8th April 2024

Report on Remote code execution via web shell upload

Presented By

Tanuj Sood

8th April 2024

Table of Contents:-

1-Introduction:

This scenario involves a website with a vulnerability in its image upload feature. This vulnerability allows users to upload files without proper validation, meaning they can upload files that are not images, such as PHP scripts disguised as images.

The objective is to exploit this vulnerability by uploading a PHP web shell. A web shell is a script that can be uploaded to a web server to gain remote access and control. Once uploaded, the attacker can execute commands on the server remotely.

In this scenario, the goal is to upload a PHP web shell and then use it to access and retrieve the contents of a specific file located at "/home/carlos/secret". This file contains sensitive information that needs to be retrieved.

2-Objectives:

The primary objective is to exploit a vulnerability within a web application's image upload function. This vulnerability enables users to upload files without adequate validation, potentially allowing for the uploading of malicious files, such as PHP scripts disguised as images.

The aim is to take advantage of this vulnerability by uploading a PHP web shell. This web shell serves as a means to gain remote access and control over the server. Once the web shell is successfully uploaded, the next step is to utilize it to access and retrieve the contents of a specific file located at "/home/carlos/secret". The ultimate goal is to retrieve the sensitive information contained within this file.

3- Requirements:

1. VirtualBox
2. Kali Linux virtual machine
3. Foxy proxy
4. Burp suite
5. Knowledge about cyber security fundamentals
6. Knowledge about web security.
7. Know about networking Fundamentals
8. Basic Knowledge of PHP

4- Scope:

The scope entails exploiting a vulnerability in a web application's image upload, lacking proper validation. Users can upload files, including malicious PHP scripts. The aim is to upload a PHP web shell to gain remote server access. Post-exploitation, the objective is to retrieve sensitive information from a specified file. This encompasses identifying, exploiting, and post-exploitation activities. The vulnerability enables unauthorized access and control over the server. The web shell facilitates remote execution of commands. Attacker gains access to the server environment upon successful upload. Accessing "/home/carlos/secret" file marks completion of the task. The focus is on the impact of the vulnerability and the severity of exploitation. The scenario underscores the importance of robust security measures. It highlights risks associated with inadequate validation in web applications. The objective is to demonstrate the potential consequences of such vulnerabilities. The scope emphasizes the need for thorough security assessments and measures in web development

5-Tools Used:

1. Virtual Box
2. Kali Linux Virtual machine
3. Burp suite
4. Foxy proxy

6- Methodology Used:

i. **Initial Assessment:** Analyze the web application to identify potential vulnerabilities, focusing on the image upload feature.

ii. **Vulnerability Identification:** Determine if the application lacks proper validation, allowing for the upload of non-image files.

iii. **Exploitation:** Develop or obtain a PHP web shell and disguise it as an image file. Upload the web shell using the vulnerable upload function.

iv. **Remote Access:** Once the web shell is uploaded successfully, use it to establish remote access to the server.

v. **Command Execution:** Utilize the web shell's capabilities to execute commands on the server remotely.

vi. **File Retrieval:** Access the "/home/carlos/secret" file using the web shell's functionality to retrieve its contents.

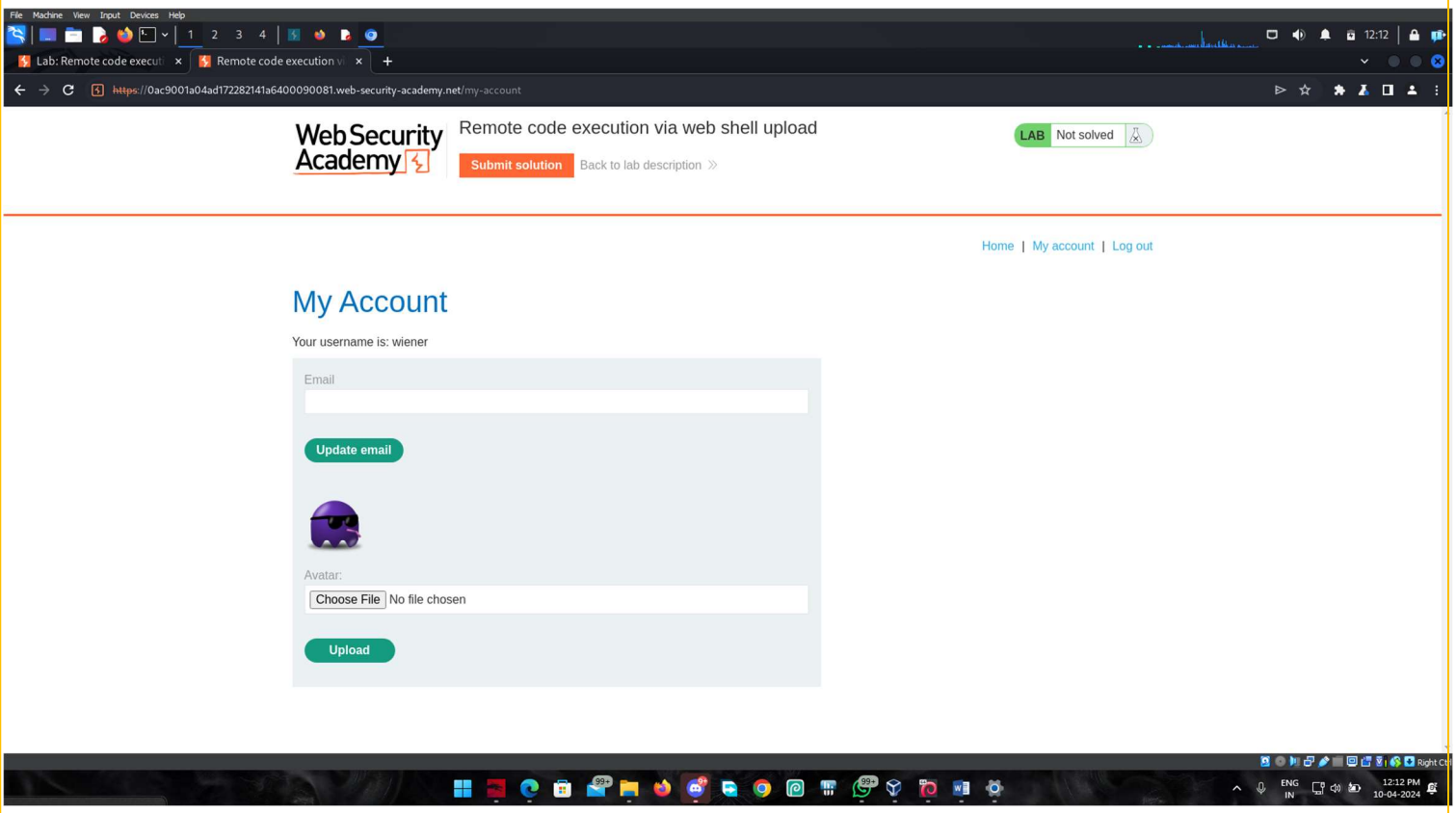
vii. **Data Exfiltration:** Extract the sensitive information from the retrieved file.

viii. **Submission:** Submit the extracted secret using the provided interface to complete the task.

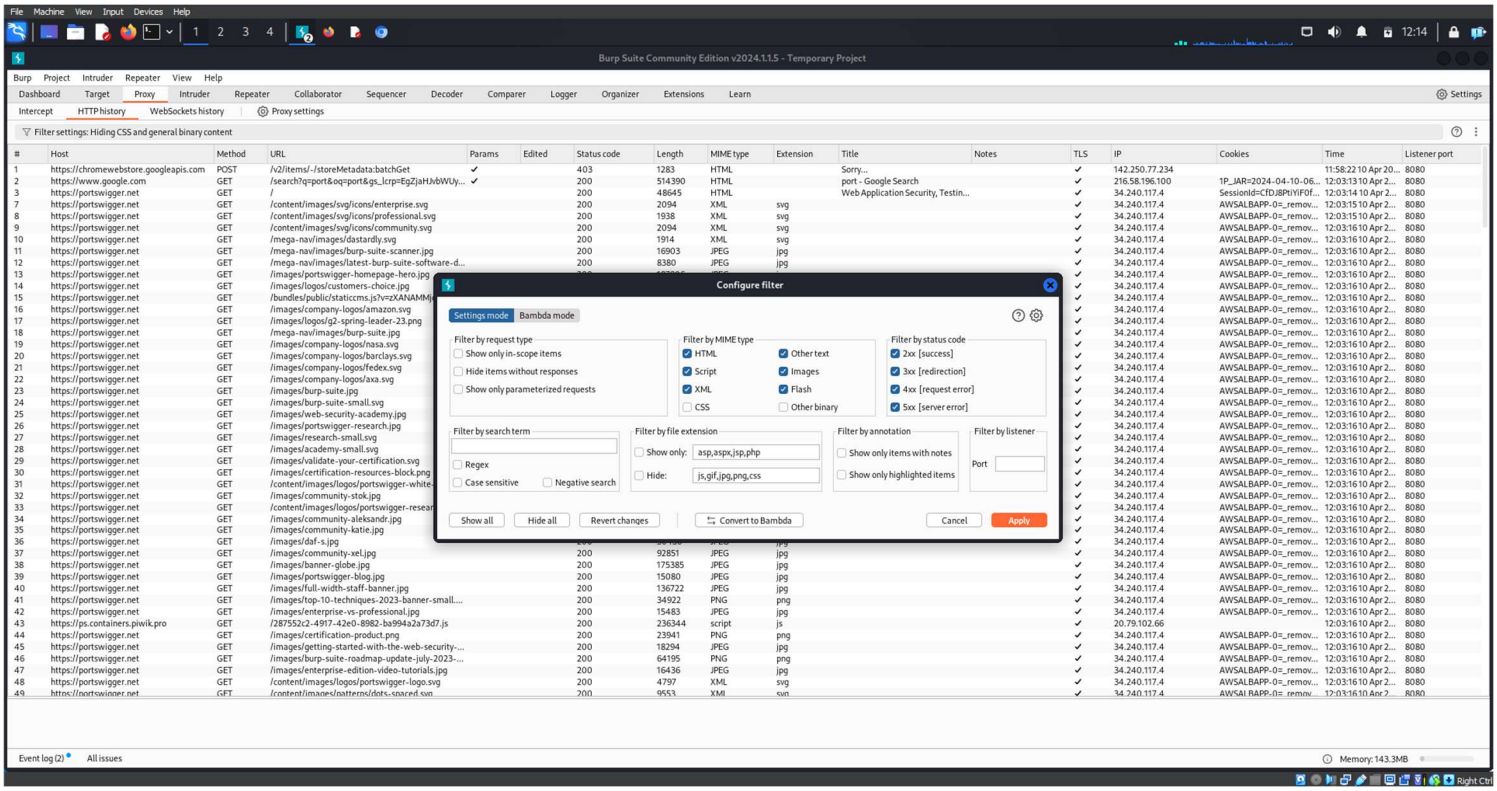
ix. **Cleanup and Mitigation:** Remove any traces of the web shell and address the vulnerability to prevent further exploitation.

x. **Documentation:** Document the steps taken, findings, and remediation measures for future reference and improvement of security practices

Step 1: log in to your account and notice the option for uploading an avatar image.Upload an arbitrary image, then return to your account page. Notice that a preview of your avatar is now displayed on the page.

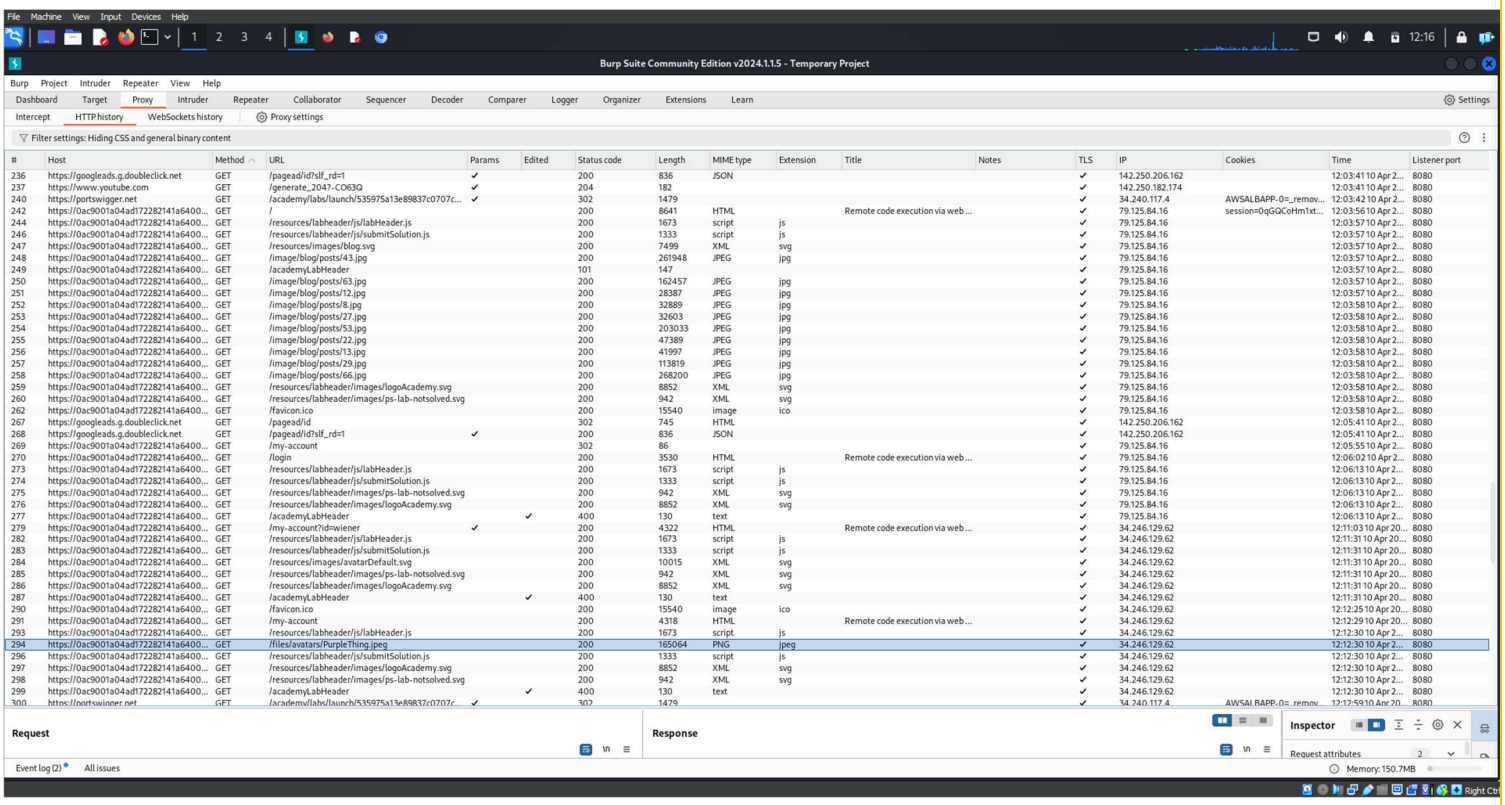


Step 2: In Burp, go to **Proxy > HTTP history**. Click the filter bar to open the **HTTP history filter** window. Under **Filter by MIME type**, enable the **Images** checkbox, then apply your changes.

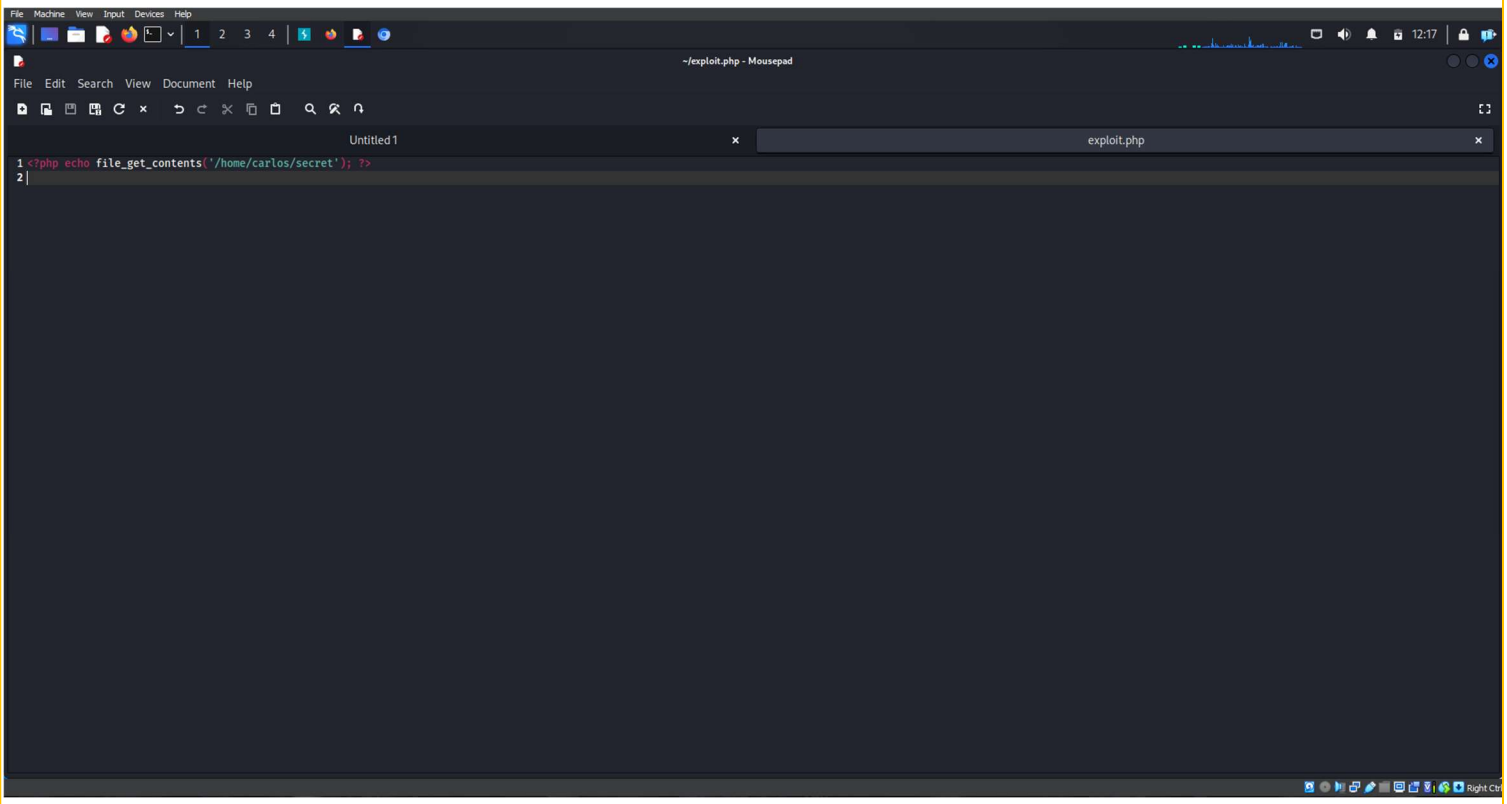


8th April 2024

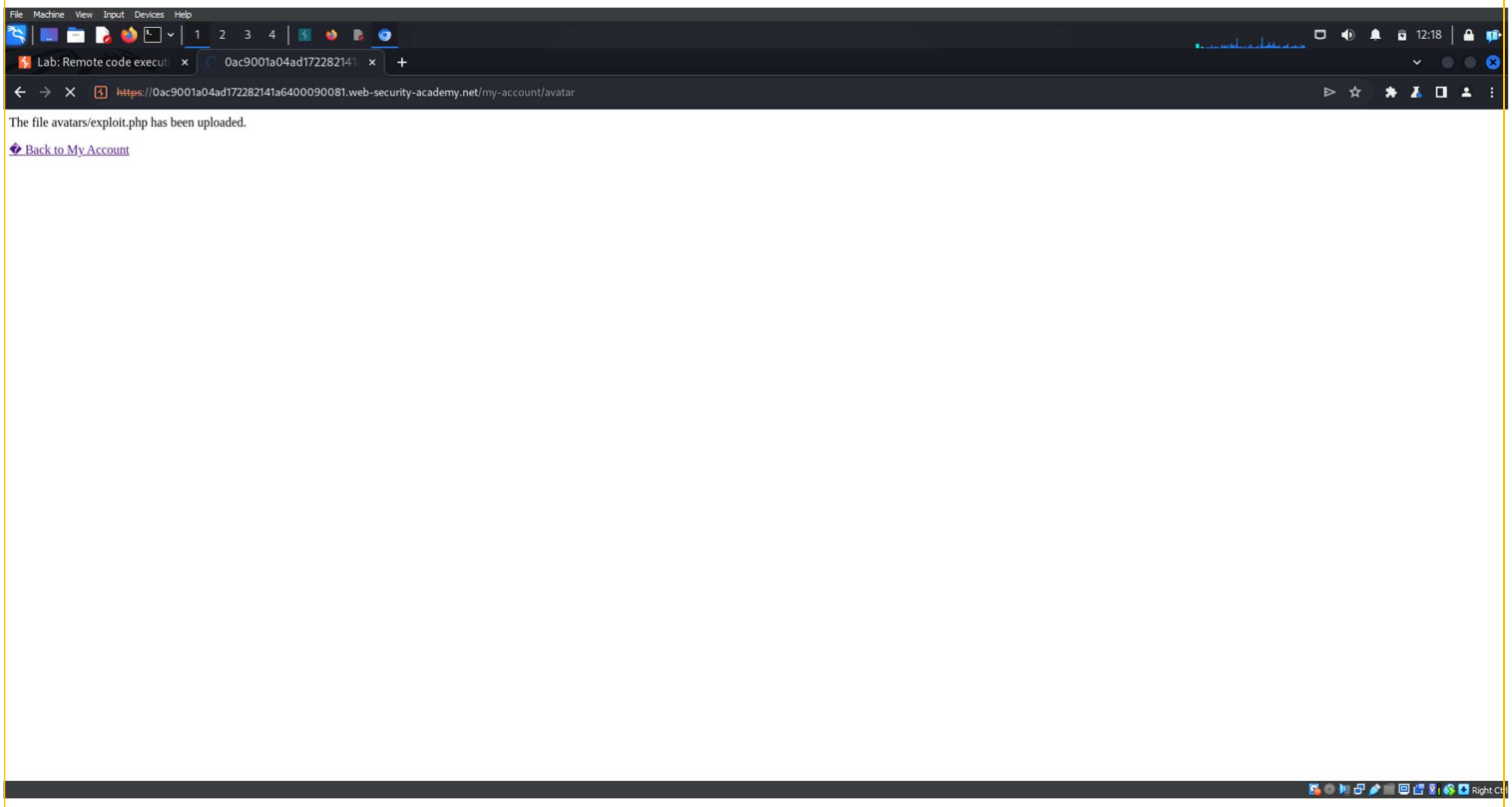
Step 3: In the proxy history, notice that your image was fetched using a GET request to /files/avatars/<YOUR-IMAGE>. Send this request to Burp Repeater.



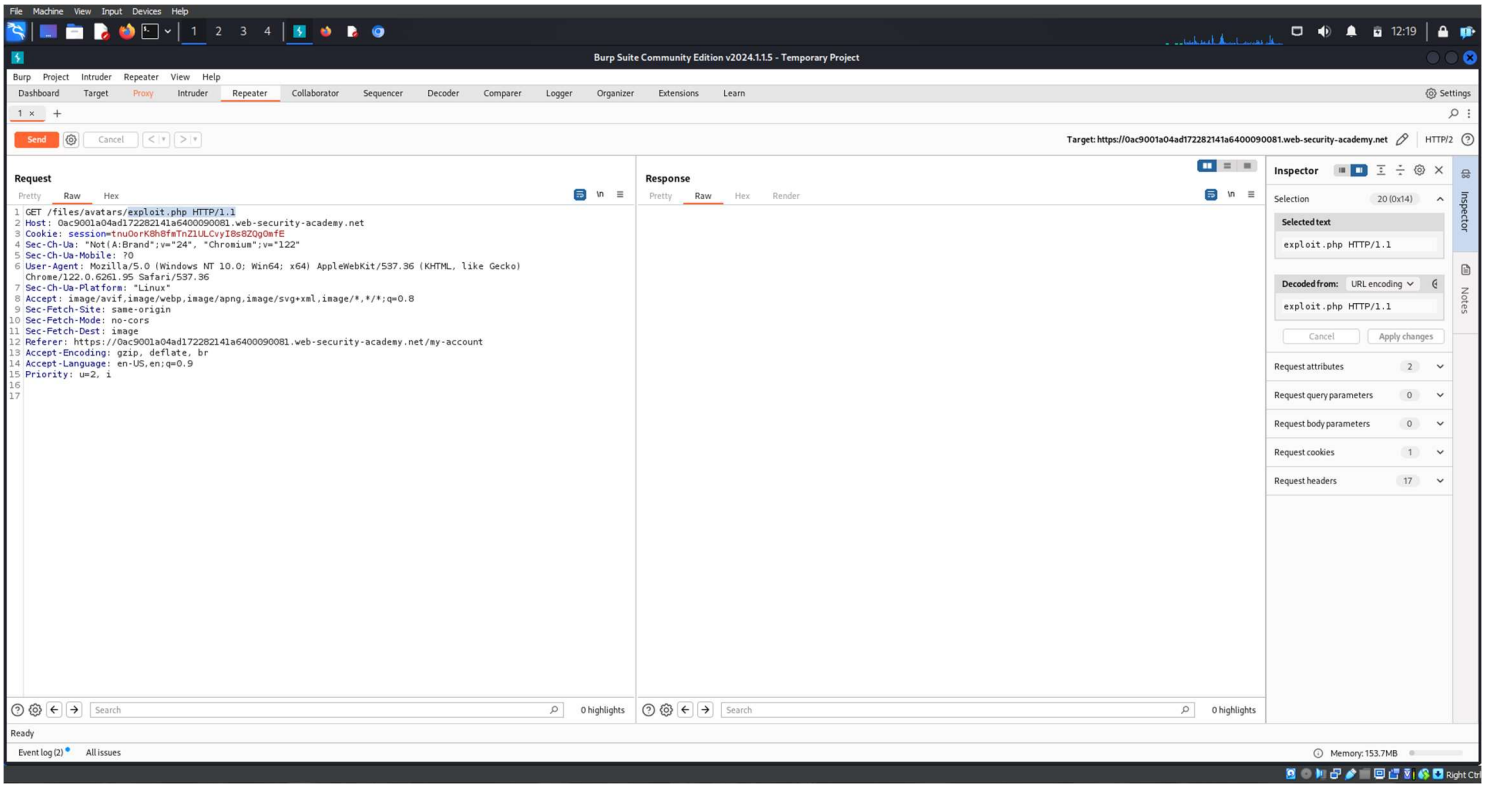
Step 4: On your system, create a file called exploit.php, containing a script for fetching the contents of Carlos's secret file.



Step 5: Use the avatar upload function to upload your malicious PHP file. The message in the response confirms that this was uploaded successfully.

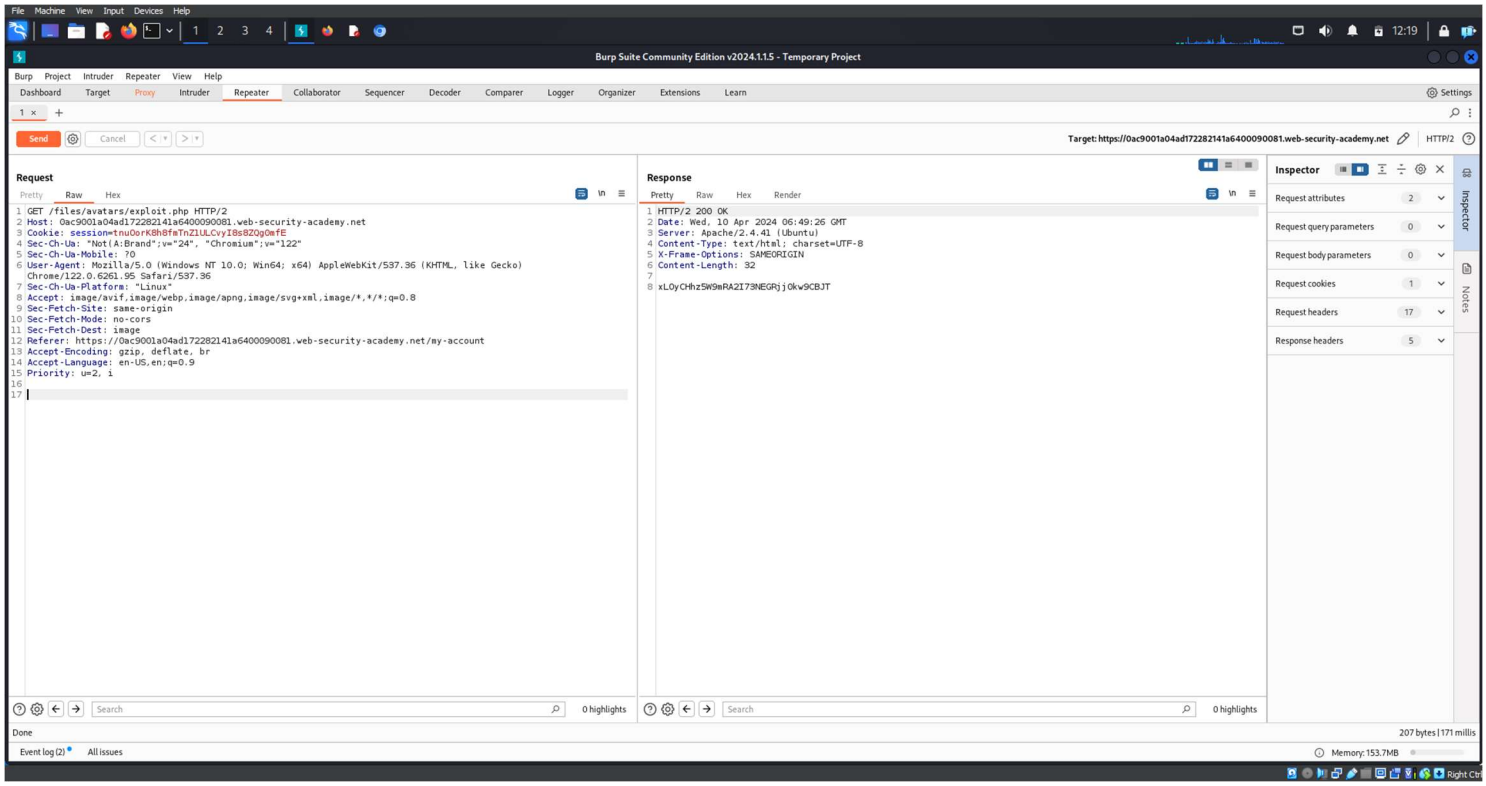


Step 6: In Burp Repeater, change the path of the request to point to your PHP file:

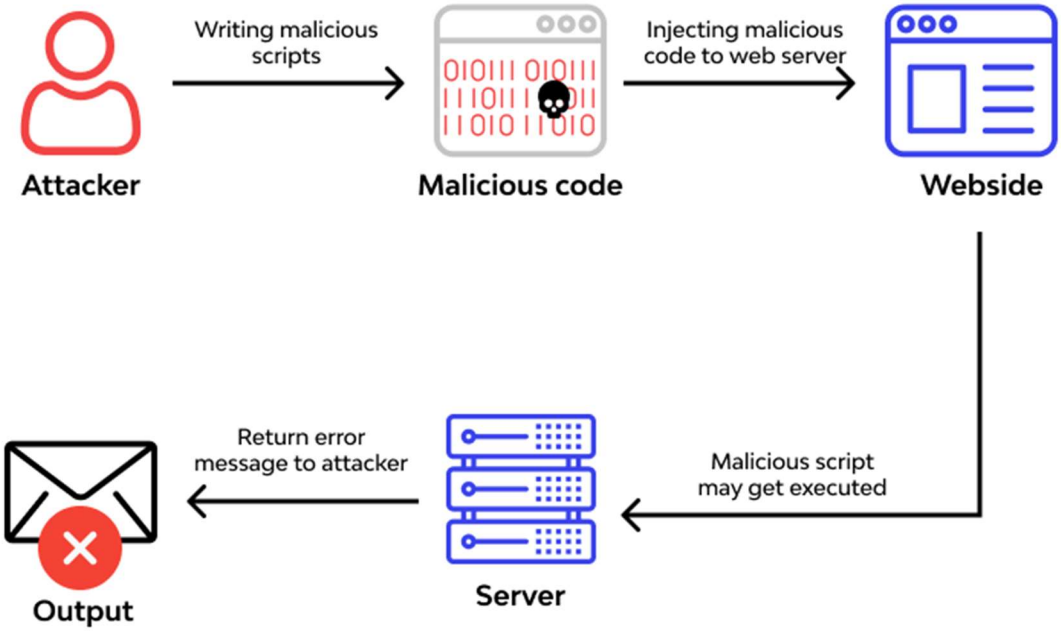


8th April 2024

Step 7: Send the request. Notice that the server has executed your script and returned its output (Carlos's secret) in the response. We successfully got the secret in response hence task completed successfully



7- Flow Diagram:



8- Recommendation: -

1- **Implement Strict File Type Validation:** Ensure that the image upload functionality strictly validates file types to only allow legitimate image files. Use server-side validation mechanisms and file type detection libraries to verify uploaded files.

2- **Secure File Permissions**: Set appropriate file permissions on the server to restrict access to sensitive directories and files. Limit the permissions of uploaded files to prevent execution of malicious scripts.

3- **Content Disposition Headers:** Set Content-Disposition headers to prevent browsers from executing uploaded files. This can help mitigate risks associated with file execution vulnerabilities.

4- **Use Content Security Policy (CSP)**: Implement CSP headers to restrict the sources from which content, such as scripts and stylesheets, can be loaded. This can help prevent cross-site scripting (XSS) attacks.

5- **Regular Security Audits**: Conduct regular security audits and penetration testing of the web application to identify and remediate vulnerabilities proactively.

6- **Secure Coding Practices**: Train developers on secure coding practices to avoid common security pitfalls, such as inadequate input validation and improper file handling.

9- Conclusion: -

In conclusion, the flow diagram highlights the potential risks associated with vulnerabilities in the image upload functionality of web applications. Exploiting these vulnerabilities can lead to unauthorized access to servers, execution of arbitrary commands, and exfiltration of sensitive data. To mitigate these risks, it is essential for organizations to implement robust security measures, including strict file type validation, secure file permissions, regular security audits, and adherence to secure coding practices. By prioritizing security measures and staying vigilant against emerging threats, organizations can bolster the resilience of their web applications and safeguard against potential exploitation.

References:-

1- Web security Academy

2- Lab: Remote code execution via web shell upload | Web Security Academy (portswigger.net)

3- 8.1 Lab: Remote code execution via web shell upload | 2023 | by Karthikeyan Nagaraj | Medium