Great, this one ties into **WSTG-SESS: Session Management Testing** 🔐

**🧾 Session Management Schema**

**1. What is Session Management?**

* A **session** is a server-maintained mechanism to identify and track a user after login.
* A **Session Management Schema** defines how sessions are:
  + **Created** (at login)
  + **Maintained** (via cookies, tokens, URL params, headers)
  + **Destroyed** (logout, timeout, invalidation)

**2. Typical Session Flow**

1. **User Authentication**
   * User provides username/password.
   * Server validates credentials.
2. **Session Token Generation**
   * Server issues a **session ID (SID)** or **JWT (JSON Web Token)**.
   * Example:
   * Set-Cookie: JSESSIONID=abc123; HttpOnly; Secure
3. **Session Maintenance**
   * Token is sent with each request:
     + **Cookie** → Most common.
     + **URL parameter** → Weak (can leak via referrer/logs).
     + **Hidden form field**.
     + **Authorization Header** → For APIs (e.g., JWT, OAuth).
4. **Session Termination**
   * Explicit logout (user clicks "Log out").
   * Idle timeout (inactivity).
   * Absolute timeout (max lifespan).

**3. Security Considerations**

**🔹 Session ID Generation**

* Must be **random, unique, unpredictable**.
* Weak IDs → attackers can guess/brute force.

**🔹 Session Transmission**

* Always over **HTTPS**.
* Use **Secure** and **HttpOnly** cookie flags.
* Avoid exposing session IDs in **URLs**.

**🔹 Session Storage**

* Server stores session data in memory, DB, or file.
* Client only stores **reference (token)**, not sensitive data.

**🔹 Session Termination**

* Logout should **invalidate session on server side**, not just clear client cookies.
* Protect against **session fixation** (force new SID on login).

**4. Common Attacks on Sessions**

* **Session Hijacking** → Attacker steals session ID (via XSS, sniffing).
* **Session Fixation** → Attacker sets a known SID for victim.
* **Session Replay** → Reuse of valid token.
* **Weak Expiration** → Sessions never expire → long-term hijack risk.

**5. Testing the Session Schema (WSTG-SESS)**

* **Check Session Tokens**
  + Predictability? (Burp Sequencer)
  + Flags? (Secure, HttpOnly, SameSite)
* **Check Session Lifecycle**
  + Does login issue a new SID?
  + Does logout kill the old SID?
* **Check Timeout Policies**
  + Idle timeout enforced?
  + Absolute session timeout?
* **Check Token Storage**
  + Stored in cookies (preferred)?
  + Stored in localStorage/sessionStorage (risk of XSS)?

**6. Best Practices**

✅ Strong, random session IDs (128+ bits entropy).  
✅ Use **Secure**, **HttpOnly**, **SameSite=Strict** flags.  
✅ Regenerate SID after login and privilege escalation.  
✅ Enforce idle & absolute timeouts.  
✅ Invalidate server-side session on logout.  
✅ Avoid exposing tokens in URLs.

✅ **Summary:**  
The Session Management Schema defines how authentication states are handled in an application. A secure schema uses strong, unpredictable tokens, transmits them only over HTTPS, protects them with cookie flags, and enforces strict expiration & termination policies. Weak schema = easy session hijacking.

👉 Do you want me to next break this down into a **Burp Suite testing workflow** (how to capture, analyze, and attack sessions) so you can practice step by step?