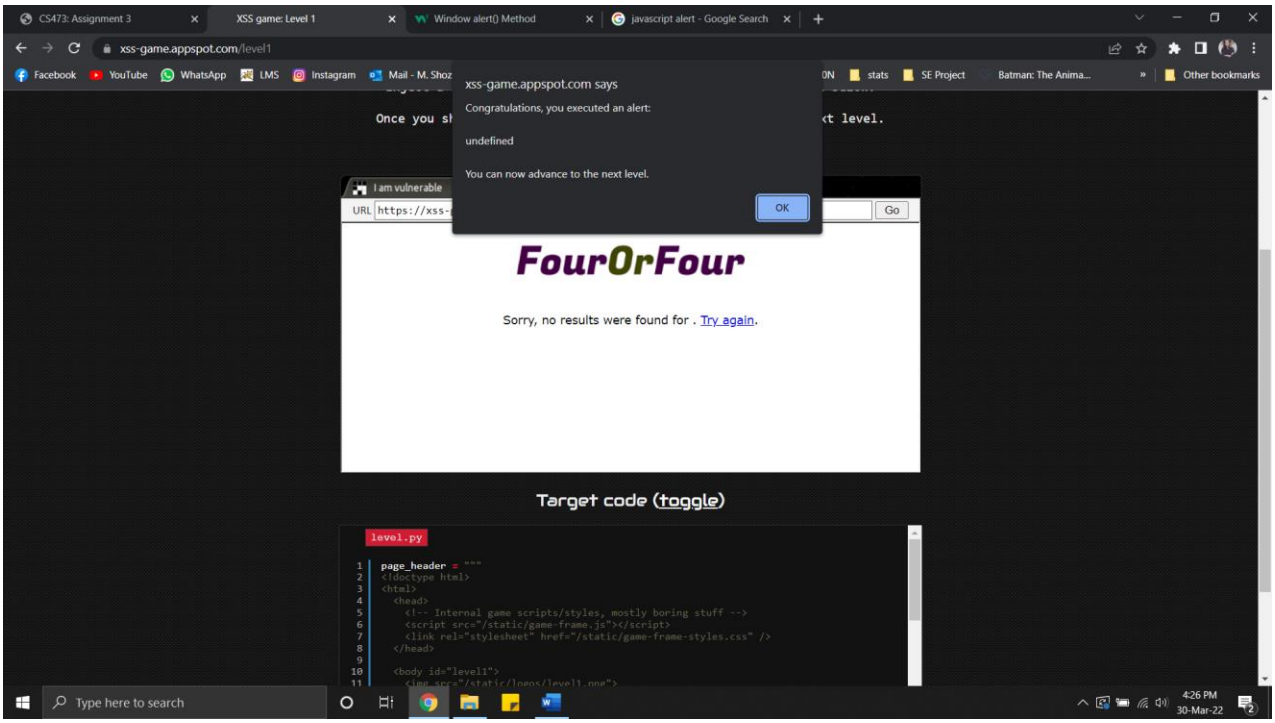
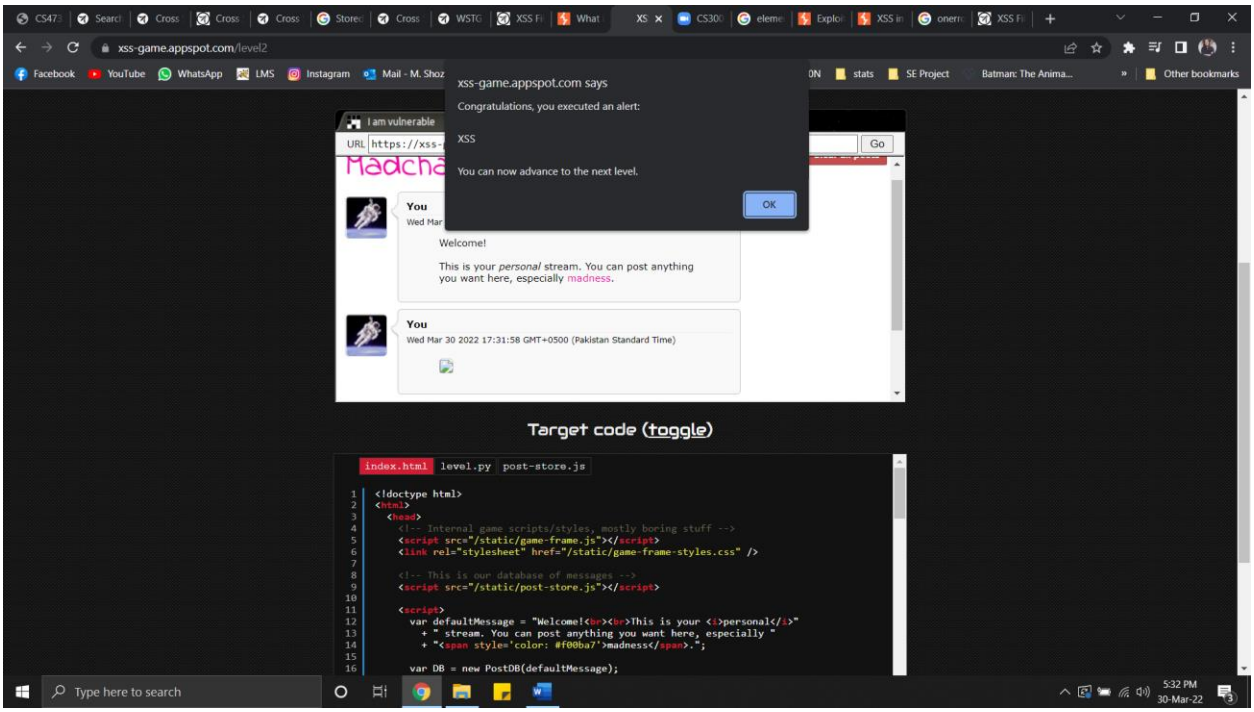
TASK 1

Level 1



I carried out the attack by providing the input **<script> alert () </script>** into the search bar. It is because anything in the script tag is interpreted as JavaScript. So, when the program will read the input from search bar and will encounter the script tags, it will have to execute whatever is in there. Since there was an alert code, the alert was popped up.

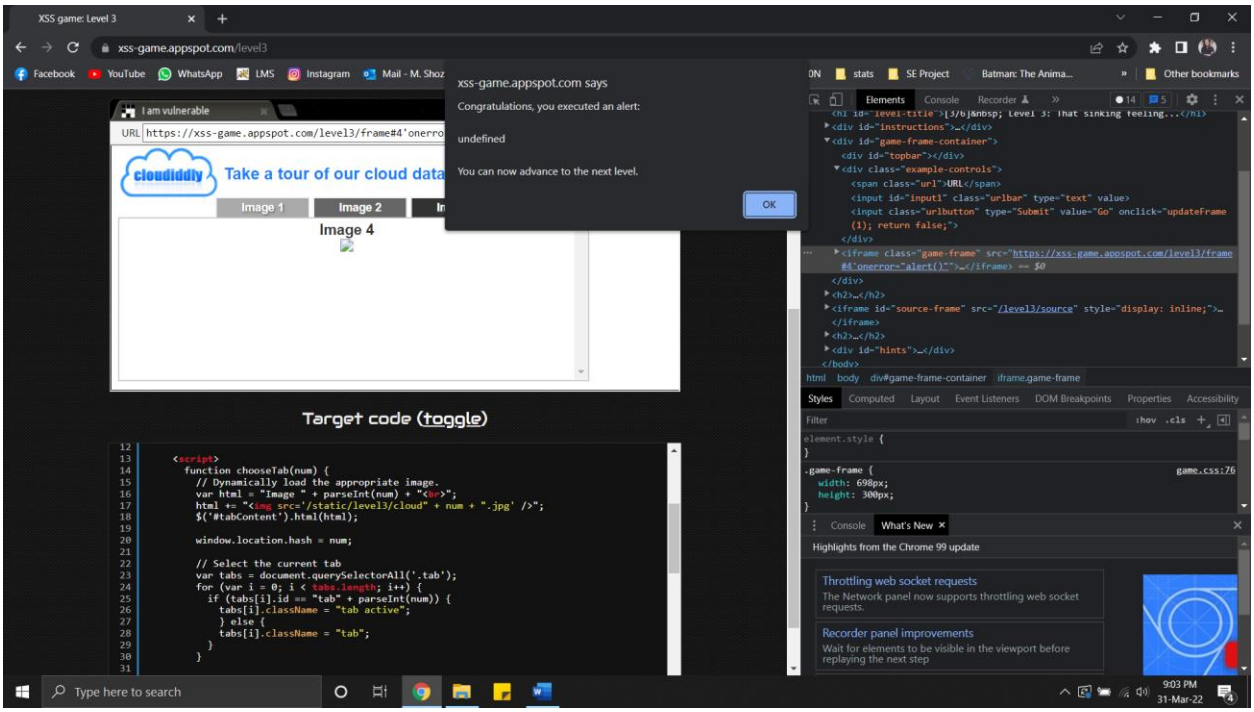
Level 2



I carried out the attack by viewing the hints which said that script tags will not work and suggested trying an element with a JavaScript attribute. It also mentioned img and onerror. So, I searched onerror alert in the XSS filter evasion cheat sheet and found the string that opened the alert. I understood that img is the element whereas the onerror is the JavaScript attribute in it. And then created my own input string. So now in this img tag, the src doesn't exist therefore, onerror is raised where we execute alert().

My input post was **<img src=/ onerror="alert('hello world')"></img>**

Level 3

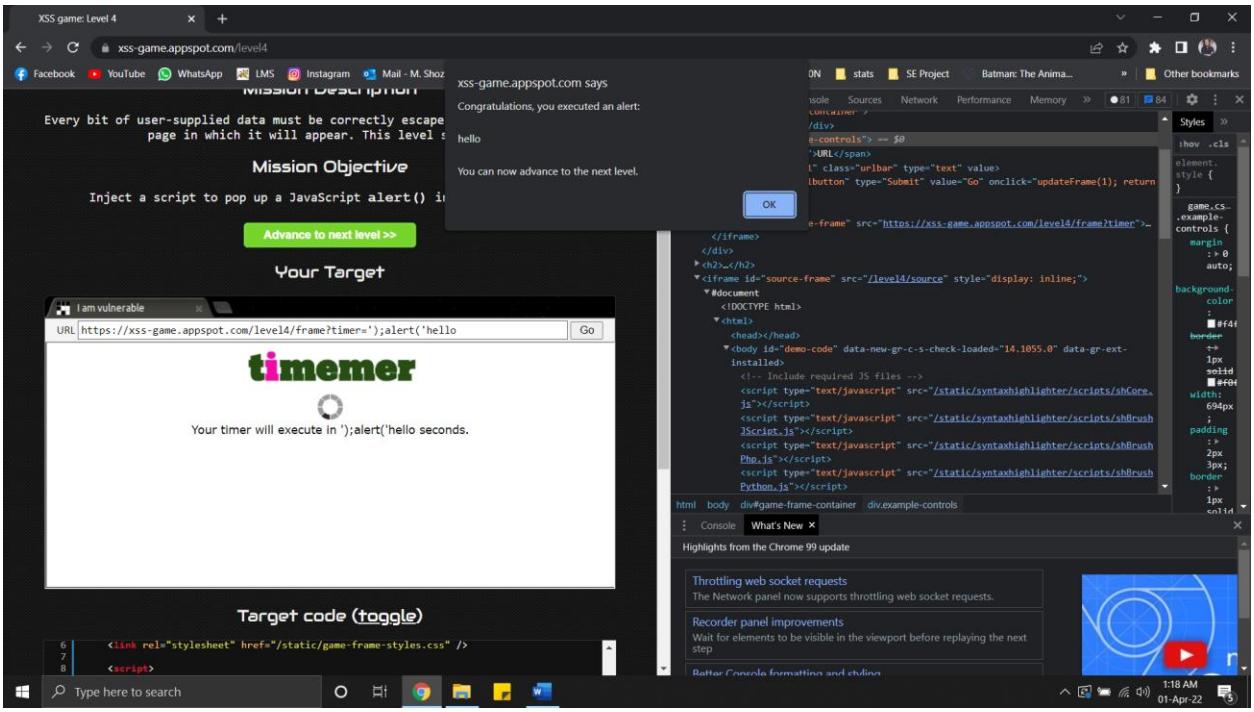


I first explored the application and noticed that when we change the picture by clicking on the image tabs, only the number of the picture changes in the whole url. Upon inspecting, I saw that this number was being input into the function choose tab. So it became apparent that whatever I write in place of the number in the URL, after the # will go into this choose tab function and then into the image tag in line 16. However, the parseInt function only takes the number and not a string so we need to escape it by using ‘. We also know that when the img tag raises onerror when the url is not correct so we can use the following input:

4’ onerror”alert()”

Now, since there is no image 4, the url in src of img tag will be wrong and hence onerror that is right after the src will be executed and since it executes an alert so an alert will pop up.

Level 4



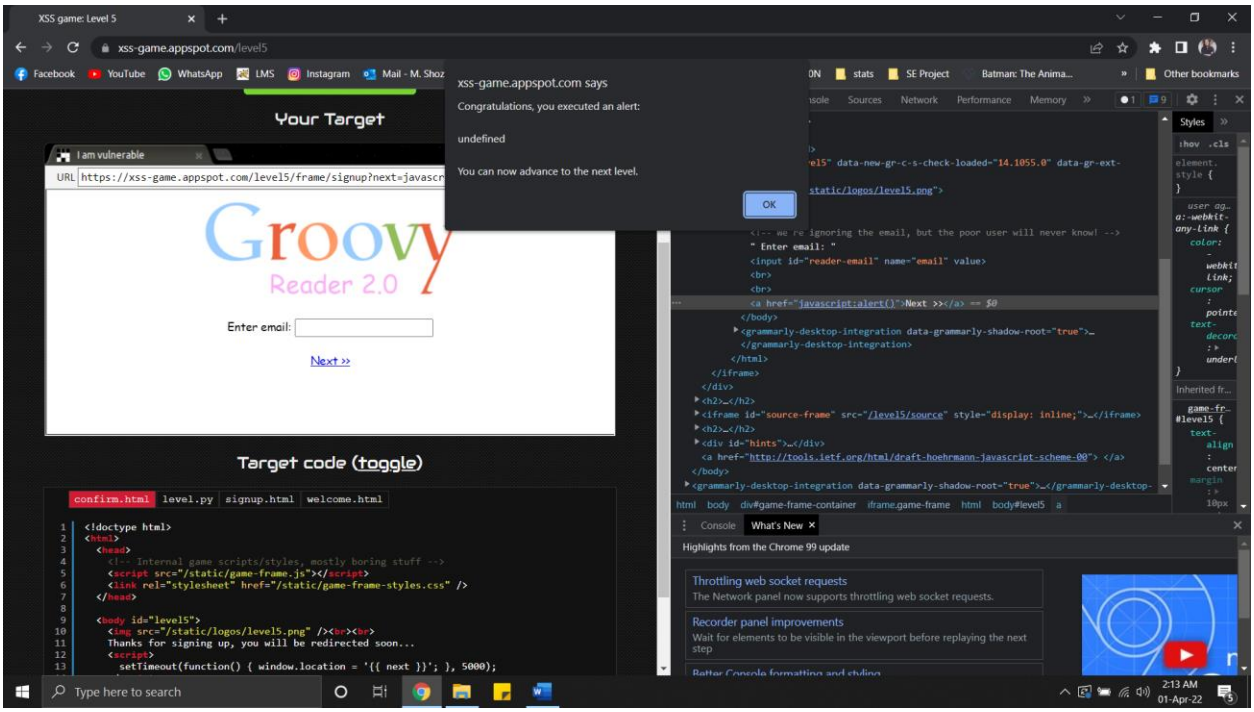
Upon inspecting the code, I found that there was an img tag where the onload was executing the function startTimer. The input to the start timer was coming from the URL. So I tried to bypass the startTimer function call by putting **');** Now after putting this, the onload looked something like this:

onload="startTimer('');');

Now to call another function in onload I just needed to append **alert(‘hello** after which the the onload looked like this onload="startTimer('');alert(‘hello');

So the input **');alert(‘hello** cracked the level 4

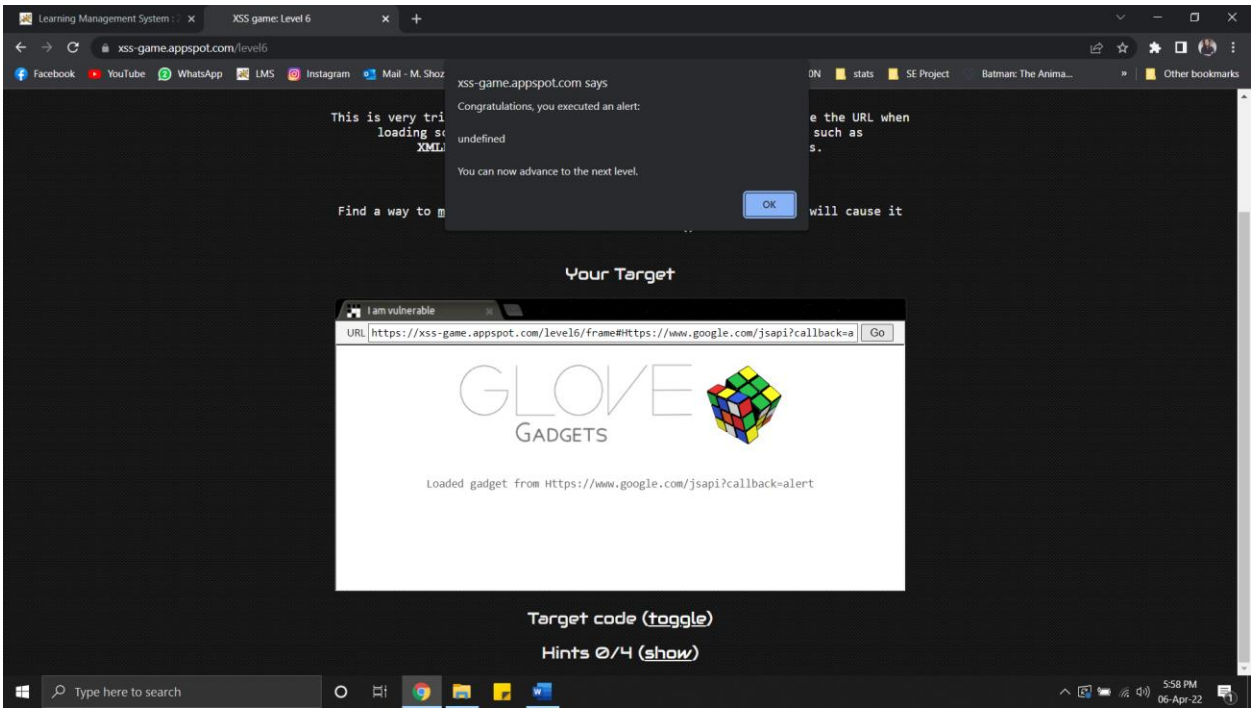
Level 5



Upon visiting the link in the last hint I got to know that we can put a javascript into a hyperlink through **... .** Then after inspecting the sign up button I noticed that the word **confirm** was being stored in the variable next and then that variable was being passed directly as the hyperlink in the sign up frame. Hence when I did **next=javascript:alert()** then **javascript:alert()** was passed into the next hyperlink in the sign up frame but since it was a javascript, it got executed and alert popped up.

My input was: **https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert()**

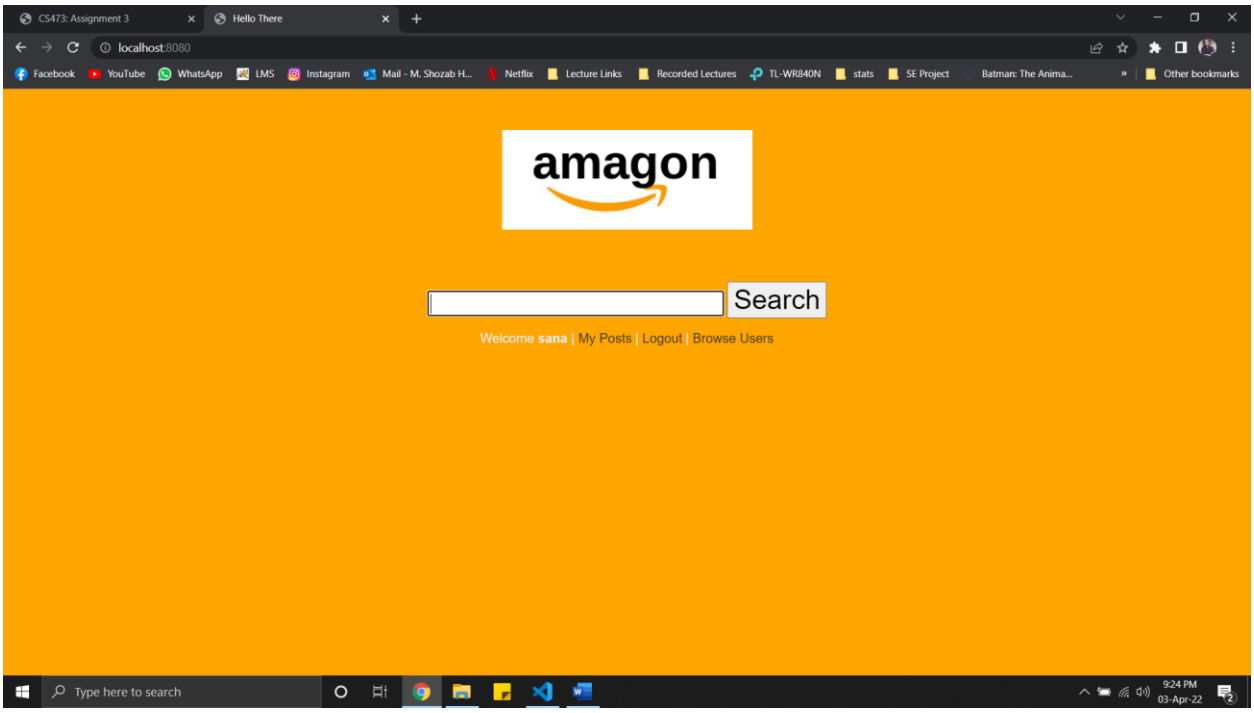
Level 6



Upon viewing the hints, I understood that anything after the # is the URL of the loaded script and that google.com/jsapi hosts js files. So if there is google.com/jsapi?calcallback=foo then there must also be google.com/jsapi?calcallback=alert which opens alert().
I tried the input https://www.google.com/jsapi?callback=alert but it didn't work because the application wont allow http. However, regex is case sensitive this means that according to regex http and Http are not same so I tried **Https://www.google.com/jsapi?callback=alert** and it worked and alert popped up.

TASK 2

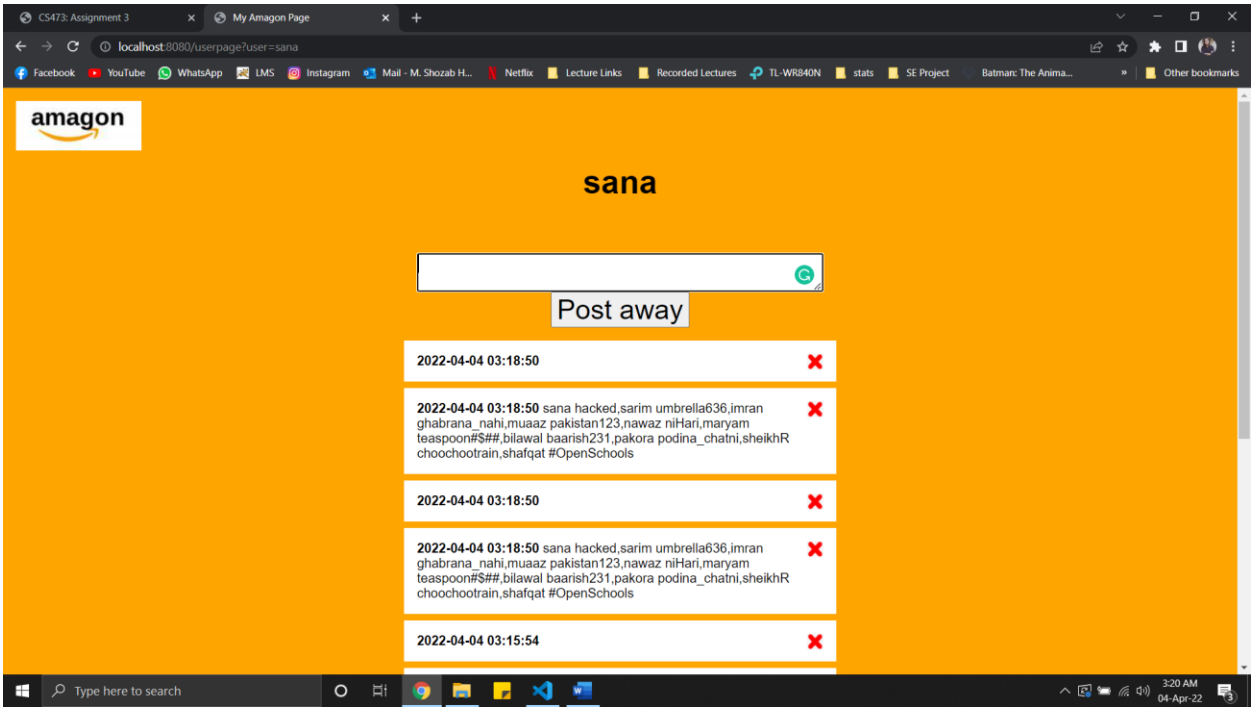
Part1



Input: **'; UPDATE accounts SET password='hacked' WHERE username='sana**

I observed this behavior because there was no check on user input hence, I used **'** to escape then **;** to end the previous query and then **UPDATE accounts SET password='hacked' WHERE username='sana** to update password. This was a malicious query that was placed and thus executed.

Part2:



Input: **',datetime('now', 'localtime'));INSERT INTO posts VALUES ('sana', (SELECT Group_concat(username||' '||password) FROM accounts), datetime('now', 'localtime'));--**

In line 124 of the code, we can see that whatever the user post away, is saved into the post variable which is then inserted into the posts table. So, I needed to send username passwords instead of post into the insert query. I first completed the default query by **',datetime('now', 'localtime'));** Then I made a similar query next to it but this time instead of post, user credentials were inserted into posts table. I got credentials by the following query **(SELECT Group_concat(username||' '||password) FROM accounts)** Now to ignore the remaining part of the default query I commented it out by using **–**

TASK3

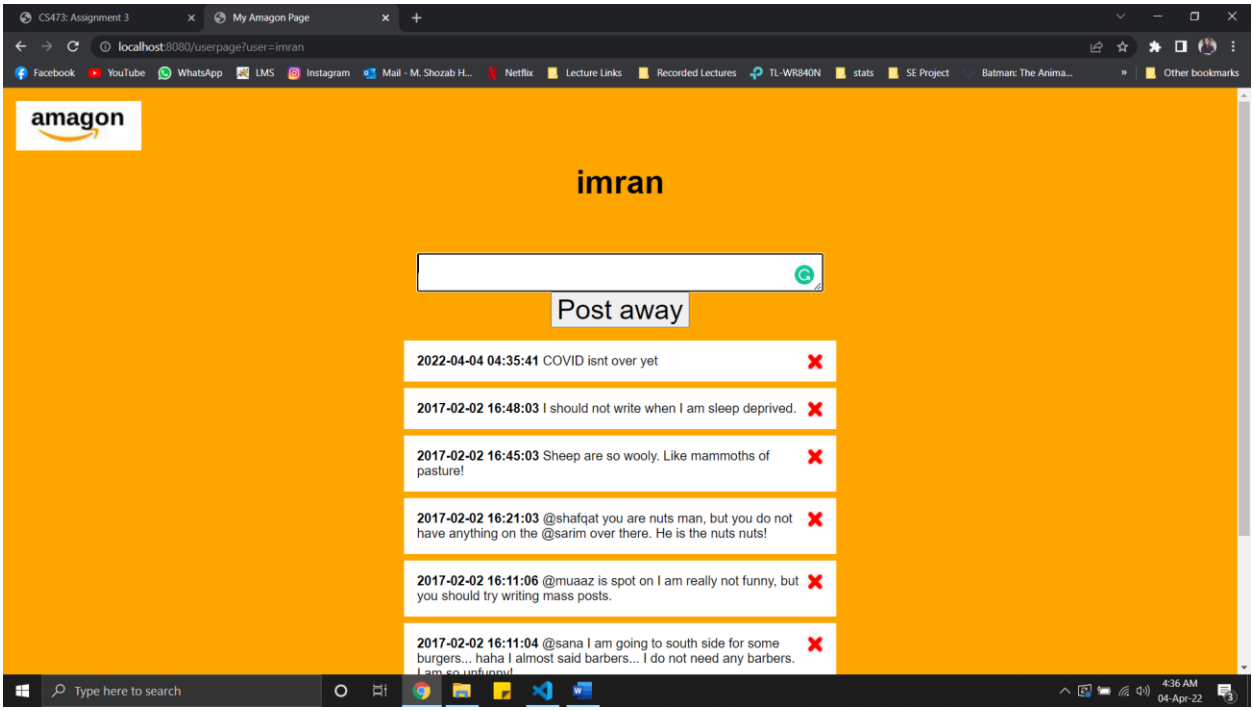
Part1:

Input: **You can buy tickets from here www.booktickets.com**

After checking the logs of webserver, I observed that **/logout?redirect=/** is received whenever a user logs out so I appended it to base URL and then posted it as a hyper link tag so that whenever the user will click on the hyperlink, they will unintentionally generate a log out request to server and will be logged out.

Assuming that the server will run on port 8080.

Part2:

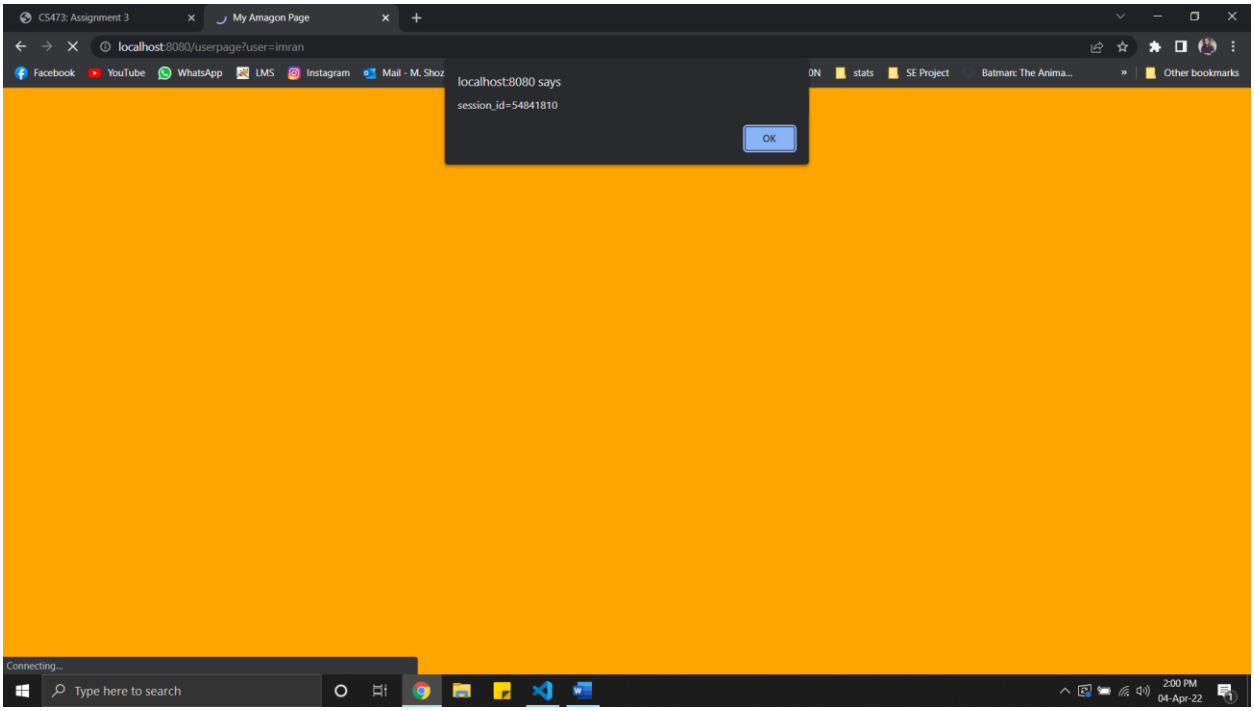


Input: **Win free iPhone: www.iphonegiveaway.com**

Just like in part 1, I observed that whenever a post is made the following request is send to the server: **do_post?redirect=%2Fuserpage%3Fuser%3Dnawaz&post=COVID isnt over yet%3C%2Fa%3E**

Here, whatever the post variable will contain, it will be posted on user's timeline. So that's why when we put post=COVID isn't over yet. Hence, when the user clicks on hyperlink, the above request will be sent to server automatically and post will be made without their consent.

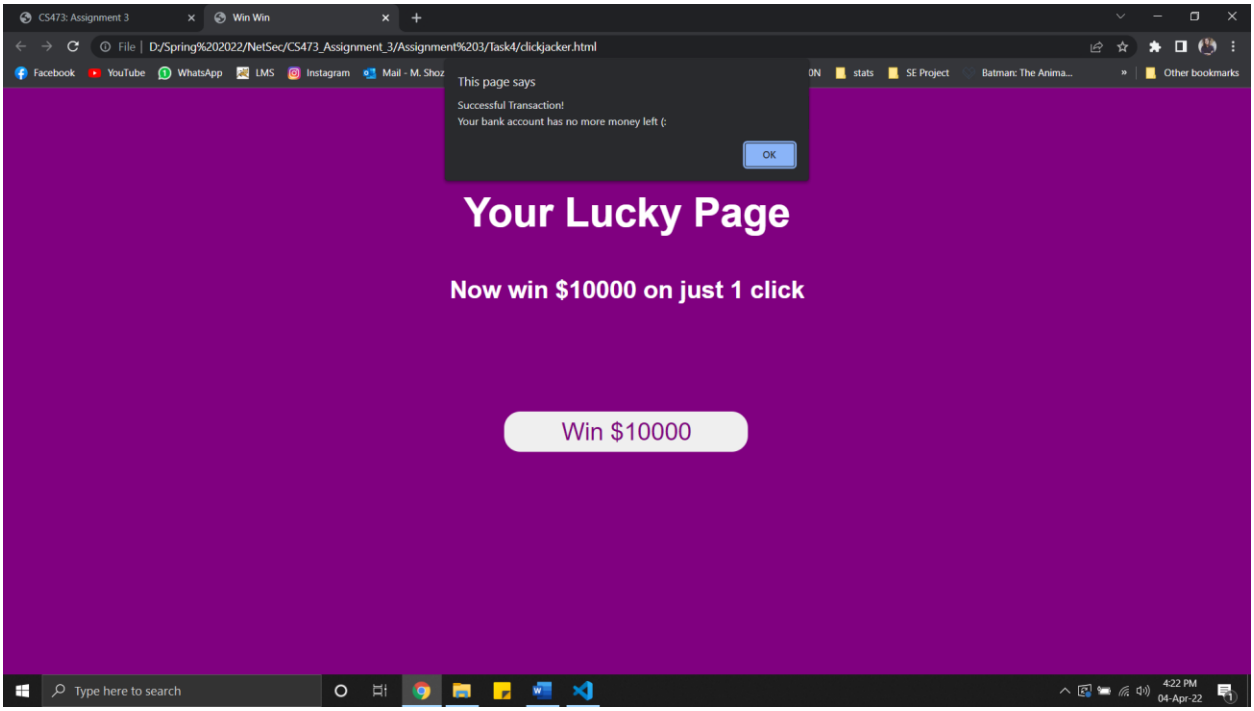
Part 3:



Input: **Download Game of Thrones for free: <a href="http://localhost:8080/do_post?redirect=%2Fuserpage%3Fuser%3Dnawaz&post=<script>alert(document.cookie)</script>%3C%2Fa%3E">www.getfreegot.com**

This part is very similar to part2, we know that whenever the timeline page will refresh or will be load, then all the latest 10 posts of the users will be printed and since there is not check, so if a post contains a script, then that script will also execute. So, I needed to make the user post the script unintentionally. From part2 I know how to make a user post something unintentionally therefore, I used same malicious string as part2 but this time in the post variable, instead of passing a normal string, I passed a script into it. Since, the alert box with cookie was required thus, the script was <script> alert(document.cookie) </script>. Hence when the user will click on the hyperlink a post containing this script will automatically be posted on their timeline. Afterwards, whenever their timeline page will be refreshed, the script will be executed.

TASK4:



At first I put the iframe tag **<iframe src="./dummy_hbl.html" frameborder="0" id="iframe"></iframe>** so that iframe can be shown on the web page. Here id is used to refer to the container, where we have done styling of the iframe through css. Frameborder was by default and is used to make the iframe borderless.

Then in the container I adjusted the width, height, top and left margin of the iframe so that the proceed button can exactly layover the win button. Z-index was specified 2 because since the body had z-index of 1 so we had to give a higher z-index value to this iframe so that it could be infront of the body and proceed button gets clicked instead of win button. Opacity is 0 so that the hbl page.