

**HTTP Request Smuggling: Documentation Summary**

**Definition:**  
HTTP Request Smuggling is a critical web vulnerability that arises when front-end and back-end servers (such as proxies, load balancers, or CDNs paired with application servers) interpret the boundaries of HTTP requests differently. This discrepancy is typically exploited by manipulating the Content-Length and Transfer-Encoding headers within a single HTTP/1.1 request[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7).

**How the Attack Works:**

* Attackers craft ambiguous HTTP requests containing both Content-Length and Transfer-Encoding headers, or use non-standard formatting of these headers.
* If the front-end and back-end servers parse these headers differently, the attacker can "smuggle" a hidden HTTP request through the front-end, which is then processed by the back-end as a separate, valid request.
* This desynchronization allows malicious requests to bypass security controls, reach protected endpoints, or interfere with other users' requests[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7).

**Common Attack Variants:**

* **CL.TE:** Front-end uses Content-Length, back-end uses Transfer-Encoding.
* [**TE.CL**](http://TE.CL)**:** Front-end uses Transfer-Encoding, back-end uses Content-Length.
* **TE.TE:** Both use Transfer-Encoding, but one server ignores the header due to obfuscation (e.g., whitespace, duplicate headers)[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[6]](#fn6).

**Potential Impacts:**

* **Bypass of security controls:** Attackers can access restricted resources or execute unauthorized actions.
* **Session hijacking:** Stealing or manipulating user session data.
* **Web cache poisoning:** Injecting malicious content into cache servers, affecting many users.
* **Cross-site scripting (XSS):** Smuggled requests can deliver XSS payloads to other users.
* **Denial of Service (DoS):** Disrupting request processing, causing service instability[[1]](#fn1)[[3]](#fn3)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7).

**Conditions for Exploitation:**

* The application must use HTTP/1.1 and have at least one intermediary (proxy, load balancer, CDN) between the client and the back-end server.
* At least one server in the chain must process ambiguous or malformed HTTP requests without proper validation[[1]](#fn1)[[3]](#fn3)[[5]](#fn5).

**Detection and Prevention:**

* **Detection:** Specialized tools (e.g., Burp Suite) and time-based or differential response testing can identify vulnerabilities[[1]](#fn1)[[3]](#fn3).
* **Prevention:** Ensure all servers in the chain consistently parse HTTP requests, reject ambiguous or malformed requests, and prefer using HTTP/2 for internal communication. Regularly update and configure all HTTP-handling components to minimize risk[[1]](#fn1)[[5]](#fn5)[[6]](#fn6).

**References:**  
This summary is based on the PortSwigger whitepaper "HTTP Desync Attacks: Request Smuggling Reborn" and supporting documentation from security industry sources[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)[[7]](#fn7).

⁂

1. http-desync-attacks.pdf

1. <https://portswigger.net/web-security/request-smuggling>

1. <https://aardwolfsecurity.com/what-is-http-request-smuggling/>

1. <https://en.wikipedia.org/wiki/HTTP_request_smuggling>

1. <https://www.vaadata.com/blog/what-is-http-request-smuggling-exploitations-and-security-best-practices/>

1. <https://www.extrahop.com/resources/attacks/http-request-smuggling>

1. <https://www.brightsec.com/blog/http-request-smuggling-hrs/>