Great, now we’re at **WSTG-INPV-05: Testing for HTTP Parameter Pollution (HPP)** 🚀

**🧪 Testing for HTTP Parameter Pollution (HPP)**

**🔎 What is HTTP Parameter Pollution?**

* HTTP Parameter Pollution occurs when **multiple parameters with the same name are injected into an HTTP request**.
* Different backends (PHP, ASP.NET, Java, Python) and intermediaries (load balancers, proxies) may **interpret duplicates differently**.
* Attackers exploit this **inconsistency** to:
  + Bypass input validation
  + Override application logic
  + Trigger unexpected behavior
  + Sometimes escalate to XSS, SQLi, or authentication bypass.

**⚡ Example Attack Scenarios**

**1. Basic Example**

Normal request:

https://example.com/search?category=books

Polluted request:

https://example.com/search?category=books&category=electronics

* Some servers **take first parameter** → books.
* Others **take last parameter** → electronics.
* Or they may **concatenate values** → books,electronics.

**2. Bypassing Security Filters**

Suppose WAF blocks "<script>" but only inspects the first parameter.

?id=123&id=<script>alert(1)</script>

* App might process the **second parameter**, bypassing the filter.

**3. Login Bypass**

POST /login

username=admin&username=guest&password=123

* If the app only reads the last username, attacker may log in as guest.

**🔧 How to Test (Step-by-Step)**

**1. Manual Testing**

* Identify input parameters (GET and POST).
* Duplicate parameter names with different values.
* Observe application behavior.

**2. With Burp Suite**

* Send a request to Repeater.
* Add duplicate parameters with variations.
* Example:
* user=admin&user=test
* Monitor responses for differences.

**3. Automated Tools**

* **OWASP ZAP**, **Burp Intruder**, **fuff** can automate injection.
* Payload examples:
* param=123&param=456
* param=normal&param=<script>alert(1)</script>

**✅ Mitigation**

* Normalize inputs → ensure app **accepts only one value per parameter**.
* Strict server-side validation and canonicalization.
* Encode/escape input before using in queries, HTML, headers.
* WAFs tuned for HPP detection (though not always reliable).

⚡ **Summary:**

* HPP abuses duplicate parameters in HTTP requests.
* Attackers use it to **bypass validation, override inputs, or chain with other attacks**.
* Testing = duplicate parameters & observe differences in processing.
* Defense = enforce strict parameter handling rules.

👉 Do you want me to also make a **cheat-sheet style table of how different backends (PHP, ASP.NET, Node.js, Python Flask) handle duplicate parameters**? That’s often useful in labs.