Alright 👍 here are **precise WSTG notes for “Testing for Client-side URL Redirect” (WSTG-CLNT-04)**.

**📝 Testing for Client-side URL Redirect**

**🎯 Purpose**

To check if the application uses **client-side logic (JavaScript, meta refresh, DOM manipulation)** to redirect users, and whether this can be **manipulated by attackers** to perform:

* **Phishing attacks** (redirecting to malicious site)
* **Open redirect exploitation** (used in OAuth token theft, spam, social engineering).

**⚡ Common Attack Scenarios**

1. **JavaScript-based redirect**
2. window.location = "https://trusted.com/home";
3. window.location.href = getParameter("url");
4. **Meta refresh redirect**
5. <meta http-equiv="refresh" content="0;url=https://trusted.com">
6. **DOM-based redirect**
   * App reads query param ?redirect=... and uses it for navigation:
   * https://victim.com/login?redirect=https://attacker.com
7. **OAuth / SSO abuse**
   * Manipulating redirect\_uri parameter to steal tokens.

**🔍 How to Test**

1. **Identify Redirect Parameters**
   * Look for params like redirect=, url=, next=, goto=, dest= in requests.
2. **Inject Payloads**
   * Replace target with malicious domain:
   * https://victim.com/login?redirect=https://attacker.com
3. **Check Bypass Filters**
   * Some apps only check for substrings → try tricks:
   * //attacker.com
   * https:attacker.com
   * https://trusted.com@attacker.com
   * /\/attacker.com
4. **Observe Redirect Mechanism**
   * Is it purely client-side (JavaScript/meta refresh)?
   * Or server-side 302 redirect (that’s WSTG-BUSL-08 instead)?
5. **Use Tools**
   * Burp Repeater → modify redirect params.
   * Browser DevTools → watch network + JavaScript execution.

**🛡️ Mitigation**

* Never trust client-side parameters for redirects.
* Use **allow-lists** of safe redirect URLs.
* For OAuth → strictly validate redirect\_uri.
* If redirect is required, use **server-side mapping** (e.g., /goto?id=1 → https://trusted.com).
* Display redirect confirmation to users.

✅ **Key Takeaway:**  
Client-side URL redirects let attackers **control navigation flow** via parameters or DOM manipulation → enabling **phishing, token theft, and open redirect abuse**. Always enforce **server-side validation with allow-lists**.

👉 Do you want me to also prepare a **list of common bypass tricks (like @, //, encoding) for open redirect fuzzing** so you can test them quickly?