

WAPH-Web Application Programming and Hack-ing

Instructor: Dr. Phu Phung

Student

Name: Bhargavi Murari

Email: muraribi@mail.uc.edu



Figure 1: Bhargavi's headshot

Repository Information

Repository's URL: <https://github.com/BhargaviMurari22/waph-muraribi.git>

Hackathon 1 Report:

Overview and outcomes:

The “Cross-site Scripting Attacks and Defenses” hackathon offers participants a thorough exploration of online application security, placing special emphasis on addressing the pervasive threat of cross-site scripting (XSS) attacks. Through hands-on lab exercises, attendees gain firsthand experience in understanding the risks associated with XSS vulnerabilities. These vulnerabilities pose serious threats to user data, session cookies, and the overall system integrity of web applications, allowing the injection and execution of scripts on legitimate web pages. The hackathon aims to empower participants with the knowledge and skills necessary to both exploit and defend against XSS vulnerabilities. The primary focus is on implementing robust defense strategies to significantly mitigate these risks. Using the attached GitHub link, access the lab materials: <https://github.com/BhargaviMurari22/waph-muraribi/tree/main/labs/hackathon1>

Task 1:

LEVEL 0:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php>)

Script used for attacking: `<script>alert("Level0: Hacked by BHARGAVI MURARI")</script>`

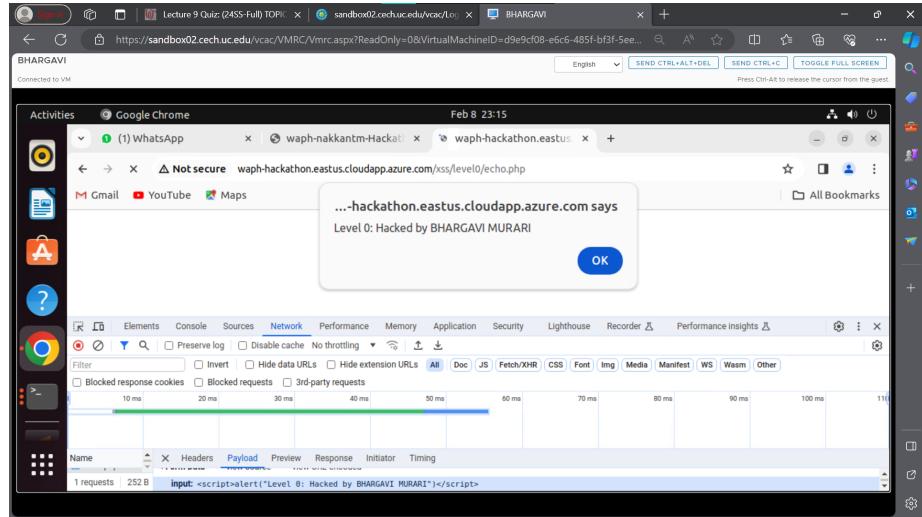


Figure 2: Hacked Level0

LEVEL 1:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php>)
 Script was written at the end of the URL as a path variable. The attacking script used was given below, `?input=<script>alert("Level 1: Hacked by BHARGAVI MURARI")</script>`

LEVEL 2:

URLused: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php>)
 This HTTP request has been converted into a basic HTML form as it does not include an input field and does not allow a path variable. The employment of a hacking script is then made easier by the attacking script being guided through this form.

```
<script>alert("Level2 Hacked by BHARGAVI MURARI")</script>
```

Source Code Guess:

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

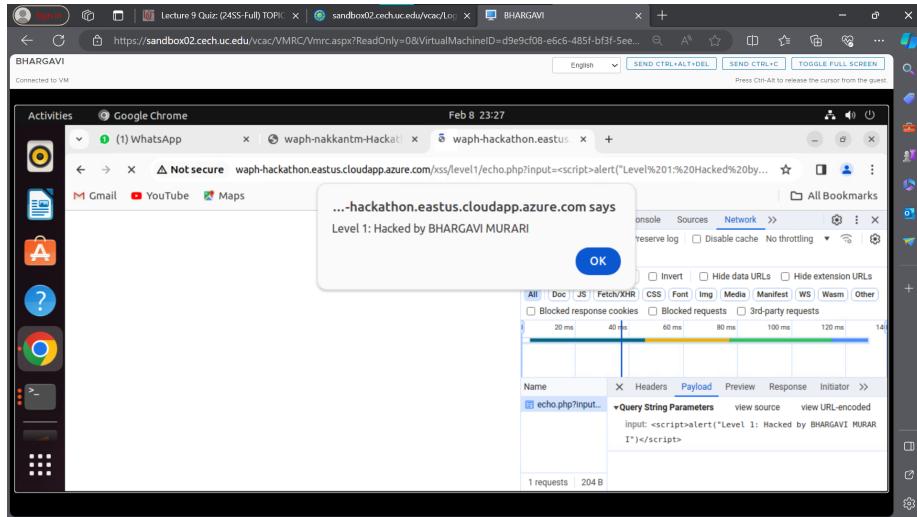


Figure 3: Hacked Level1

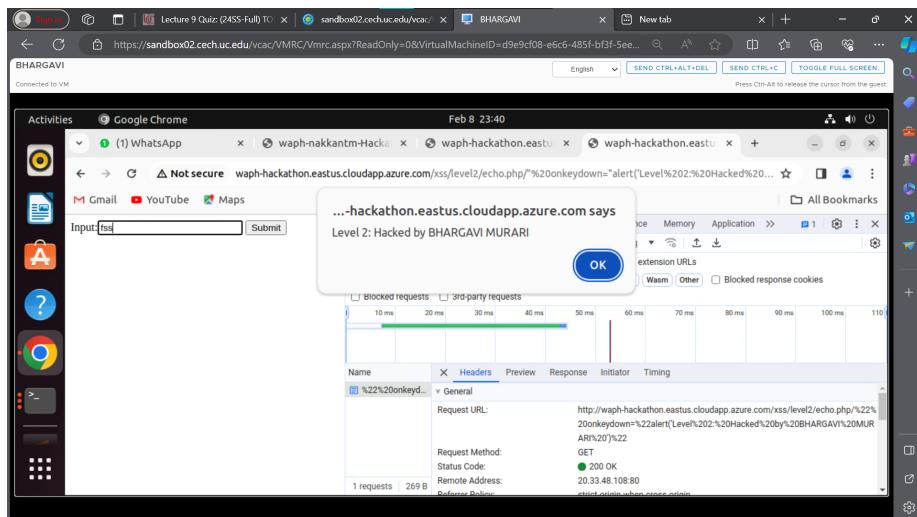


Figure 4: Hacked Level2

LEVEL 3:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php>)
This degree of security prevents the script tag from being directly entered into the input variable. To take use of this URL, the code was broken up into several pieces and connected to cause a warning to appear on the website.

Script tag used for attacking:

```
?input=<script><script>>alert("Level 3 Hacked by BHARGAVI MURARI")</script></script>t>
```

Source code guess: Script tag may be substituted with a blank, `str_replace(['<script>', '</script>'], ' ', $input)`

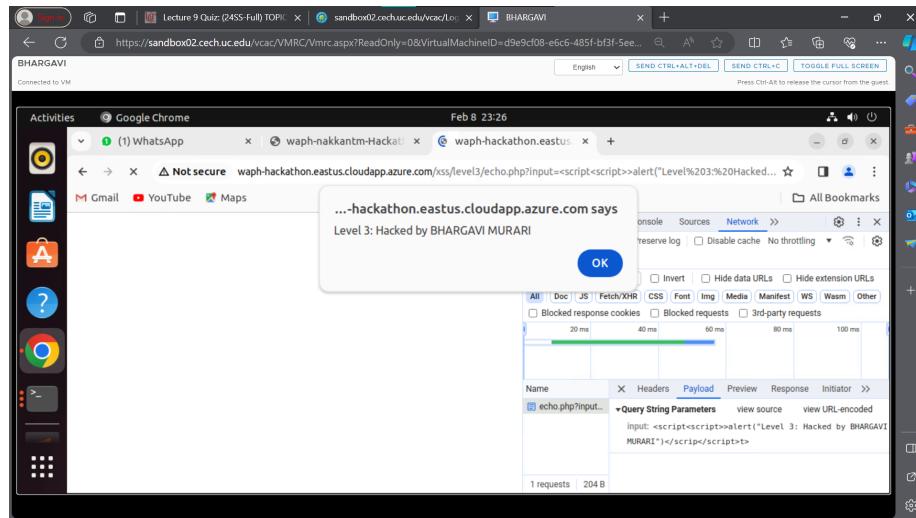


Figure 5: Hacked Level3

LEVEL 4:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php>)
At this point, the script tag is fully filtered, thus it won't be able to be blocked even if the string is broken up and then joined together. Using the `onerror()` method of the tag to inject the XSS script allowed me to set off an alarm.

Script tag used:

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

Code injected:

```
?input=<button onclick="alert('Level4')"></button>
```

Source code guess:

```

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

```

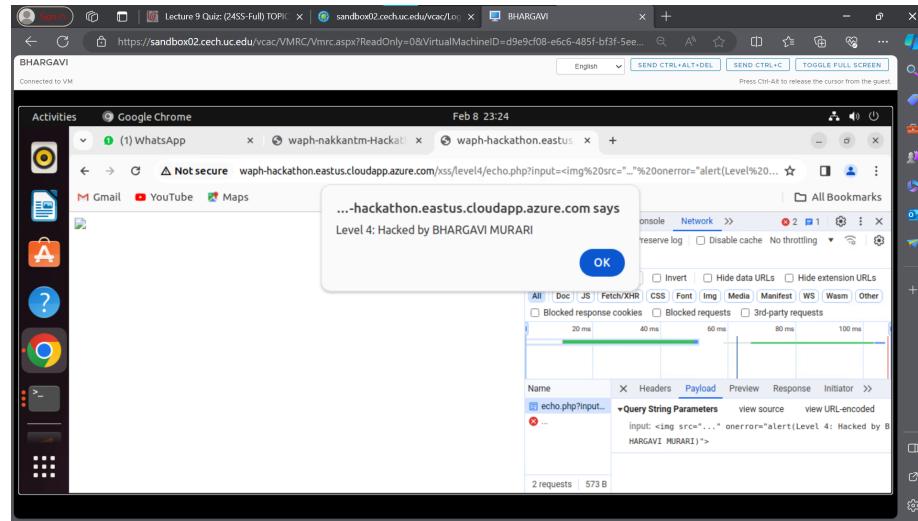


Figure 6: Hacked Level4

LEVEL 5:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php>)
Both the alert function and the tag are filtered at this level. I combined the button tag's onerror function with unicode encoding to raise the popup alert.

Code injected:

```
?input=
```

Source Code Guess:

```

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

```

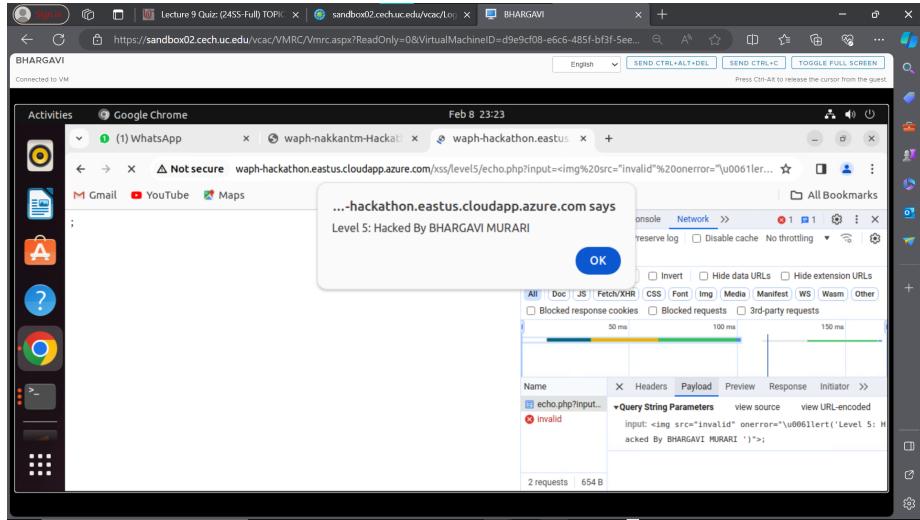


Figure 7: Hacked Level5

LEVEL 6:

URL used: (<http://waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php>)
 This level still takes input even though I think the original code uses the htmlentities method to translate necessary characters into their correct HTML entities. As a result, the webpage shows user input as plain text. In certain situations, JavaScript eventListeners like onmouseover, onclick, onkeyup, and onmouseenter can be used to initiate an alert. In this case, every time a key is hit in the input field, the onmouseenter eventListener that I used triggers an alarm on the website. The input form element is altered when the script is inserted through the URL, as shown in the illustration below. It appends to the code.

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

Source Code Guess:

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

Task 2.

A. By adding code for input validation and editing the echo.php file, Lab 1's defensive measures against XSS were put into place. After a preliminary check to make sure the input is empty, PHP is stopped from running. When the input is verified as genuine, the htmlentities technique is used to clean it up. It then gets converted into the appropriate HTML characters so that it shows up on the

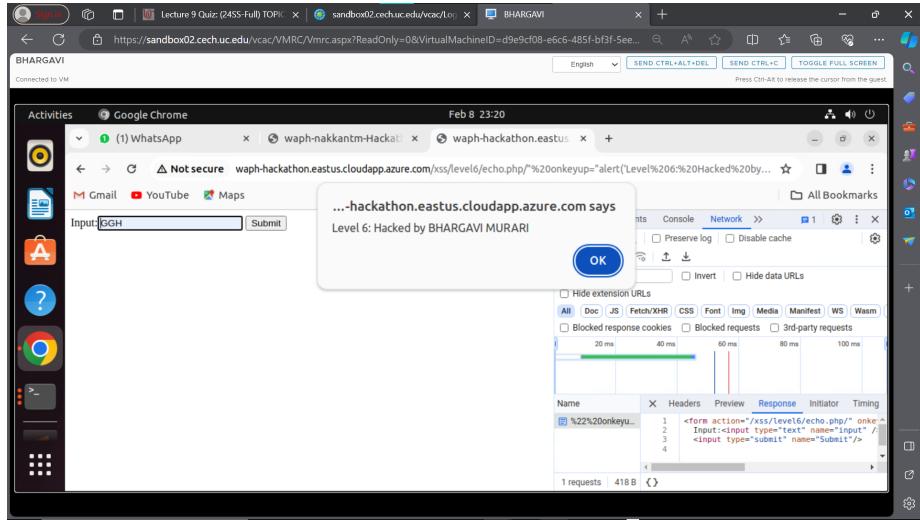


Figure 8: Hacked Level6

page as text only.

```
echo.php defence <?php if(!isset($_REQUEST["data"])){ die("{\"error\":\n\"Please provide 'data' field\"}"); } echo htmlentities($_REQUEST['data']); ?>
```

B. Once external input points were identified, the waph-muraribi.html code underwent a significant change. After each of these inputs was verified, the result texts were edited. i) There is now validation applied to the input data for the HTTP GET and POST request forms. The request cannot be handled until the user input has been validated thanks to the introduction of a new function called validateInput.

- ii) InnerText was substituted for innerHTML in cases when the display was plain text and HTML rendering was not required.
- iii) To improve security by cleaning replies, a newly added function named encodeInput has been developed. To prevent cross-site scripting attacks, this entails transforming special characters into the proper HTML entities before adding them to the HTML content. As a result, the material is not executable and is handled as text only. InnerText is the material that is injected into a newly generated div element by the code. It is then given back as HTML content.

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

The screenshot shows a GitHub commit page for a file named echo.php. The commit title is "echo.php revised; added defense mechanism". The commit message indicates it was made by BhargaviMurari22 1 minute ago and is verified. The commit has 1 parent, b07b031, and a commit hash of 5cac2298f9ddacd130e67262dd53b0fb5164bf9. The code diff shows a single change: line 2 was modified from "echo \$_REQUEST['data'];" to "if(!isset(\$_REQUEST['data'])){ die('Error' . "\nPlease provide 'data' field"); } echo htmlentities(\$_REQUEST['data']);". The commit interface includes tabs for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. It also shows 0 comments on the commit and a Write and Preview button.

Figure 9: Revised echo.php with added defense mechanism

The screenshot shows a GitHub commit page for a file named waph-muraribi.html. The commit message is "Interaction with forms". The code diff shows numerous changes, with many lines added (green) and deleted (red). The changes include the addition of form elements like input fields and submit buttons, along with associated JavaScript logic for validation and logging key presses. The commit interface includes tabs for Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings. It shows 0 comments on the commit and a Write and Preview button.

Figure 10: waph-muraribi.html defense code

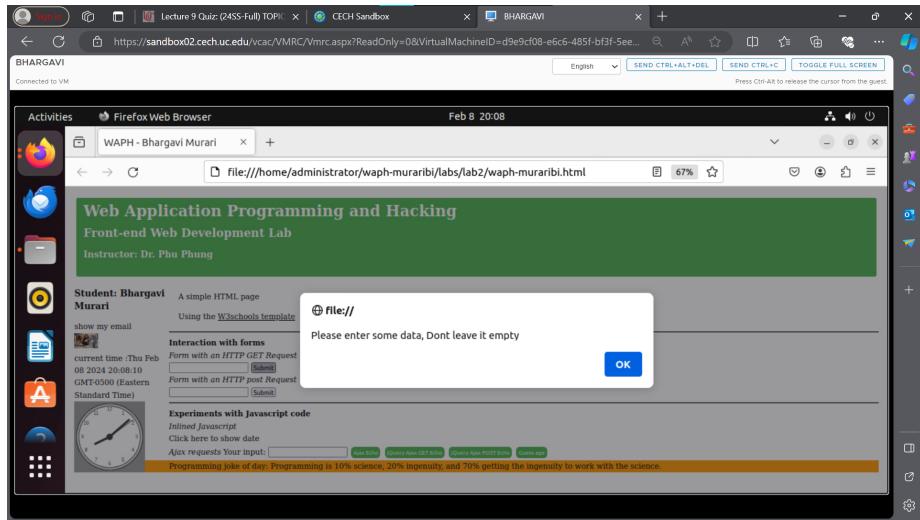


Figure 11: Validating inputs

```

Showing 1 changed file with 38 additions and 19 deletions.

@@ -152,6 +158,19 @@ <h3> Student: Bhargavi Murari</h3>
158     ctx.translate(radius, radius);
159     radius=radius * 0.99;
160     setinterval(drawclock, 1000);
161 +   function validateinput(inputid) {
162 +     var input = document.getElementById(inputid).value;
163 +     if(input.length == 0){
164 +       alert("Please enter some data, Dont leave it empty");
165 +       return false;
166 +     }
167 +     return true;
168 +   }
169 +   function encodeInput(input){
170 +     const encodeddata = document.createElement('div');
171 +     encodeddata.innerText=input;
172 +     return encodeddata.innerHTML;
173 +   }
155 174   function drawClock(){
156 175     drawFace(ctx, radius);
157 176     drawNumbers(ctx, radius);

```

Figure 12: Modified html to text



Figure 13: Validated Output

}

```

Showing 1 changed file with 38 additions and 19 deletions.

55 +             <input type="text" name = "data" id = "data-get" onkeyup="console.log('Just clicked a key')">
56 +             <!--
57 +             Your input: <input name="data" -->
58 +             </input type="submit" value="Submit">
59 +         </div>
60 +     </div>
61 +     <div>
62 +         <div>
63 +             <div> <form with an HTTP post Request/>
64 -             <form action="echo.php" method="POST">
65 -                 Your input: <input name="data">
66 -                 <input type="text" name = "data" id = "data-post" onkeyup="return validateInput(data-post)">
67 -                 <input type="submit" value="Submit">
68 -             </form>
69 +             <div>
70 +                 <div> Experiments with Javascript code </div><br>
71 +                 <div> Inlined Javascript</div>
72 -                 <div id="date" onclick="document.getElementById('date').innerHTML=Date()">click here to show date</div>
73 +                 <div id="date" onclick="document.getElementById('date').innerText=Date()">click here to show date</div>
74 -             </div>
75 +             <div> Ajax requests</div>
76 +             Your Input:
77 +             <input name="data"
78 +                 onkeyup="console.log('you have pressed a key')"(id="data");
79

```

Figure 14: Encode input function

- iv) Updates have been made to the joke retrieval API to guarantee the accuracy of the results that are received. New validations have been added, which verify that jokes in JSON are not empty. An error notice appears when the result is null.

```

Showing 1 changed file with 38 additions and 19 deletions.

88 85     async function guessAge(name){
89 86         const response =await fetch(`https://api.agify.io/?name=${name}`);
90 87         const result=await response.json();
91 88         // ${"#response"},html(`Ht "+name + ",your age should be "+ result.age);
92 -         ${"#response"},html(`Ht "+name + ",your age should be 30");
93 +         if(result.age == null || result.age==0)
94 +             return ${"#response"},text("An error occurred at this moment");
95 +         ${"#response"},html(`Ht "+name + ",your age should be "+ result.age);
96
97         }
98         function queryAjax() {
99             var input = ${"#data").val();
100            if (input.length == 0) return;
101            $.get("echo.php?data="input,
102            function(result) {
103 -                ${"#response"},html("Response from server:" + result);
104 +                ${"#response"},html("Response from server:" + encodeinput(result));
105             }
106             );
107             ${"#data").val("");
108         }
109
110     }

```

Figure 15: Defense to displaying joke

- v) It is confirmed that the asynchronous method guessAge's received result is either empty or non-zero. In addition, the data entered by the user is checked to make sure it is not null or empty. An error notification appears on each of these occasions.

```

if(result.age==null || result.age==0)
return ${"#response")
.text("An error occurred at this moment, So age can't be displayed");
${"#response").text("Hello "+name+" ,your age should be "+result.age);

```



Figure 16: Validated Guess Age