Perfect 👍 Let’s cover **precise notes for “Testing for Cross-Site Script Inclusion (XSSI)” (WSTG-CLNT-08)**.

**📝 Testing for Cross-Site Script Inclusion (XSSI)**

**🎯 Purpose**

To check if sensitive data (e.g., JSON, config files, API responses) can be **stolen by including them as scripts** in an attacker’s page.

* Unlike XSS, here the **attacker doesn’t inject JS into the victim site**, but instead **tricks the victim’s browser** into running trusted site’s JS/data in attacker’s page.

**⚡ Common Attack Scenarios**

1. **Sensitive JSON API without CSRF/XSSI protections**
   * Victim authenticated → attacker includes:
   * <script src="https://target.com/api/userinfo"></script>
   * If response is valid JS/JSON → attacker can access or leak it.
2. **Array/Function Wrapping Attacks**
   * If server returns JSON like:
   * ["username","email","token"]
   * Attacker can wrap:
   * <script src="https://target.com/api/data"></script>
   * <script>
   * alert(data[0]); // Access victim’s info
   * </script>
3. **JSONP Abuse**
   * Legacy APIs with callback= parameter:
   * https://target.com/api/user?callback=evil
   * Attacker defines function evil() → sensitive data is leaked.

**🔍 How to Test**

1. **Identify Candidate Endpoints**
   * Look for API endpoints returning **JSON, JavaScript, config, tokens**.
   * Common paths: /api/, /data/, /config/.
2. **Try Script Inclusion**
   * On attacker page:
   * <script src="https://target.com/api/user"></script>
   * Observe browser console → check if data loads.
3. **Check for Executable JSON**
   * If response is **not wrapped** (pure JSON), browser throws error but attacker may still leak using **side-channel techniques**.
   * If response **is valid JavaScript (JSONP, var assignment)** → high risk.
4. **Test callback / jsonp Parameters**
   * Modify request:
   * ?callback=evil
   * See if server wraps response.

**🛡️ Mitigation**

* **Never return raw JSON/JS without protections.**
* Use **CSRF tokens** or SameSite cookies to prevent cross-site inclusion.
* Return JSON with **Content-Type: application/json**, not text/javascript.
* Avoid JSONP — replace with **CORS + proper headers**.
* Prefix JSON with **non-executable strings** (e.g., )]}',\n) to prevent execution when included as a script.

✅ **Key Takeaway:**  
XSSI allows attackers to **steal authenticated JSON/JS data** by including it as a script.  
Fix it by **disabling JSONP, enforcing CSRF protections, and returning pure JSON with correct headers.**

👉 Do you want me to also create a **mini attack PoC HTML template** that you can reuse for testing XSSI quickly (just swap out target URL)?