# HTTP Parameter Pollution (HPP)

**HTTP Parameter Pollution (HPP)** is a vulnerability where an attacker injects **multiple parameters with the same name** into an HTTP request, exploiting how servers or backend systems **parse and interpret parameter collisions differently**.

This can lead to:

* **Bypassing security controls**
* **Modifying application logic**
* **Access control bypass**
* **Cache poisoning**
* **Open redirect / XSS**
* **WAF evasion**

# Why It Happens:

Different frameworks interpret multiple parameters with the same name **in different ways**:

| Framework / Language | Behavior |
| --- | --- |
| PHP | Takes the last value |
| Java (Servlets) | Takes the first value |
| ASP.NET | Returns all values in an array |
| Node.js | May return a comma-separated string |
| Nginx / Apache | Often take first occurrence or merge |

# Example Attack

Original Secure URL: <https://example.com/profile?role=user>  
HPP Payload: <https://example.com/profile?role=user&role=admin>  
If backend uses the **last parameter**, user may get admin access.

# Types of HPP

## 1. Query String Pollution

* Parameters repeated in **URL**
* Example: /profile?user=1&user=admin
* Impact: Access control bypass

## 2. Body Parameter Pollution

* Duplicate keys in **POST body**
* Example:  
  POST /login  
  user=admin&user=guest
* Impact: Unexpected logic

## 3. Header Pollution

* Inject duplicate headers (e.g., X-Forwarded-For)
* Example:  
  X-Forwarded-For: 127.0.0.1  
  X-Forwarded-For: attacker.com
* Impact: Bypass IP filters, logs

## 4. Cookie Parameter Pollution

* Duplicate cookie names
* Example: Cookie: sessionid=abc; sessionid=evil
* Impact: Hijack or mislead authentication

# Impact

| Area | Risk |
| --- | --- |
| Access Control Bypass | Switch roles (e.g., user → admin) |
| Logic Tampering | Trick app into wrong behavior |
| Data Leakage | Trick cache or logs into storing wrong info |
| WAF Bypass | Send two versions of a blocked parameter |
| Bypass Validation Filters | App may only check first param, execute last |

# Real-World Scenario:

**Use case:** A password reset form checks: /reset?token=VALID

Attacker sends: /reset?token=INVALID&token=VALID   
App may check first (INVALID), fail, but backend applies the last (VALID) — enabling token misuse.

# Testing Techniques

| Tool | What To Do |
| --- | --- |
| Burp Suite Repeater | Inject multiple parameters: |

/endpoint?id=1&id=2

| **POST Body Duplication** | Modify raw body:  
[email=test@example.com&email=admin@evil.com](mailto:email=test@example.com&email=admin@evil.com)  
  
| **Parameter Pollution Scanner (Burp Extension)** | Automates parameter injection |  
| **Observe Differences in Behavior** | Compare normal vs polluted requests |  
| **Cache Poisoning Payloads** | Cache different responses based on param position

# Tools for Detection

| **Tool** | **Use** |
| --- | --- |
| **Burp Suite** | Manual or automated testing with parameter fuzzing |
| **Param Miner (Burp Extension)** | Detects hidden parameters and duplicates |
| **HPP Fuzzer / Custom Scripts** | Automate HPP payloads |
| **ZAP / Postman** | Manual testing for multiple param values |

# Mitigation

| Strategy | Implementation |
| --- | --- |
| Whitelist Known Parameters | Reject duplicates by default |
| Normalize Parameters Server-Side | Accept only the first or reject if multiple received |
| Use Strong Frameworks | Use libraries that handle parameter parsing securely |
| Sanitize Input & Remove Duplicates | During input pre-processing |
| WAF Rules | Block repeated params or param flooding |
| Log Unexpected Parameter Behavior | Alert on anomalies |

# Secure Code Example (PHP)

$role = $\_GET['role']; // insecure

// Secure approach:

$allowed\_roles = ['user', 'admin'];

if (isset($\_GET['role']) && in\_array($\_GET['role'], $allowed\_roles)) {

$role = $\_GET['role'];

} else {

die("Invalid role");

}

# Points

“HTTP Parameter Pollution is about **inconsistent parameter parsing** between frontend, backend, or proxies.”

“Insecure backends may **prioritize the wrong value** from multiple occurrences.”

“Test by **injecting repeated parameters**, observing behavior changes, bypasses, or logic manipulation.”

# Real-World Exploits

| Case | Description |
| --- | --- |
| Akamai & CDNs | HPP used for cache poisoning and WAF evasion |
| Facebook (Bug Bounty) | Duplicate POST param bypassed authentication checks |
| WordPress Plugins | Double redirect\_url led to open redirects |
| ASP.NET MVC | Array parsing of duplicate keys exploited in access controls |