing the user input Gifts:

SELECT name, description FROM products WHERE category = 'Gifts'

An attacker can submit the input:

' UNION SELECT username, password FROM users--

This causes the application to return all usernames and passwords along with the names and descriptions of products.

Basically used to execute another SQL query resulting in potential data retrieval from the database.

DATABASE SPECIFIC SYNTAX: when dealing with an Oracle database every SELECT keyword must be used with the FROM keyword to specify a valid table. There is a build in table called “dual” which can be used to identify the number of tables in an Oracle database. This looks like:

' UNION SELECT NULL FROM DUAL—

\*\*\* In a MySQL database the -- MUST be followed by a space \*\*\* alternatively a “#” also works to comment

FINDING COLUMNS OF USEFUL DATATYPES:

To find columns that hold strings we can inject specific queries to check this if the column datatype is not a string (or whatever datatype were testing for) it will result in an error if its not the same type. Note this must be done once we determine the number of columns in the database. All this is shown and described below:

After you determine the number of required columns, you can probe each column to test whether it can hold string data. You can submit a series of UNION SELECT payloads that place a string value into each column in turn. For example, if the query returns four columns, you would submit:

' UNION SELECT 'a',NULL,NULL,NULL--

' UNION SELECT NULL,'a',NULL,NULL--

' UNION SELECT NULL,NULL,'a',NULL--

' UNION SELECT NULL,NULL,NULL,'a'--

If the column data type is not compatible with string data, the injected query will cause a database error, such as:

Conversion failed when converting the varchar value 'a' to data type int.

If an error does not occur, and the application's response contains some additional content including the injected string value, then the relevant column is suitable for retrieving string data.

Once we identify a column with strings we can try injecting certain things where the ‘a’ is in hopes of retrieving that data from the column.

Example of making queries to retrieve data:

When you have determined the number of columns returned by the original query and found which columns can hold string data, you are in a position to retrieve interesting data.

Suppose that:

* The original query returns two columns, both of which can hold string data.
* The injection point is a quoted string within the WHERE clause.
* The database contains a table called users with the columns username and password.

In this example, you can retrieve the contents of the users table by submitting the input:

' UNION SELECT username, password FROM users--

In order to perform this attack, you need to know that there is a table called users with two columns called username and password. Without this information, you would have to guess the names of the tables and columns. All modern databases provide ways to examine the database structure, and determine what tables and columns they contain.

Retrieving multiple values within a single column:

In some cases the query in the previous example may only return a single column.

You can retrieve multiple values together within this single column by concatenating the values together. You can include a separator to let you distinguish the combined values. For example, on Oracle you could submit the input:

' UNION SELECT username || '~' || password FROM users--

This uses the double-pipe sequence || which is a string concatenation operator on Oracle. The injected query concatenates together the values of the username and password fields, separated by the ~ character.

The results from the query contain all the usernames and passwords, for example:

...

administrator~s3cure

wiener~peter

carlos~montoya

...

Different databases use different syntax to perform string concatenation. For more details, see the [SQL injection cheat sheet](https://portswigger.net/web-security/sql-injection/cheat-sheet).

**\*\*\*\*** When we determine the number of columns and then which column is a string (or whatever our desired datatype is) we can inject our query right into the column of compatible datatype while leaving the other null in the query as shown below: in this scenario we knew the column names and the name of the users table allowing us to make this query.

' UNION SELECT NULL,username || '~' || password FROM users—

**EXAMINING THE DATABASE IN SQL INJECTION ATTACKS**

Two main things that will help us get the most out of and SQLi attack are: 1. The type and version of database software and 2. The tables and columns that the database contains.

QUERYING DATABASE TYPE AND VERSION:

Use the “Database Version” section of SQLi Cheat Sheet for useful payloads for detecting type and version. We can also combine these with a ‘ UNION attack vector to get this valuable info.

\*\* with oracle try also using SELECT \* FROM v$version

\*\* ALLLL DATABASE INQURIES MUST USE FROM dual (or a valid table) in ORACLE DATABASES

So, to test for column number in oracle we must do ‘ UNION SELECT NULL,NULL,… FROM dual

\*\*\* also we may sometimes need to put + in between all words as shown below: no + between columns

'+UNION+SELECT+'abc','def'+FROM+dual—

Again with finding out version info we first need column (if ‘ UNION – doesn’t work try adding + in between) then once we have it we need to use the form NULL,version.. as shown below. Note that this version part if the query will not need quotes around it even if the column holds strings. 'UNION+SELECT+NULL,banner+FROM+v$version—

\*\* TEST ALL COMMENT METHODS FOR EAHC DATABASE \*\*

LISTING CONTENTS OF A DATABASE:

Most database types (except Oracle) have a set of views called the information schema. This provides information about the database.

For example, you can query information\_schema.tables to list the tables in the database:

SELECT \* FROM information\_schema.tables

This returns output like the following:

TABLE\_CATALOG TABLE\_SCHEMA TABLE\_NAME TABLE\_TYPE

=====================================================

MyDatabase dbo Products BASE TABLE

MyDatabase dbo Users BASE TABLE

MyDatabase dbo Feedback BASE TABLE

This output indicates that there are three tables, called Products, Users, and Feedback.

You can then query information\_schema.columns to list the columns in individual tables:

SELECT \* FROM information\_schema.columns WHERE table\_name = 'Users'

This returns output like the following:

TABLE\_CATALOG TABLE\_SCHEMA TABLE\_NAME COLUMN\_NAME DATA\_TYPE

=================================================================

MyDatabase dbo Users UserId int

MyDatabase dbo Users Username varchar

MyDatabase dbo Users Password varchar

This output shows the columns in the specified table and the data type of each column.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'+UNION+SELECT+table\_name,+NULL+FROM+information\_schema.tables-- we can use table\_name as a sort of default table to display the info

ID,INFO,EXPLOIT - STEPS:

1, identify SQLi by seeing different reponses from things like ‘-- and ‘1=1,’1=2 etc also try all possible comment types and closely observe the server response

2. Identify number of columns using ‘ UNION SELECT NULL,NULL,…-- and looking for anomalies/ differences in the server responses.

3. Find the data types of columns by replacing ‘a’ for NULL

4. Find version of database

5. research database type and craft injections for specific databases look up the official website for each database and use it to craft your exploits the cheat sheet can also help here. \*\* look up information\_schema.tables “postgres” to see a list of all available table names for example in postgres we use table\_name as shown below to list the contents. This will vary for other SQLs.

6. Note for postgres we can use the following to pull all table names: (of course well need to be using the correct amount of “NULL” but notice the table\_name which is specific to this database. '+UNION+SELECT+table\_name,+NULL+FROM+information\_schema.tables--

7. Once we list all tables we can begin to query individual columns using the database specific build in names to list all columns for a specific table using a comment similar to the one shown below: \*\*\* (ALWAYS TRY SWITCHING WHERE YOU HAVE THE NULL)

'+UNION+SELECT+column\_name,+NULL+FROM+information\_schema.columns+WHERE+table\_name='users\_abcdef'--

Now we are querying information\_schema.columns and specifying our own table that we found in above steps. Of course this will vary for each database we encounter things like column\_name and table\_name= will likely be different. Then we fill in the table were curious about where the table\_name=”interesting\_table\_name\_here”

8. Finally when we want to pull specific data from an interesting column we can then do this:

'+UNION+SELECT+username\_abcdef,+password\_abcdef+FROM+users\_abcdef--

Where replace the FROM+interstingColumnName and the username,password with whatever data we want to pull out. (BASICALLY SAYING: column1,column2 FROM table

ORACLE how to:

First of course detect SQLi then move to finding database version: (Cheat Sheet has other oracle version payloads that may work for oracle this is not end-all be all.

'+UNION+SELECT+BANNER,+NULL+FROM+v$version--

Must use FROM dual when finding column size as shown:

'+UNION+SELECT+'abc','def'+FROM+dual--

Retrieving all table names: \*\*note we use all\_tables here

'+UNION+SELECT+table\_name,NULL+FROM+all\_tables--

Retrieve all column names in a desired table: or course add your desired ‘tableName’

'+UNION+SELECT+column\_name,NULL+FROM+all\_tab\_columns+WHERE+table\_name='tableName --

After dropping all columns for a table we know what to display the specific column names:

'+UNION+SELECT+column1,+column@+FROM+tableName--

Of course switch with desired column & table names!