Great! Now we’re at **WSTG-INPV-01: Testing for Reflected Cross-Site Scripting (XSS)** 🚀

**🧪 Testing for Reflected Cross-Site Scripting (XSS)**

**🔎 What is Reflected XSS?**

* **Reflected XSS** happens when **user-supplied input is immediately returned in the HTTP response** without proper validation or encoding.
* Example: a search box that reflects your query back into the page.
* Unlike stored XSS, payloads are **not saved on the server** → they only work when the malicious link is clicked.

**⚡ Example Attack Flow**

1. Attacker crafts a malicious URL:
2. https://vulnerable.com/search?q=<script>alert('XSS')</script>
3. Victim clicks it → server reflects q parameter back into the page without sanitization.
4. Victim’s browser executes the script → attacker can:
   * Steal cookies/session (document.cookie)
   * Redirect users (window.location)
   * Load external scripts (keyloggers, phishing UI, etc.)

**🔧 How to Test (Step-by-Step)**

**1. Manual Testing with Payloads**

* Inject classic payloads in parameters or input fields:
* <script>alert(1)</script>
* <img src=x onerror=alert(1)>
* '"><svg/onload=alert(1)>
* If alert box pops → XSS exists.

**2. Burp Suite Testing**

* Send request to Burp Repeater.
* Modify parameters (query strings, headers, cookies, POST body).
* Observe reflection in HTML/JS context.

**3. Automated Scanning**

* Use **Burp Active Scanner** or **OWASP ZAP** to fuzz input points.
* Tools like XSStrike or Dalfox specialize in XSS fuzzing.

**4. Context-Aware Testing**

Check how input is reflected:

* Inside HTML: <p>Hello {input}</p>
* Inside JavaScript: var name = "{input}";
* Inside attributes: <input value="{input}">
* Different contexts require **different payloads**.

**✅ Mitigations**

* **Input Validation**: whitelist expected input.
* **Output Encoding**:
  + HTML encode (< → &lt;) when outputting in HTML.
  + JavaScript escape when injecting into scripts.
  + Attribute encode when inserting into attributes.
* **Content Security Policy (CSP)** to block inline JS.
* **HttpOnly Cookies** → prevent theft via JS.

⚡ **Summary:**

Reflected XSS = input is immediately echoed in response.  
Testing = inject payloads in parameters/inputs, use Burp/ZAP, check context.  
Defense = proper input validation + output encoding + CSP.

👉 Do you want me to also write **ready-to-use XSS payload cheat sheet (basic → advanced)** so you can plug it into Burp/ZAP while testing?