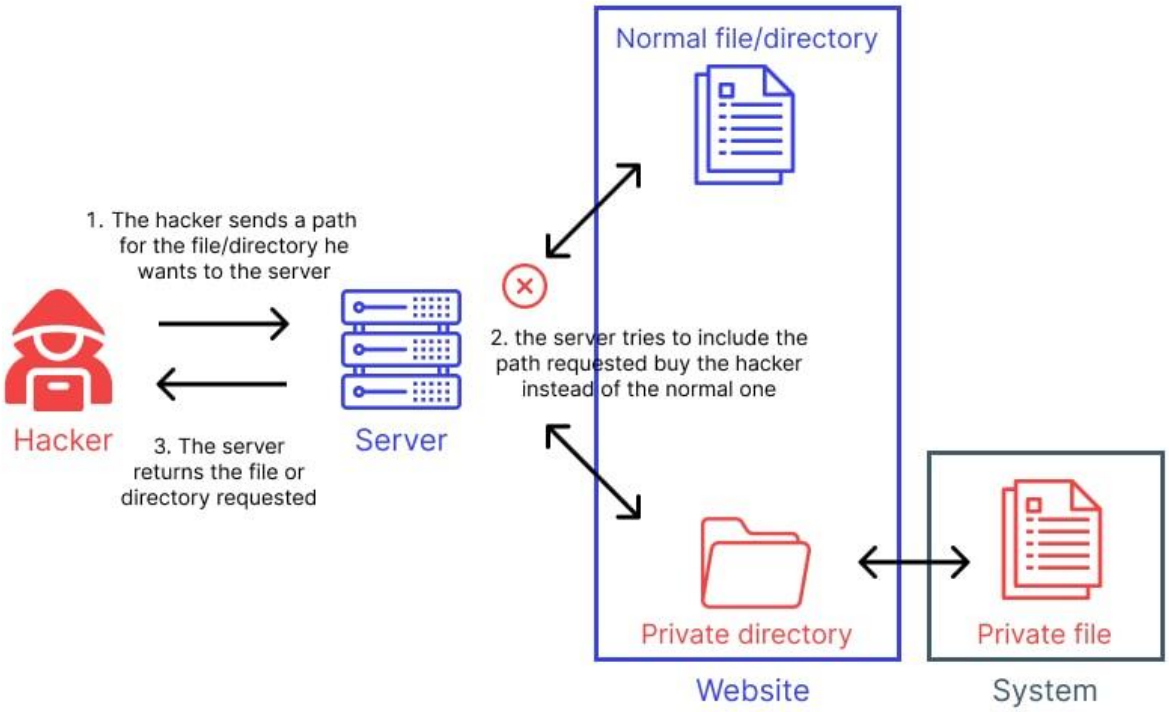
A Deep Dive into File Path Traversal, Inclusion, and Upload Vulnerabilities

Lurking beneath the polished veneer of websites, complex software code can harbor hidden vulnerabilities waiting to be exploited. Among these digital landmines lie file path traversal, file inclusion, and file upload vulnerabilities, posing significant threats to data security and website integrity. This report delves into these intricate threats, exposing their nature, dissecting their impact, and equipping you with the knowledge and strategies to secure your online journeys.

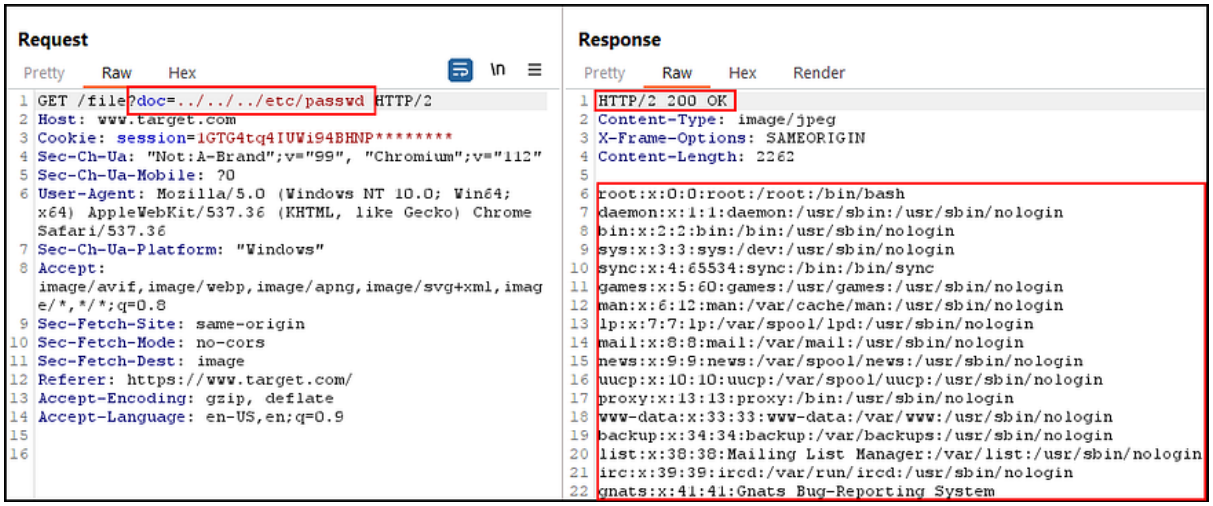
File Path Traversal

Imagine a bustling city with streets and avenues, each leading to a specific building. File path traversal (FPT) vulnerabilities are akin to manipulating street signs, tricking the city explorer (the application) into venturing into forbidden alleys and accessing unauthorized buildings (files) beyond intended locations. Attackers exploit user-controlled input – an address with malicious twists – to manipulate the application's internal navigation system, enabling them to access sensitive data, credentials, even source code residing in restricted areas.



How to Find the Vulnerability

- **Target Input Fields:** Focus on areas where users provide file paths or URLs, such as:
    - File upload forms
    - File download links
    - URL parameters
    - Search bars
    - User profile settings
- **Inject Malicious Traversal Sequences:** Attempt to access unauthorized files by appending sequences like ../ or %2e%2e/ to input fields. For example, instead of uploading "image.jpg", try "../../../../etc/passwd".
- **Observe Responses:** Monitor for unexpected error messages, file content disclosures, or successful retrieval of files outside the intended directory.



Path Traversal Bypass:

- **UTF-8 Encoding**
    - Attackers can leverage multiple percent encodings or overlong encodings to represent traversal characters like ../ or \.
    - Multiple percent encodings: %c1%1c (equivalent to /)

    - Overlong encodings: %c0%ae (equivalent to .)
- **UTF-7 Encoding**
    - ASCII characters can be embedded within Unicode blocks to bypass standard validation processes.
    - Embedding ASCII characters: +ADw- (equivalent to <)
- **URL Encoding**
    - Percent-encoding characters like %2e%2e%2f (equivalent to ../) can be used if applications fail to sanitize URL-encoded input.
    - Percent-encoded traversal: %2e%2e%2f (equivalent to ../)
- **Double URL Encoding**
    - Double encoding traversal sequences can potentially evade filters and firewalls.
    - Double-encoded slash: %252F
- **16-bit Unicode (UTF-16) Encoding**
    - Traversal characters can be represented using UTF-16 encoding, which is used in certain operating systems and APIs.
    - Traversal characters: %u002e (equivalent to .)
- **NULL Bytes**
    - Null bytes (%00) can sometimes be used to bypass file type checks by truncating file names.
    - Truncating file names: ../../../../../passwd%00.jpg
- **Mangled Paths**
    - If filters for traversal sequences aren't applied recursively, attackers can bypass them by nesting sequences within each other.
    - Nesting traversal sequences: ....//, ...\//, ..//..//..\

Impact of FPT:

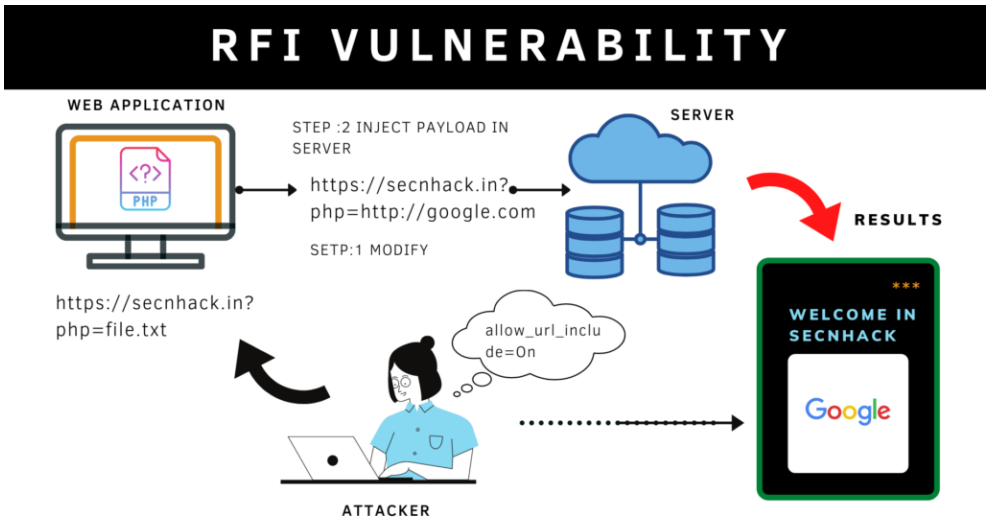
- Data Exfiltration: Sensitive information like user data, financial records, and confidential business documents can be extracted and used for nefarious purposes like identity theft, fraud, and blackmail.
- System Compromise: Gaining access to critical system files allows attackers to inject malware, manipulate website content, or even orchestrate complete server takeovers, unleashing chaos and disruption.
- Information Disclosure: Leaking internal documents or logs can expose confidential information, reveal vulnerabilities, and damage an organization's reputation.

File Inclusion Vulnerability

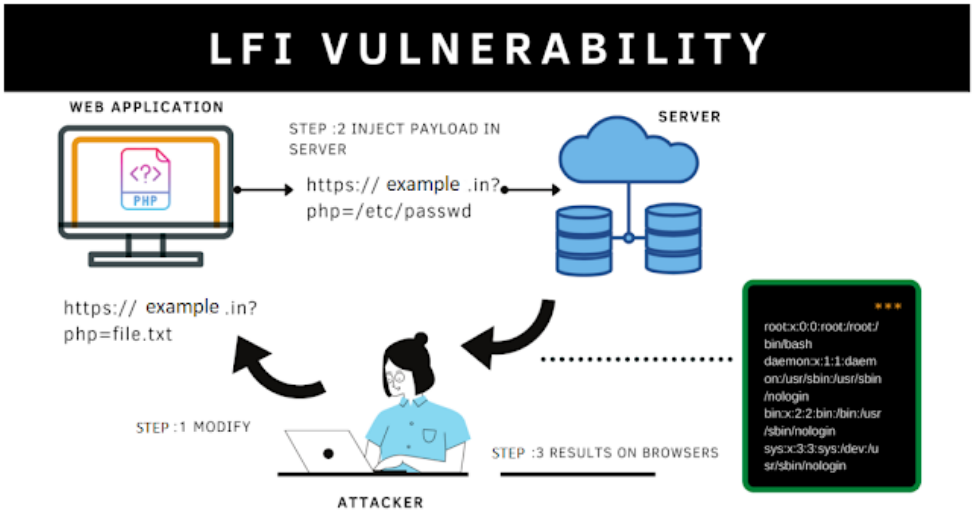
Think of a recipe where ingredients can be dynamically added depending on your preferences. File inclusion vulnerabilities introduce a sinister twist to this culinary analogy. Applications may dynamically include user-controlled code or files based on external input, essentially allowing attackers to add their own nefarious ingredients (malicious code) to the recipe. This results in unexpected and often harmful outcomes, impacting the website's functionality and jeopardizing data security.

Types of File Inclusion Vulnerabilities:

- Remote File Inclusion (RFI): Malicious code is included from a remote server controlled by the attacker, granting them remote control over the application's execution. Imagine ordering contaminated ingredients from a remote vendor, poisoning the entire dish (the website



- Local File Inclusion (LFI): Malicious code is included from the server's own file system. Like using tainted spices from your pantry, this can still compromise the final dish (the website) even if the source seems familiar.



Impact of File Inclusion Vulnerabilities:

- Code Execution: Injected code can run on the server, granting attackers control over critical functions, manipulating data, or even launching further attacks. Think of the recipe turning the oven into a bomb instead of baking a cake.
- Denial-of-Service (DoS): Malicious code can consume excessive resources, crashing the website and rendering it inaccessible to legitimate users. Imagine adding so much salt that the entire kitchen (the server) explodes.
- Data Theft and Manipulation: Attackers can access, steal, or modify sensitive data stored on the server, compromising user privacy and potentially causing financial or reputational damage.

How to find File inclusion

- Identify Potential Entry Points:
    - Focus on areas where users provide file paths or URLs, such as:
        - File upload forms
        - File download links
        - URL parameters
        - Search bars
        - User profile settings
- Inject Traversal Sequences:
    - Attempt to traverse the file system using sequences like ../ or %2e%2e/ to access unauthorized files.
    - Examples:
        - In a file upload form, try uploading "../../../../etc/passwd" instead of a legitimate image.
        - In a URL parameter, inject ../ to see if you can access files outside the intended directory.

- Observe Responses:
    - Monitor for:
        - Unexpected error messages
        - Disclosure of file content
        - Successful retrieval of files outside the intended directory

Difference between LFI and RFI

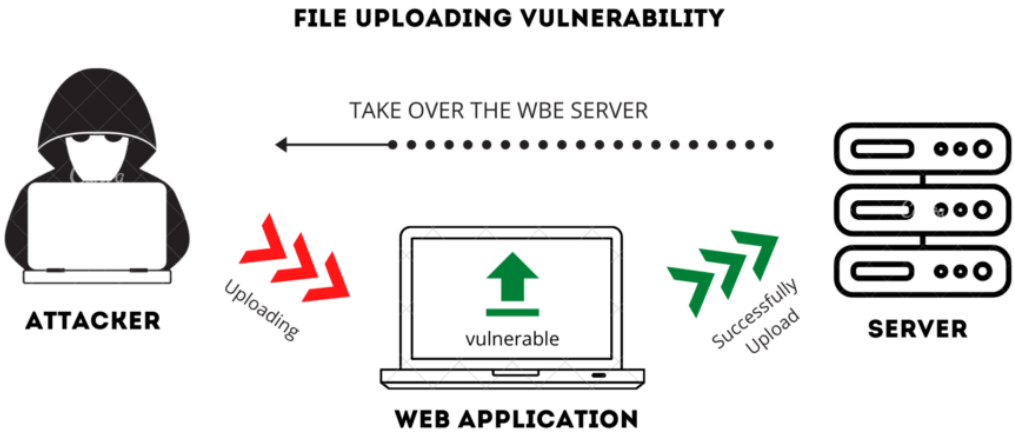
Parameter	LFI	RFI
Source	The attacker includes a file from the server's own file system.	The attacker includes a file from a remote server controlled by them.
Exploitation	User-controlled input with crafted paths or filenames	User-controlled input with URLs pointing to malicious file
Impact	Data exfiltration, system compromise, DoS attacks	Same as LFI, plus potential server dependency and third-party code injection

Difference between File Inclusion and Path Traversal

Parameter	File Inclusion	Path Traversal
Goal	Execute malicious code	Access and read files
Mechanism	Include unintended files	Traverse directory structure
Impact	Remote code execution, data theft, system compromise	Exposure of sensitive information

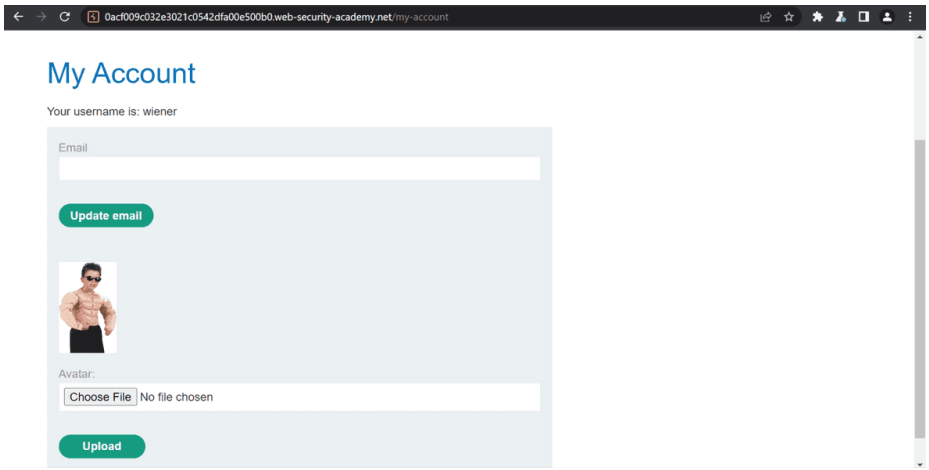
File Upload Vulnerability

Websites often allow users to upload files, creating a seemingly innocent avenue for interaction. However, this digital door can swing open to reveal vulnerabilities when uploaded files are not properly sanitized or validated. Think of an online marketplace where anyone can bring any item without inspection. Attackers can exploit this to introduce malicious content disguised as harmless files, causing significant damage.

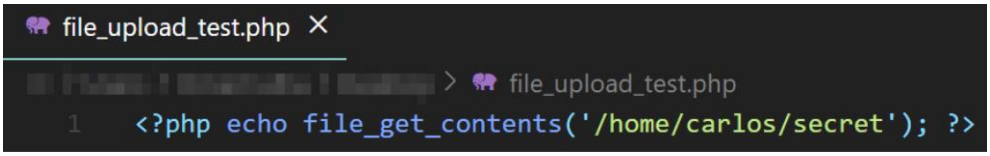


How to find the file upload vulnerability

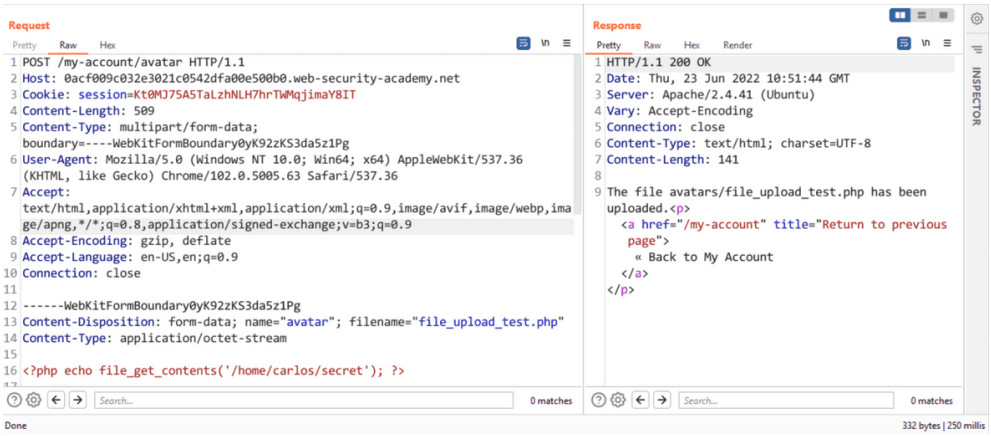
1. Identify Vulnerable Application: Attacker locates a web application with an unsecure file upload feature.



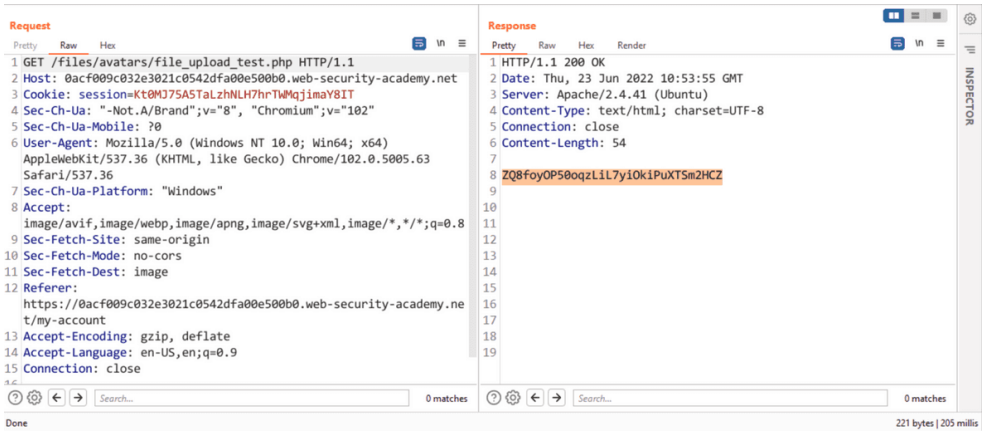
2. Craft Malicious File: Attacker prepares a file containing malicious code or scripts.



3. Upload File: Attacker submits the malicious file through the application's upload form.



4. Exploit Weak Validation: If the application fails to validate and sanitize the file properly, the attacker's file is stored on the server.



5. Trigger Execution: Attacker accesses the uploaded file through a URL or other means, triggering the execution of the malicious code.

Bypass

1. **File extension**

   Developers may blacklist specific file extensions and prevent users from uploading files with extensions that are considered dangerous. This can be bypassed by using alternate extensions or even unrelated ones. For

example, it might be possible to upload and execute a .php file simply by renaming it file.php.jpg or file.PHp.

Alternate extensions

Type	Extension
php	phtml, .php, .php3, .php4, .php5, and .inc
asp	asp, .aspx
perl	.pl, .pm, .cgi, .lib
jsp	.jsp, .jspx, .jsw, .jsv, and .jspf
Coldfusion	.cfm, .cfml, .cfc, .dbm

2. **MIME type**

   Blacklisting MIME types is also a method of file upload validation. It may be bypassed by intercepting the POST request on the way to the server and modifying the MIME type.

   Normal php MIME type:

   `Content-type: application/x-php`

   Replace with:

   `Content-type: image/jpeg`

Impact of File Upload Vulnerability:

- Malware Deployment: Uploadable files can be laced with malware that infects server and user systems upon upload. Imagine buying a seemingly harmless apple at the market, only to discover it's a robotic worm that starts infecting everything it touches.
- Phishing Attacks: Malicious files disguised as legitimate documents can be used for phishing attacks to steal user credentials or sensitive information. Picture a box labeled "important documents" containing forged tax forms used to trick users into revealing personal data.

- DoS Attacks: Uploading large or resource-intensive files can consume server resources, causing outages and disrupting legitimate user access. Imagine flooding the marketplace with an infinite number of watermelons, preventing anyone from entering or buying anything.

Mitigation Strategies

Combating these vulnerabilities demands a multi-layered defense:

- Input Validation: Thoroughly sanitize all user-controlled input, filtering out malicious characters and paths that could manipulate file system navigation. Think of installing security cameras and scanners at the market to identify and remove dangerous items before they enter.
- Restrict File Types: Limit allowed file types for uploads to only those necessary for website functionality. This minimizes the risk of unexpected and potentially harmful content entering the system. Imagine only allowing apples and oranges at the market,

Reference

- https://medium.com/@ch3t4nn/path-traversal-vs-file-inclusion-vulnerability-how-to-tell-the-difference-cdac2a129c88
- https://vulp3cula.gitbook.io/hackers-grimoire/exploitation/web-application/file-upload-bypass
- https://portswigger.net/web-security/file-upload