Obfuscation of File Extension

Introduction

This scenario focuses on bypassing file upload validation by **obfuscating file extensions**. Many applications rely on native filename checks, often case-sensitive or based on incomplete filtering logic. If the backend later normalizes or interprets the filename differently, this mismatch can allow attackers to upload files that should have been blocked.

When validation inspects only the raw filename and not the normalized or decoded path, an attacker can disguise a dangerous file so that validation accepts it, while the server later treats it as an executable script.

This class of vulnerability occurs when the server:

- performs **weak, case-sensitive or partial extension checks**,
- fails to normalize filenames before validating them,
- decodes them only after storing or processing the file,
- or strips prohibited extensions **without applying this transformation recursively**.

Such discrepancies enable bypasses that may result in remote code execution.

Examples

- **case obfuscation** - if validation is case-sensitive, files such as `exploit.pHp` may bypass the filter, but the server may still execute them as `.php`.
- **multiple extensions** - depending on how the filename is parsed, a file like `exploit.php.jpg` may be treated as an image during validation but executed as PHP by the server.
- **trailing characters** - some systems ignore trailing dots or spaces:
  - `exploit.php.` $\rightarrow$ effectively becomes `exploit.php`.

- **URL-encoding (single or double)** - if validation doesn't decode the filename, but the server later does:
  - `exploit%2Ephp` → becomes `exploit.php` server-side.
- **semicolons or null bytes** - high-level languages may read the full filename, while low-level handlers stop at a semicolon or null byte:
  - `exploit.asp;.jpg`
  - `exploit.asp%00.jpg`
- **non-recursive stripping of dangerous extensions** - if `.php` is removed only once, filenames like: `file.php.pHp` may still resolve to a valid executable extension after stripping.

LAB

goal: examine a vulnerable file upload mechanism that is vulnerable to extension obfuscation.
requirements: Internet connection, BurpSuit, PortSwigger account
LAB: link

Hints

hint 1: find unrestricted file upload function and try to exploit it.

hint 2: based on last lab, prepare payload file and upload it.

hint 3: send to `Repeater` file upload `POST` and try to obfuscate extension.

hint 4: send `GET` request for payload file to `Repeater` and use it to execute `PHP` code.