WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Tejeswar Reddy Nalijeni

Email: nalijety@mail.uc.edu

Short-bio: Currently I am pursuing Master's in Information Technology and I am interested in Cybersecurity

Hackathon 1

The Hackathon overview

This Hackathon is all about learning how to stop hackers from using a sneaky trick called Cross-Site Scripting (XSS) to mess with websites. In this hackathon, I looked for weak spots in code, follow some safety rules from OWASP, and make sure our websites are safe from these attacks. The lab has two main parts(Task 1 and Task 2). In first task, I found out different attack levels on a website called http://waph hackathon.eastus.cloudapp.azure.com/xss/ from 0 to 6 levels. Then, I learnt how to make our websites stronger by checking. After each part, we'll make README.md file of what I have done in simple way and make a PDF report using a tool called Pandoc.

https://github.com/nalijety/waph-nalijety/tree/main/Hackathon/Hackathon 1.

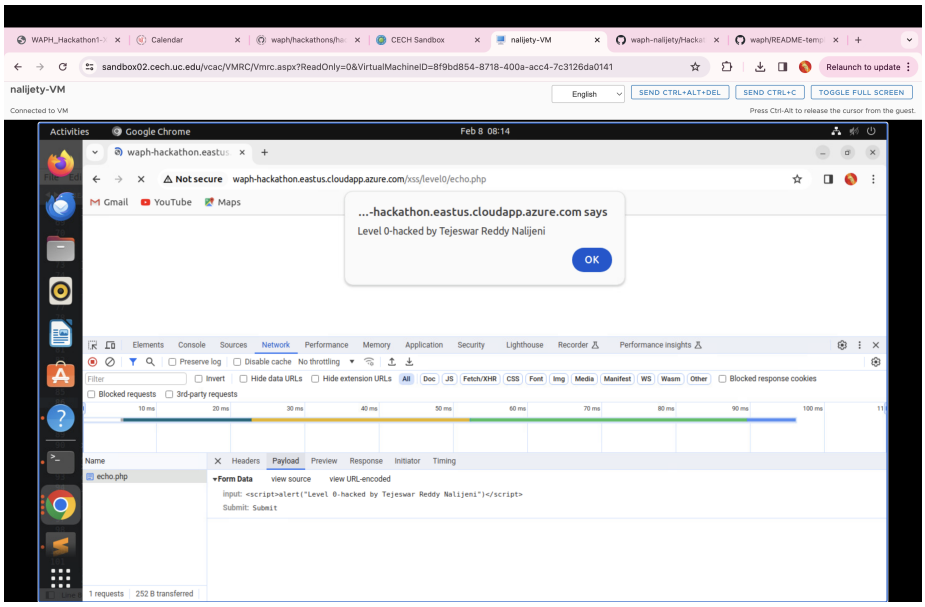
Task 1: Attacks

Level 0:

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

```
<script>alert ("Level 0-hacked by Tejeswar Reddy Nalijeni")</script>
```

The above script is injected in the input. The below screenshot shows the pop-up alert for the Level-0 attack.

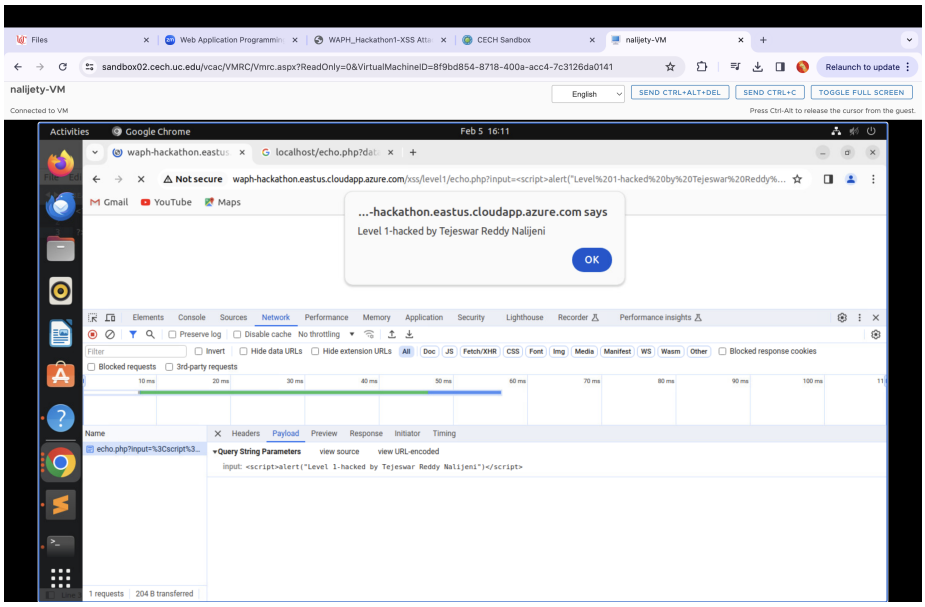


Level 1:

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

```
<script>alert("Level 1-hacked by Tejeswar Reddy Nalijeni")</script>
```

The above script is used as pathvariable for attacking. The below screenshot shows the pop-up alert for the Level-1 attack.



Level 2:

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

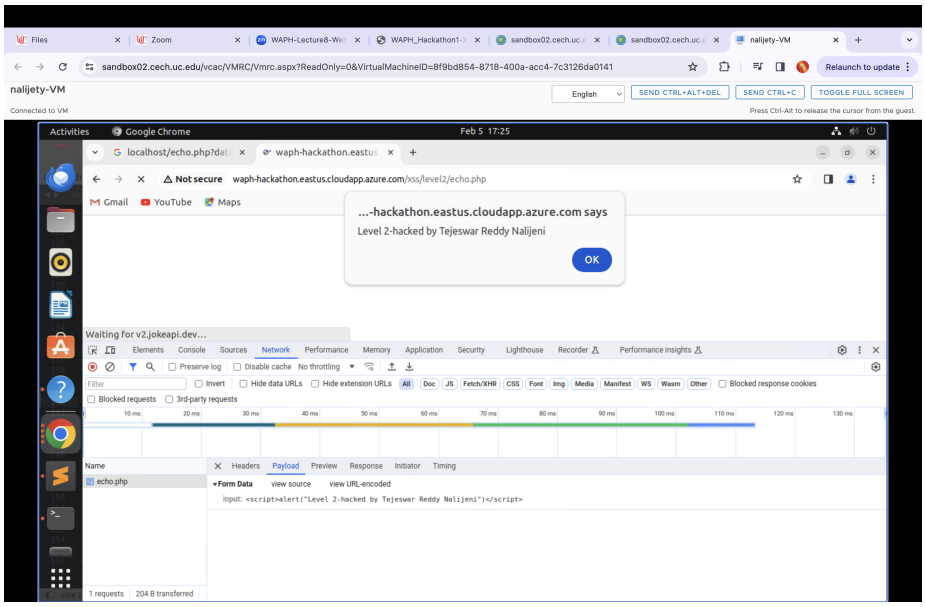
```
<script>alert("Level 2-hacked by Teieswar Reddy Naliieni")</script>
```

In this the URL is given in the POST form in HTML file and then the above script is used for attacking using the POST input field with HTTP request as it has no path variable.

Source code guess:

```
<?php
if($_SERVER['REQUEST METHOD']==='POST'{
echo $_POST"input"];
}
else{
echo "{\"error!": \"Please provide 'input' field in an HTTP POST Request\"}";
}
?>
```

The below screenshot shows the pop-up alert for the Level-2 attack.



Level 3:

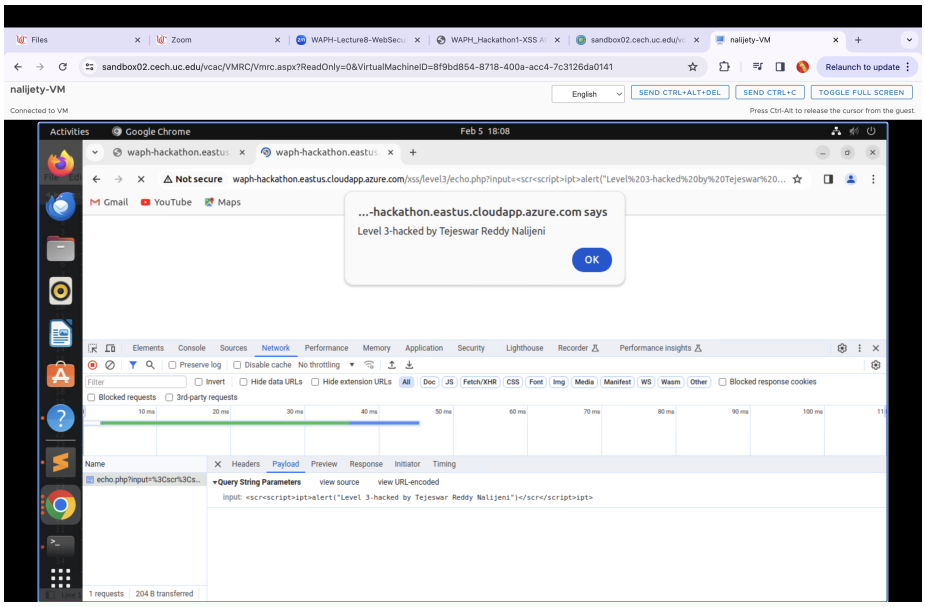
URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php

```
<scr<script>ipt>alert("Level 3-hacked by Tejeswar Reddy Nalijeni")</scr/script>ipt>
```

Source code guess:

```
<?php
    if(strpos($_REQUEST["input"])){
        $filteredData = preg_replace('/<\/?script\b[^>]*>/','',$_REQUEST["input"]);
        echo $filteredData;
    }
    else{
        echo "{\"error\": \"Please provide 'input' field\"}";
    }
?>
```

In this when the input is given, then it filters and passes the attack script. The below screenshot shows the pop-up alert for the Level-3 attack.



Level 4:

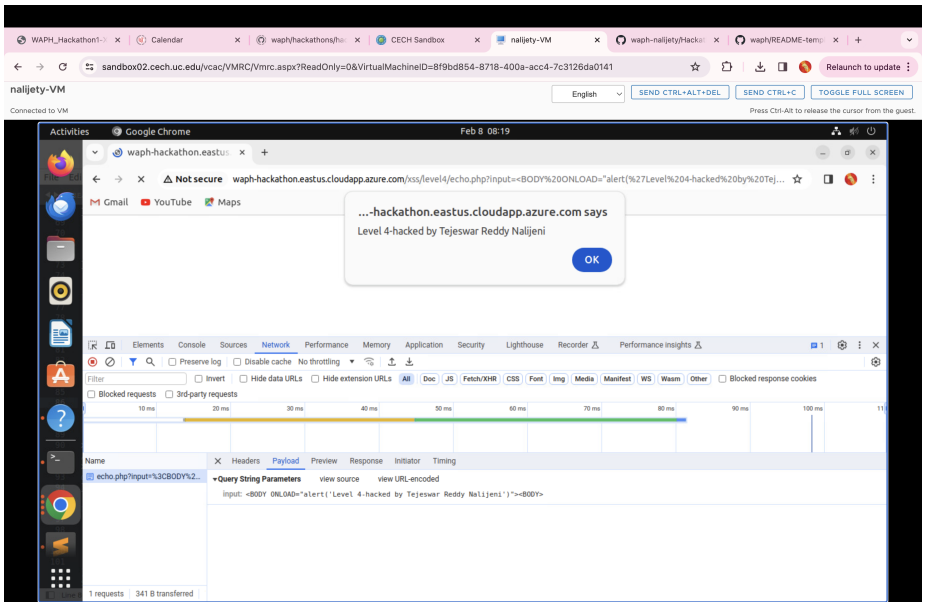
URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php

```
<BODY ONLOAD="alert('Level 4-hacked by Tejeswar Reddy Nalijeni')"></BODY>
```

Source code guess:

```
<?php
    if(isset($_REQUEST['input'])){
        if(strpos($_REQUEST["input"],"script")!==false){
            echo "{\"error\": \"No 'script' is allowed!\"}";
        }
        else{
            echo $_REQUEST['input'];
        }
    }
    else{
        echo "{\"error\": \"Please provide 'input' field\"}";
    }
?>
```

In this, it filters the script tag even if it pass after breaking. The below screenshot shows the pop-up alert for the Level-4 attack.



Level 5:

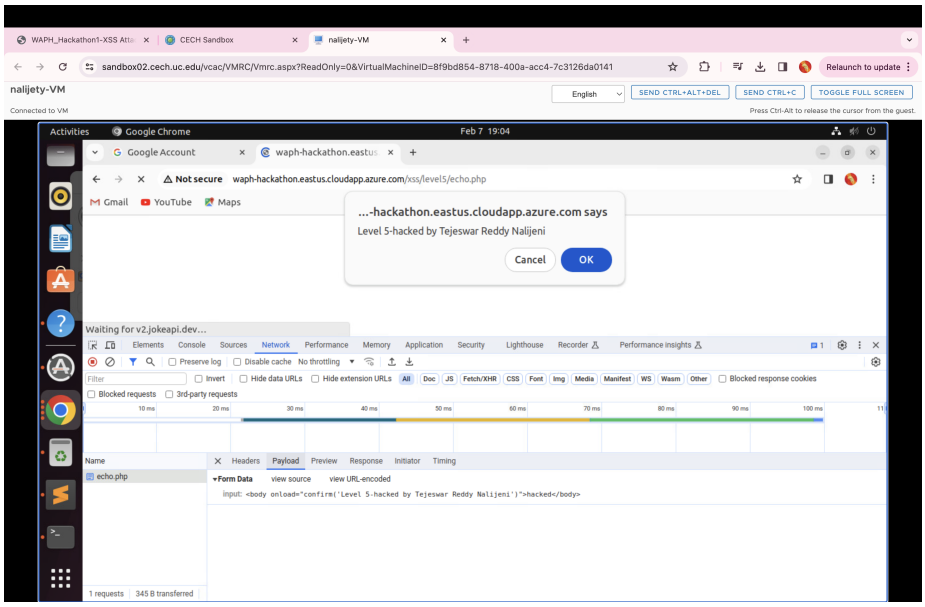
URL: http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php

```
<body onload="confirm('Level 5-hacked by Tejeswar Reddy Naliieni')">hacked</body>
```

Source code guess:

```
<?php
    if(isset($_REQUEST['input'])){
        if(strpos($_REQUEST["input"],"script")!==false){
            echo "{\"error\": \"No 'script' is allowed!\"}";
        }
        elseif (strpos($_REQUEST["input"],"alert")!==false) {
            echo "{\"error\": \"No 'alert' is allowed!\"}";
        }
        else{
            echo $_REQUEST['input'];
        }
    }
    else{
        echo "{\"error\": \"Please provide 'input' field\"}";
    }
?>
```

In this, the script tag and alert needs to be filtered. I used the above script in the HTML file and got the pop-up alert as shown in the below screenshot.



Level 6:

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

`"/xss/level6/echo.php/" onkeyup="alert('Level 6-hacked by Tejeswar Reddy Nalijeni')"`

Source code guess:

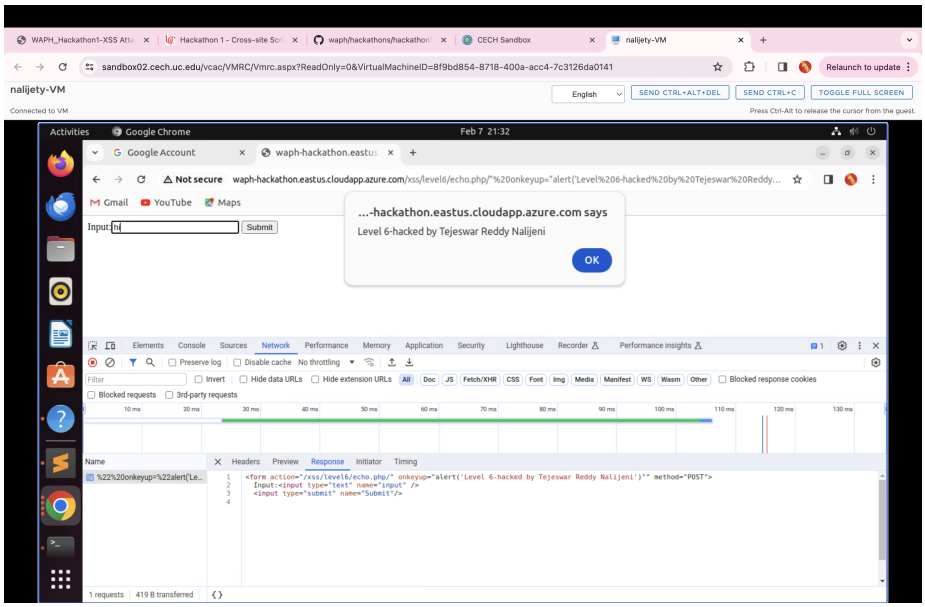
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title></title>
</head>
<body>

<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $inputValue = isset($_POST['input']) ? $_POST['input'] : '';
    echo htmlspecialchars($inputValue);
}
?>

<form method="post" action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>">
    <label for="input">Input:</label>
    <input type="text" id="input" name="input" value="<?php echo htmlspecialchars($inputValu
    <button type="submit">Submit</button>
```

```
</form>
</body>
</html>
```

At this level, the input undergoes processing like using the htmlentities() method, which converts applicable characters to their corresponding HTML entities. This ensures that user input is displayed strictly as text on the webpage. Triggering an alert on a webpage under these conditions can be accomplished using JavaScript event listeners such as onclick(), or onkeyup(). In this case, the onkeyup() event listener is utilized, causing an alert to appear on the webpage whenever a key is pressed in the input field s shown in the below screenshot.



Task 2: Defense

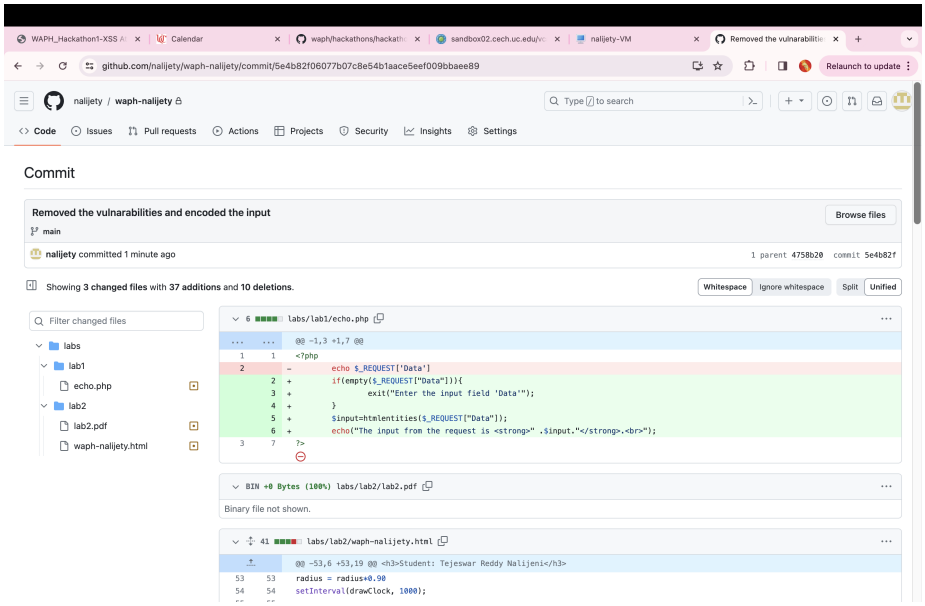
a. Input Validation Implementation for Echo.php file:

In Lab 1, modifications were made to the echo.php file, introducing input validation and XSS defense measures. The process begins with a check for empty input, halting PHP execution if the condition is met. Valid input is then displayed on the webpage as text only. This is accomplished using the htmlentities() method, ensuring the input is sanitized and converted to appropriate HTML characters. The addition of input validation enhances the security of the application by preventing harmful input from being processed.

```
if(empty($ REQUEST["Data"'])){
exit("Enter the input field 'Data'");
}
$input=htmlentities($_REQUEST["Data"]);
```

```
echo("The input from the request is <strong>" .$input."</strong>.<br>");
```

The below screenshot shows the committed changes in the “echo.php” file.

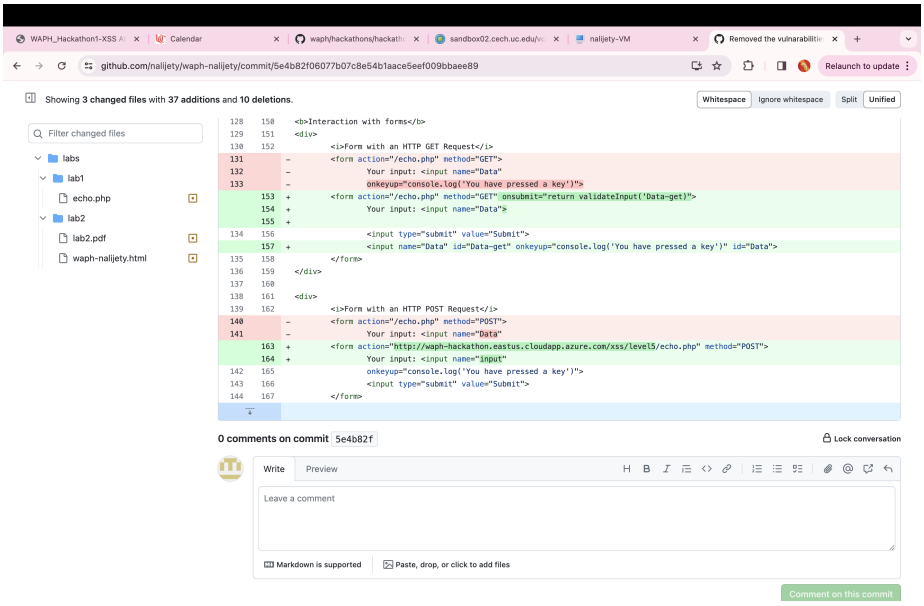


b. Input Validation Implementation for HTML file:

The waph-nalijety.html code had been modified, during which all external input points were carefully identified. Each of these input sources underwent thorough verification, and subsequent measures were taken to ensure that the output texts were thoroughly cleansed and sanitized. This meticulous approach aimed to enhance the security and reliability of the code, minimizing the potential for vulnerabilities or malicious exploitation.

1. HTTP GET and POST request: Both HTTP GET and POST request forms now undergo thorough accuracy checks for input data. The implementation of a new function named validateInput() mandates that users input text before proceeding with their request execution. The below code is the modified change in the HTML file.

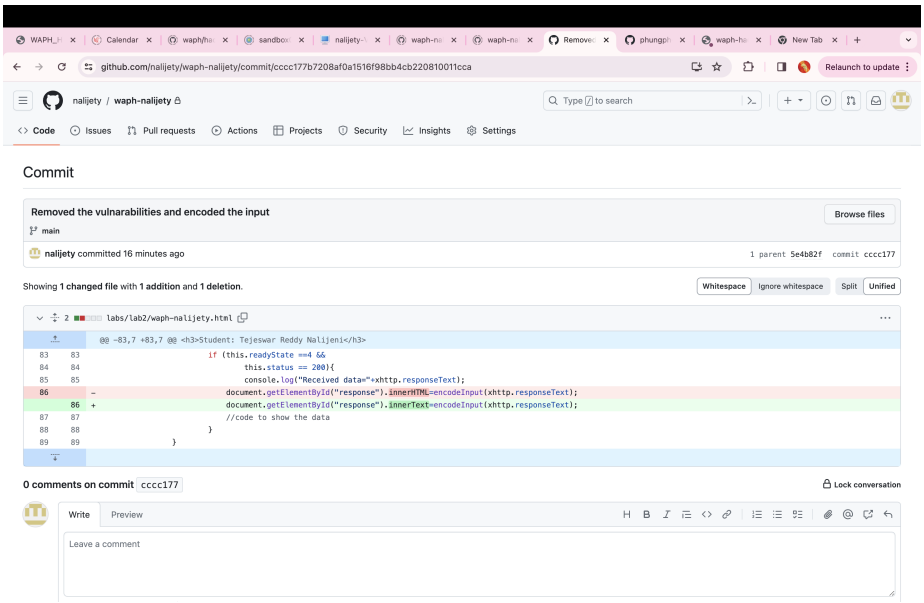
```
<form action="/echo.php" method="GET" onsubmit="return validateInput('Data-get)"
Your input: <input name="Data"s
<input type="submit" value="Submit">
<input name="Data" id="Data-get" onkeyup="console.log('You have pressed a key')" id="Data">
```



2. innerHTML to innerText: In this where HTML rendering serves no purpose and only plain text is required, the code has been adjusted to replace .innerHTML with .innerText. This modification ensures that content presentation is simplified and minimizes the risk of potential security vulnerabilities.

```
document. getElementById("response").innerText=encodeInput(http.responseText);
```

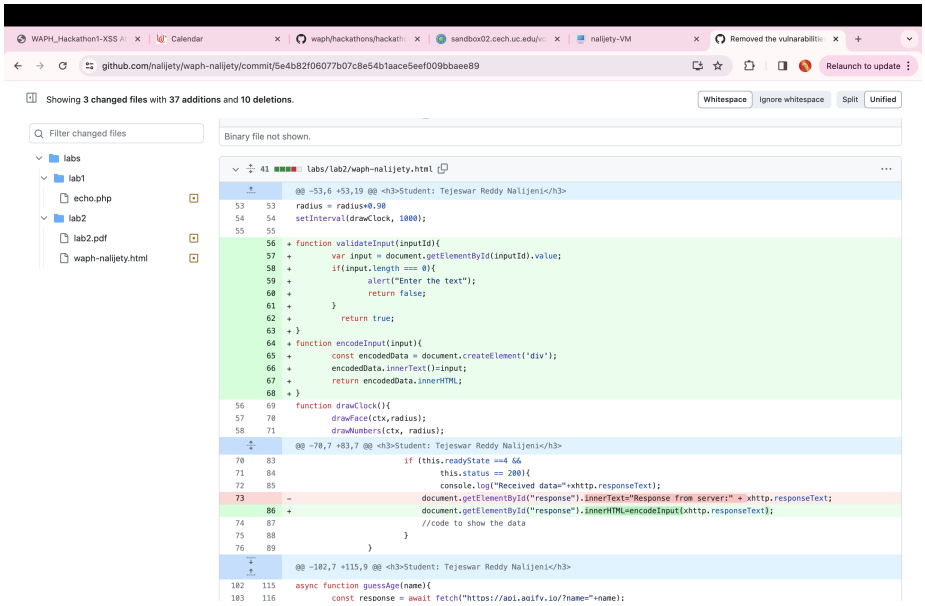
The below screenshot shows the commited change for HTML file.



3. Encode and validate: To address concerns regarding cross-site scripting attacks, a new function called encodeInput() has been introduced. This function sanitizes responses by converting special characters into the appropriate HTML entities before insertion into the HTML document. By rendering the content non-executable and strictly textual, the application's defense against XSS attacks is significantly strengthened.

```
function validateInput (inputId) {
var input = document.getElementById(inputId).value;
if(input. length === 0){
alert ("Enter the text");
return false;
}
return true;
}
function encodeInput (input) {
const encodedData = document. createElement ('div');
encodedData.innerText ()=input;
return encodedData. innerHTML;
```

The below screenshot shows the commited change for encode and validation.



4. Joke API and Guess age API: Validation measures have been integrated to enhance the reliability of interactions with the API endpoint https://v2.jokeapi.dev/joke/Programming?type=single. These validations meticulously inspect both the received result and the result.joke fields within the JSON response to ensure they are not empty. In cases where null values are encountered, an error message is promptly generated, thereby improving the

application's handling of unexpected scenarios and ensuring a smoother user experience. The asynchronous function 'guessAge()' ensures that the received result is not empty or zero. Additionally, it validates that the input data is neither null nor empty. In both scenarios, an error message is displayed to alert the user. This proactive approach enhances the reliability of the application by preventing potential issues associated with empty or invalid data.

```
$("#response").html("A programming joke of the day: " + result. joke);
if(result && result. joke) {
var encodedJoke = encodeInput(result. joke);
§("#response").text("Programming joke for the day: " + encodedJoke);
elsef $("#response").text ("Unable to get a joke at the moment.");
});
```

```
if(result.age==null || result.age==0)
return $("#response").text("Sorry, the server cannot get your age");
$("#response" ).text("Hi! " + name + ", your age should be" + result.age);
```

The below screenshot shows the commited change for joke API and Guess age API.