

WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Bheemreddy Vikhyath Reddy Email: bheemrvy@mail.uc.edu



Figure 1: Vikhyath's Headshot

Repository Information Repository's URL: <https://github.com/Vikhyath-Reddy/waph-bheemrvy> 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

Lab's overview

This Hackathon focuses on raising awareness about XSS attacks, understanding code vulnerabilities, adhering to OWASP guidelines, and implementing secure coding practices to defend against cross-site scripting attacks. The lab comprises two tasks:

Task 1 involves attacking the URL <http://waph-hackathon.eastus.cloudapp.azure.com/xss/>, which contains six levels of vulnerabilities. Task 2 entails mitigating XSS attacks through secure coding practices, emphasizing input validation and output sanitization.

Link to Hackthon1 code : <https://github.com/Vikhyath-Reddy/waph-bheemrvy/tree/main/Hackthons/Hackthons1>

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 Vikhyath Reddy Bheemreddy')</script>
```

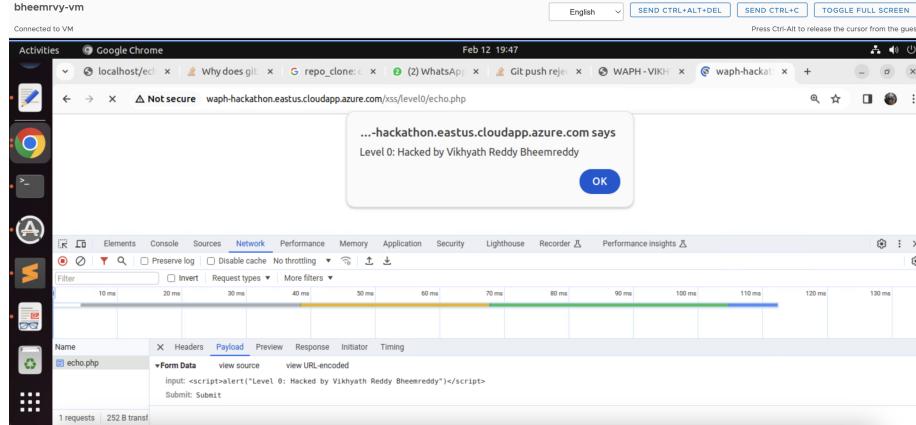


Figure 2: Level 0

Level 1

URL : <http://waph-hackathon.eastus.cloudapp.azure.com/xss//level1/echo.php>
Leveraging XSS vulnerabilities typically entails appending a malicious script at the end of the URL.

```
input=<script>alert('Level 1 - Hacked by Vikhyath Reddy Bheemreddy')</script>
```

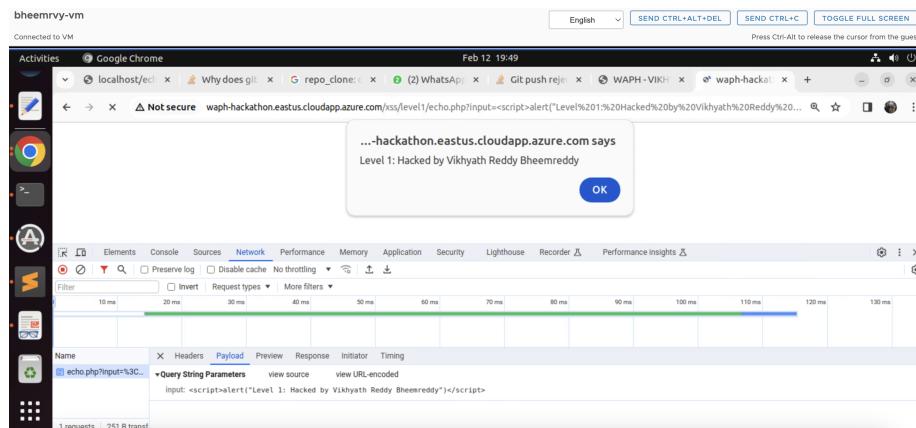


Figure 3: Level 1

Level 2

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

This URL is associated with an HTML form rather than engaging with user inputs or path variables within the HTTP request. Through this form, it becomes

possible to directly insert attacking scripts. This approach facilitates the injection of malicious code into the web application, thereby enabling the examination of XSS vulnerabilities.

```
input=<script>alert('Level 2 - Hacked by Vikhyath Reddy Bheemreddy')</script>
```

Possible Code:

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

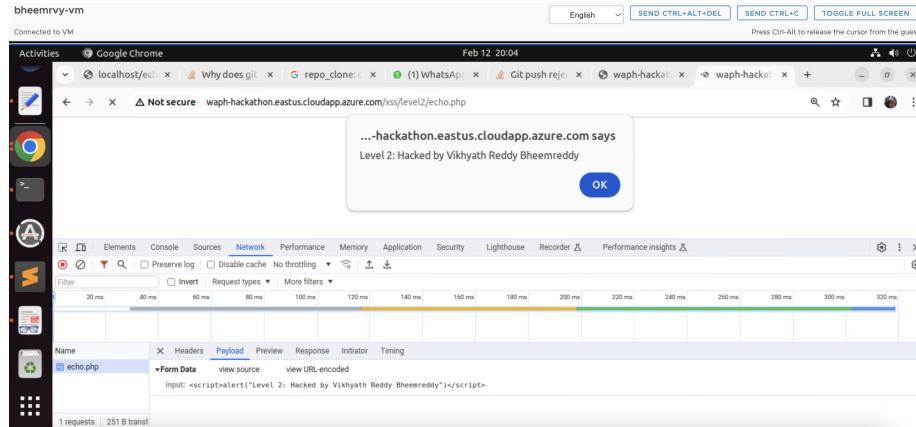


Figure 4: Level 2

Level 3

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

In order to successfully attack the URL and evade the filtering of `<script>` tags sent through input variables directly, the attacking code needs to be fragmented and then reassembled. This approach showcases the persistence and ingenuity necessary in XSS attacks, as it allows for the injection of malicious code capable of triggering alerts on specific web pages.

```
input=<scr<sc<script>ript>ipt>alert('Level 3 - Hacked by Vikhyath Reddy Bheemreddy')</scr</sc</script>
```

Code Possibility:

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

Level 4

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

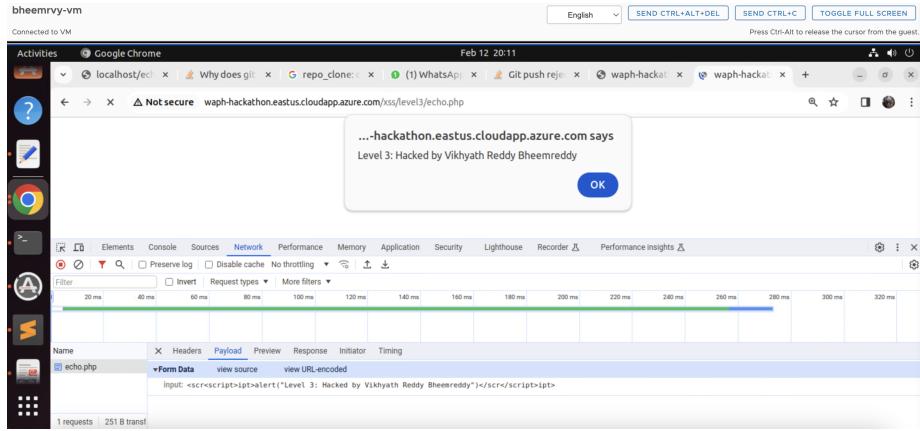


Figure 5: Level 3

I tried using the `onload()` event of the `<body>` tag for an XSS script, but even with script manipulation, the `<script>` element remained fully filtered. However, embedding the script within the `onload()` event still triggered an alert on page load, bypassing the filter and enabling the injection of harmful code without relying on the `<script>` tag.

```
input = <body onload="alert('Level 4 - Hacked by Vikhyath Reddy Bheemreddy')">This website is hacked!
```

Possible Source code:

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

Level 5

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

Enhancing security measures in level 5 involved encoding the `<body>` tag with the `onload()` function. This tactic bypasses direct filters on `<script>` and `alert()` functions, allowing for the indirect execution of JavaScript code. Utilizing Unicode encoding ensures proper interpretation of characters as JavaScript code by the browser, enabling the desired functionality of triggering a popup alert.

```
input=<body onload="\u0061lert('Level 5 - Hacked by Vikhyath Reddy Bheemreddy')">This website is hacked!
```

Possible Source Code:

```
$input = $_GET['input']
```

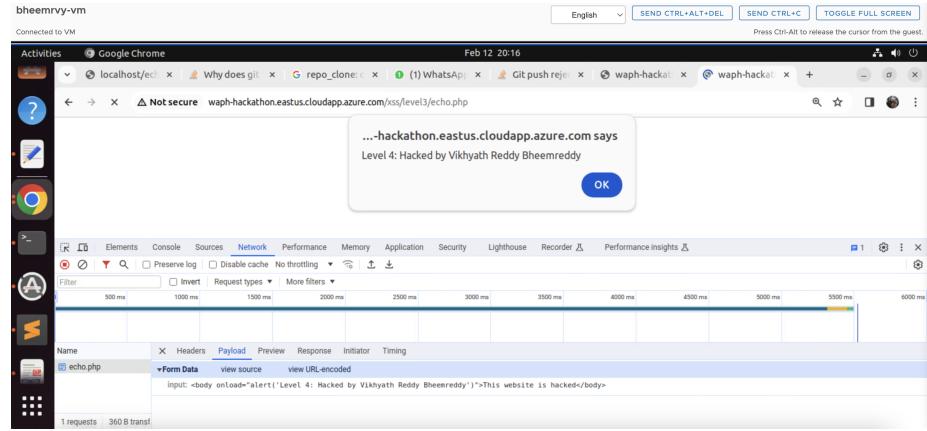


Figure 6: Level 4

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

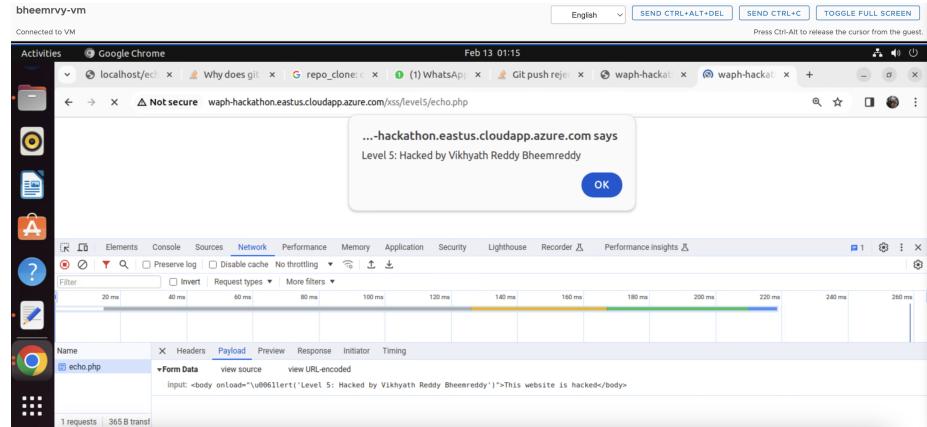


Figure 7: Level 5

Level 6

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

Using the `htmlentities()` method, user input is converted into HTML entities, displaying it solely as text on the webpage. JavaScript event listeners, such as `onclick()`, are then employed to initiate an alert whenever a key is entered in

the user input, enabling the execution of JavaScript code while maintaining the security precaution of displaying user input as plain text.

Possible Source code:

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

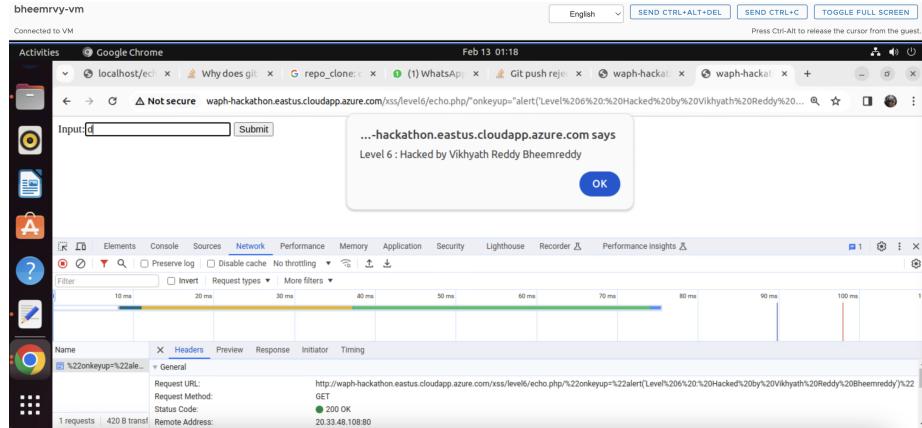


Figure 8: Level 6

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.

```
diff --git a/Labs/Lab1/echo.php b/Labs/Lab1/echo.php
--- a/Labs/Lab1/echo.php
+++ b/Labs/Lab1/echo.php
@@ -1,3 +1,8 @@
 <?php
 1   1     echo $_REQUEST['data'];
 2   2     if(empty($_REQUEST['data']))
 3   3     {
 4   4         exit("Please enter the input field data");
 5   5     }
 6   6     $input=htmlentities($_REQUEST['data']);
 7   7     echo("The input from the request is <strong>".$input."</strong>.<br>");
 8   ?>
```

Figure 9: Git Changes for echo.php

b. Front-end prototype

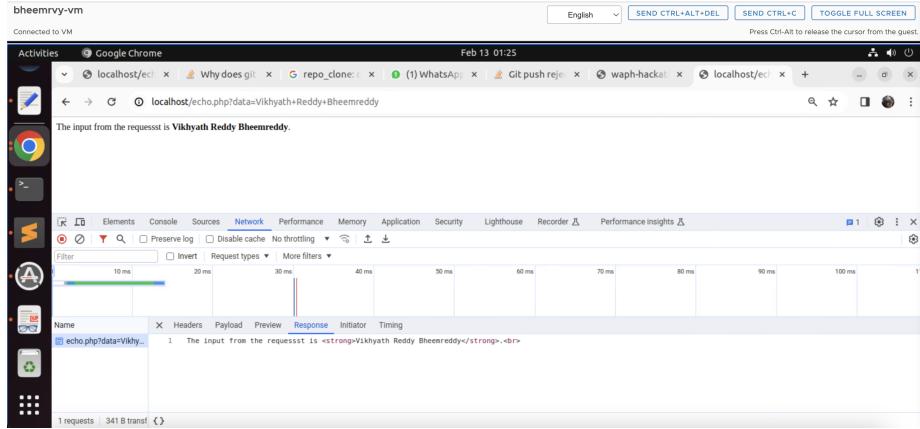


Figure 10: Echo Response

1. To enhance the security of both POST and GET requests, a validation function called validateInput() has been introduced. This function mandates user input text to ensure the data's validity before executing the request. Additionally, instances where plain text is displayed instead of HTML rendering have been identified to mitigate the risk of XSS attacks. To mitigate the possibility of executing any attacking scripts, the innerHTML property has been substituted with innerText for checking plain text. These measures collectively fortify the web application's security protocols concerning input validation and output rendering.

```

53 53      00 -53,17 +53,17 @@ -n3><div id="body">
54 54          <h3>Student : Vikhyath Reddy Bheemreddy</h3>
55 55      <div>
56 56          <h4>Interaction with HTTP Forms</h4>
57 57          <div>
58 58              <h5>Form with HTTP GET Request</h5>
59 59                  <form action="/echo.php" method="GET">
60 60                      <input type="text" name="data" onkeyup="console.log('you have clicked a Key')">
61 61                  </form>
62 62          <h5>Form with HTTP POST Request</h5>
63 63                  <form action="/echo.php" method="POST" onsubmit="return validateInput('post-data')">
64 64                      <input type="text" name="data" onkeyup="console.log('you have clicked a Key')">
65 65                      <input type="text" name="data" id="post-data" onkeyup="console.log('you have clicked a Key')">
66 66                      <input type="submit" value="Submit" id="submit" onsubmit="validateInput('post-data')"/>
67 67                  </form>
68 68          </div>
69 69      </div>
70 70      00 -74,14 +74,15 @@ -n3><div id="response">
71 71          <h3>Student : Vikhyath Reddy Bheemreddy</h3>
72 72          <div>
73 73              <input class="button round" type="submit" value="Ajax Echo" onclick="getEcho()"/>
74 74              <input class="button round" type="submit" value="jQuery Ajax Echo" onclick="getjQueryAjax()"/>
75 75              <input class="button round" type="submit" value="jQuery Ajax Post" onclick="getjQueryAjaxPost()"/>
76 76              <input class="button round" type="submit" value="Guess Age" onclick="guessAge($('data').val())"/>
77 77              <input class="button round" type="submit" value="guess Age" onclick="guessAge($('#data').val())"/>
78 78          </div>
79 79      </div>
80 80      </div>
81 81      </div>
82 82      <h4>Experiments with JavaScript code</h4>
83 83      <div id="inlineDate" onClick="document.getElementById('inlineDate').innerHTML=Date()><Click to display time and date</div>
84 84      <div id="inlineId" onClick="document.getElementById('inlineDate').innerText=Date()><Click to display time and date</div>
85 85      </div>
86 86

```

Figure 11: Git Changes for HTTP Requests

2. EncodeInput() function, introduced recently, aims to prevent XSS attacks

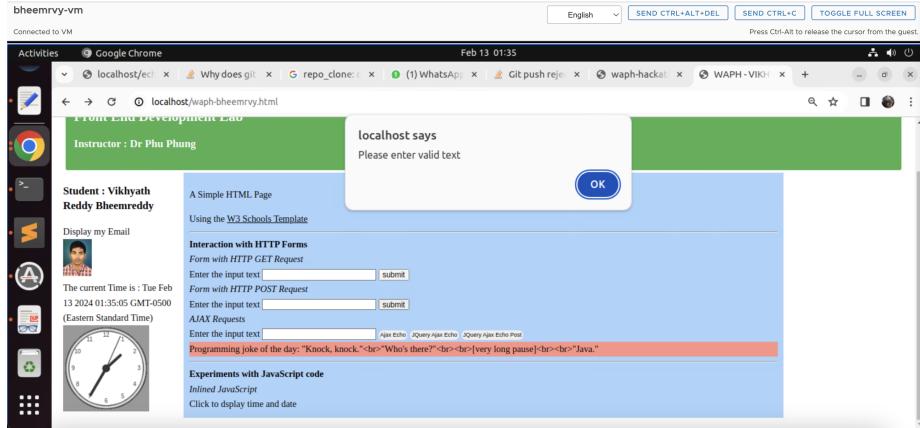


Figure 12: GET, POST Response when input field is empty

```

83 84      </div><div>
84 85      <div><input type="button" value="Click to display time and date"></div>
85 86      <div><input type="button" value="Click to display time and date"></div>
86 87    </div>
87 88  
```

88 -94.6 +95.28 @@ <h3>Student : Vikhyath Reddy Bheemreddy</h3>

94 95 document.getElementById('digital-clock').innerHTML=" The current Time is : "+ Date();

95 96 }
96 97 setInterval(displayTime,500);
97 98 function validateInput(inputId)
98 99 {
100 100 var input=document.getElementById(inputId).value;
101 101 if(input.length == 0){
102 102 alert("Please enter valid text");
103 103 return false;
104 104 }
105 105 return true;
106 106 }
106 107 function encodeInput(input)
107 108 {
108 109 const encodedData=document.createElement('div');
109 110 encodedData.innerHTML=input;
110 111 return encodedData.innerHTML;
111 112 }
112 113 </script>
113 114 <script type="text/javascript">
114 115 var canvas=document.getElementById("analog-clock");
115 116

116 -117,7 +132,7 @@ <h3>Student : Vikhyath Reddy Bheemreddy</h3>

117 117 //alert("readystate "+ this.readyState +" , status "+this.status+" , statusText "+this.statusText);
118 118 if(this.readyState==4 && this.status==200)
119 119 console.log("Received data "+http.responseText);
120 120 document.getElementById("response").innerHTML= http.responseText;
121 121 document.getElementById("response").innerHTML= encodeInput(http.responseText);
122 122 }
123 123 xhttp.open("GET", "echo.php?data="+input, true);
124 124

124 -144,15 +159,26 @@ <h3>Student : Vikhyath Reddy Bheemreddy</h3>

Figure 13: Git Changes for new functions added

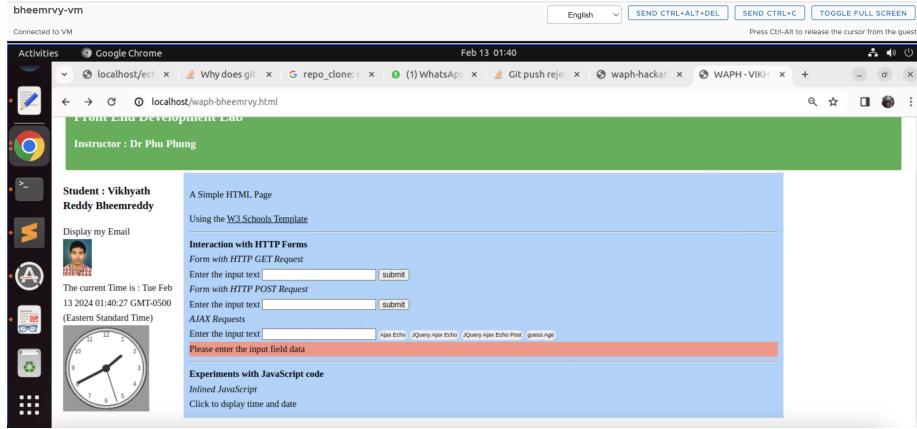


Figure 14: AJAX Response when input field is empty

by transforming special characters into HTML entities, rendering HTML content as plain text, thus immune to malicious scripts. It generates a new `<div>` element, appends the sanitized content (`innerText`), and returns the HTML content within this div, ensuring enhanced security against XSS vulnerabilities.

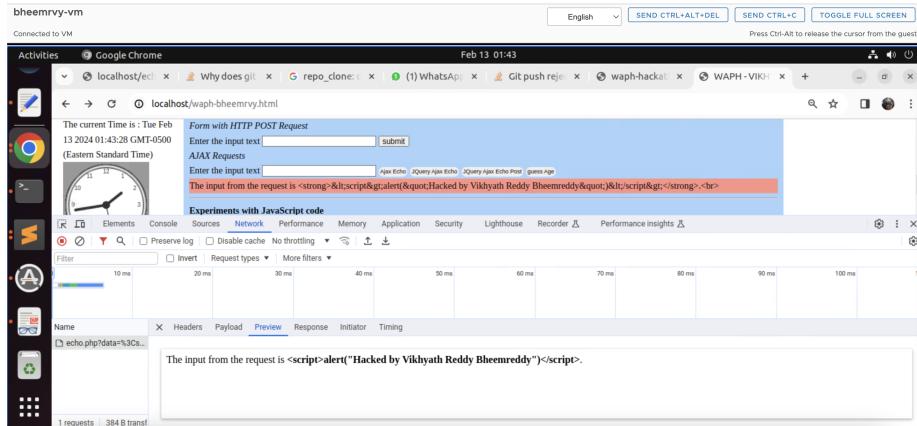
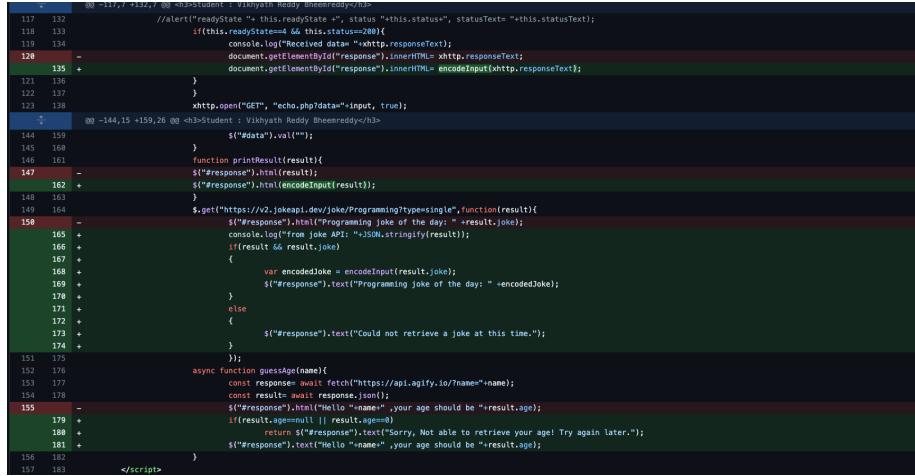


Figure 15: Ajax Response

3. To enhance security and reliability, additional validation checks have been implemented in API calls. Specifically, when fetching jokes from the JokeAPI, checks are now in place to ensure that both the JSON response and the `result.joke` property are not empty. If either of these variables is found to be null, an error message is promptly displayed to alert the user. These measures aim to bolster the dependability of the application

by ensuring that valid data is retrieved from the API.



```

diff --git a/index.js b/index.js
index 90-111,7 112,7 @@ -<h3>Student : Vikhyath Reddy Bheemreddy</h3>
 117 132      //alert("readystate "+ this.readyState +", status "+this.status+", statusText= "+this.statusText);
 118 133      if(this.readyState==4 && this.status==200)
 119 134          console.log("Received data: "+http.responseText);
 120 135      document.getElementById("response").innerHTML= http.responseText;
 121 136      }
 122 137      }
 123 138      xhttp.open("GET", "echo.php?data"+input, true);
 144 159      $($("#data").val(""));
 145 168      }
 146 161      function printResult(result){
 147 162      $("#response").html(result);
 148 163      }
 149 164      $.get("https://v2.jokeapi.dev/jokes/Programming?type=single",function(result){
 150 165      $("#response").html("Programming joke of the day: "+result.joke);
 151 166      console.log("From joke API: "+JSON.stringify(result));
 152 167      if(result && result.joke)
 153 168      {
 154 169          var encodedJoke = encodeInput(result.joke);
 155 170          $("#response").text("Programming joke of the day: "+encodedJoke);
 156 171      }
 157 172      else
 158 173      {
 159 174          $("#response").text("Could not retrieve a joke at this time.");
 160 175      });
 161 176      async function guessAge(name){
 162 177          const response = await fetch("https://api.agify.io/?name="+name);
 163 178          const data = await response.json();
 164 179          if(data.name == null || result.age == 0)
 165 180              return $("#response").text("Sorry, Not able to retrieve your age! Try again later.");
 166 181          $("#response").text("Hello "+name+", your age should be "+result.age);
 167 182      }
 168 183  
```

Figure 16: Joke Api Git changes

- To enhance the reliability and security of the application, further validation steps have been integrated into the guessAge() function. These steps include verifying that the user's input is not null or empty and confirming that the obtained output is neither empty nor zero. If any of these criteria are not met, a relevant error message notifies the user. These enhancements aim to minimize errors, ensure data accuracy, and bolster the application's dependability and security.

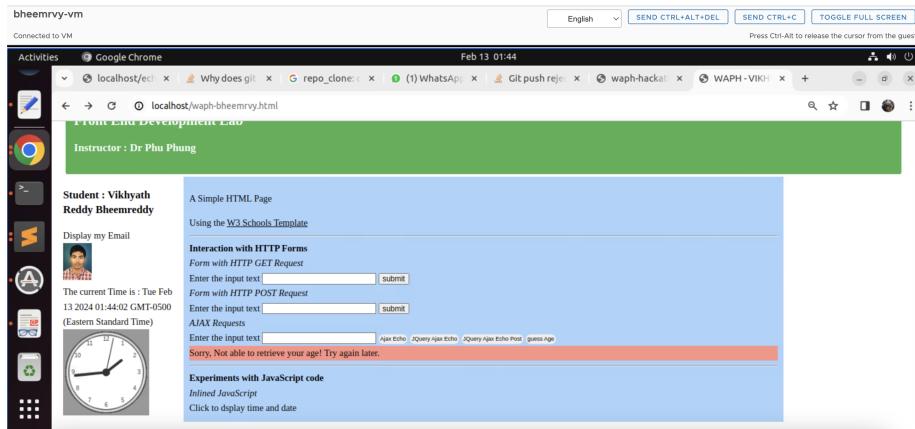


Figure 17: Age API