CS 5340 INTRODUCTION TO INFORMATION AND COMPUTER SECURITY

HANDS-ON PROJECT LAB-2 ON WEB ATTACKS

FALL 2022

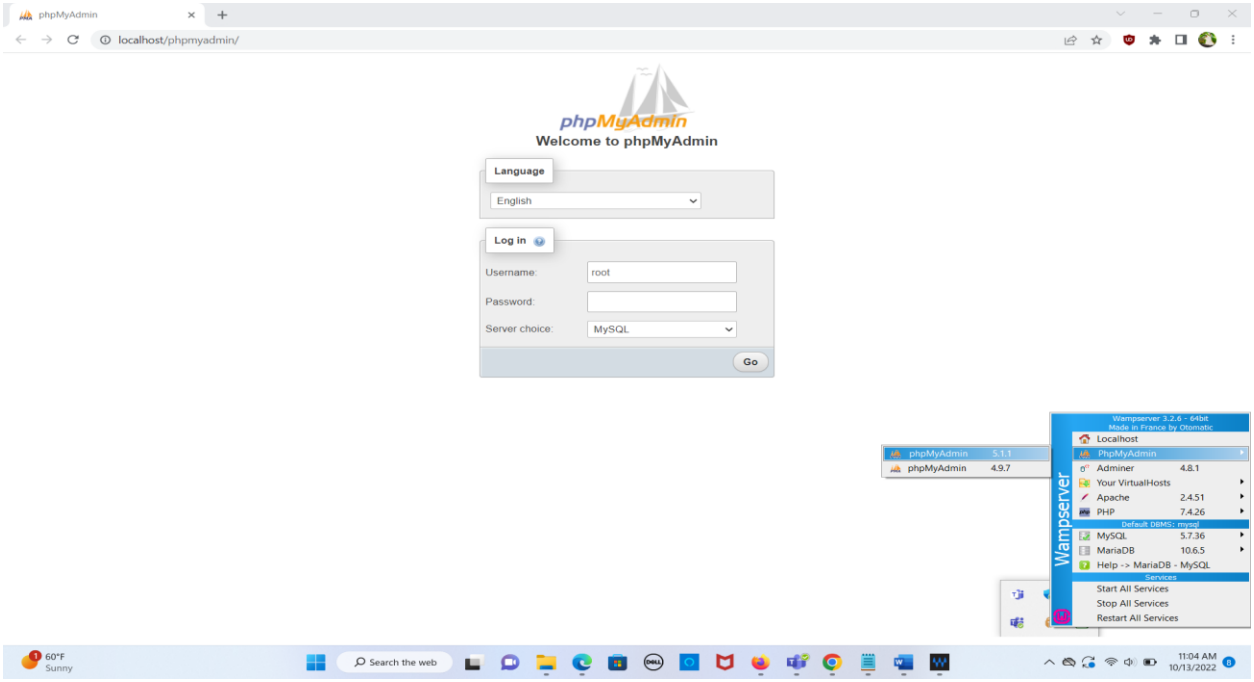
Name : SubhashChandra Bose Goud Muchchunoor

R# : R11850821

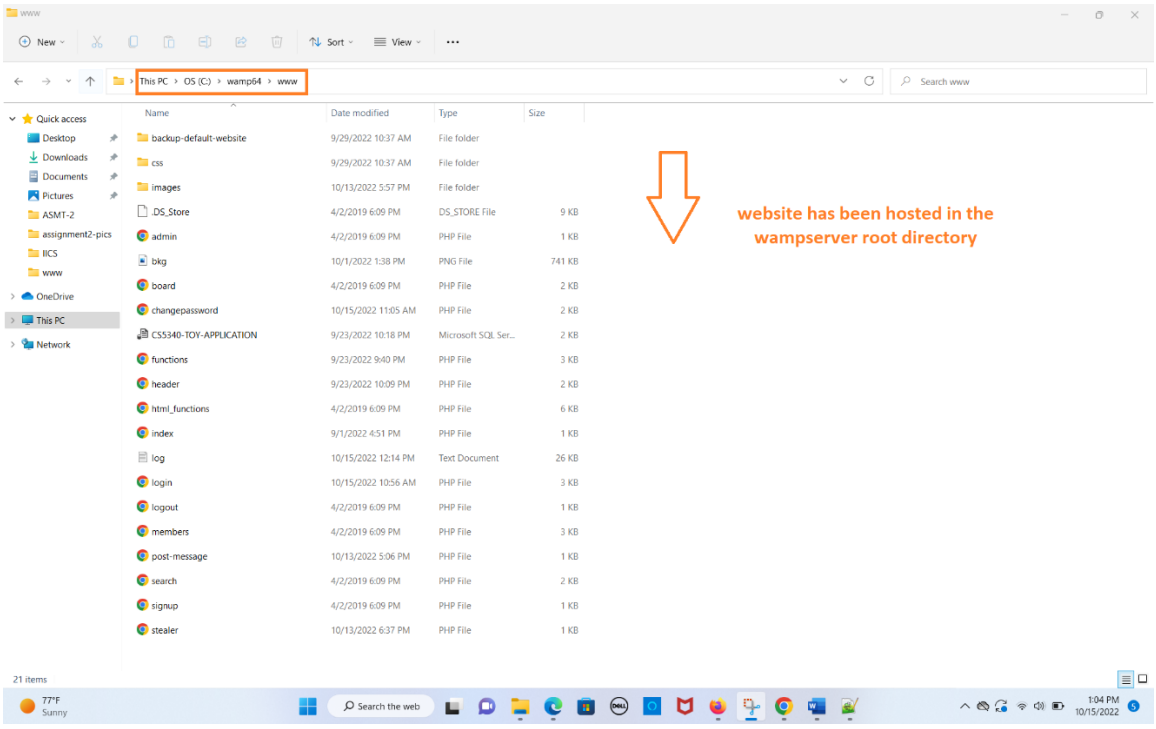
SECTION 1 – XSS ATTACKS

1. Lab Setup:

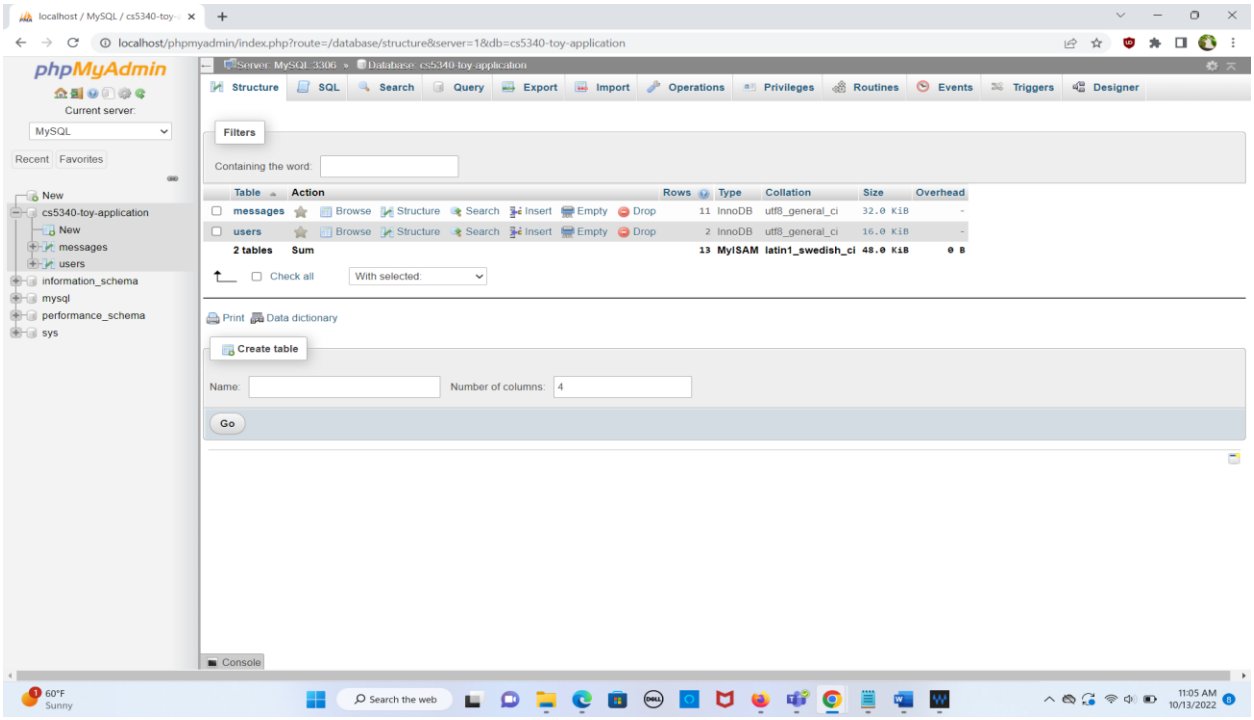
Wamp server with MYSQL is installed in my local system



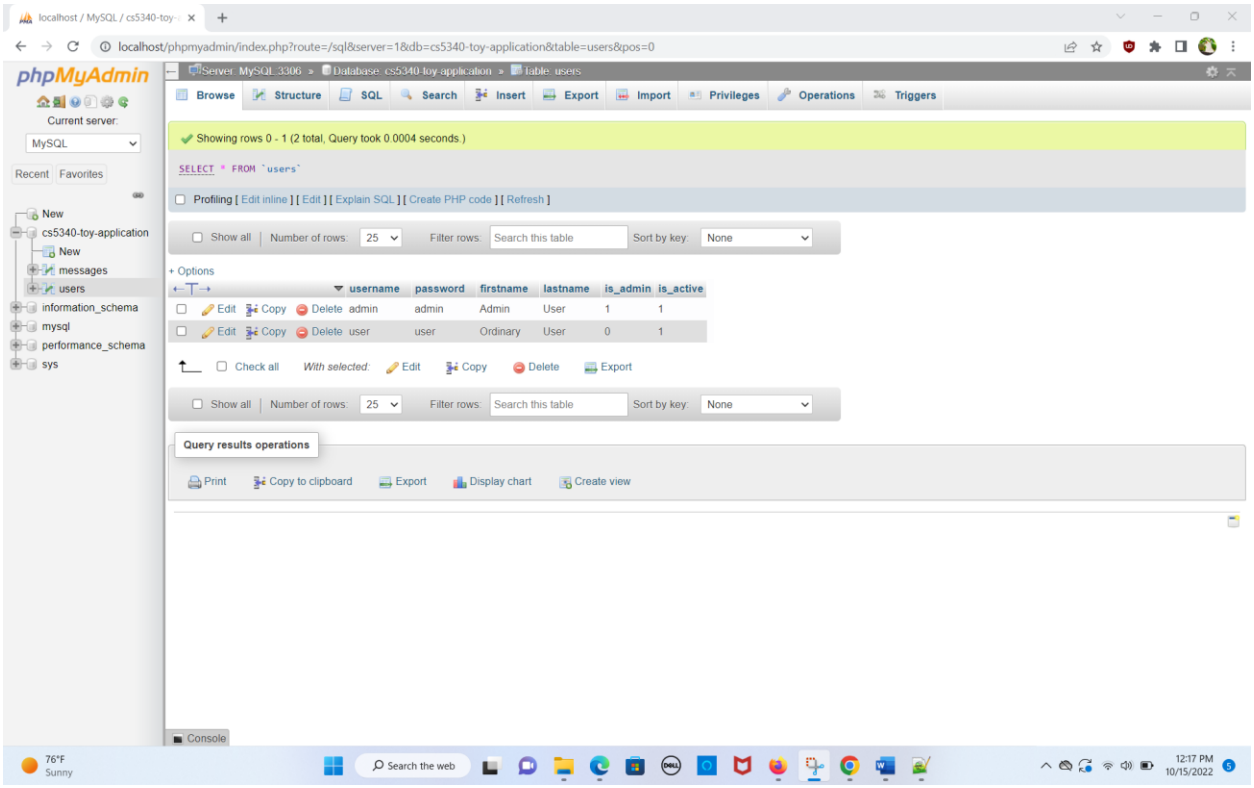
CS5340-toy-application code is downloaded and extracted. Website is hosted in the WAMP server root directory as in below screenshot.



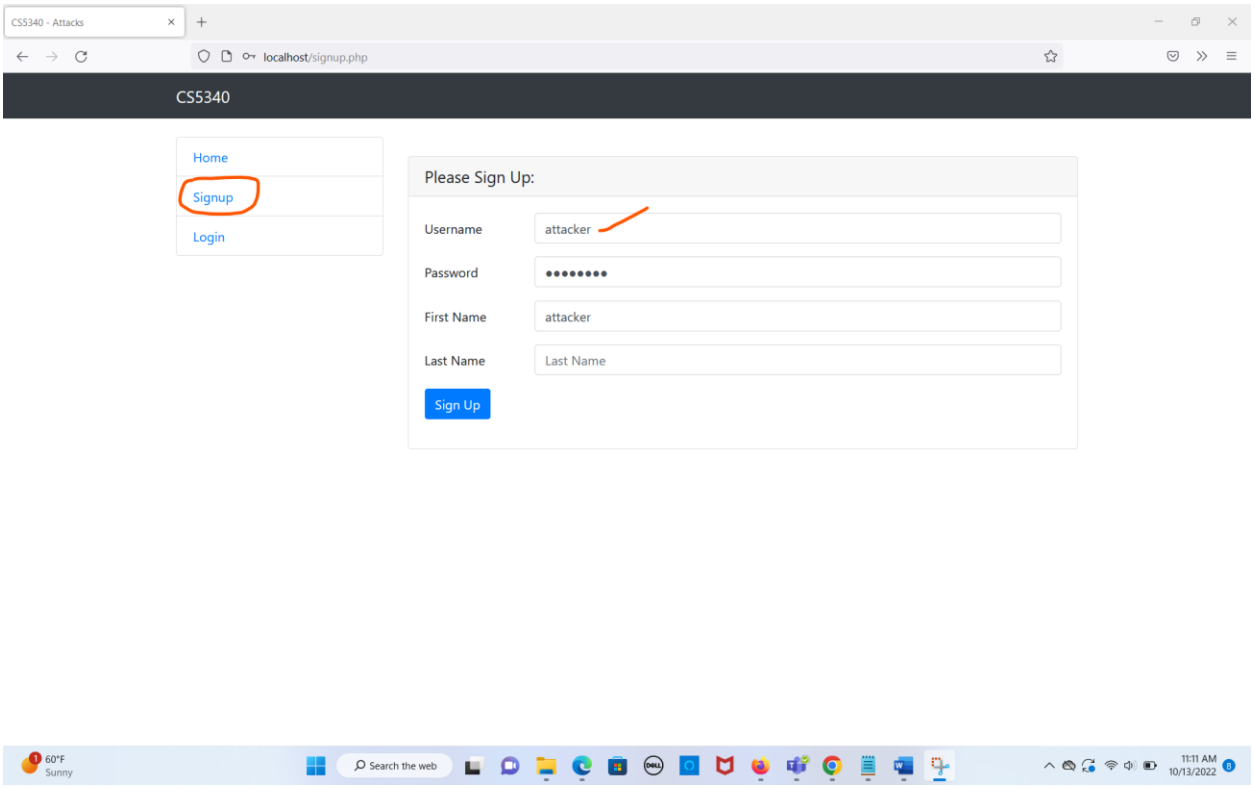
The database file is imported into MYSQL DB sever. The below picture illustrates table view of database

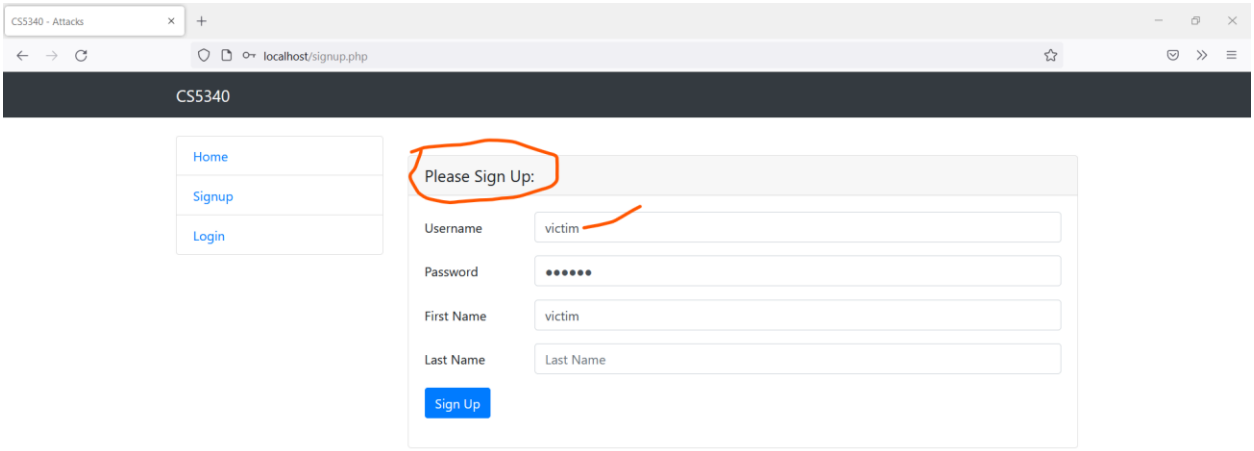


The below picture displays the user login details in the users table

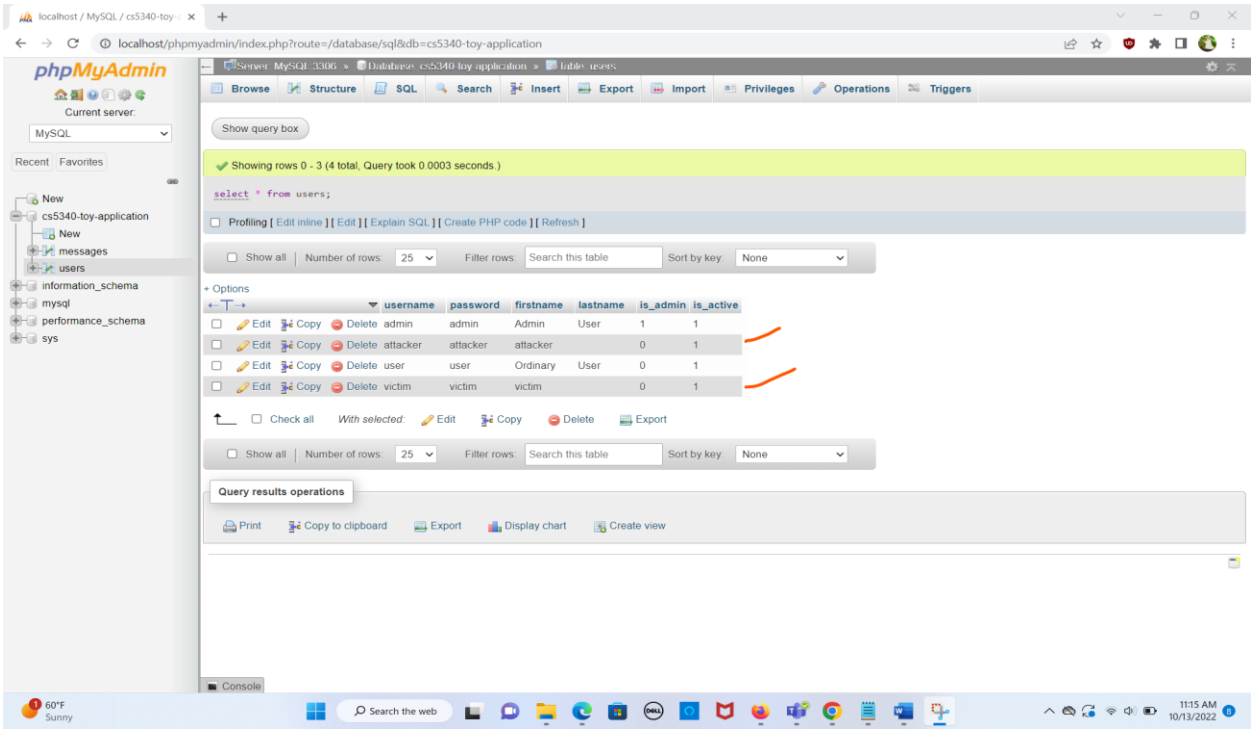


Created 2 users from the website signup option. They are 1- attacker, 2- victim





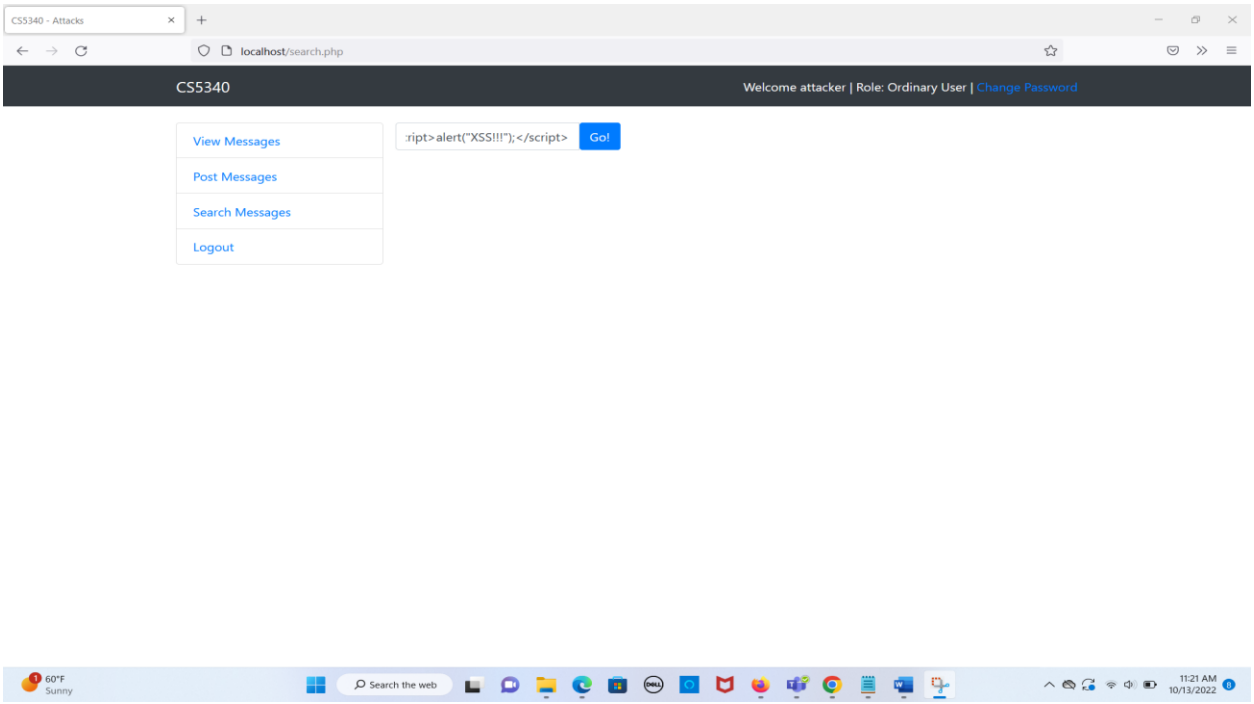
Both the users are not admin's they are ordinary users

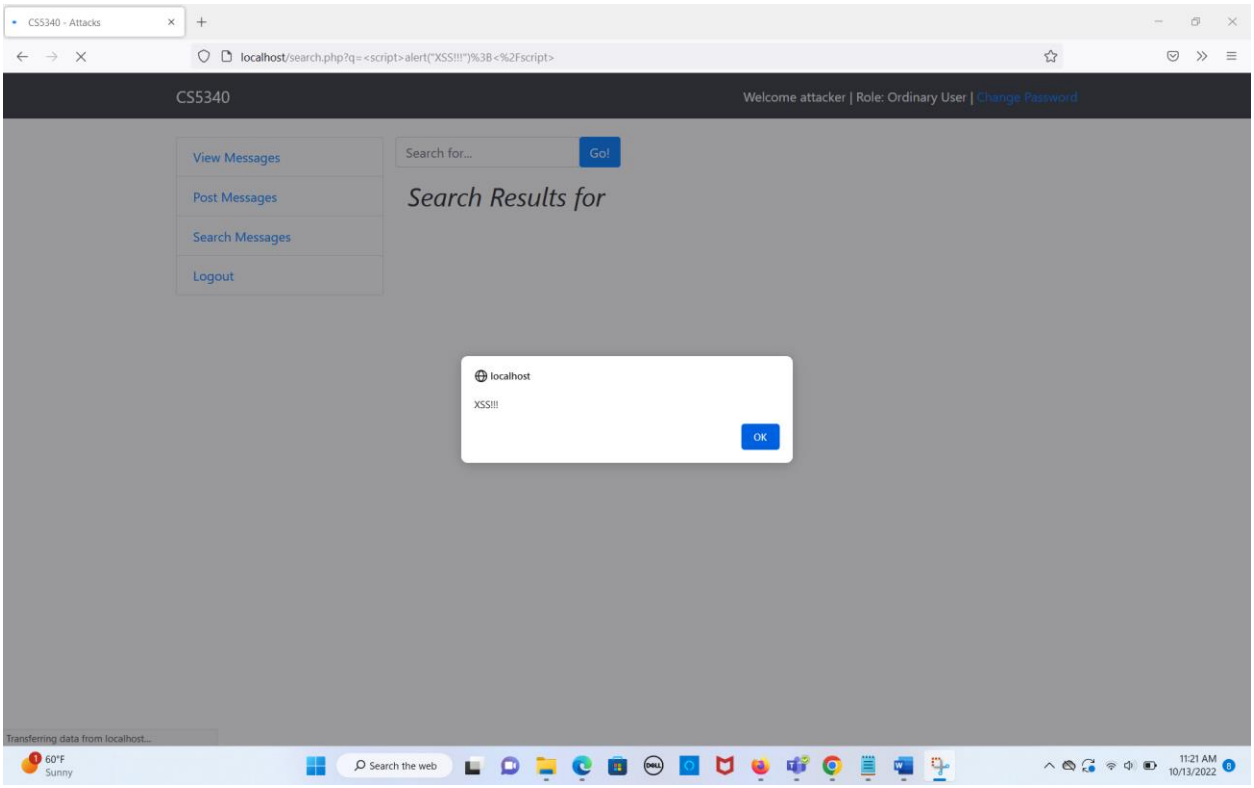


2. Non-Persistent XSS:

Task 1: XSS Vulnerability Check

<script>alert("XSS!!!");</script> is performed using XSS attack



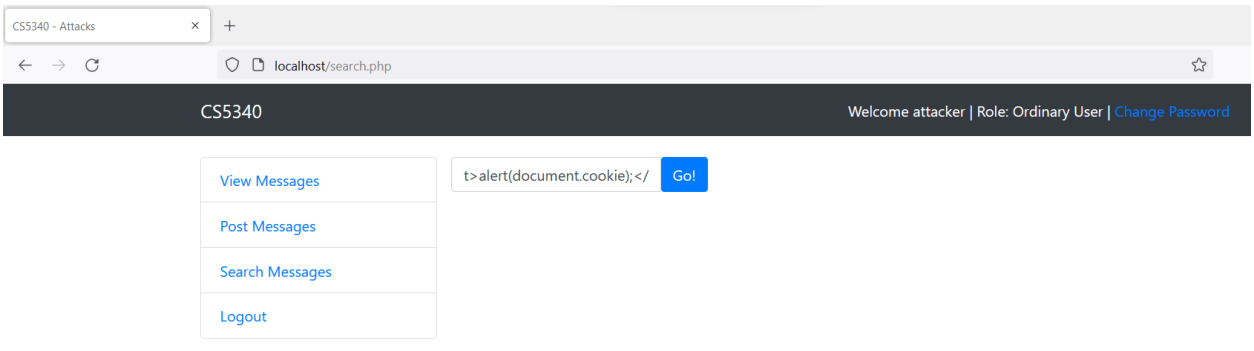


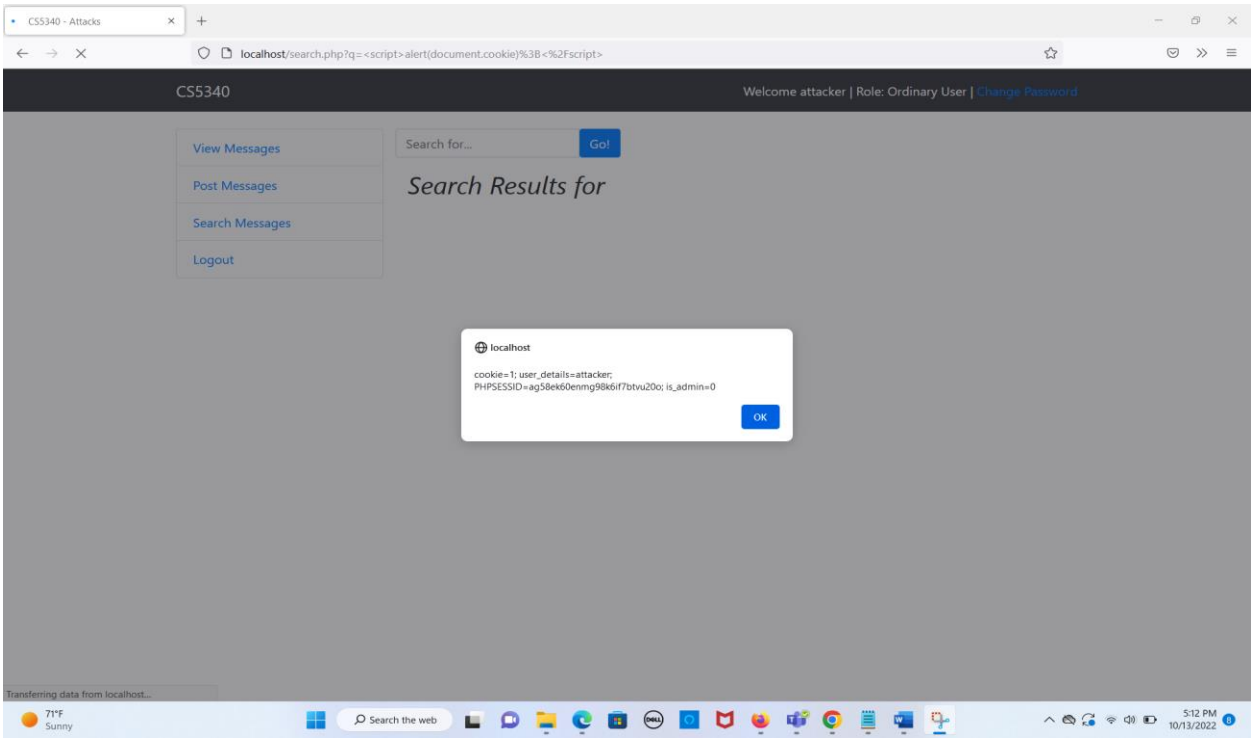
The alert script was successfully executed which means application is allowing malicious content.

What do you observe? Explain the reason behind this observation.

The Script we passed as a message got executed as a command. Here we exploited the XSS vulnerability of the application.

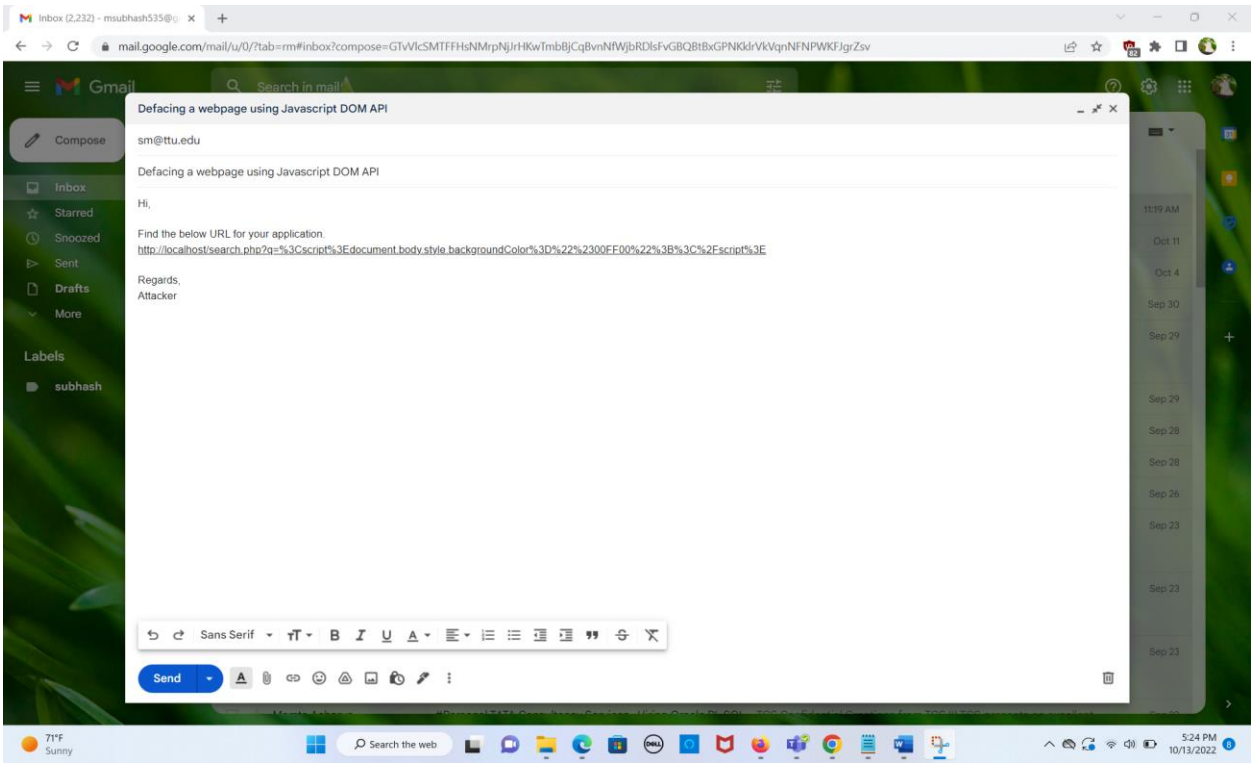
Modify the statement used in (b) above to display the cookie in the alert box.



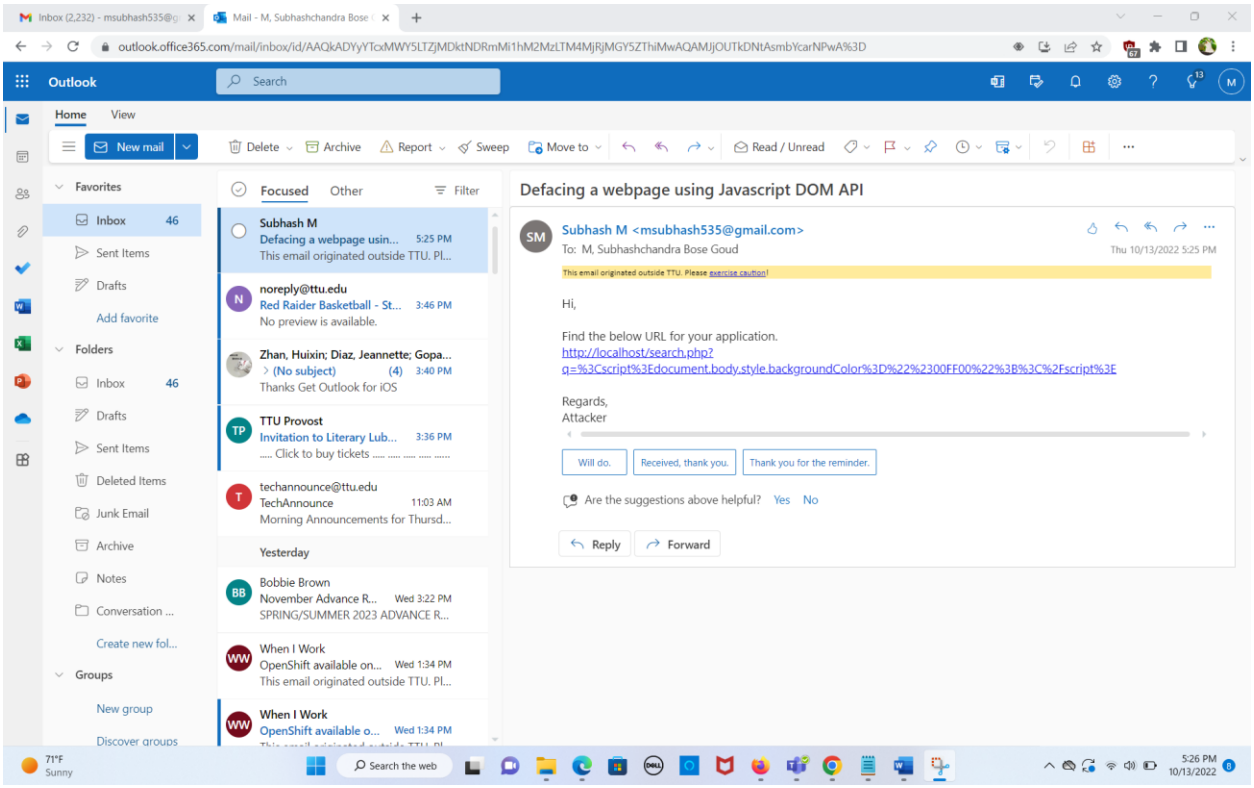


Task 2: Defacing a webpage using Javascript DOM API

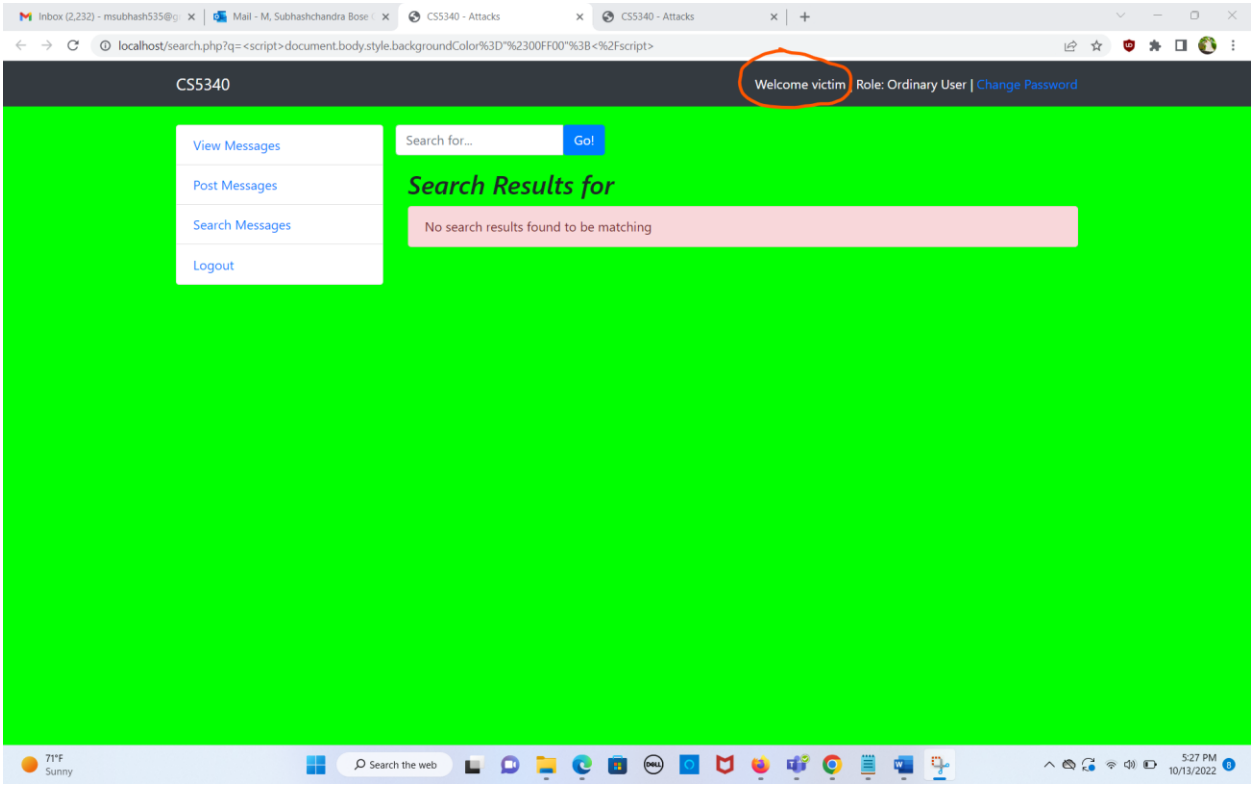
A mail is sent to victim from attacker with a malicious link attached in it



Victim received the mail sent from attacker

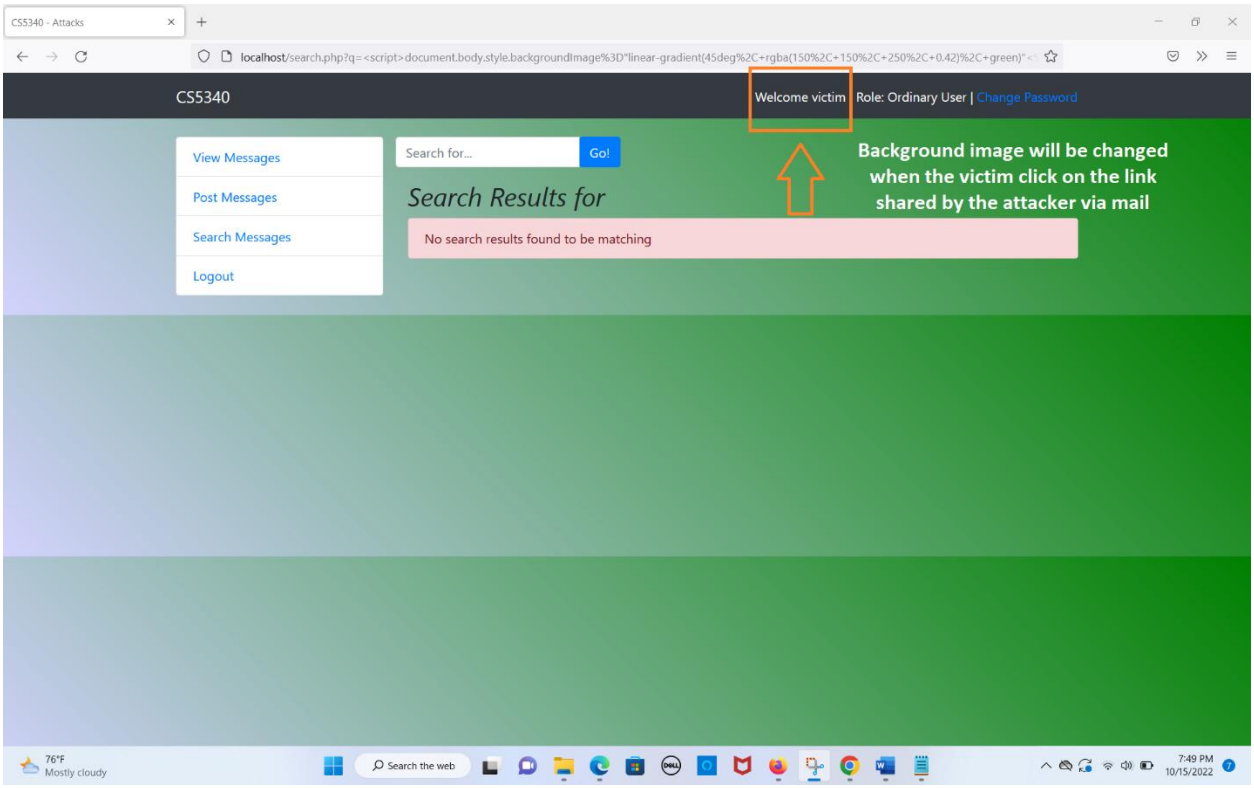


Victim clicks on the link sent by the attacker



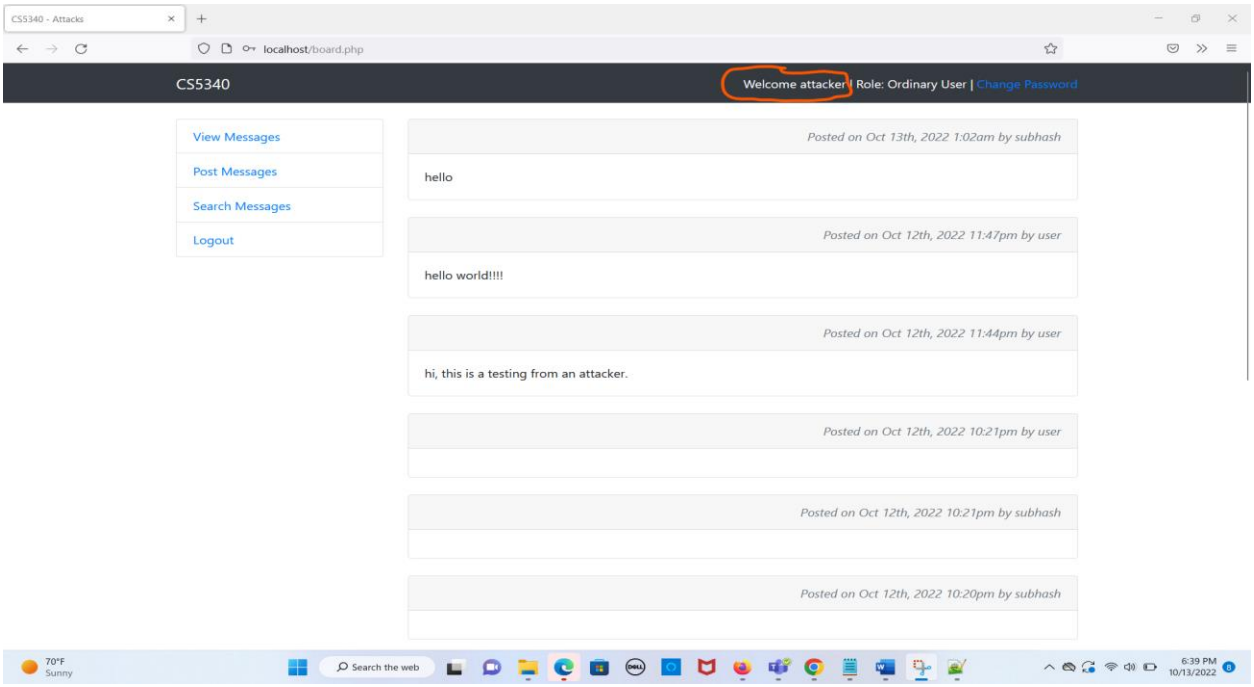
Similarly, background image can be changed as below-

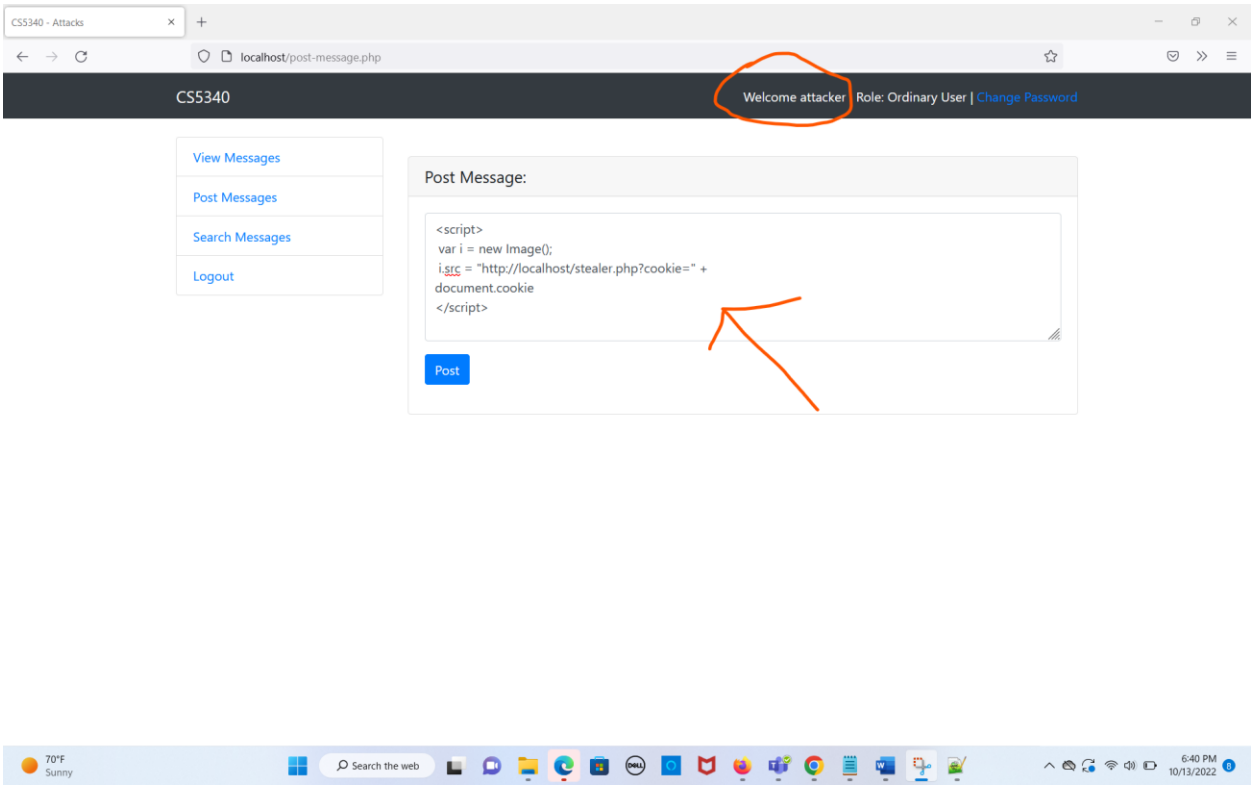
http://localhost/search.php?q=%3Cscript%3Edocument.body.style.backgroundImage%3D%22linear-gradient%2845deg%2C+rgba%28150%2C+150%2C+250%2C+0.42%29%2C+green%29%22%3C%2Fscript%3E

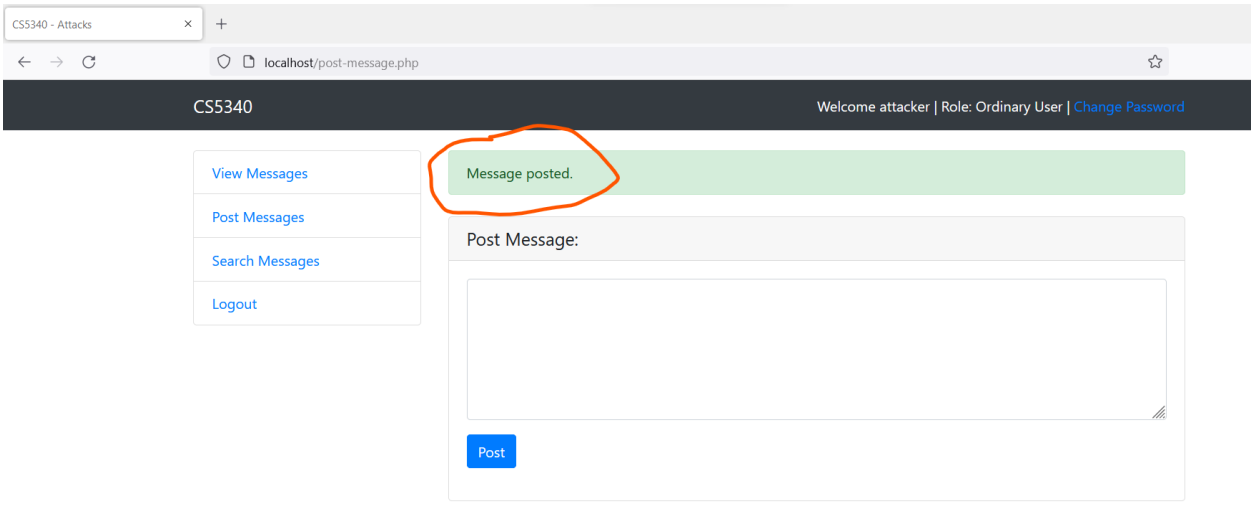


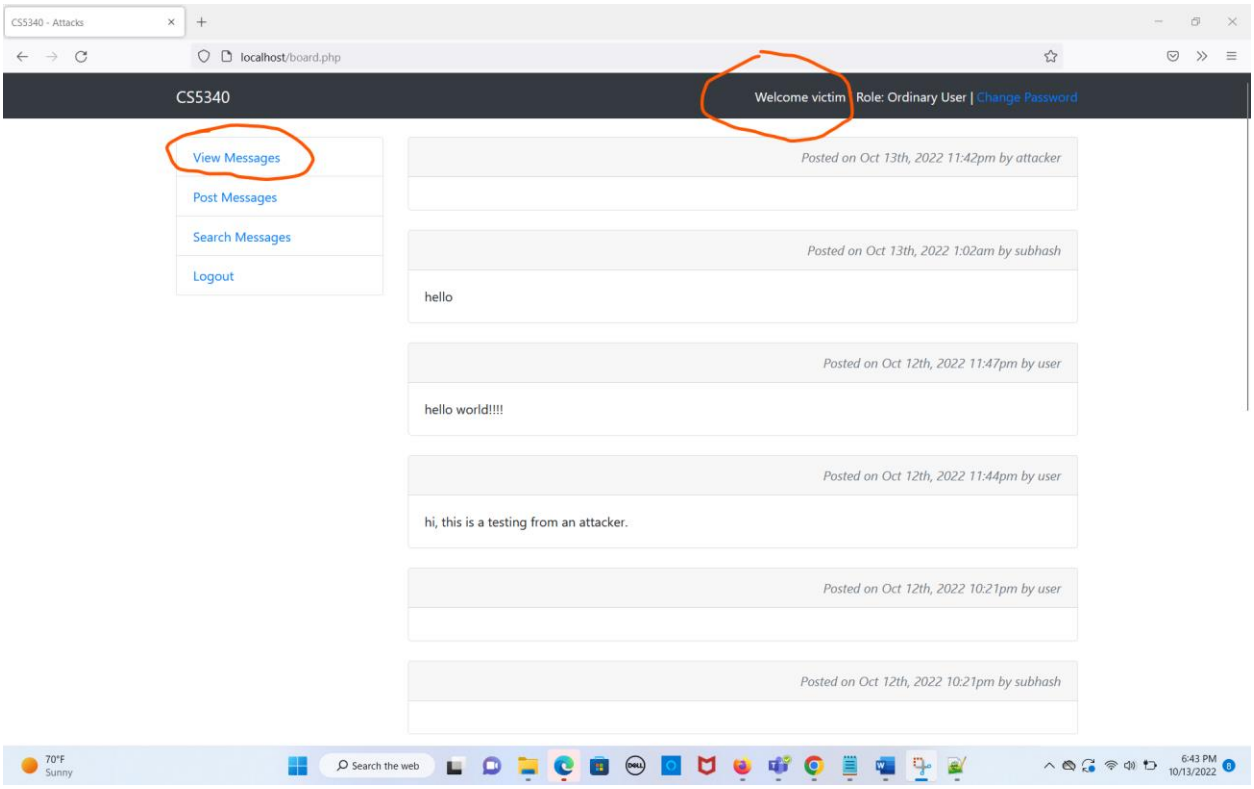
3. Persistent XSS:

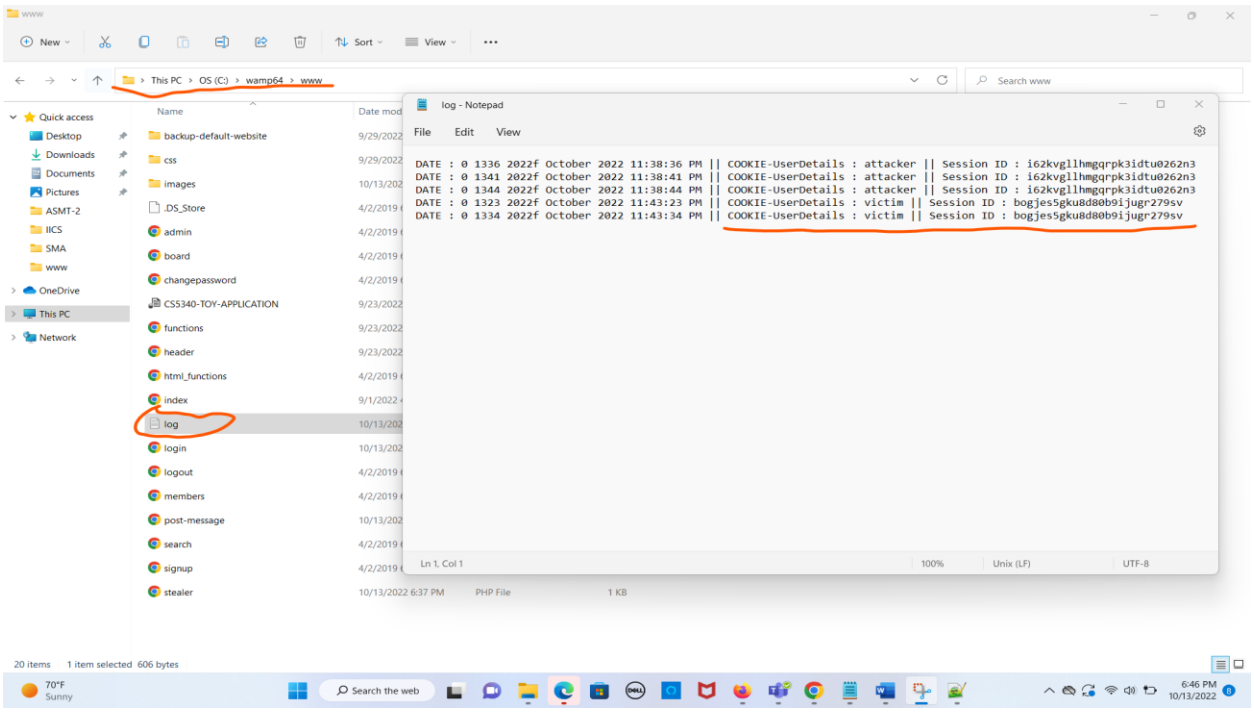
Task 1: Stealing user's credentials e.g., cookies






CS5340 - Attacks
localhost/post-message.php
CS5340
Welcome attacker | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Message posted.
Post Message:
Post


CS5340 - Attacks
localhost/board.php
CS5340
Welcome victim | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Posted on Oct 13th, 2022 11:42pm by attacker
Posted on Oct 13th, 2022 1:02am by subhash
hello
Posted on Oct 12th, 2022 11:47pm by user
hello world!!!!
Posted on Oct 12th, 2022 11:44pm by user
hi, this is a testing from an attacker.
Posted on Oct 12th, 2022 10:21pm by user
Posted on Oct 12th, 2022 10:21pm by subhash
70°F
Sunny
Search the web
6:43 PM
10/13/2022

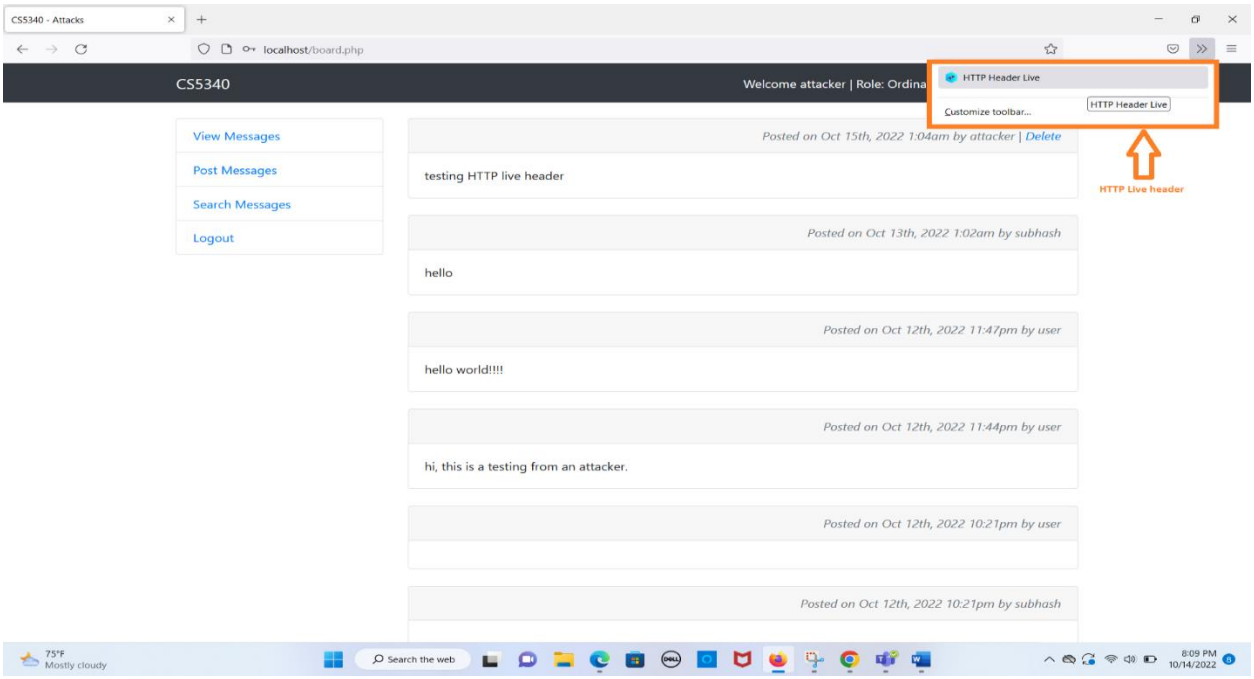


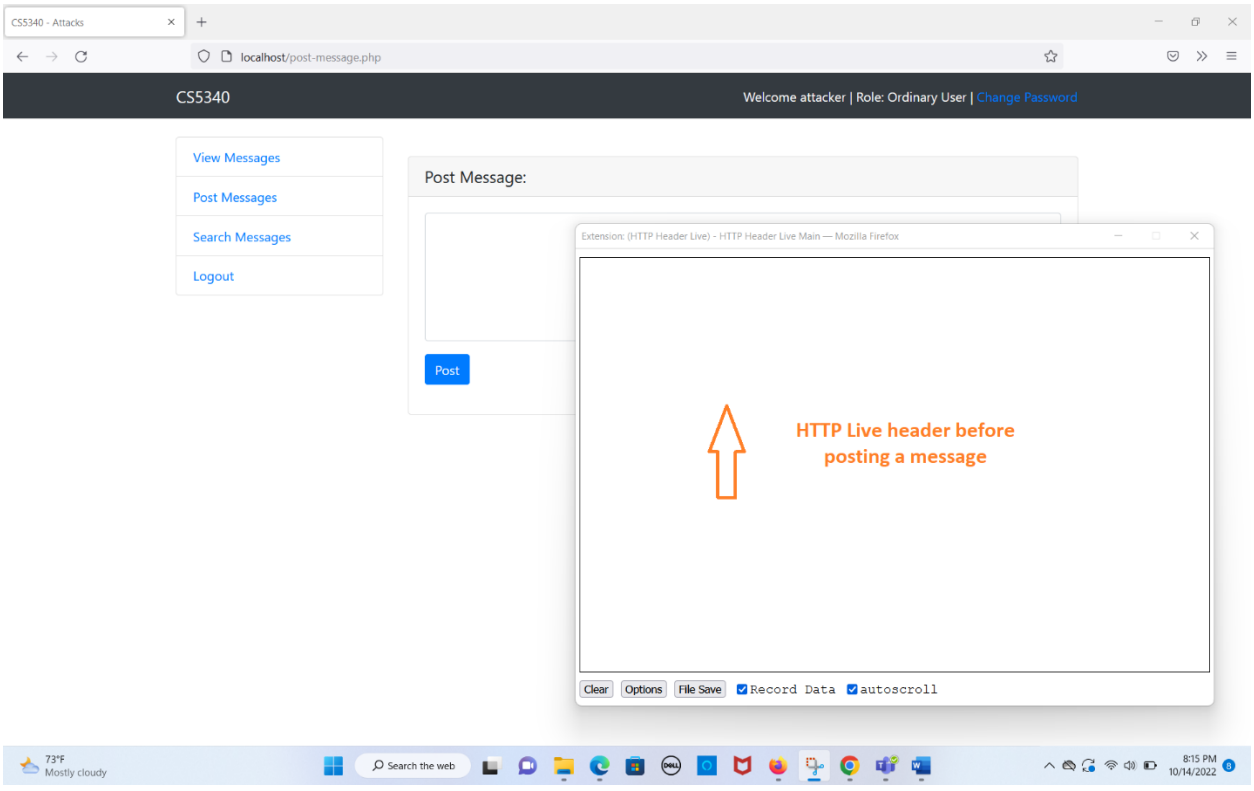
What do you observe? Explain the reason behind this observation.

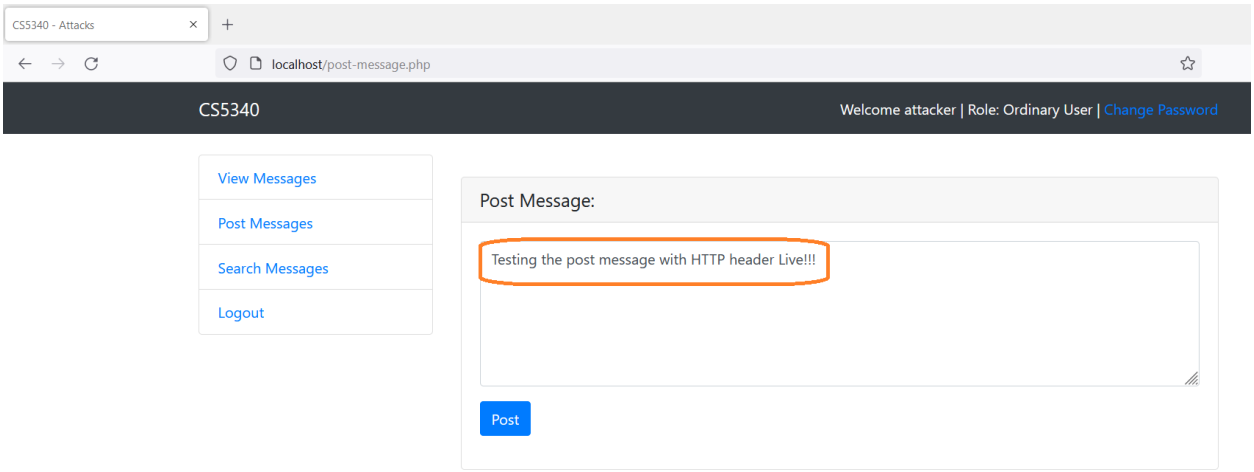
Here, attacker has stole victim login details from the cookies by posting javascript code from post message window.

Task 2: Illegitimately posting messages on behalf of the user

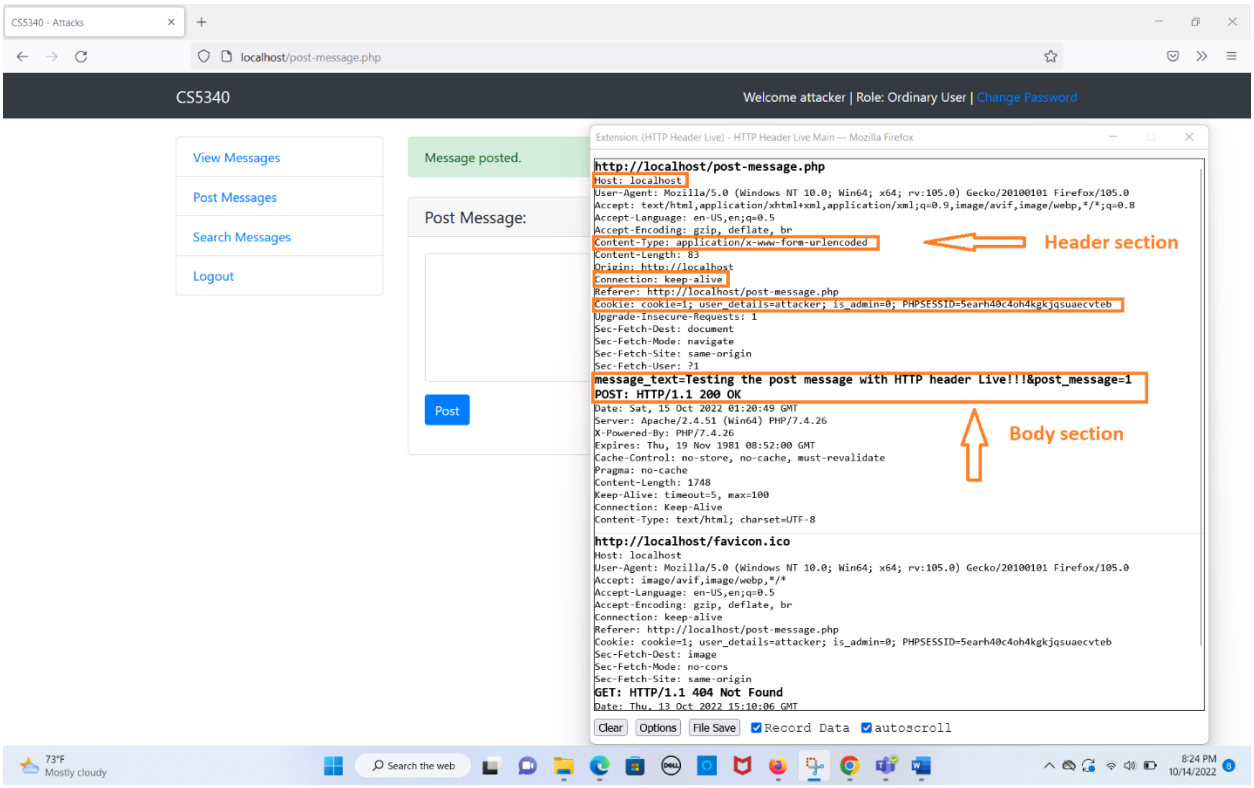
HTTP header live addon has been added to the firefox browser.


CS5340 - Attacks
localhost/board.php
CS5340
Welcome attacker | Role: Ordina
HTTP Header Live
Customize toolbar...
HTTP Header Live
HTTP Live header
View Messages
Post Messages
Search Messages
Logout
Posted on Oct 15th, 2022 1:04am by attacker | Delete
testing HTTP live header
Posted on Oct 13th, 2022 1:02am by subhash
hello
Posted on Oct 12th, 2022 11:47pm by user
hello world!!!!
Posted on Oct 12th, 2022 11:44pm by user
hi, this is a testing from an attacker.
Posted on Oct 12th, 2022 10:21pm by user
Posted on Oct 12th, 2022 10:21pm by subhash
75°F
Mostly cloudy
Search the web
8:09 PM
10/14/2022

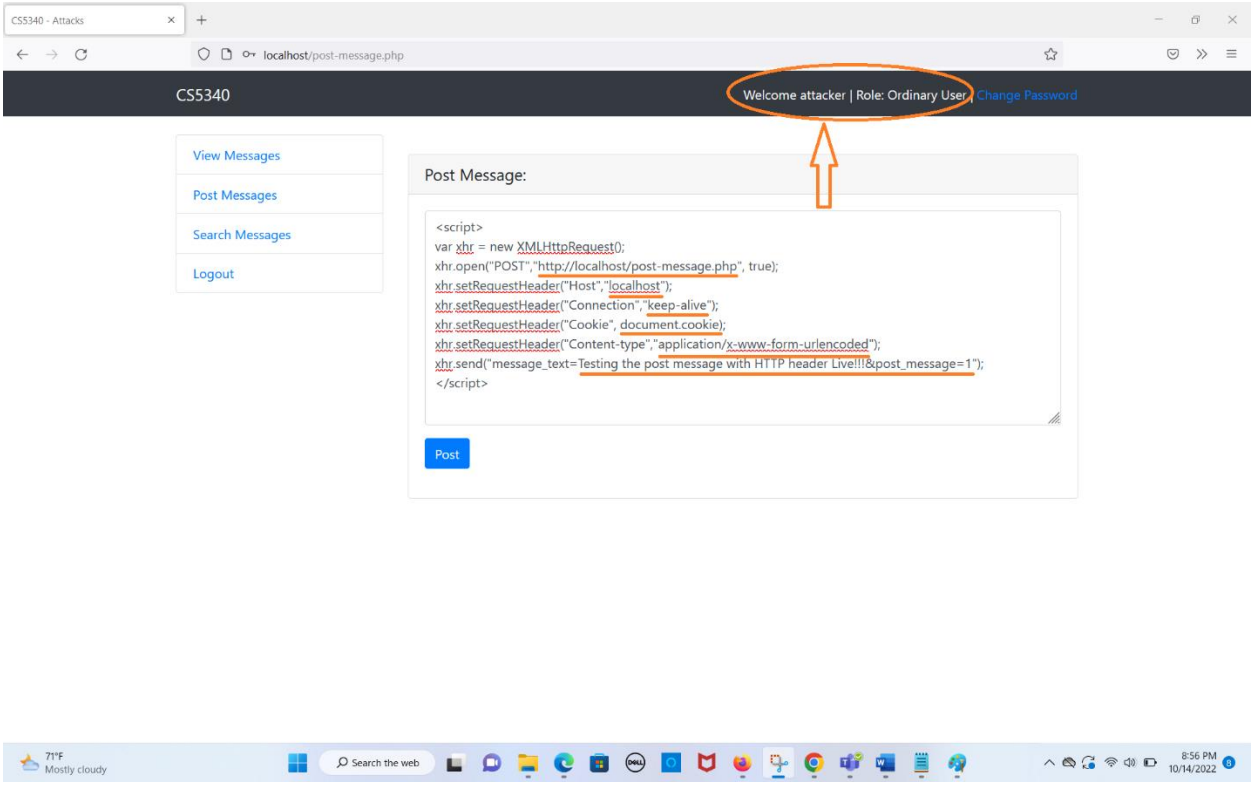

CS5340 - Attacks
localhost/post-message.php
CS5340
Welcome attacker | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Post Message:
Post
Extension: (HTTP Header Live) - HTTP Header Live Main — Mozilla Firefox
HTTP Live header before posting a message
Clear
Options
File Save
Record Data
autoscroll
73°F
Mostly cloudy
Search the web
8:15 PM
10/14/2022



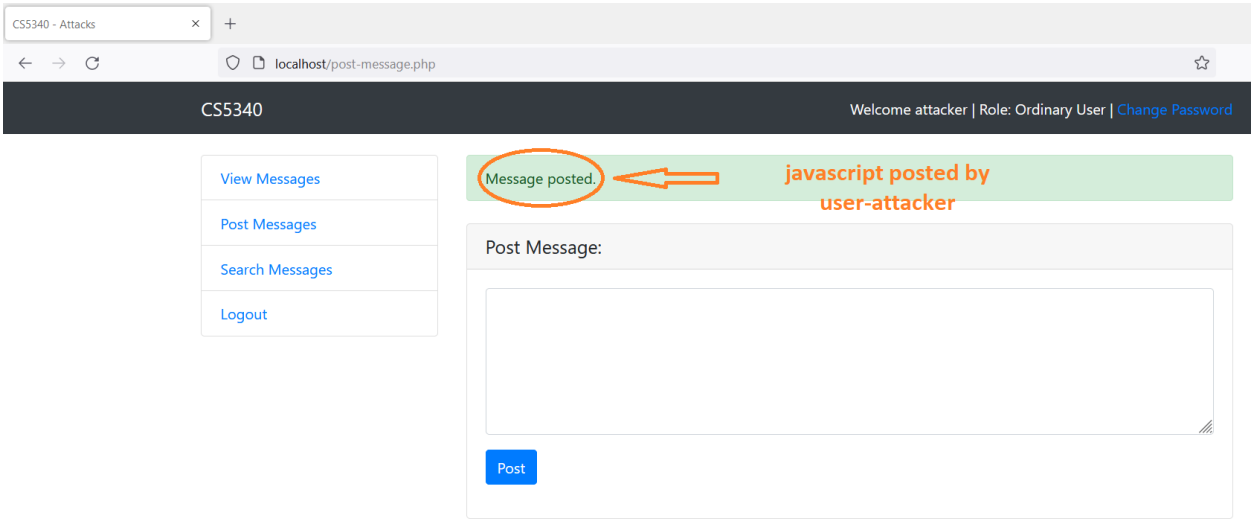
All the information is tracked in the HTTP header live when the attacker posts a message.



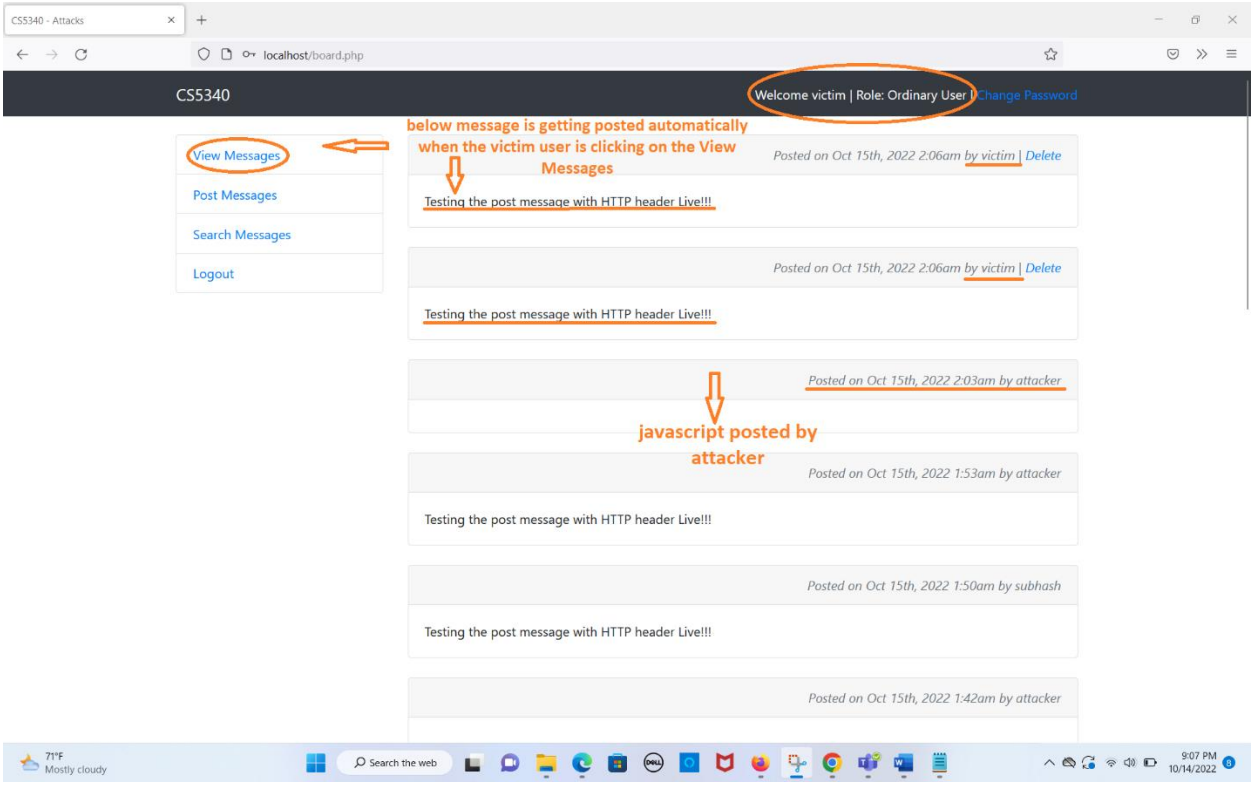
Attacker uses the information from the HTTP header live and prepare script and post it in the post message screen as shown the in below screenshot.



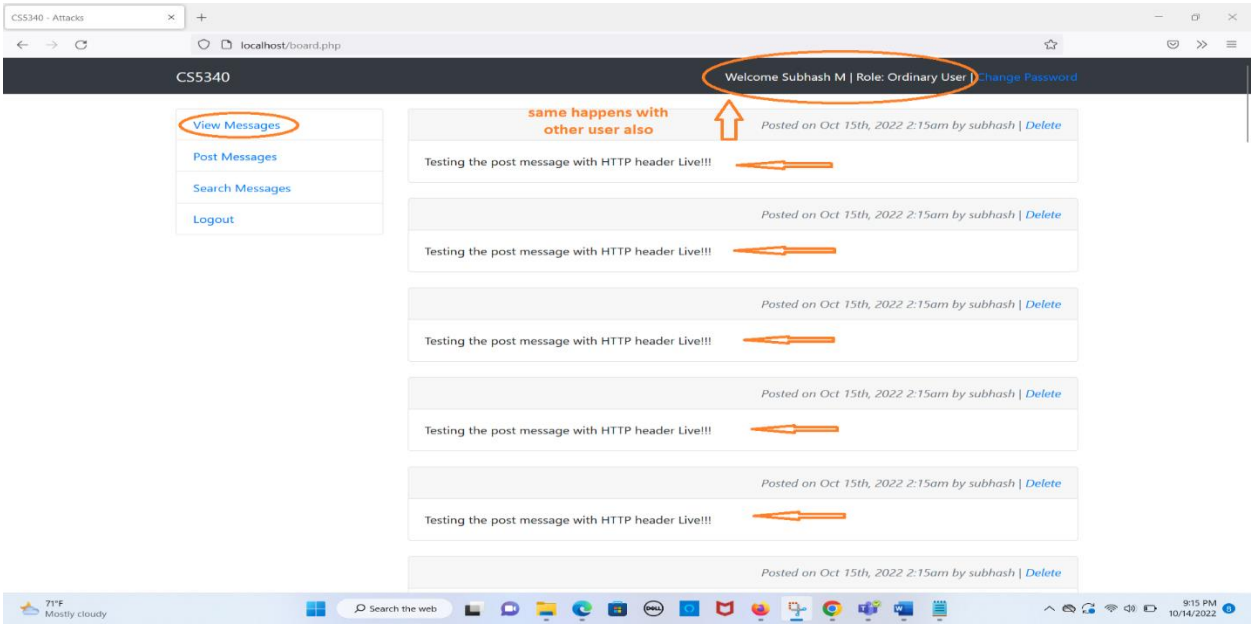
Script has been posted by the attacker



Now, when the victim logs into the application and try to clicks on View Messages page then malicious code posted by the attacker will run and illegitimately post the messages on behalf of the victim user as highlighted in the below screen.



Same happens when the other user also try to open View Messages page as below.

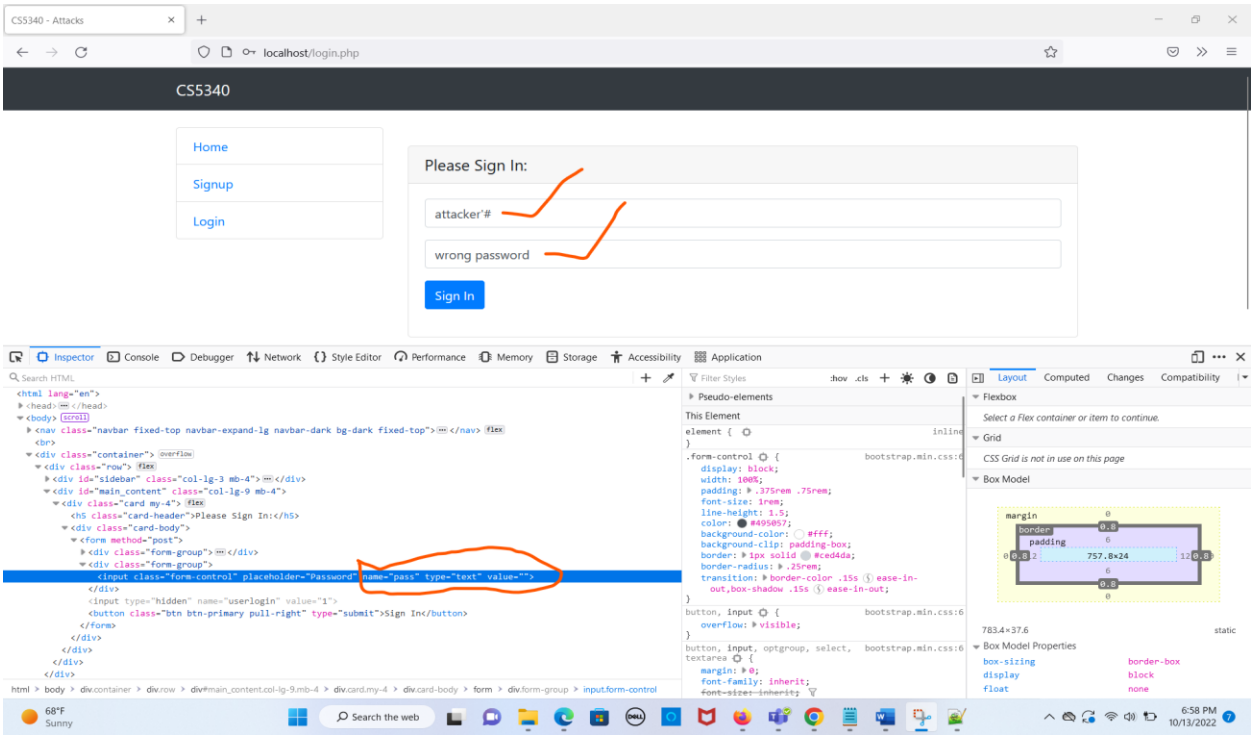


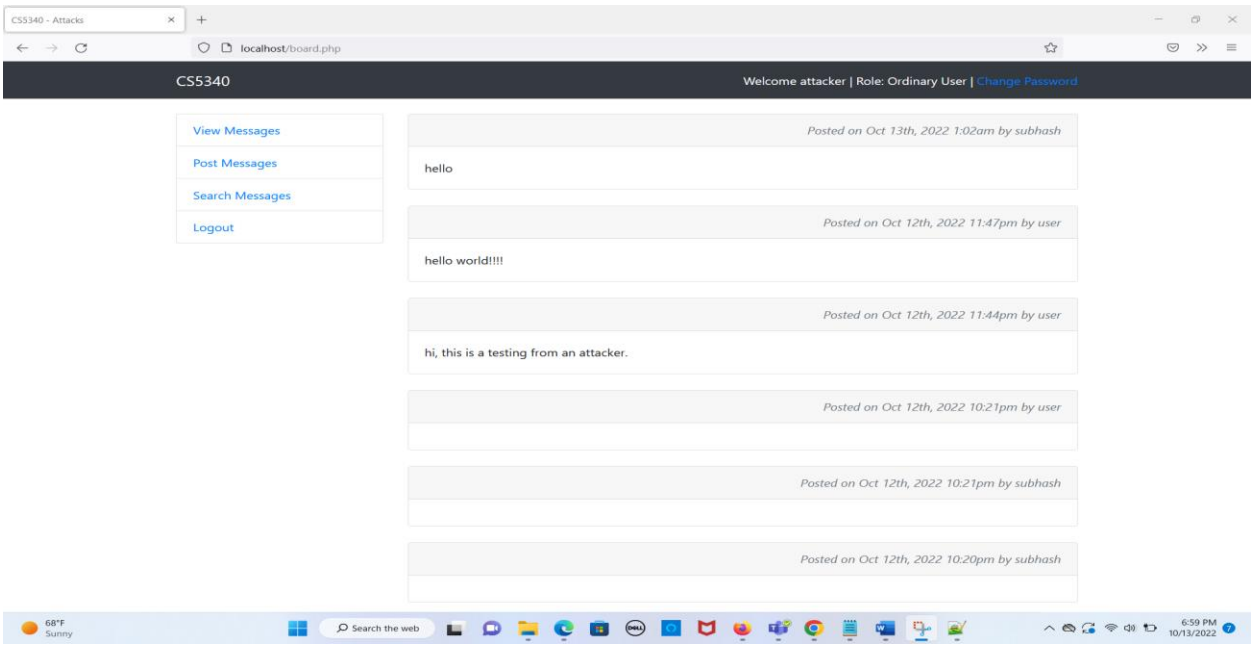
What do you observe? Explain the reason behind this observation.

Attacker has posted the javascript, which will illegitimately post a message on behalf of victim when victim or any other user clicks on View Messages button.

SECTION 2 – SQL INJECTION ATTACKS

Task 1: Log into the web application without correct password



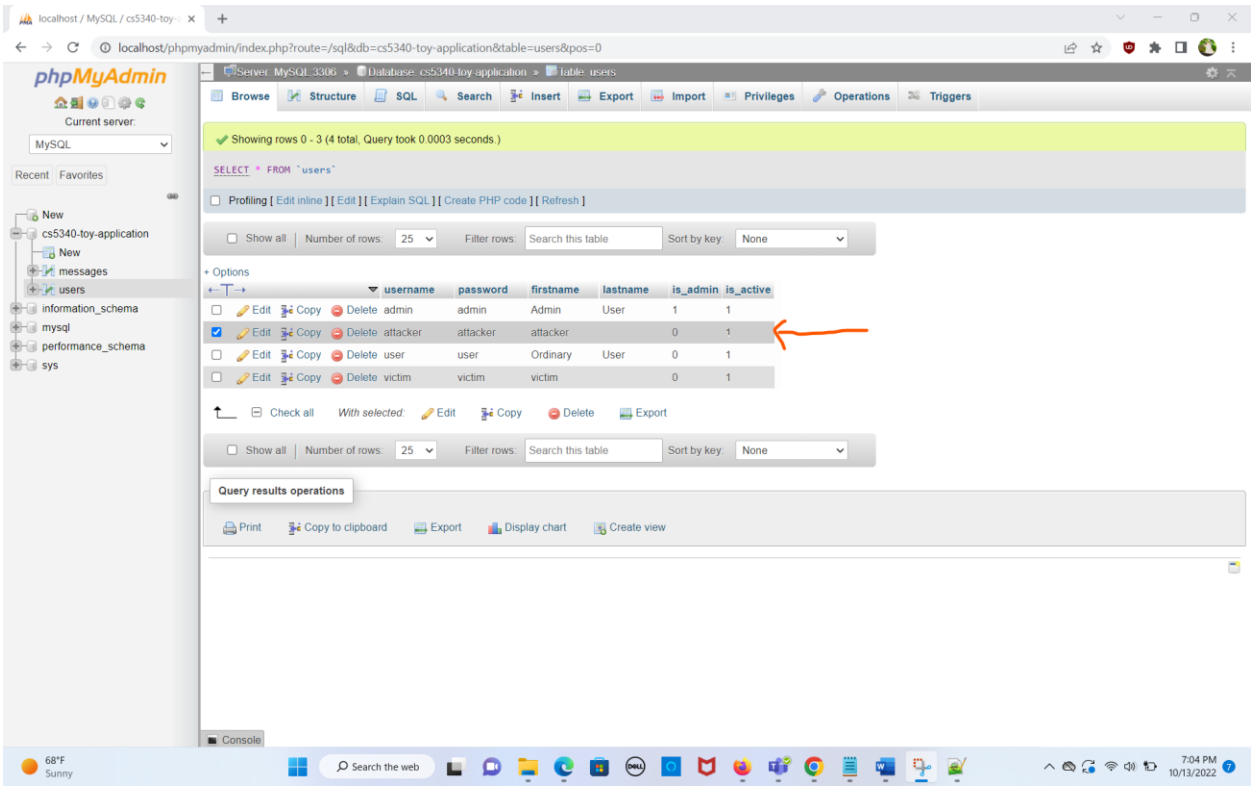


Successfully logged in using wrong password by using sql injection

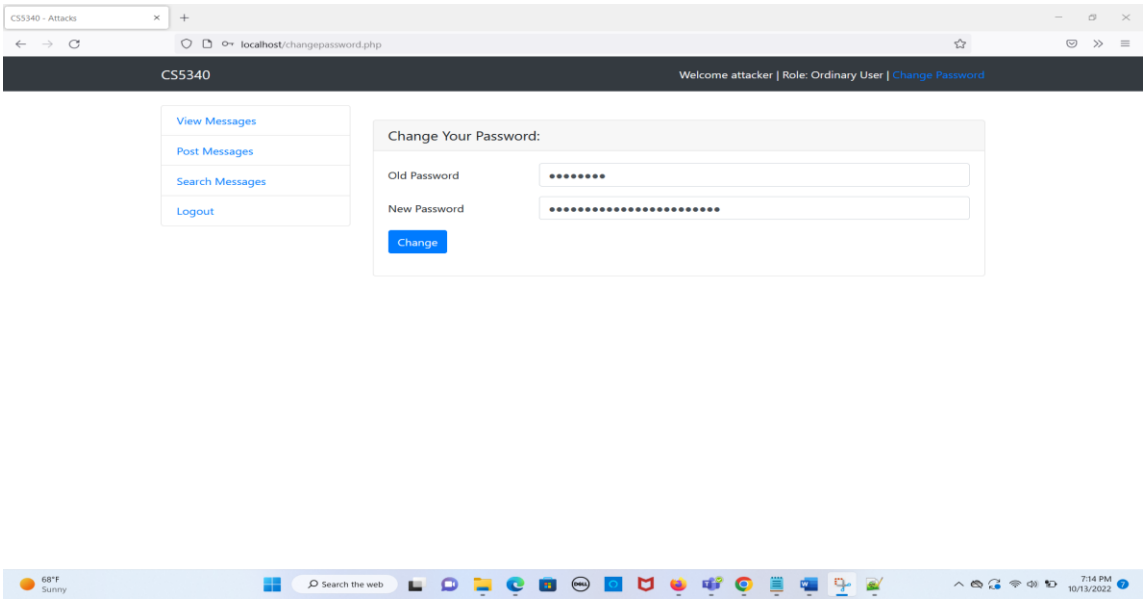
What does the SQL statement become when the above operation is done?

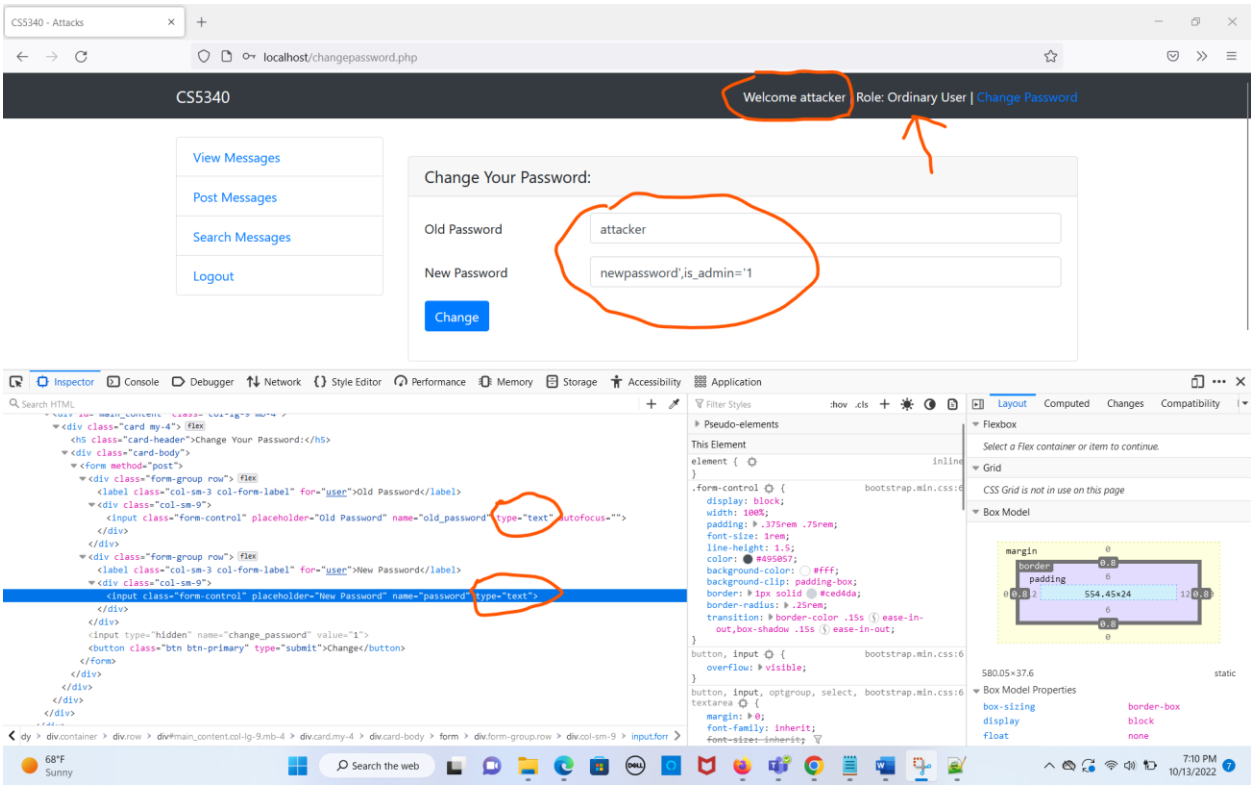
The SQL statement after the '#' are commented, in here the password verification query part is commented. This leads to login without verifying the password.

Task 2: Elevate user's account privileges

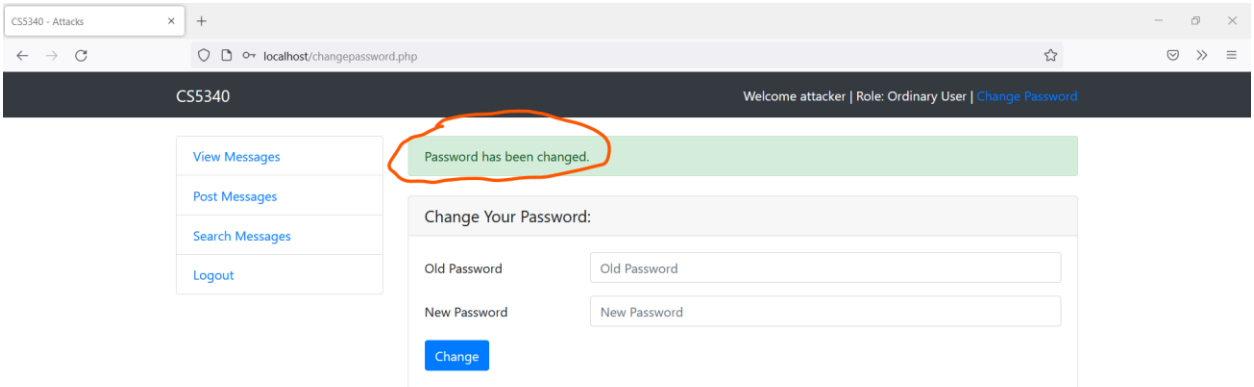


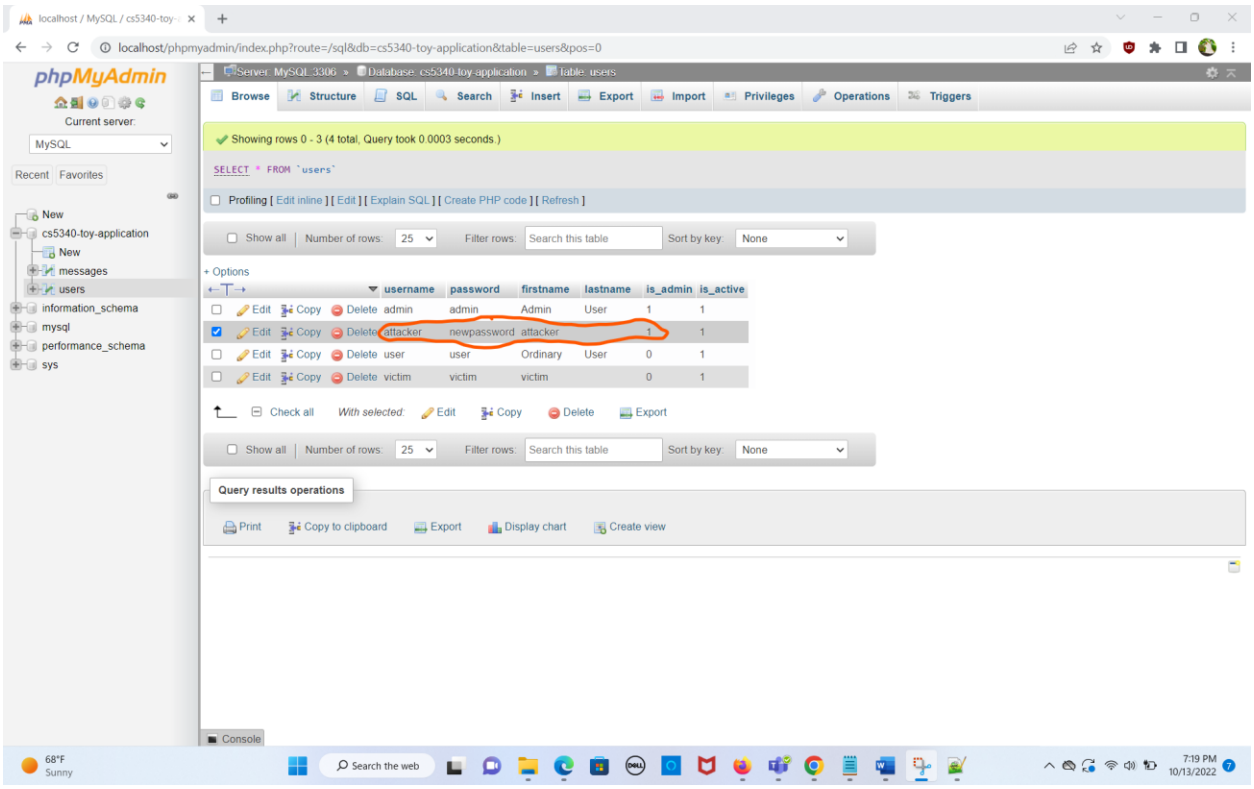
The attacker is an ordinary user, here we are going to perform SQL injection to modify user access role to administrator



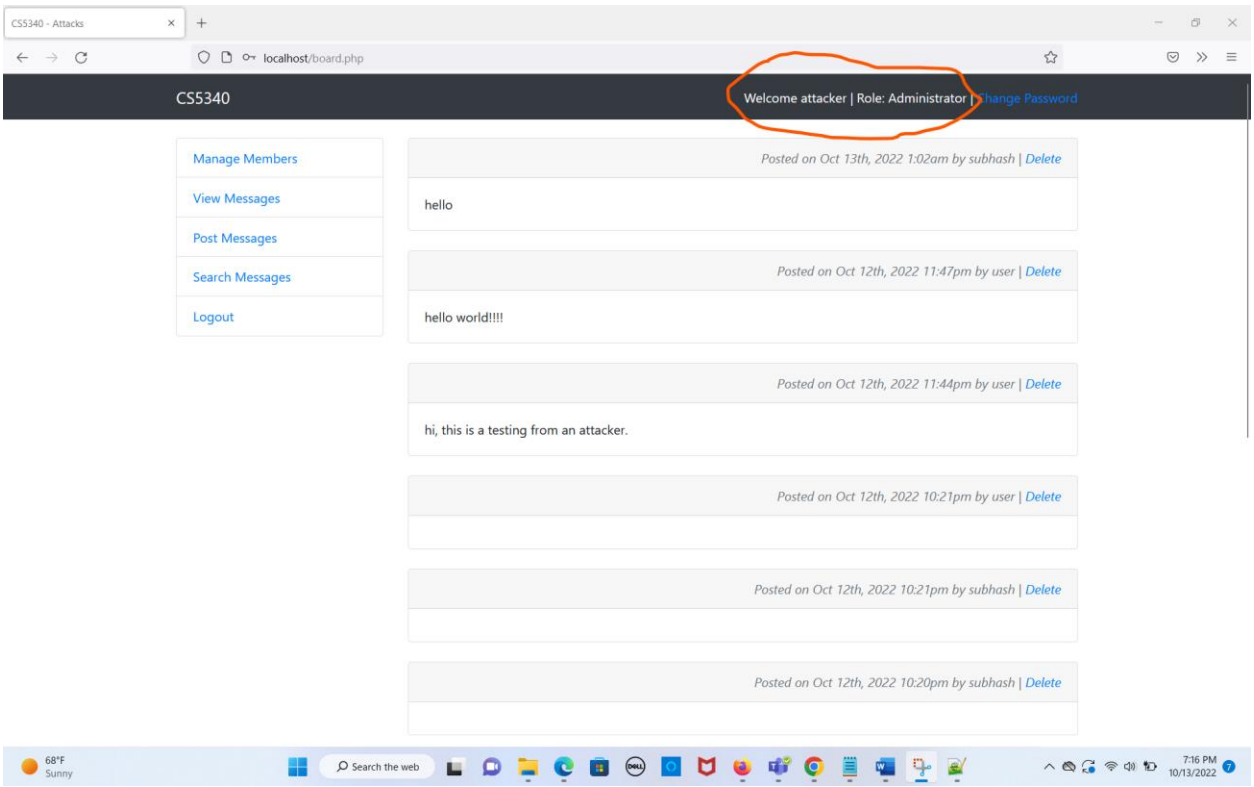

CS5340 - Attacks
localhost/changepassword.php
CS5340
Welcome attacker | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Change Your Password:
Old Password
attacker
New Password
newpassword',is_admin='1
Change

Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Search HTML
<h5 class="card-header">Change Your Password:</h5>
<label class="col-sm-3 col-form-label" for="user">Old Password</label>
<input class="form-control" placeholder="Old Password" name="old_password" type="text" autofocus="">
<label class="col-sm-3 col-form-label" for="user">New Password</label>
<input class="form-control" placeholder="New Password" name="password" type="text">
<input type="hidden" name="change_password" value="1">
<button class="btn btn-primary" type="submit">Change</button>
Pseudo-elements
This Element
Layout
Computed
Changes
Compatibility
Flexbox
Select a Flex container or item to continue.
Grid
CSS Grid is not in use on this page
Box Model
Box Model Properties
box-sizing
display
float
border-box
block
none
68°F
Sunny
Search the web
7:10 PM
10/13/2022


CS5340 - Attacks
localhost/changepassword.php
CS5340
Welcome attacker | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Password has been changed.
Change Your Password:
Old Password
Old Password
New Password
New Password
Change
68°F
Sunny
Search the web
7:13 PM
10/13/2022



Here the attacker got administrator access.



What does the SQL statement become when the above operation is done?

Here the sql the password has ',is_admin='1 concatenated to its newly created password. Which will act as an additional condition. Therefore it will change the password and grant the user administrator access to the application.

SECTION 2 – DEFENCE MECHANISMS.

(a) (i) Provide a description of the various defense mechanisms that can help mitigate the attacks showcased in this lab. Describe in detail the core ideas behind each of the defense mechanisms you describe.

XSS attacks:

for XSS attacks we can use the following prevention methods

1.Filter the input data(received from user) on arrival to system. This can reduce the chances of using invalid input fields.

2. Using appropriate response headers like content-type, x-content-type- options are help us prevent XSS in http responses. These response headers are not intended to contain any html or JavaScript.

3. Enforcing content security policy is one of the popular option of last line defense in XSS attacks. By using this we can reduce severity XSS vulnerabilities that still occur.

SQL Injection:

The ideal way to prevent SQL injection attacks is validating input fields and parameterized queries.

The single quotes should not be allowed in input fields and best practice is to avoid displaying database errors to users in browser.

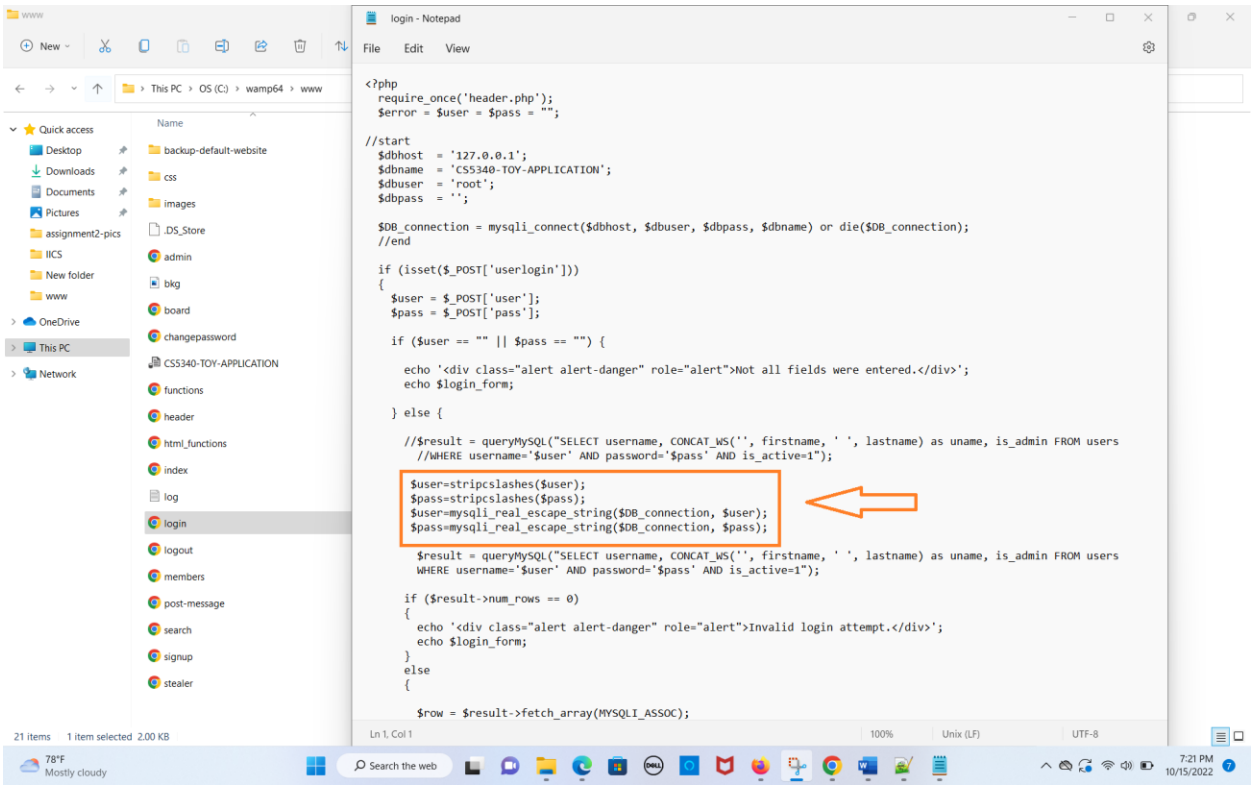
(ii) Why do these attacks continue to plague today's web applications despite the existence of the defense mechanisms that you describe above?

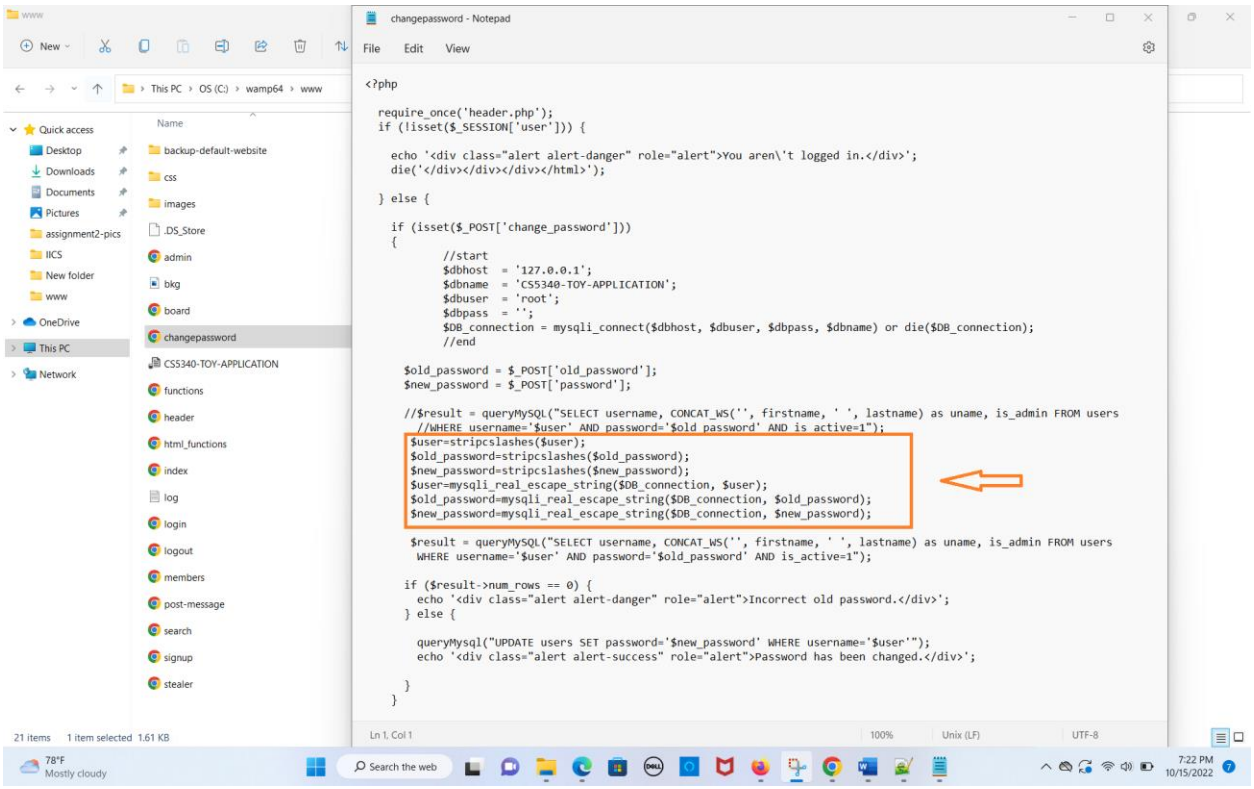
1. Extensive use of third party libraries (open source)
2. Bad coding practices make application vulnerable to attacks
3. Untrained developers on security issues of the application
4. Application with security as least policy
5. Unawareness of user can compromise security of the application by clicking on links that can contain malicious content

(b) Modify the application to ensure that it is immune to these attacks. Using screen shots as appropriate, demonstrate that the application cannot fall to these attacks

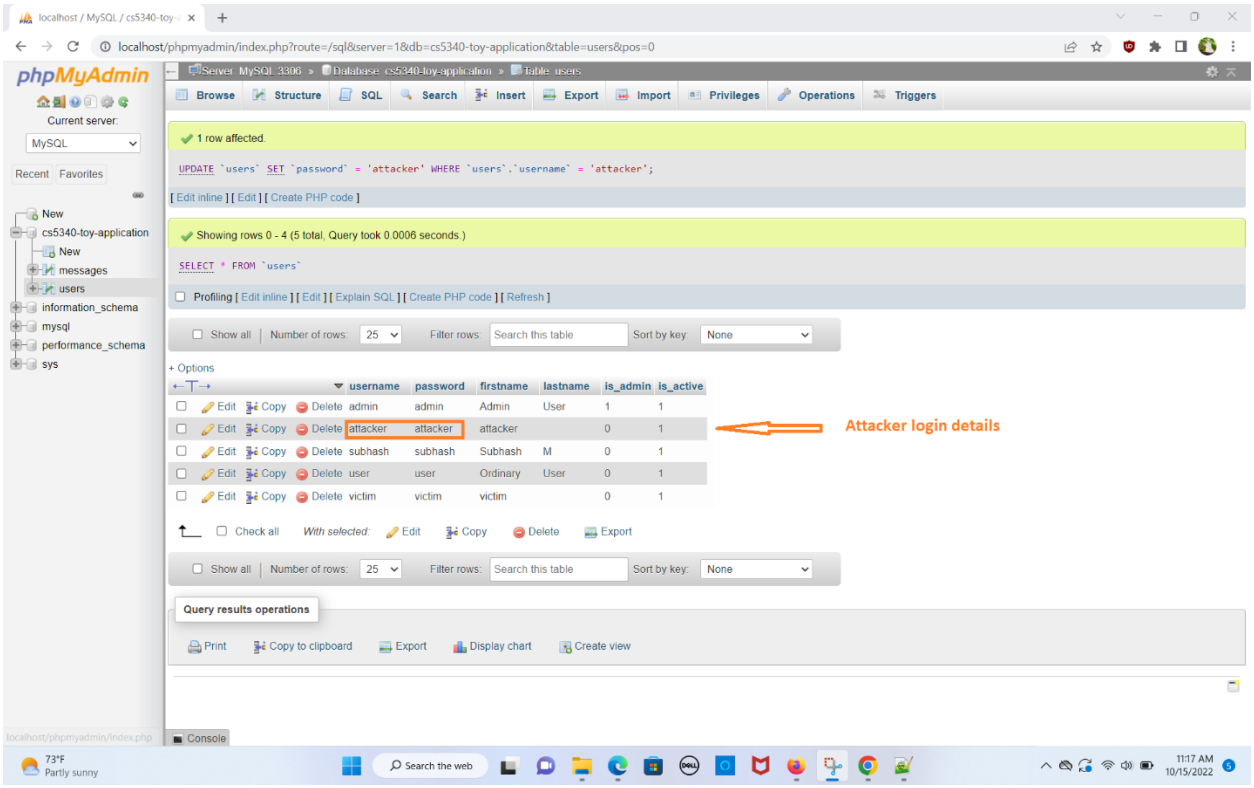
Prevention for SQL Injection:

SQL injection is prevented by using the sql functions : **stripcslashes()** and **mysqli_real_escape_string()** as highlighted in the below screenshots.

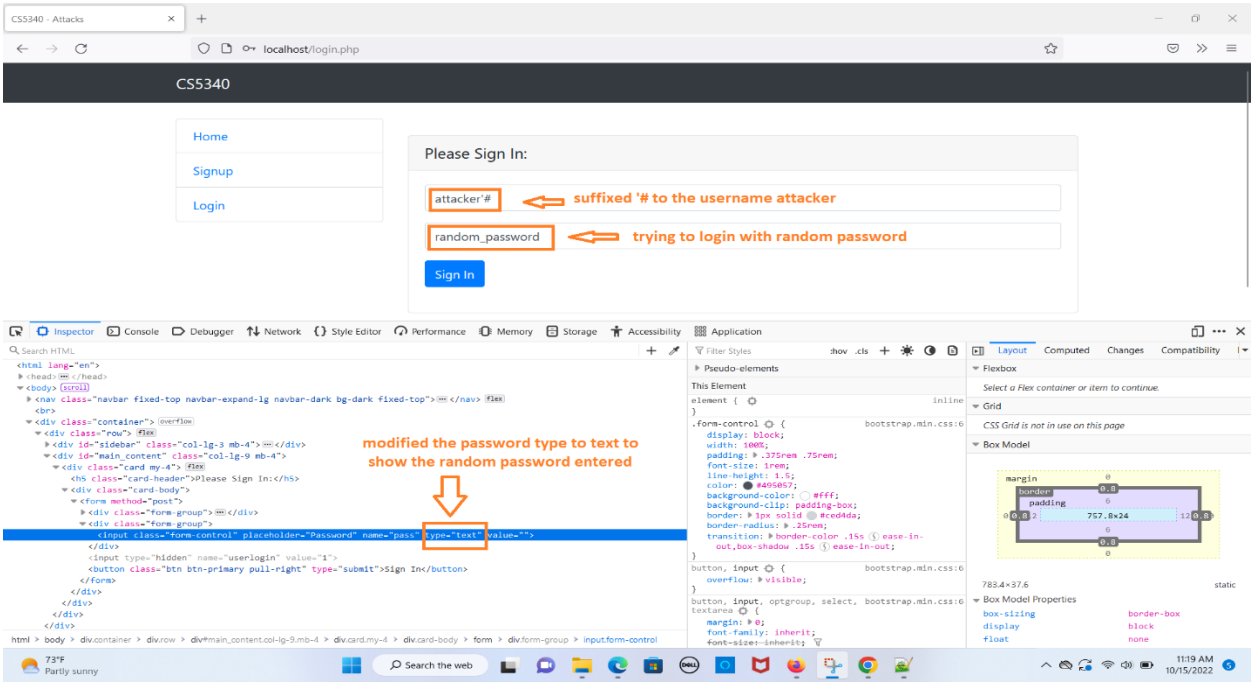

login - Notepad
File Edit View
<?php
require_once('header.php');
$error = $user = $pass = "";
//start
$dbhost = '127.0.0.1';
$dbname = 'CS5340-TOY-APPLICATION';
$dbuser = 'root';
$dbpass = '';
$DB_connection = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname) or die($DB_connection);
//end
if (isset($_POST['userlogin']))
{
$user = $_POST['user'];
$pass = $_POST['pass'];
if ($user == "" || $pass == "") {
echo ' Not all fields were entered. ';
echo $login_form;
} else {
//$result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users
//WHERE username='$user' AND password='$pass' AND is_active=1");
$user=stripcslashes($user);
$pass=stripcslashes($pass);
$user=mysqli_real_escape_string($DB_connection, $user);
$pass=mysqli_real_escape_string($DB_connection, $pass);
$result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users
WHERE username='$user' AND password='$pass' AND is_active=1");
if ($result->num_rows == 0)
{
echo ' Invalid login attempt. ';
echo $login_form;
}
else
{
$row = $result->fetch_array(MYSQLI_ASSOC);
Ln 1, Col 1
100%
Unix (LF)
UTF-8
21 items
1 item selected 2.00 KB
78°F
Mostly cloudy
Search the web
7:21 PM
10/15/2022


changepassword - Notepad
File Edit View
<?php
require_once('header.php');
if (!isset($_SESSION['user'])) {
echo ' You aren\'t logged in. ';
die(' </html>');
} else {
if (isset($_POST['change_password']))
{
//start
$dbhost = '127.0.0.1';
$dbname = 'CS5340-TOY-APPLICATION';
$dbuser = 'root';
$dbpass = '';
$DB_connection = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname) or die($DB_connection);
//end
$old_password = $_POST['old_password'];
$new_password = $_POST['password'];
//$result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users
//WHERE username='$user' AND password='$old_password' AND is_active=1");
$user=stripcslashes($user);
$old_password=stripcslashes($old_password);
$new_password=stripcslashes($new_password);
$user=mysqli_real_escape_string($DB_connection, $user);
$old_password=mysqli_real_escape_string($DB_connection, $old_password);
$new_password=mysqli_real_escape_string($DB_connection, $new_password);
$result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users
WHERE username='$user' AND password='$old_password' AND is_active=1");
if ($result->num_rows == 0) {
echo ' Incorrect old password. ';
} else {
queryMysql("UPDATE users SET password='$new_password' WHERE username='$user'");
echo ' Password has been changed. ';
}
}
Ln 1, Col 1
100%
Unix (LF)
UTF-8
21 items
1 item selected 1.61 KB
78°F
Mostly cloudy
Search the web
7:22 PM
10/15/2022

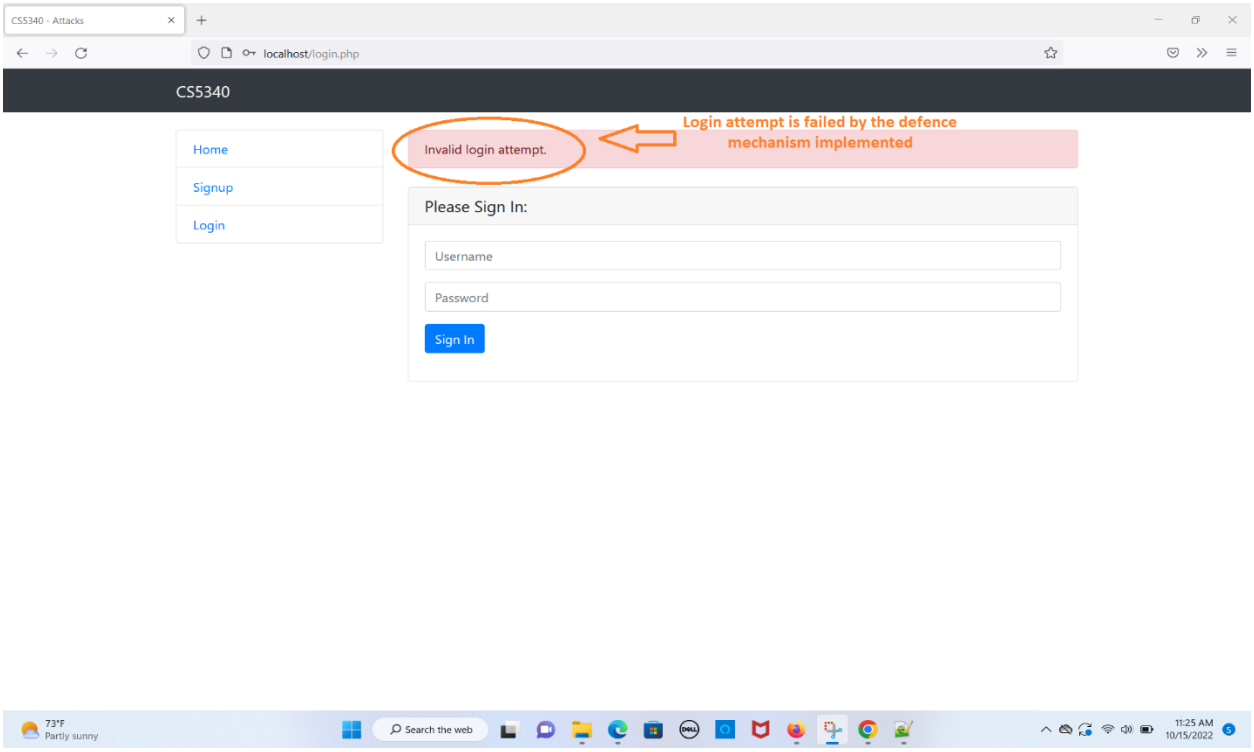
1)Attacker user login details before entering the application in the screenshot below-



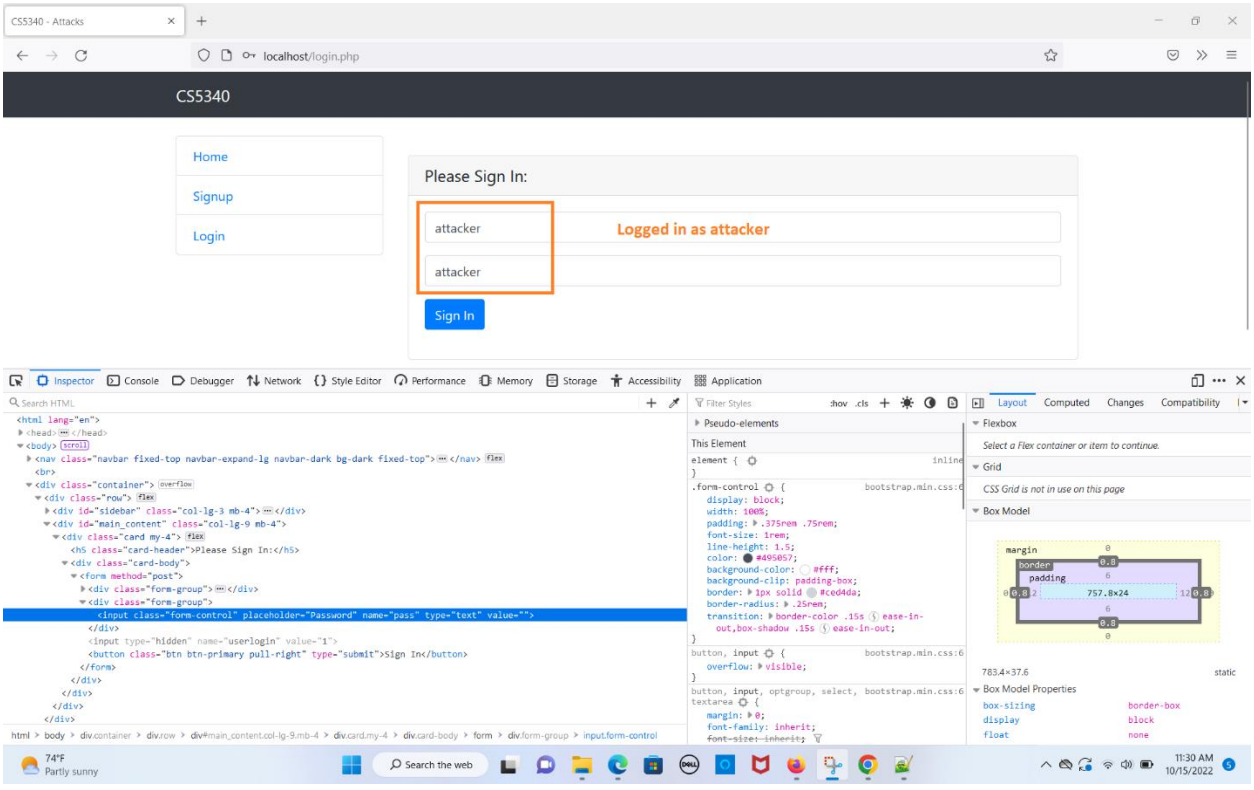
Attacker with username attacker'# and random password is trying to enter the application with SQL injection attack.



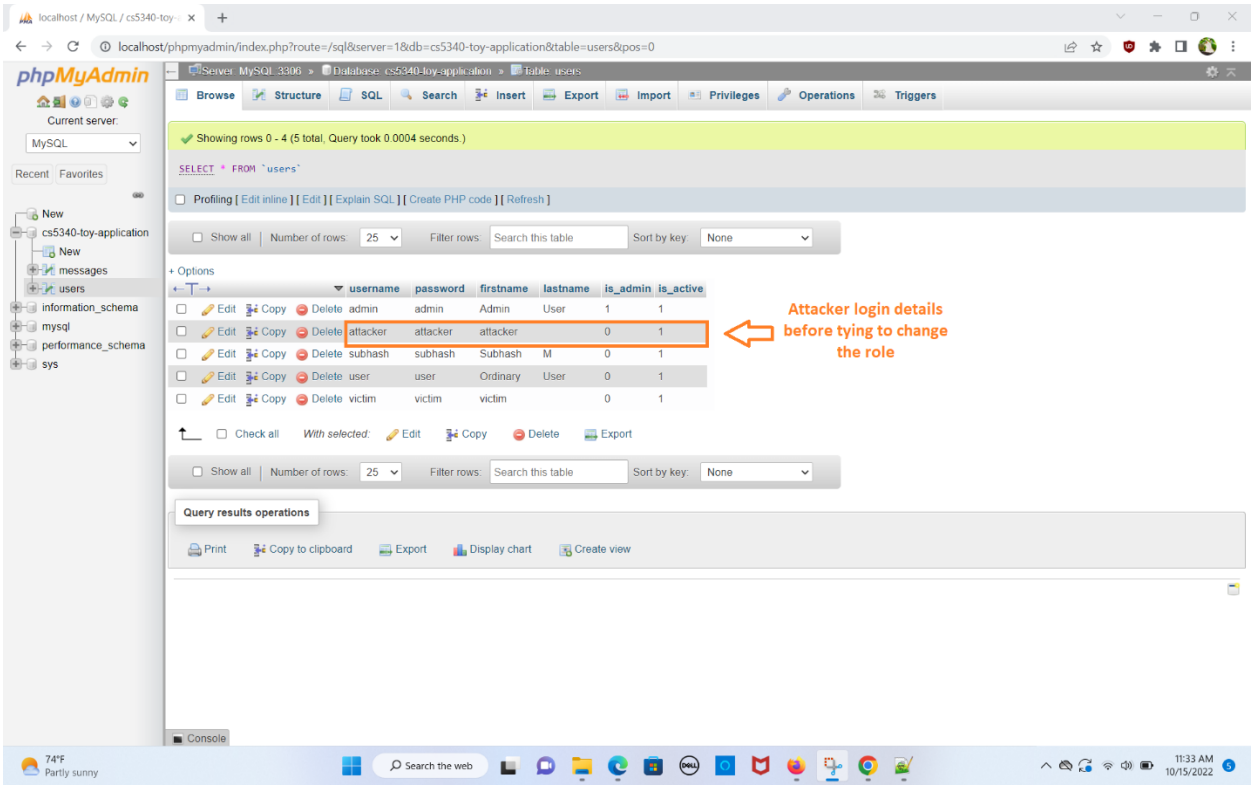
Here the defense mechanism implemented is sanitized inputs and login details are detected as invalid. Hence login is denied to the system.



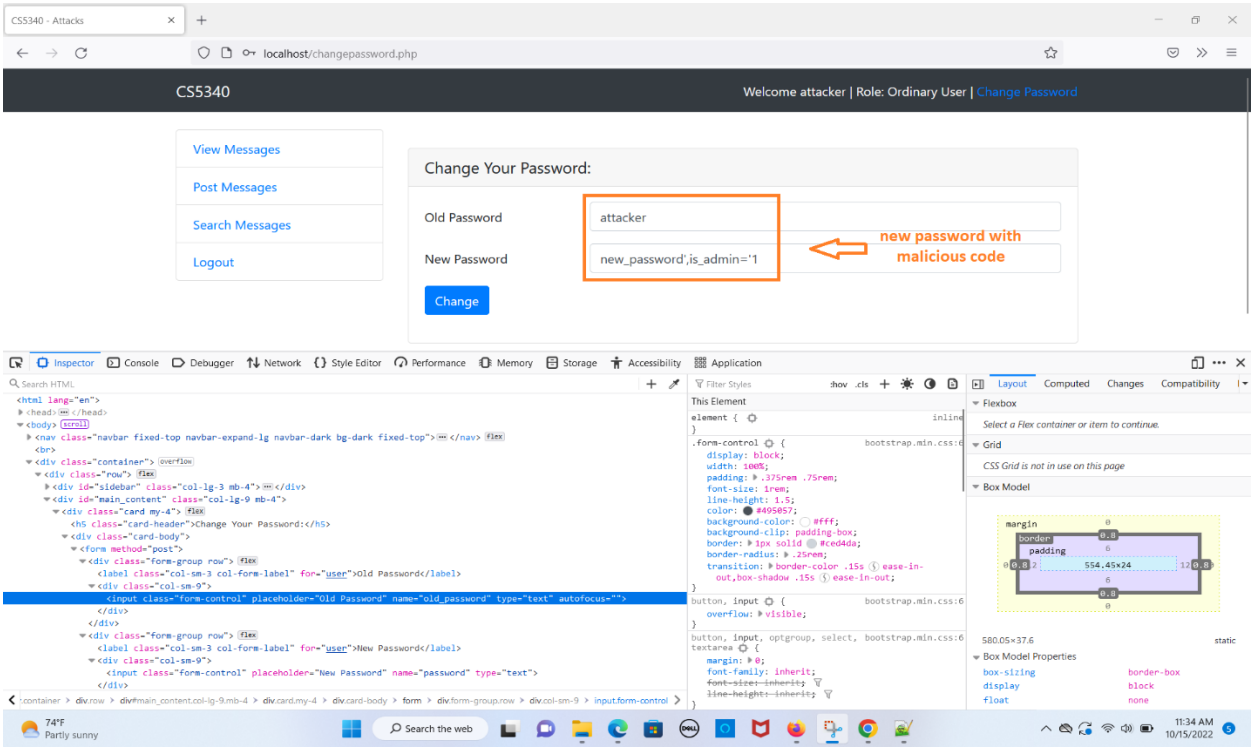
2)Attacker is logging into the application with valid credentials



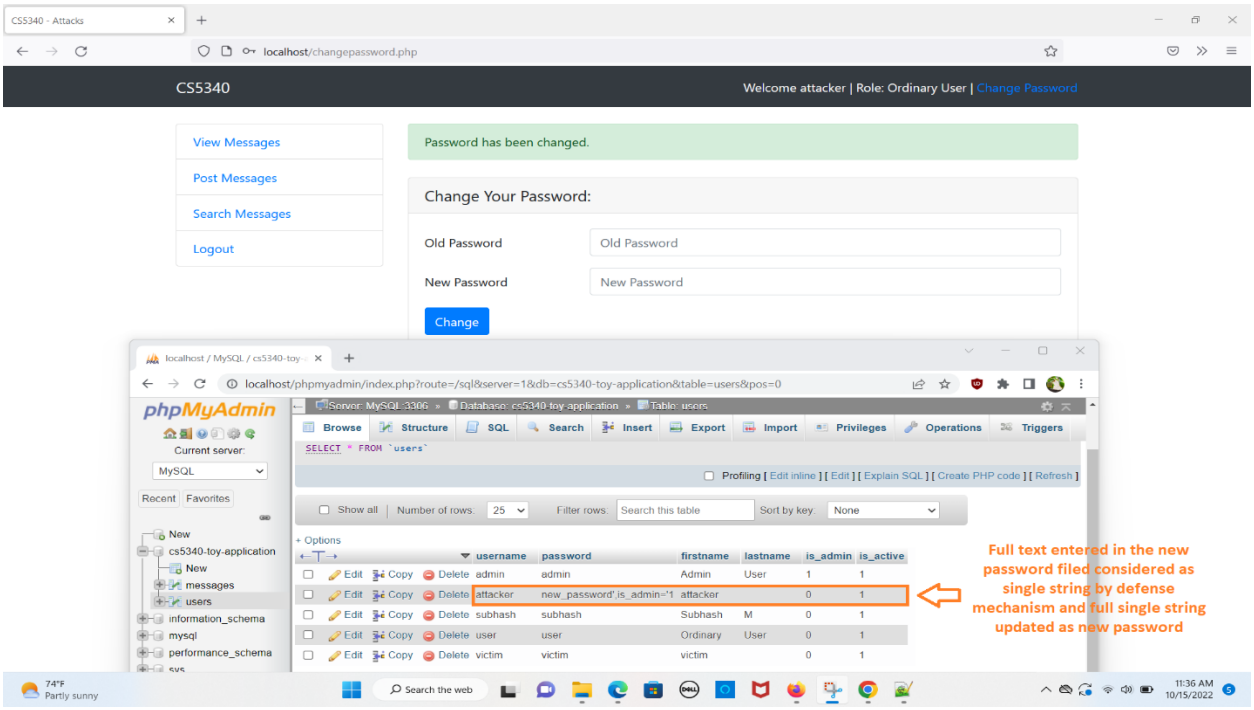
valid user details in the database users table below.



Attacker is trying to perform SQL injection with malicious command in the new password input field.

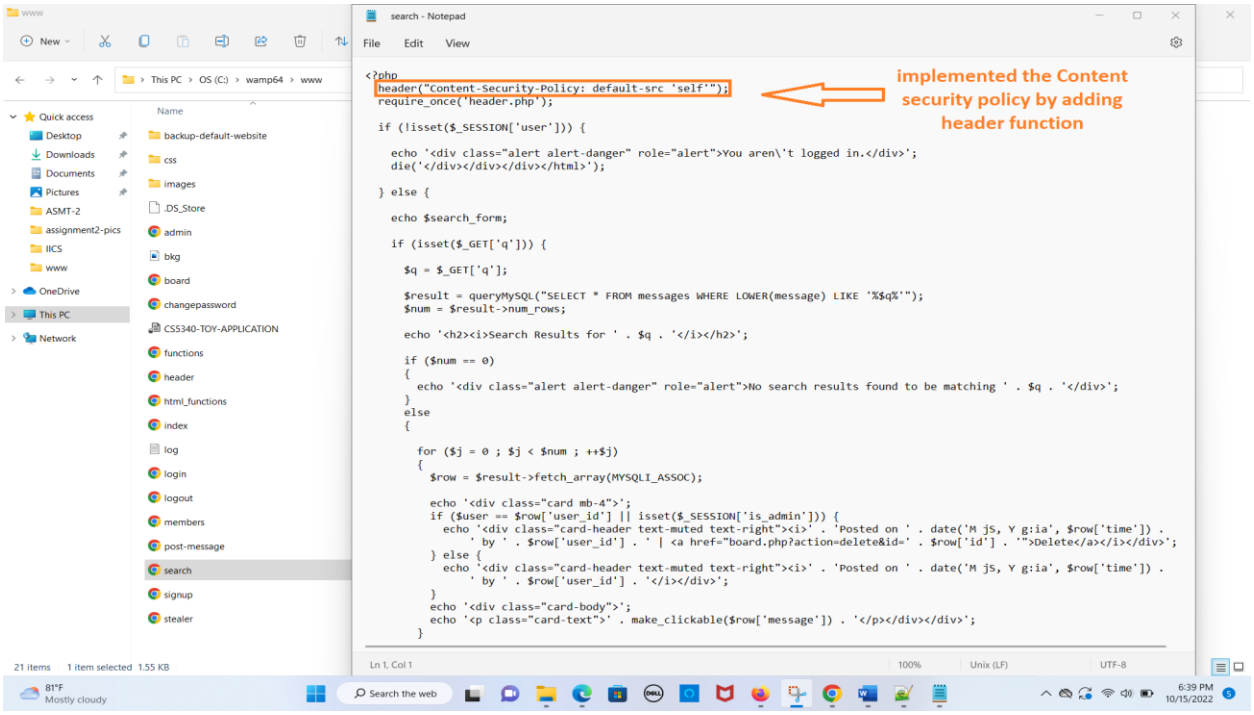


Here in this, the input fields are sanitized and defensing the malicious commands. Hence the text entered in the new password is considered as a single string and getting inserted into the database as new password which is displayed in the below picture.

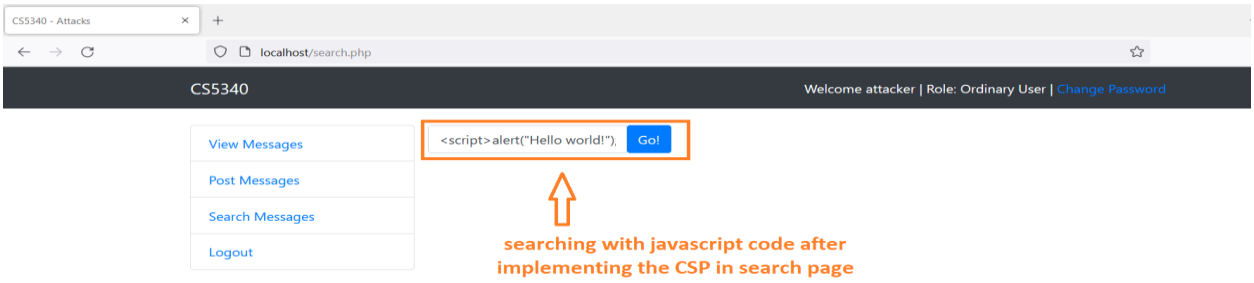


Content Security Policy:

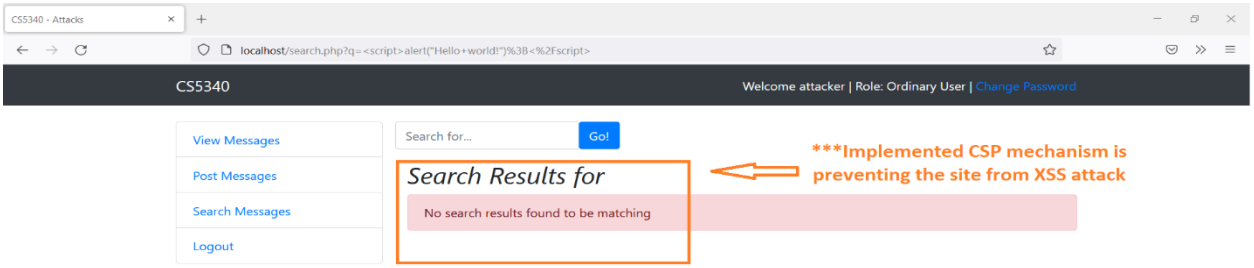
3)Content security policy has been implemented by adding the header function as in screenshot below-



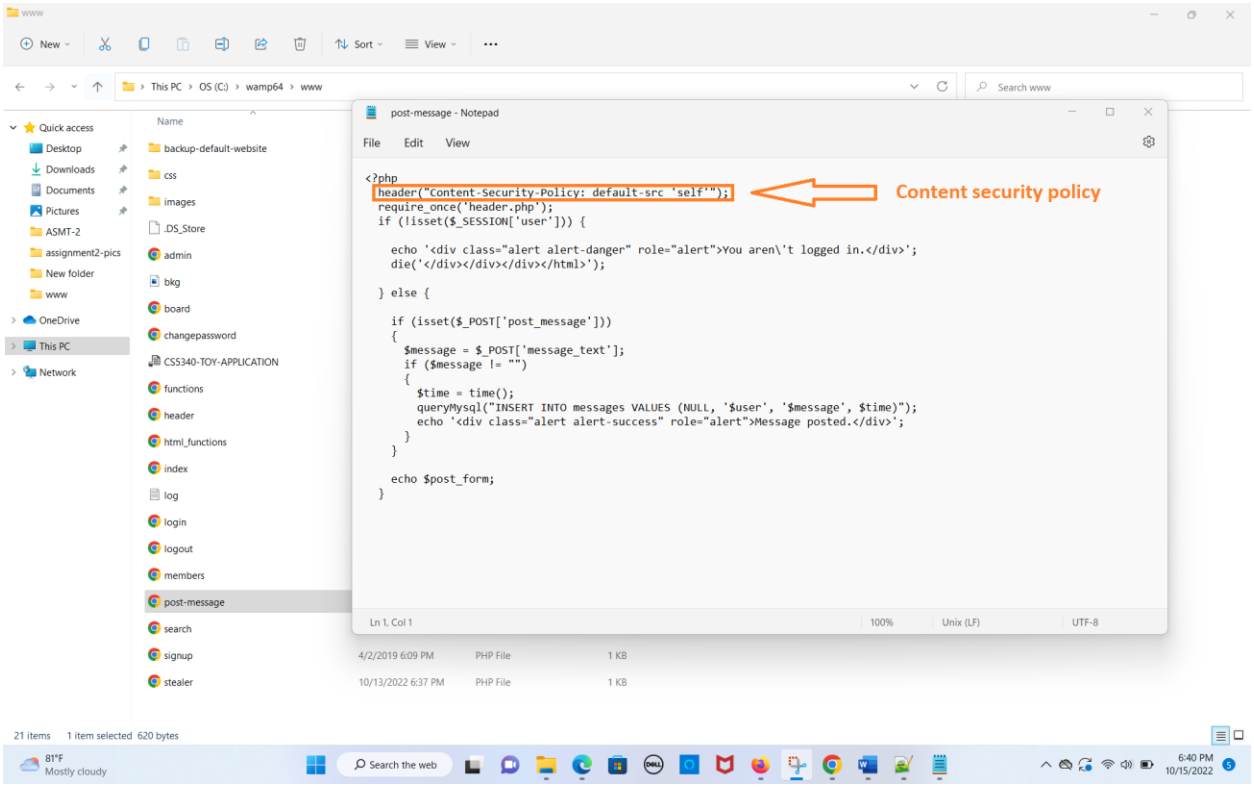
Trying to check the site it is prone to the attack by sending javascript code for alert box.



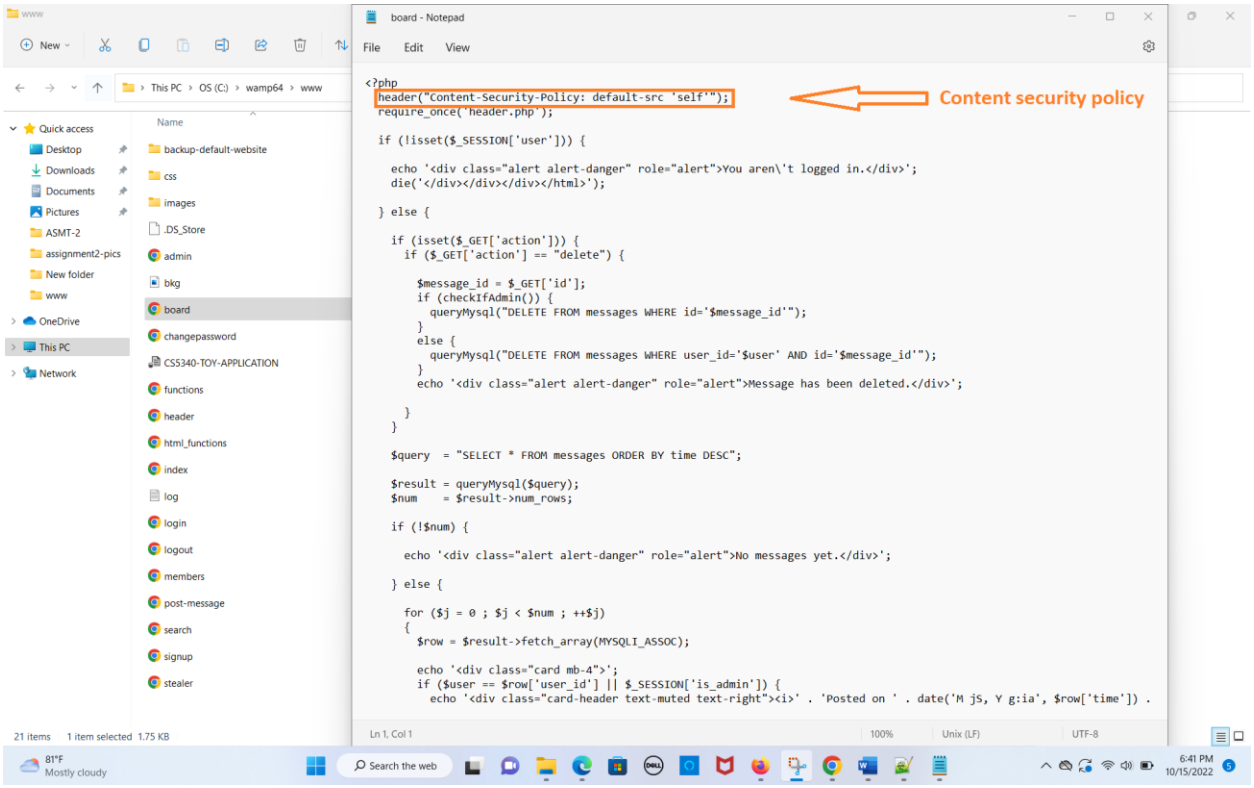
Alert box is not popped up hence implemented CSP is working fine and preventing the XSS attack.



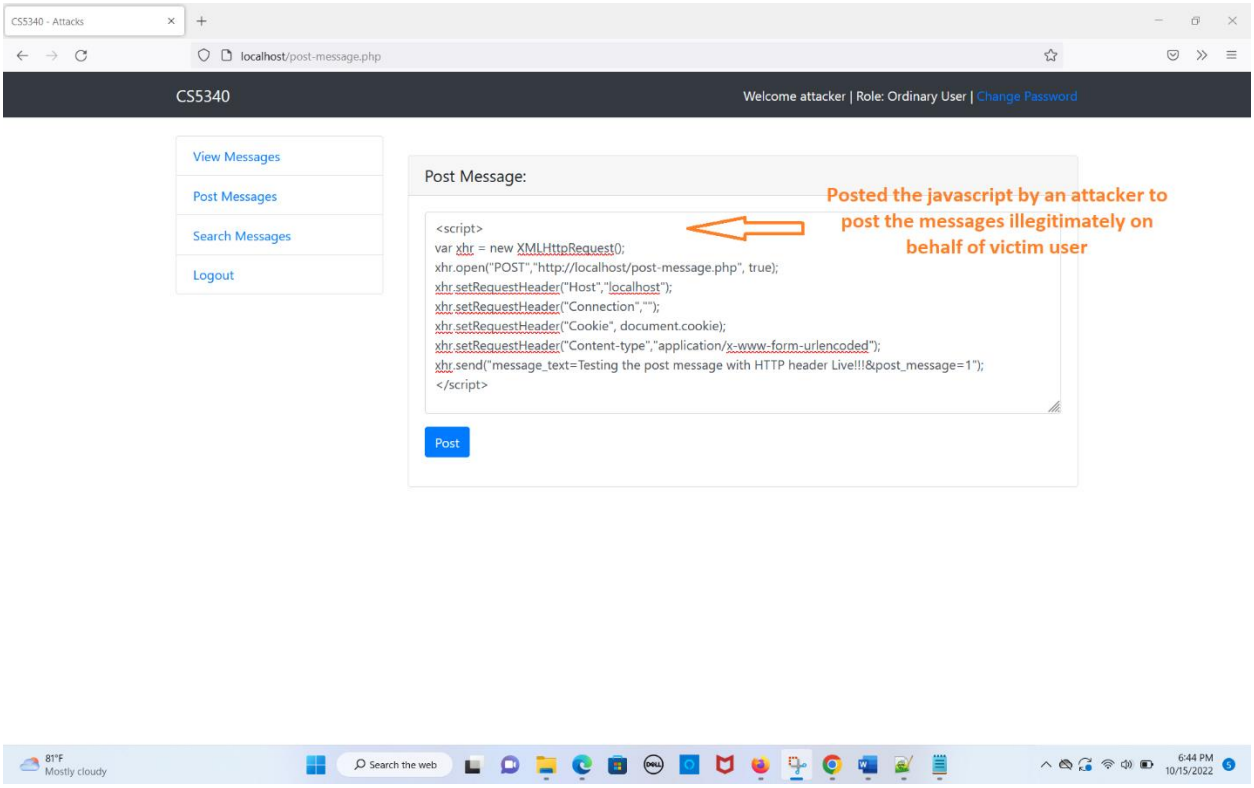
Content security policy has been implemented by adding the header function as in screenshot below-



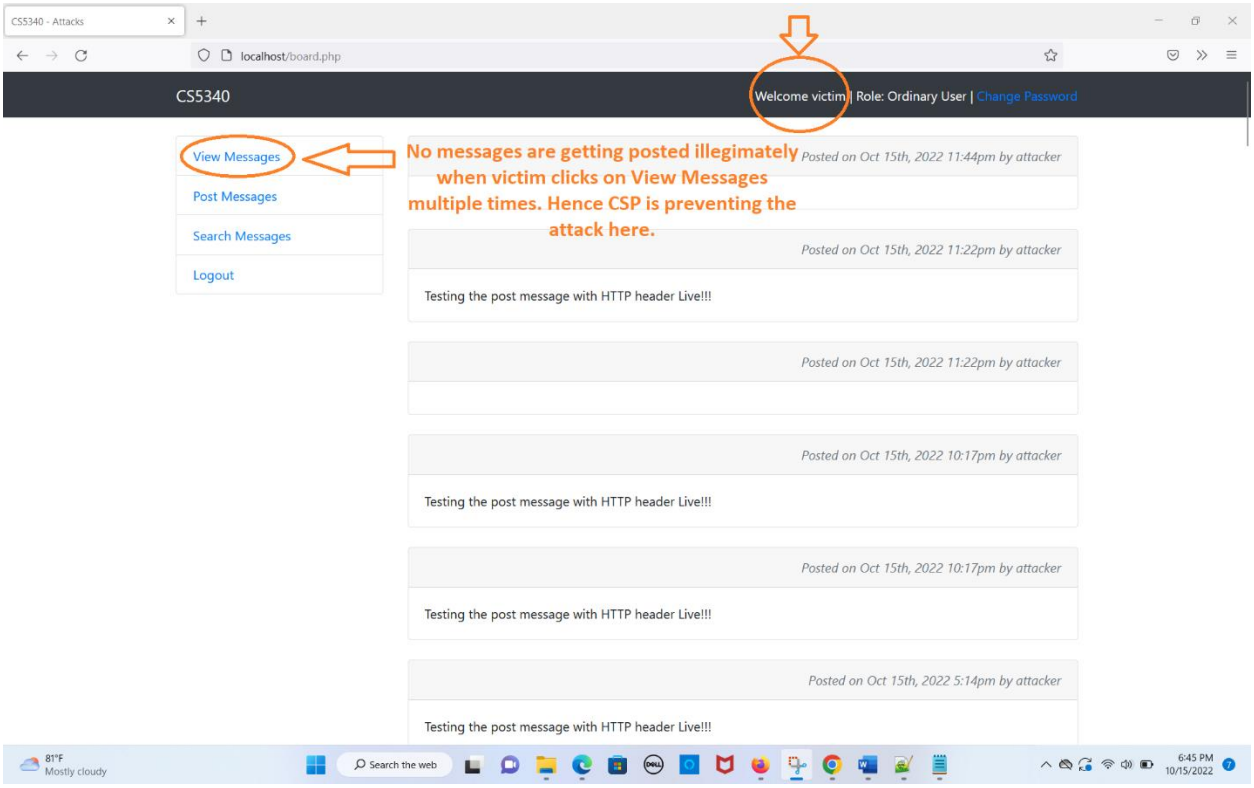
Content security policy has been implemented by adding the header function as in screenshot below-



Attacker has Posted the JavaScript to post the messages illegitimately on behalf of the victim.



Posting the messages illegitimately on behalf of the victim is not happening now hence CSP implemented is preventing the attack as in the below screenshot-



(c) Two other attacks discussed in class but not included in this lab are clickjacking and the CSRF attack. Discuss defense mechanisms for these attacks.

ClickJacking:

- This can be mitigated by preventing browser from loading the page in frame using x-frame-options or CSP (Content security policy)
- Application should be prevented from including session cookies in a frame using the same site cookie attribute.
- Using frame busters (JavaScript code) to prevent it being loaded in frame.

Cross site Request forgery (CSRF):

CSRF defense mechanisms are

1. Use built-in CSRF protection

2. for stateful application use the sync token pattern

3. for stateless application use double submit cookies

4. CSRF tokens should not be transmitted using cookies