

SEED LABS

SQL INJECTION 2.0



Contents

[Environment Setup 2](#_Toc151834158)

[Task 1 8](#_Toc151834159)

[Task 2 10](#_Toc151834160)

[Task 2.1 11](#_Toc151834161)

[Task 2.2 12](#_Toc151834162)

[Task 2.3 14](#_Toc151834163)

[Task 3 17](#_Toc151834164)

[Task 3.1 21](#_Toc151834165)

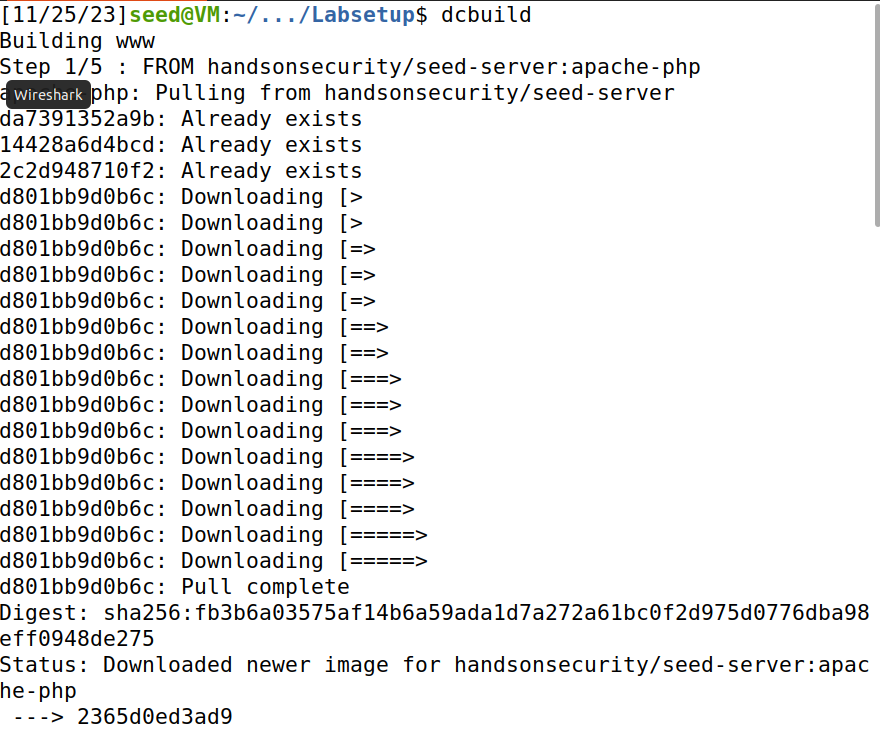
[Task 3.2 24](#_Toc151834166)

[Task 3.3 25](#_Toc151834167)

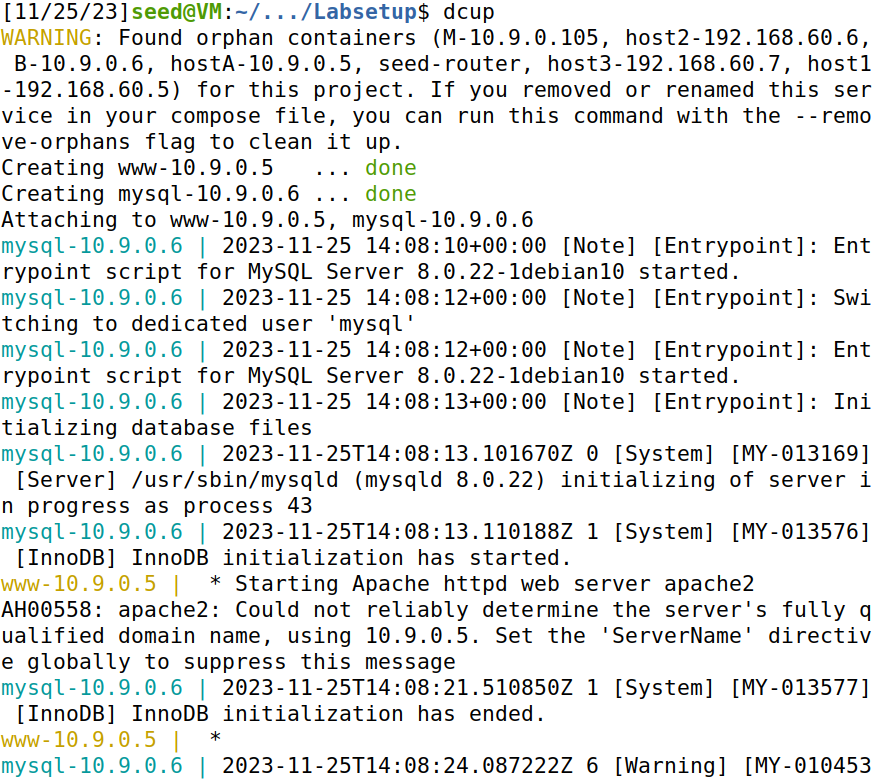
[Task 4 28](#_Toc151834168)

# Environment Setup

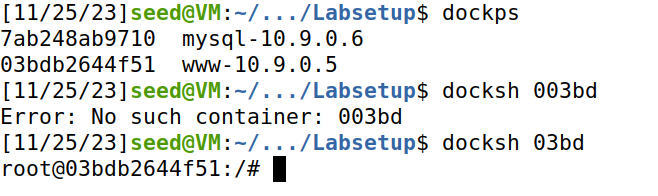
Building Dockers.



Setting up the Containers.



Setting up the Docker for URL.



Setting up MySQL Database container.

A close-up of a number

Description automatically generated

Now I copied this link as it will be required for URL of the web application.

A screenshot of a computer error

Description automatically generated

Adding the entry with the above URL in SQL Injection Lab section.

A screenshot of a computer

Description automatically generated

Now the URL is live.

A screenshot of a computer

Description automatically generated

Using the provided data.

A table with numbers and numbers

Description automatically generated

Logging in as Admin to test if the provided data works.

A screenshot of a user details

Description automatically generated

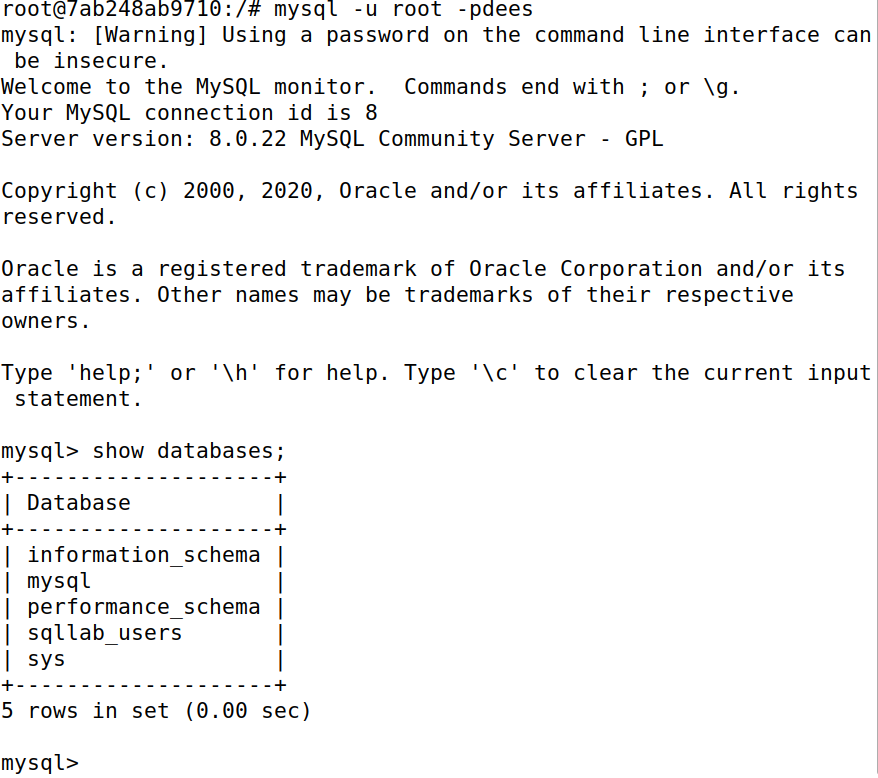
Now logging in as Alice.

A screenshot of a computer

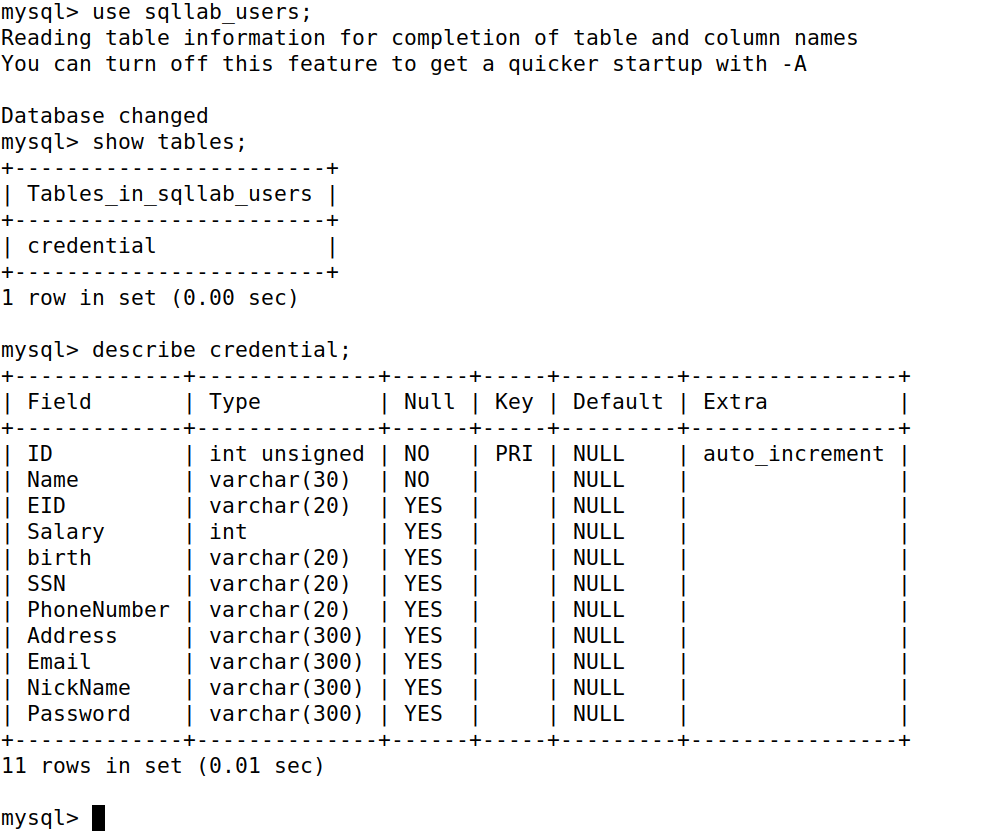
Description automatically generated

# Task 1

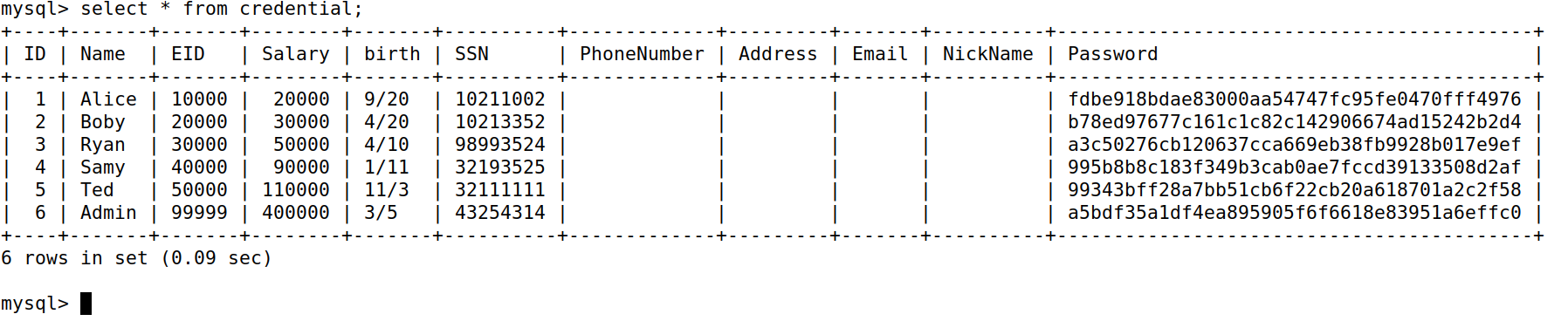
Checking the databases in MySQL Database where the target is “sqllab\_users” as shown in the database and manual.



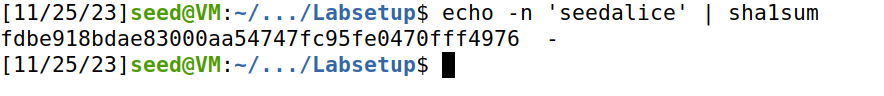
Using the target database and checking the schema of credentials table.



Now showing all the records with credentials of the Users where the passwords are in hashes.

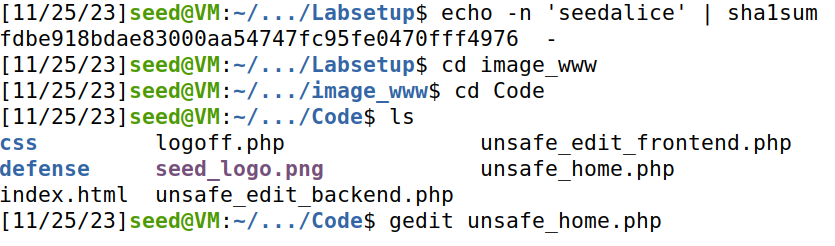


Just for verification I opened a new terminal and checked the sha1sum of the password to see if the provided password in the manual matches the one in database which it does.

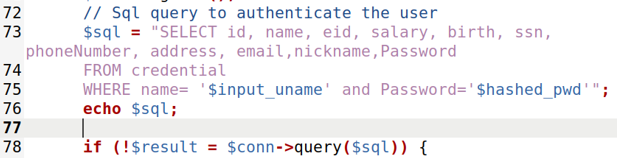


# Task 2

As mentioned in the manual I am checking the **unsafe\_home.php** file.



Now while looking for vulnerability, I found this line which asks for parameters so I will simply add the code on line 76.

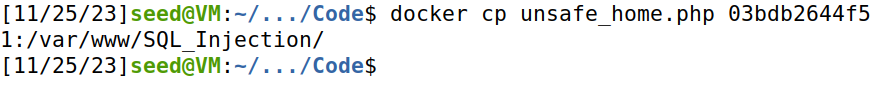


Going to the directory from the Web container to where the above file is present.

A close up of a computer code

Description automatically generated

Copying the file in the directory mentioned above.



## Task 2.1

Now trying the Attack.

A screenshot of a login screen

Description automatically generated

Now while trying to login as Admin, I receive the SQL statement meaning that the input in the User section didn’t comment out the remaining section.

A screenshot of a computer error message

Description automatically generated

Now with the mentioned statement I got I changed the query and tried again.

A screenshot of a login screen

Description automatically generated

And I the attack is successful which means the query **Admin’ #** caused to comment the password section in the statement and made possible the attack.

A screenshot of a user details

Description automatically generated

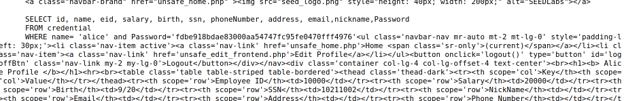
## Task 2.2

I tried the curl command **curl ’www.seed-server.com/unsafe\_home.php?username=alice&Password=11’**  in the manual to get the following information but the password here is hashed. Which actually is a GET request.

A screenshot of a computer code

Description automatically generated

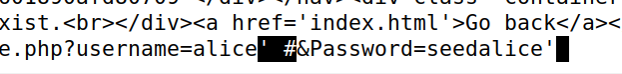
Now if I try with the provided password of Alice in the manual while using the command **curl ’www.seed-server.com/unsafe\_home.php?username=alice&Password=seedalice’**. It is noticeable that the above command didn’t provide the write hashed password.



Now while trying to login without password with command **curl ’www.seed-server.com/unsafe\_home.php?username=alice’ #&Password=seedalice’**. Which failed.



Now to make it work I encoded the displayed part below.



I encoded the highlighted part on the site <https://www.urlencoder.org/>.

A screenshot of a computer

Description automatically generated

Now trying the modified command **’www.seed-server.com/unsafe\_home.php?username=alice%27%20%23&Password=seedalice’** which totally worked and caused the login.

A screenshot of a computer screen

Description automatically generated

## Task 2.3

Now in the SQL Database it is possible to select 2 rows with appended SQL commands.

A screenshot of a computer program

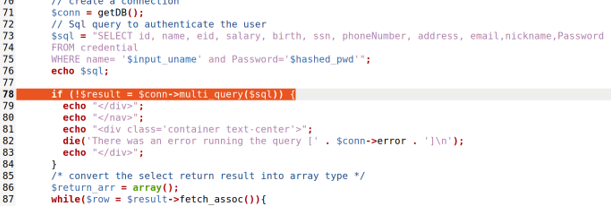
Description automatically generated

Now in the code here in line 78 there is one query allowed only.

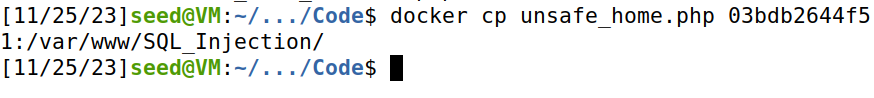
A screenshot of a computer code

Description automatically generated

With this modification I can get around the countermeasure easily.



Now copying to the Web docker the code file.



Now sending the appended SQL statement.

A screenshot of a login screen

Description automatically generated

The attack worked and the reason is mentioned with the screenshot ahead.

A screenshot of a computer

Description automatically generated

As a note that why isn’t anything displayed when I sent appended query is because the select 1 worked as the array where the data here was on the index 0 but given that with select 1 it treated as index 1 there is no data present there to show. Hence, the attack worked.

A screen shot of a computer program

Description automatically generated

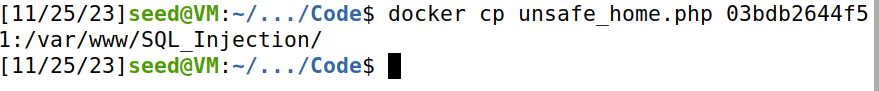
# Task 3

For the ahead tasks I removed the modifications in the home page code.

A computer screen shot of a computer code

Description automatically generated

Copied the file to the Web container.

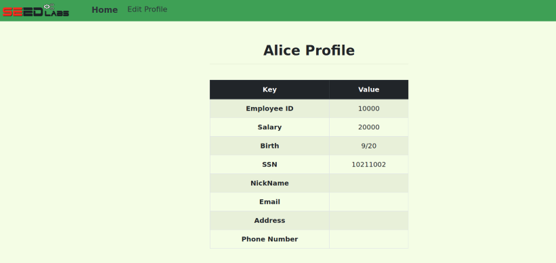


Logging in as Alice.

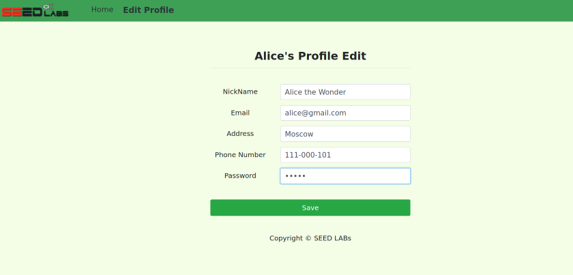
A screenshot of a login screen

Description automatically generated

Now I am logged in as Alice.



Updating the Profile of Alice.



The changes are visible on the profile.

A screenshot of a computer

Description automatically generated

It can also be observed that the database has also been updated.

A close-up of a list

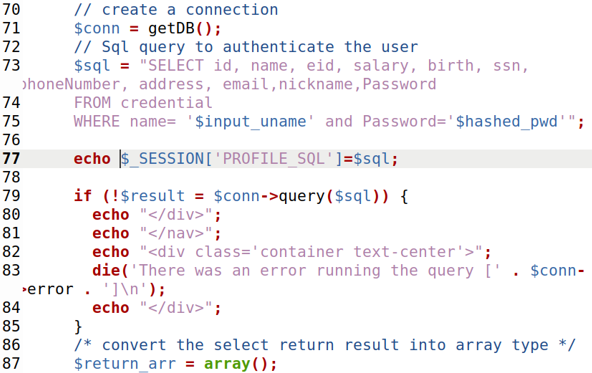
Description automatically generated

## Task 3.1

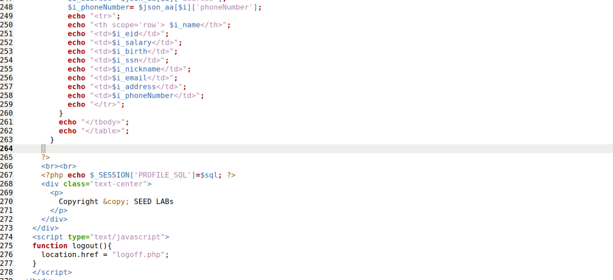
Now I modified the **unsafe\_edit\_backend.php** code file at line 56 and 57.



And echoing it in the **unsafe\_home.php** file.



Another modification in the file **unsafe\_home.php** on line 267.



Finally copying the modified files to the Web continer.

A close-up of words

Description automatically generated

Based on these code lines I have found the vulnerability where I can add salary as a parameter as a part of SQL statement which is a part of the database used here.

A close-up of a computer screen

Description automatically generated

Sending the query.

A screenshot of a computer

Description automatically generated

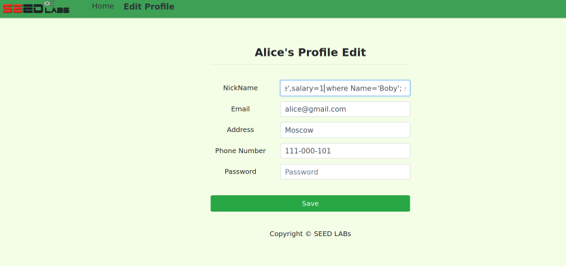
And the salary has been changed from 10,000 to 30,000.

A screenshot of a computer

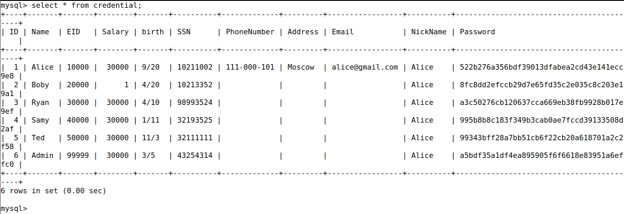
Description automatically generated

## Task 3.2

Now trying to change Boby’s Salary with the SQL statement **Alice',salary=1 where Name='Boby'; #**



Now in the SQL Database container I have compared the Salary and it has been changed to 1 from 30,000.



## Task 3.3

Now changing Boby’s password from Alice’s profile with the help of SQL Statement **Alice',password=sha1('boby') where Name='Boby'; #**

A screenshot of a computer

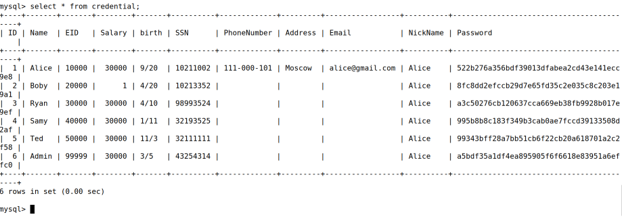
Description automatically generated

The previous screenshot of the Database is like this.

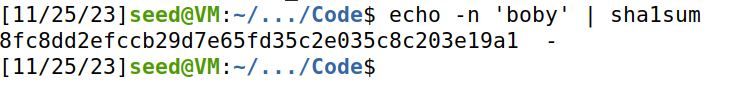
A close-up of a number

Description automatically generated

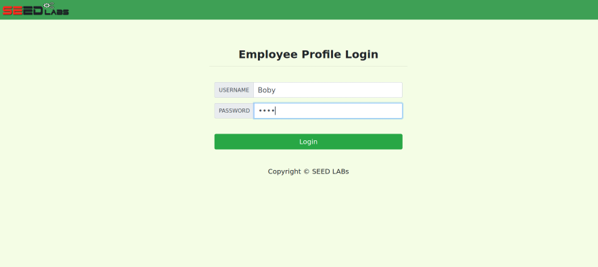
It can be confirmed that the password has been changed by the SQL Database and in the screenshot below it is proven by how the hash of the password is different than what it was before.



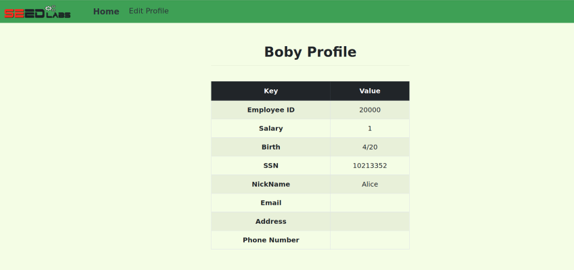
It can also be confirmed with sha1sum which is a match to the change observed in the database.



Further confirmation is to login with the changed password set as “boby”.

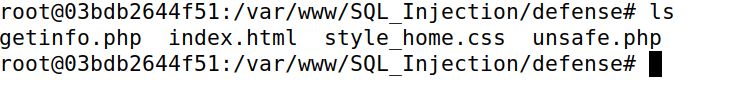


Hence, the task has been completed successfully as the login is successful. Moreover, the salary update performed in previous subtask is also visible.

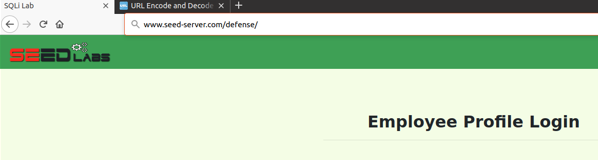


# Task 4

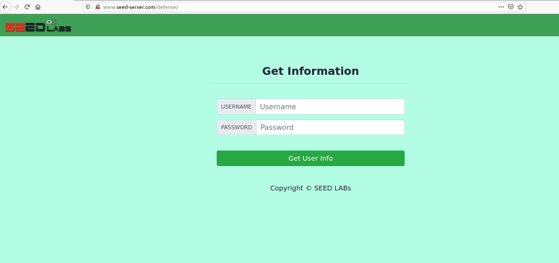
These are the files for the defense of the site as in the web docker.



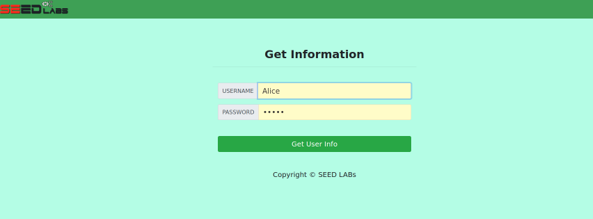
Now visiting the following URL for the task.



And this is how it looks like.



Trying to get user info of Alice.



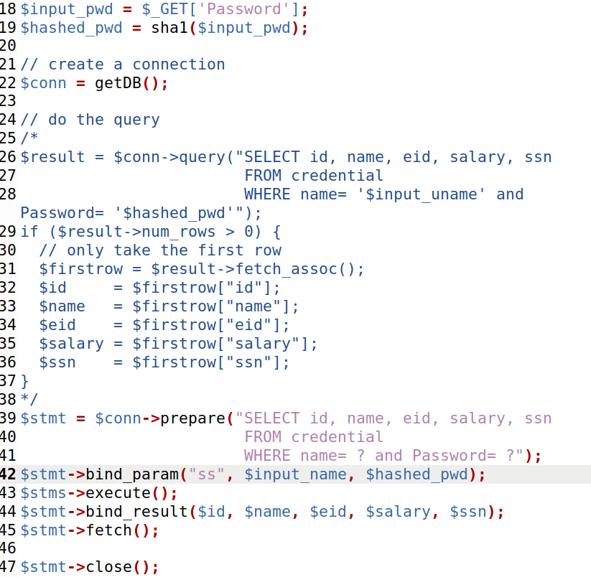
Which is as following as the site accepted Alice’s modified credentials.



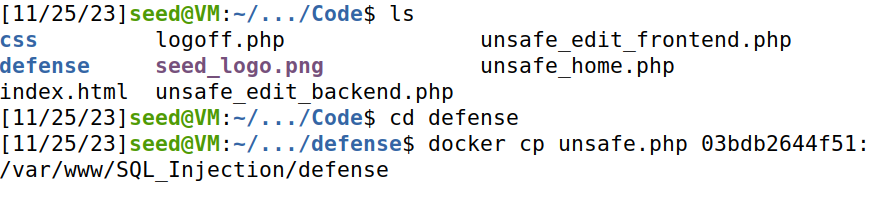
A vulnerability has been spotted in the code of defense site in the **unsafe.php** file. This vulnerability even allows as simple an SQL statement as **Alice’ #**



To make prepared statements I have commented the already present vulnerability from line 25 to 38. Then continuing from line 39 I wrote the code where username and password are not just accepted as any input placed in the parameter. Then I bound the parameter with username input and hashed password continuing to the statement where only these parameters will be executed. When executed upon success the user id, name, eid, salary and ssn parameters will be fetched from the database. Finally closing it to prevent anymore vulnerabilities as to include other statements.



Now copying the **unsafe.php** file to the Web container in the defense folder.



Now trying the attack.

A screenshot of a computer screen

Description automatically generated

And the attack didn’t work as nothing appeared which is more like no result and closed.

A screenshot of a computer

Description automatically generated