Nice one 👍 let’s document **WSTG notes for “Test Upload of Malicious Files” (WSTG-BUSL-09)**.

**📝 Notes: Test Upload of Malicious Files**

**🎯 Purpose**

To verify whether the application **properly defends against malicious files** uploaded by attackers.

* Even if file type validation exists, attackers may upload files that **execute code, deliver malware, or exploit client-side weaknesses**.

**⚡ Common Attack Scenarios**

1. **Web Shell Upload**
   * Uploading a .php, .jsp, .asp, .aspx file → attacker executes commands remotely.
2. **Malicious Document Files**
   * Uploading .docm, .xlsm, .pdf with embedded macros or exploits → infects users when opened.
3. **Polyglot Files**
   * A file valid in **multiple formats** (e.g., image + JavaScript) → bypasses filters but executes in browser.
4. **Client-Side Malware Hosting**
   * Uploading .exe, .apk, .bat, .ps1 → tricking users into downloading malware.
5. **Stored XSS in File Metadata**
   * Malicious payload in **EXIF data of images** or PDF metadata → triggers in web viewer.

**🔍 How to Test**

1. **Attempt Dangerous File Uploads**
   * Executable files: .exe, .dll, .apk, .bat, .sh, .ps1.
   * Scripts: .php, .jsp, .asp, .html, .js.
2. **Check Storage & Access**
   * Does the file get stored in **web root** and accessible directly?
   * Try accessing https://target.com/uploads/malicious.php.
3. **Tamper File Headers**
   * Modify Content-Type to bypass filters.
   * Example: upload PHP shell but set Content-Type: image/png.
4. **Test Polyglot Payloads**
   * Example: image with appended PHP code or HTML inside PDF.
5. **Test Metadata Injection**
   * Insert <script>alert(1)</script> in EXIF fields of an image.

**🛡️ Mitigation**

* Enforce **strict server-side validation** (extensions + MIME + content inspection).
* Store files **outside web root** (never directly executable).
* Rename files to **random UUIDs**.
* Use **file scanning (AV, sandboxing)** for malicious content.
* Strip **metadata and active content** from uploaded documents/images.
* Forbid upload of **scripts and executables** unless business-critical.
* Apply **least privilege** → uploaded files shouldn’t inherit execution rights.

✅ **Key Takeaway:**  
Even allowed file types may carry **malicious payloads**. Without strong controls, attackers can gain **RCE, malware distribution, or stored XSS** through file uploads.

👉 Do you want me to also prepare a **red team style payload list** (sample malicious PHP shells, polyglot examples, EXIF XSS strings) so you can practice hands-on?