Perfect 👍 Let’s break down **WSTG notes for “Test for Process Timing” (WSTG-BUSL-05)**.

**📝 Notes: Test for Process Timing**

**🎯 Purpose**

To check whether the application’s **business processes depend on timing assumptions** that an attacker can exploit.  
Timing flaws often arise when:

* Operations are executed **faster/slower than expected**, or
* Users can **trigger multiple requests in parallel** to abuse logic.

**⚡ Common Attack Scenarios**

1. **Race Conditions / TOCTOU (Time-of-Check vs Time-of-Use)**
   * Attacker submits multiple purchase/payment requests in parallel → receives goods/services multiple times but pays once.
2. **Coupon / Voucher Abuse**
   * Submitting the same discount code simultaneously before it’s marked “used.”
3. **Double-Spending**
   * Banking/fintech apps failing to lock balances during transaction processing.
4. **Session Expiry Gaps**
   * Performing actions after timeout but before backend fully clears session.
5. **Order Processing Delays**
   * Cancelling/refunding while the original order is still being processed, gaining both money and goods.

**🔍 How to Test**

1. **Identify Critical Processes**
   * Payment, refund, coupon redemption, fund transfer, password reset.
2. **Send Concurrent Requests**
   * Use **Burp Intruder**, **OWASP ZAP**, curl loops, or race testing tools like **Turbo Intruder**.
3. seq 1 20 | xargs -I % -P 20 curl -X POST https://target.com/checkout -d "order=123"
4. **Delay or Replay Requests**
   * Test if delaying or repeating a request (after supposed expiry) is still accepted.
5. **Analyze Server-Side Controls**
   * Does the backend enforce **atomic operations** and **locking mechanisms**?
   * Is “first come, first served” guaranteed, or can multiple requests slip in?

**🛡️ Mitigation**

* Implement **atomic transactions** (DB-level locking, unique constraints).
* Use **idempotent operations** (same request cannot succeed twice).
* Validate **timestamps** and enforce expiration strictly on the server.
* Apply **rate limiting** on critical operations.
* Conduct **race-condition testing** during QA.

✅ **Key Takeaway:**  
If timing flaws exist, attackers can exploit **race conditions, delays, or expired states** → leading to fraud (double spending, coupon abuse, privilege misuse).

👉 Do you want me to also prepare a **ready-made tester’s playbook with exact Burp/Turbo Intruder scripts** for simulating race conditions in process timing?