waph-peddinry

WAPH-WEB APPLICATION PROGRAMMING AND HACKING

instructor : Dr.Phu Phung

Name: Narendra Reddy Peddireddy **Email**: peddirny@mail.uc.edu

Repository's URL: https://github.com/narendra2105/waph-peddinry.git

Narendra Reddy Peddireddy uses this private repository to house all of the course's code. The following is how this repository is organized.

Hackathon1 :Attacks and Defenses Against Cross-Site Scripting

Overview of hackathon_1

The goal of this hackathon is to learn about and strengthen defenses against Cross-Site Scripting (XSS) attacks by investigating code vulnerabilities and following OWASP principles. One job is to attack a website that has six levels of cross-site scripting (XSS) vulnerabilities; the other aim is to employ secure coding techniques, such as input validation and output sanitization, to mitigate XSS attacks. After everything was finished, a markdown compilation of the documentation was made, and a PDF report was produced using the Pandoc application. You may find the project's GitHub repository at https://github.com/narendra2105/waph-peddinry/blob/main/Hackathons/Hackathon1/README.md.

Hands-on hacking exercises

Task_1: Attacks:

Level_0

link : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php
assaulting code:

Figure 1: Narendra Reddy Peddireddy Headshot

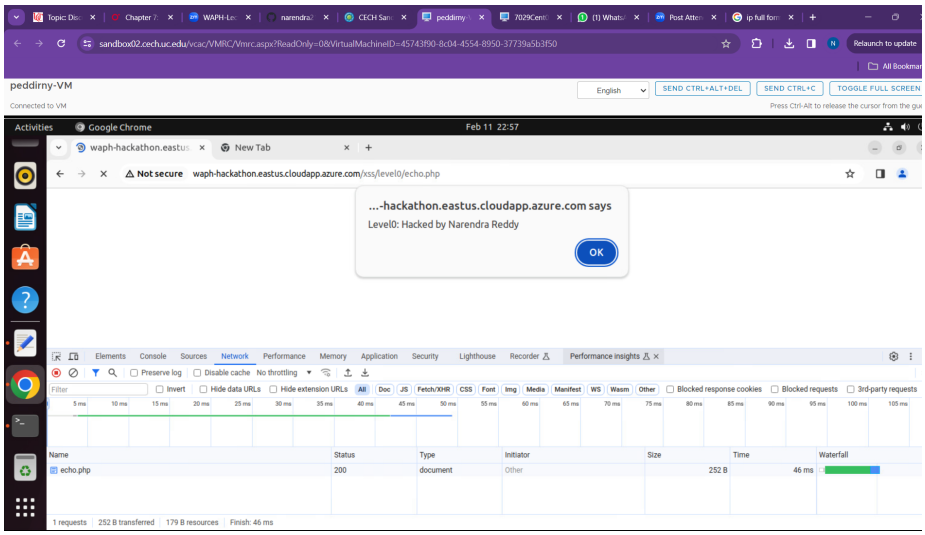
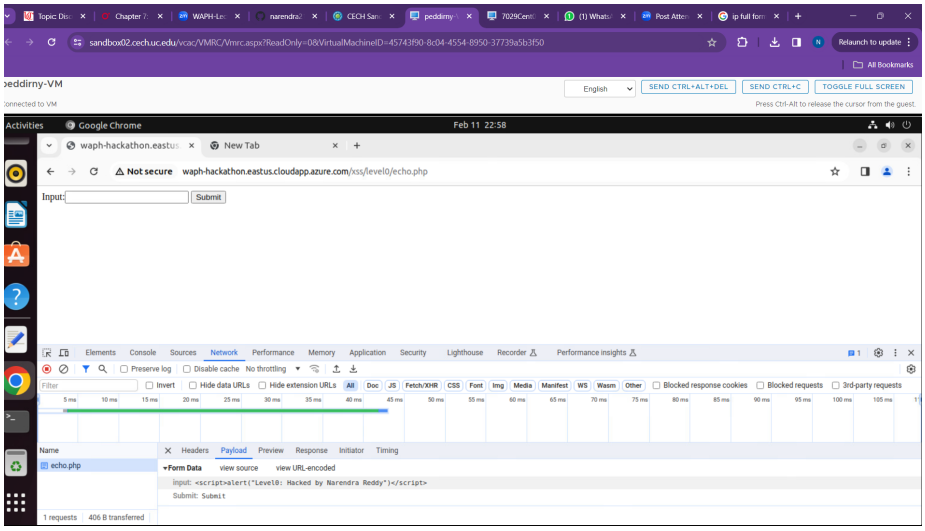
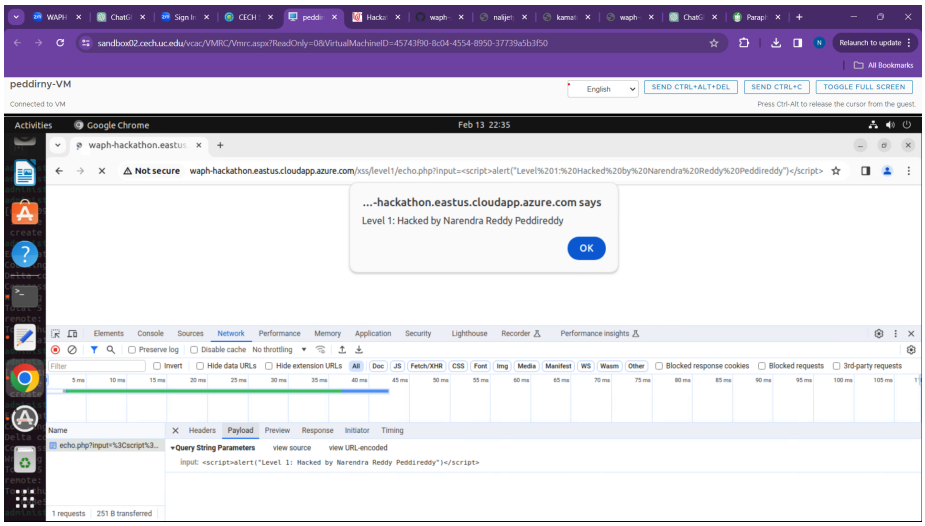


Figure 2: Level0



Level_1

link : http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php assaulting code:’’’



’’’

Level_2

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

The URL has been translated to a simple HTML form without any input fields or path variables, and a POST request sent through the form is used to inject the malicious script.

assaulting code:’’’

’’’

Estimated source code:’’’

Level_3

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

assaulting code:’’’?input=<script

ipt>’’’

Estimated source code: ’’‘ tags from the input $filteredData = preg_replace(’/<script[^>]>(.?)</script>/is’, ’‘, $input); echo $filteredData; } else { echo “{"error": "Please provide ‘input’ field"}”; } ?>”’

Level_4

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

Because the coding forbids using the

tag at all, the XSS script is executed by using the tag with the onerror property, which gets over the filtering method.

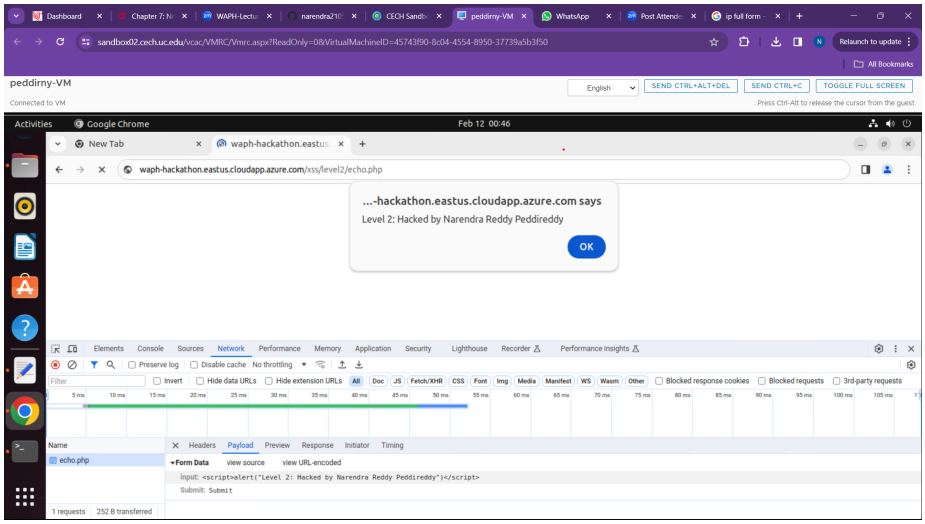


Figure 3: Level2

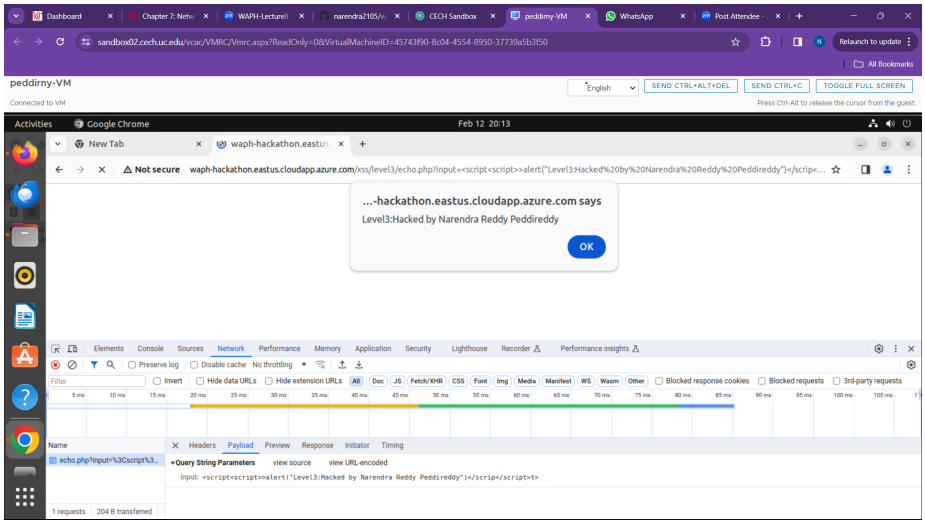


Figure 4: Level_3

assaulting code:’ ’‘?input=<img%20src=“P” onerror=“alert(Level4:Hacked by Narendra Reddy Peddireddy)”>” ’

Estimated source code: ’ ’‘” ’

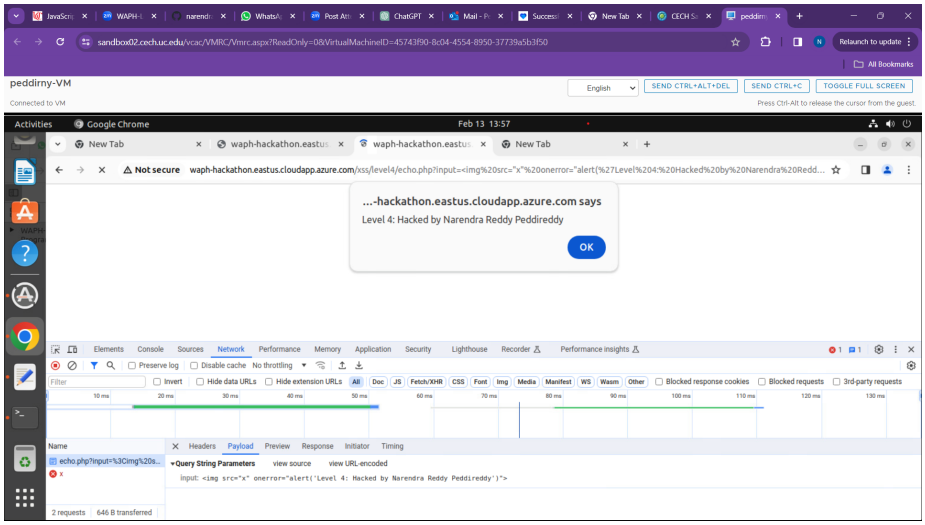


Figure 5: Level_4

Level_5

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

In order to get around the code’s limitations, a popup alert is triggered by utilizing the onerror() function of the tag in conjunction with Unicode encoding to overcome the filtering of both

tags and alert() methods in this level.

assaulting code:’ ’‘?input=” ’

Estimated source code:

’ ’‘” ’

Level_6

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

At this point, the input is processed using htmlentities(), which converts relevant characters to the appropriate HTML entities and guarantees that user input is only shown as text on the page. Despite this processing, JavaScript event listeners like onclick() and onkeyup() are used to generate an alert. More specifically, an alarm is raised each time a key is pressed in the input field by utilizing the

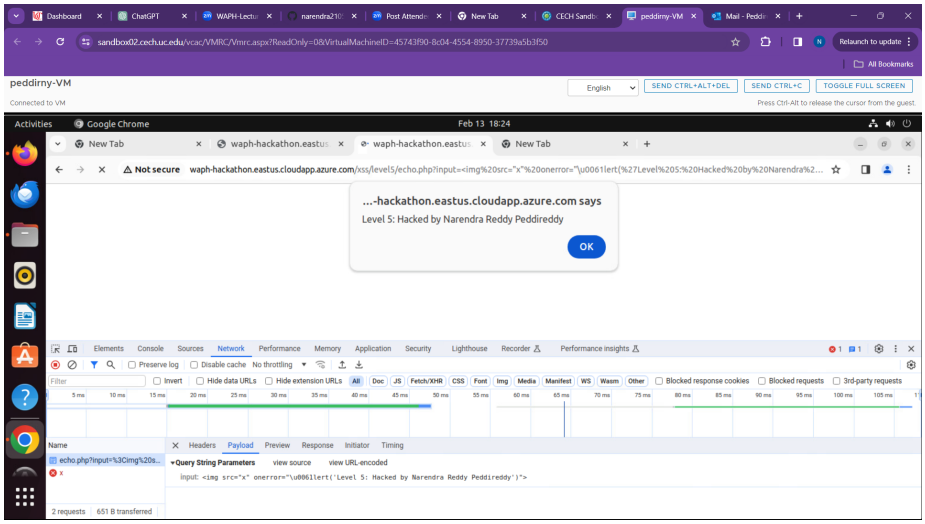


Figure 6: Level_5

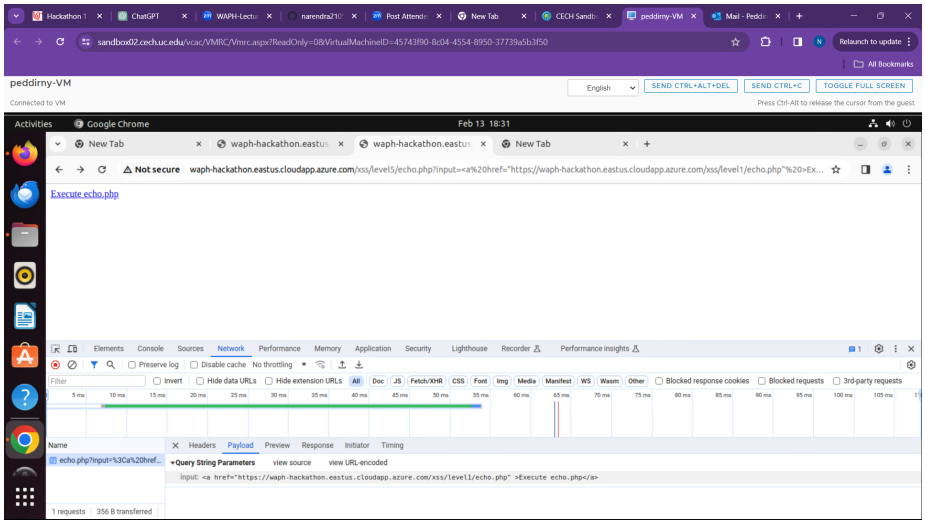


Figure 7: Level5_using link tag

onkeyup() event listener. An XSS attack can be carried out by changing the form tag within the URL. This will change the behavior of the form to include the script required to trigger the alert, avoiding the htmlentities() processing.

assaulting code:’ ’‘/" onkeyup=“alert(‘Level 6:Hacked by Narendra Reddy Peddireddy’)” ” ’

Estimated source code: ’ ’‘” ’

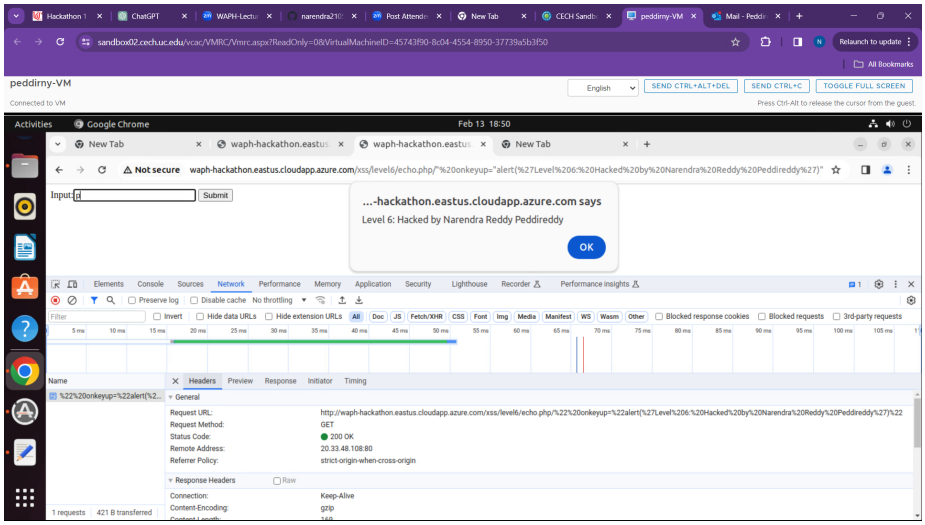


Figure 8: Level_6

Task_2:Defenses:

A echo.php from lab1

The echo.php file was modified in Lab 1 to provide XSS protection and input validation. The script first looks for empty input; if it finds any, PHP execution is stopped. The htmlentities() method is then used to show valid input simply as text on the webpage, making sure the input has been cleaned and transformed into the proper HTML characters. By catching potentially hazardous data before processing, input validation greatly strengthens the application’s security and reduces the chance of cross-site scripting (XSS) attacks. The modified echo.php code that incorporates these security precautions is shown below. ### code explanation The first thing the offered code snippet does is see if the request’s “data” field is empty. If so, a message requesting data entry into the input area is displayed before the script ends. If not, the input is formatted and repeated back to the user after being cleaned up using htmlentities() to stop XSS attacks.

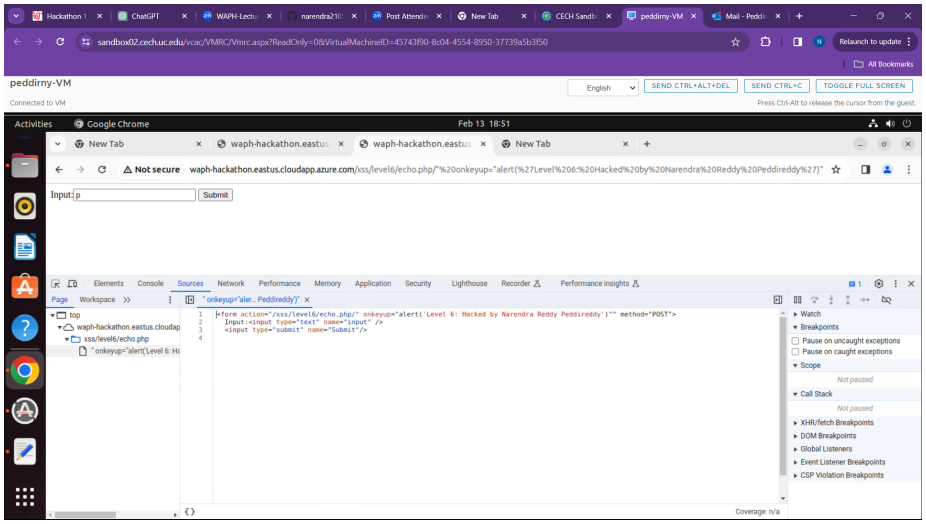


Figure 9: Level_6 code

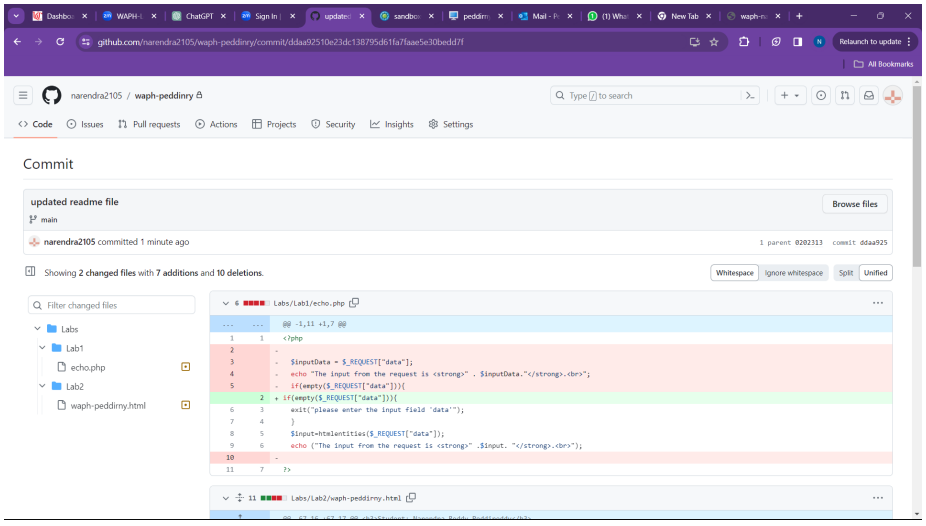


Figure 10: echo.php code change

B Now prototype for the front end

i HTTP GET AND POST Changes

Many changes were made to the waph-peddirny code in order to improve its security and dependability. Every external input point that was included into the code was carefully located and put through rigorous verification procedures. In order to guarantee the accuracy of the input data, a new function called validateInput() was created. This function requires users to input text before executing their requests, especially for HTTP GET and POST request forms. By taking a careful approach to input validation and output sanitization, the codebase's overall security posture is greatly improved, reducing the possibility of vulnerabilities or hostile exploitation.

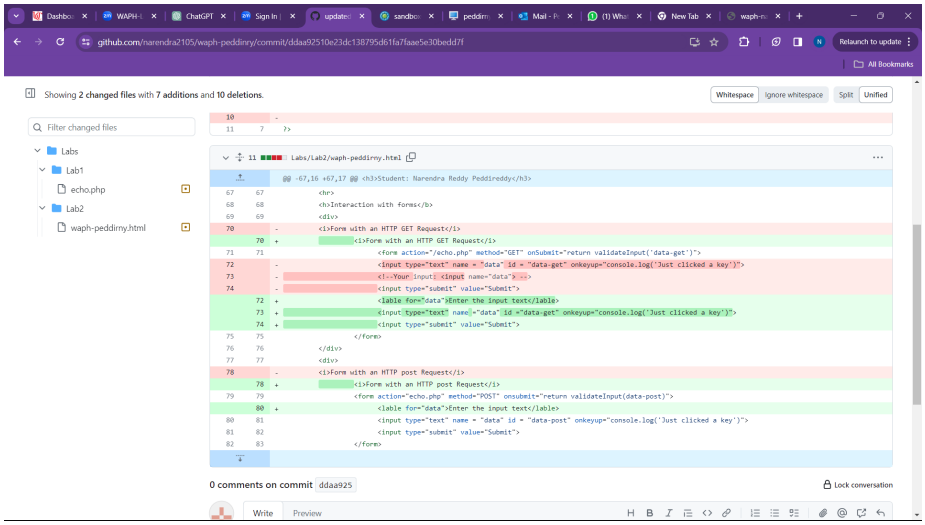


Figure 11: http get and post change

ii inner Text

The code has been modified to use.innerText instead of.innerHTML when plain text is required and HTML rendering is not required. By limiting the amount of HTML that is rendered to plain text, this change both streamlines the presentation of content and reduces the risk of security flaws. When it was determined that plain text simply needed to be displayed and HTML rendering was not required, the.innerHTML to.innerText conversion was applied.

iii Encodeinput()

Introduced to strengthen defenses against cross-site scripting (XSS) attacks is a new function named encodeInput(). In order to sanitize responses, this function converts special characters into the appropriate HTML entities before inserting

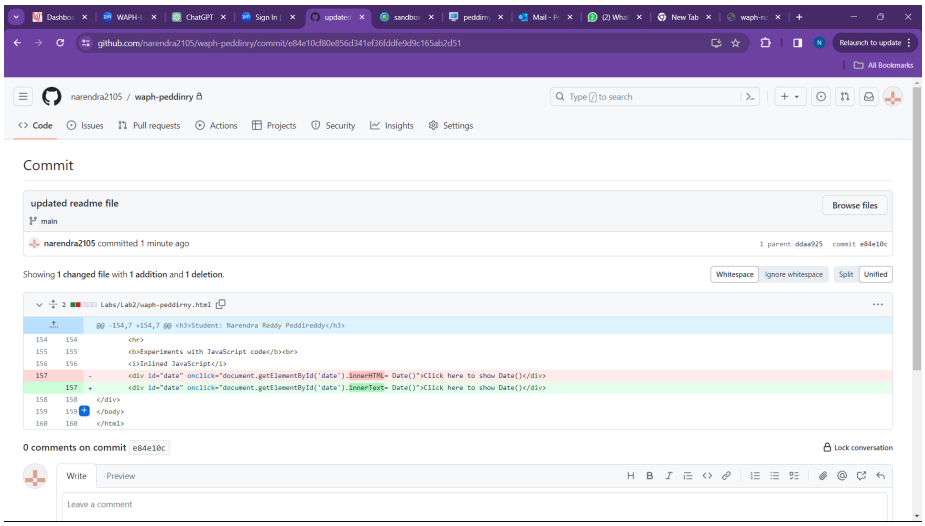


Figure 12: innerText code change

them into the HTML text. The content is rendered as non-executable text, significantly strengthening the application's defense against XSS assaults. In order to accomplish this, the encodeInput() function creates a new div element and sets its innerText attribute to the input that has been sanitized, thereby turning it into plain text. After that, the HTML content that has been cleaned up is returned, guaranteeing that any potentially dangerous scripts are eliminated before they can be run. Furthermore, the implementation of the validateInput() function guarantees that input is given prior to taking any further action.

iv

To increase the dependability of interactions with the API endpoint, rigorous validation procedures have been put in place. These validations carefully examine the result as well as the result that was received.make sure the humor fields in the JSON response are not empty. When null values are found, an error message is issued promptly, which improves the application's ability to handle unforeseen situations and makes the user experience more seamless. Moreover, the asynchronous procedure 'guessAge()' has been designed to confirm that the value obtained is neither null or empty. Additionally, it verifies that none of the input data is null or empty. The user is instantly alerted with an error message in both cases. By proactively resolving possible problems before they arise, this proactive method greatly improves the application's dependability.

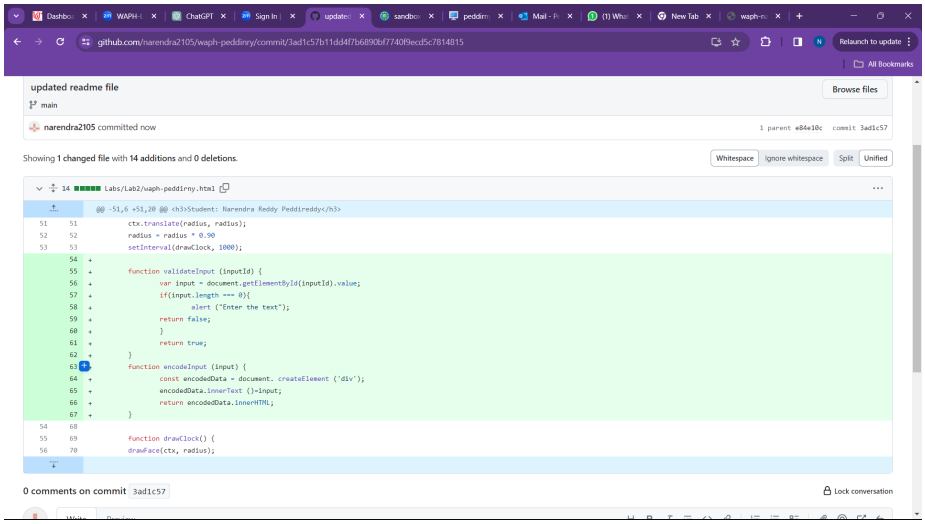


Figure 13: Encodeinput

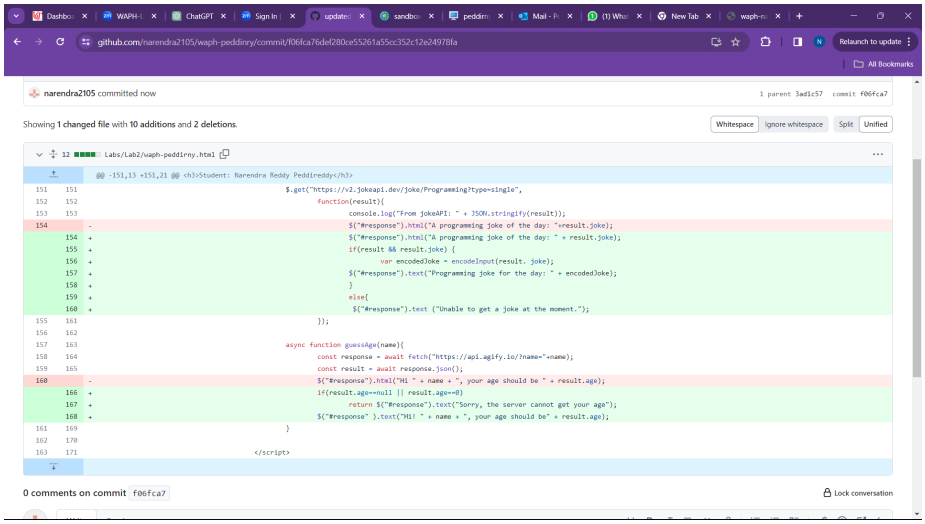


Figure 14: joke api and guess age