# HTTP Smuggling

**HTTP Request Smuggling (HRS)** is a critical vulnerability where attackers craft **desynchronized HTTP requests** that exploit **inconsistent parsing between front-end (proxy/load balancer) and back-end (origin server)**, resulting in:

* Web cache poisoning
* WAF bypass
* Request hijacking
* Credential theft
* Privilege escalation
* Remote Code Execution (RCE) in chained exploits

It **abuses differences in how multiple systems interpret HTTP headers like Content-Length, Transfer-Encoding, and chunked encoding**.

# Why It Happens

HTTP/1.1 allows for:

* Multiple headers in one request
* Both Content-Length and Transfer-Encoding headers
* Various chunked encoding formats

When front-end and back-end servers **parse those differently**, they **disagree on where a request ends**, enabling an attacker to "smuggle" hidden malicious requests.

# Impact

| Type | Description |
| --- | --- |
| WAF Bypass | Hidden request never seen by WAF |
| Credential Hijacking | Poison internal response headers |
| Desync Attacks | One user’s request hijacked by another |
| Information Disclosure | Reveal internal routing, proxy behavior |
| Cache Poisoning | Trick cache into storing poisoned pages |

# Types of HTTP Request Smuggling

## 1. CL.TE (Content-Length vs. Transfer-Encoding)

* **Frontend trusts Content-Length**, backend trusts Transfer-Encoding.
* Smuggled body interpreted differently.

POST / HTTP/1.1

Host: target.com

Content-Length: 44

Transfer-Encoding: chunked

0

GET /admin HTTP/1.1

Host: target.com

## 2. TE.CL (Transfer-Encoding vs. Content-Length)

* Frontend processes chunked, backend uses Content-Length.

**Example:**

POST / HTTP/1.1

Host: target.com

Transfer-Encoding: chunked

Content-Length: 6

5\r\n

hello\r\n

0\r\n

\r\n

GET /admin HTTP/1.1

## 3. TE.TE or CL.CL (Advanced Cases)

Double Transfer-Encoding or Content-Length headers

Exploits **header normalization** differences

## 4. H2.C (HTTP/2 → HTTP/1 Smuggling)

Exploiting **downgrade inconsistencies** between HTTP/2 frontends and HTTP/1 backends.

# **How Attackers Exploit It**

| Step | Description |
| --- | --- |
| 1️⃣ | Craft dual-header HTTP request |
| 2️⃣ | Send to front-end (proxy, CDN, WAF) |
| 3️⃣ | Proxy splits request differently than backend |
| 4️⃣ | Backend processes injected request unknowingly |
| 5️⃣ | Attacker gains internal path access, injects cache poison, hijacks sessions, or triggers SSRF |

# Testing Tools

| Tool | Description |
| --- | --- |
| Burp Suite | Intruder, Repeater for custom headers and smuggling |
| HTTP Request Smuggler (Burp Extension) | Automated scanning for CL.TE, TE.CL, etc. |
| Smuggler.py | CLI-based smuggling tool by PortSwigger |
| Curl | For raw request crafting (advanced) |
| TLS-proxy-aware Proxies | Use mitmproxy or Tyk to analyze header normalization |

# Mitigation

| Strategy | Control |
| --- | --- |
| Strict RFC Compliance | Ensure front-end and back-end parse HTTP headers consistently |
| Remove Ambiguous Headers | Strip Transfer-Encoding or Content-Length where unnecessary |
| Enforce Single Header Use | Reject requests with both Content-Length & Transfer-Encoding |
| Upgrade Proxies & Libraries | Patch known bugs in NGINX, HAProxy, Apache, AWS CloudFront, etc. |
| Normalize Requests Early | Use middlewares to reformat headers |
| Implement Rate Limiting & Logging | Flag multiple malformed requests from same IP |

# Points

“HTTP Smuggling is an advanced vulnerability that exploits **desync between HTTP devices** — often invisible to scanners.”

“Always test CL.TE and TE.CL variations, especially when proxies, CDNs (like Cloudflare, Akamai), or load balancers are in place.”

“Modern secure setups **enforce strict header parsing** and reject dual presence of Content-Length and Transfer-Encoding.”

# Real-World Examples

| Org | Vulnerability |
| --- | --- |
| Amazon (2019) | Header injection allowed internal SSRF via smuggled request |
| Netflix | Used to bypass WAF & cache poison |
| Cloudflare | HTTP/2 → HTTP/1 desync exploited in multi-tenant apps |
| HackerOne Reports | Frequent RCE and authentication bypass via TE.CL |

# Secure-by-Design Principles

| Practice | Why |
| --- | --- |
| Use latest HTTP libraries | Avoid old parsing behavior |
| Disallow dual headers | Prevent ambiguity |
| Validate content length/TE headers at ingress | Detect smuggling patterns |
| Set max request size | Prevent long-body smuggling |