File Uploading

↳ This vulnerability occurs when the application support file uploading features (profile pic update etc.,) which is saved to server without any validation

Filtering :-

↳ client side filtering
↳ Server side filtering

client side filtering:

↳ validation done in client side level so easy to bypass.
↳ most of the times allowed extensions are whitelisted or disallowed extensions are blacklisted.

Filters can be applied on

changing content-type bypass → * Filetype Filtering - by MIME type (Content-Type)
changing file headers bypass → * Magic number validation - by file headers [hex editor <file name>]
* File name filtering - by whitelisting / blacklisting allowed / disallowed characters in file name
* File length filtering - by size of file
* File content filtering - Scan the content of file & Filter

↳ Sometimes, on client side (JS is used to apply filter) (i) It can be bypassed by intercepting the request in burp and drop the (.js) file if external javascript is used and forward the remaining request.

To intercept the external javascript in burp

Note :
Goto burp intercept → intercept options → edit → remove the $js and ok
Burp usually doesn't intercept javascript file by default.

ii) If javascript code is within html, then do intercept the response to the request and edit the html response (delete or comment) and forward the requests.

↳ If client side file type filtering is applied, then it can be bypassed by changing the Content-Type to the accepted one. eg:) If app allows .jpeg then, changing the content type of our php shell from text/x-php to img/jpg

Server-side filtering :-

↳ Filtering can be done at server side. Since code is not accessible to Attacker, it is hard to bypass.

double extension —bypass→ * Filter by extension — Same as above —bypass→ using different type of extensions like phtml, php-s, phar for PHP

changing file headers —bypass→ * Filter by Magic numbers — Same as above → using hex editor.

Note: In php version 5, we can bypass the extension filter by adding null byte to the end of the file extension
eg:) shell.php%00.jpg

Sometimes, developer whitelist/blacklist accepted file extension. (i.e) if shell.php is blacklisted. Then shell.php-s (or) shell.phtml is used. which will bypass the filter

How to Approach File upload Vulnerability :

i) Find the Technologies which the Website is built with → Wappalyzer
Sometimes result may inaccurate, try to find manually from header like server, x-powered-by using intercepting request from Burp.

ii) Find the upload directory (the place where our files are stored) Using tools like gobuster. eg:) gobuster dir -u <url> -w <wordlist> -t <threads>

ii) If client side filter, then go through the source code & try to bypass. We can take control on client side.

iii) If Server side filter,

↳ If you can successfully upload a file with a totally invalid file extension (e.g. testingimage.invalidfileextension) then the chances are that the server is using an extension blacklist to filter out executable files. If this upload fails then any extension filter will be operating on a whitelist.

↳ Try re-uploading your originally accepted innocent file, but this time change the magic number of the file to be something that you would expect to be filtered. If the upload fails then you know that the server is using a magic number based filter.

↳ As with the previous point, try to upload your innocent file, but intercept the request with Burpsuite and change the MIME type of the upload to something that you would expect to be filtered. If the upload fails then you know that the server is filtering based on MIME types.

↳ Enumerating file length filters is a case of uploading a small file, then uploading progressively bigger files until you hit the filter. At that point you'll know what the acceptable limit is. If you're very lucky then the error message of original upload may outright tell you what the size limit is. Be aware that a small file length limit may prevent you from uploading the reverse shell we've been using so far.

iv) After uploading, try to access the uploaded file. Sometimes, Servers can use different file names than your upload one.

Here, use gobuster to find the file name in the upload directory which we found in above steps.

eg) gobuster dir -u <url> -w <wordlist> -t <threads> -x .php, phar

(here we did n't know file name of the file which server changes) → Extension to search

v) Then Finally, access the file.

found in payloadallthings github repo

Note:

↳ We can use both web-shell or reverse shell according to your needs

↳ Combination of filtering methods can be used.