

**Detailed Explanation of HTTP Desync Attacks (Request Smuggling)**

**Overview:**  
HTTP Desync Attacks, also known as HTTP Request Smuggling, exploit inconsistencies in how front-end and back-end servers parse HTTP requests. By crafting ambiguous requests—typically involving both Content-Length and Transfer-Encoding headers—attackers can desynchronize the interpretation of request boundaries, allowing them to inject ("smuggle") malicious requests that bypass security controls and impact other users[[1]](#fn1)[[2]](#fn2)[[3]](#fn3).

**Core Concepts**

* **Multi-tier Architecture Vulnerability:**  
  Modern web applications often use multiple HTTP-aware components (proxies, load balancers, CDNs, and web servers). These components may interpret the end of an HTTP request differently, especially when ambiguous headers are present[[1]](#fn1)[[2]](#fn2)[[3]](#fn3).
* **Desynchronization Mechanism:**  
  The attacker sends a single HTTP request that is interpreted as two distinct requests by the back-end server, while the front-end sees only one. This allows the attacker’s payload to be prepended to the next legitimate user's request, leading to a variety of exploits[[1]](#fn1)[[3]](#fn3).
* **Key Headers Involved:**
  + Content-Length: Specifies the size of the request body.
  + Transfer-Encoding: chunked: Indicates chunked encoding for the body.
  + Ambiguity arises when both are present, or when header names are obfuscated (e.g., extra spaces, unusual capitalization, or non-ASCII characters)[[1]](#fn1)[[4]](#fn4).

**Attack Techniques**

|  |  |  |
| --- | --- | --- |
| Technique | Description | Typical Scenario |
| **CL.TE** | Front-end uses Content-Length; back-end uses Transfer-Encoding. | Attacker sends both headers. The front-end reads up to the content length, but the back-end expects chunks, causing the rest of the data to be interpreted as a new request. |
| [**TE.CL**](http://TE.CL) | Front-end uses Transfer-Encoding; back-end uses Content-Length. | The front-end processes chunked data, but the back-end uses the content length, leading to desynchronization. |
| **Header Obfuscation** | Manipulating headers with whitespace, tabs, or non-standard formatting so one server ignores the header while another processes it. | Bypasses security controls that only check for standard header formats. |
| **HTTP/2 to HTTP/1.1 Downgrade** | Exploiting flaws in protocol translation, especially when a proxy converts HTTP/2 requests to HTTP/1.1, reintroducing ambiguity. | Allows smuggling even in environments that partially use HTTP/2[[2]](#fn2). |
| **Pause-Based/Client-Side Desync** | Exploiting timing or client-side parsing differences to force desynchronization. | Advanced, less common variant[[2]](#fn2). |

**Attack Methodology**

1. **Detection:**
   * Send ambiguous requests and observe anomalies (e.g., timeouts, unexpected responses).
   * Use time-based or error-based techniques to identify desynchronization without impacting other users[[1]](#fn1).
2. **Confirmation:**
   * Poison a back-end socket and verify that a subsequent request (your own or a victim's) is affected, confirming the vulnerability[[1]](#fn1).
3. **Exploration:**
   * Leak internal headers or routing logic by reflecting smuggled requests and analyzing the responses.
   * Adjust headers and payloads to maximize the impact or reach internal APIs[[1]](#fn1).
4. **Exploitation:**
   * Launch attacks such as web cache poisoning, session hijacking, credential theft, privilege escalation, or XSS at scale[[1]](#fn1)[[5]](#fn5)[[3]](#fn3).

**Real-World Impact**

* **Web Cache Poisoning:**  
  Attackers can poison web caches, causing malicious or sensitive data to be served to many users until the cache is cleared[[1]](#fn1)[[5]](#fn5)[[3]](#fn3).
* **Session Hijacking:**  
  By manipulating request boundaries, attackers can steal session cookies or credentials from other users[[1]](#fn1)[[3]](#fn3).
* **API Abuse:**  
  Smuggled requests can bypass API gateways or access internal endpoints not meant for public use[[3]](#fn3).
* **Mass Exploitation:**  
  Vulnerabilities can be chained with XSS or open redirects to compromise large numbers of users without their interaction[[1]](#fn1)[[5]](#fn5)[[4]](#fn4).
* **Bypassing Security Controls:**  
  Attackers can circumvent firewalls, authentication, and input validation implemented only at the front-end layer[[1]](#fn1)[[3]](#fn3).

**Advanced Variants**

* **HTTP/2 Request Smuggling:**  
  While HTTP/2 is designed to prevent these attacks, vulnerabilities can arise when HTTP/2 is downgraded to HTTP/1.1 by intermediaries, reintroducing ambiguity[[2]](#fn2).
* **Client-Side Desynchronization:**  
  Exploits differences in how browsers and servers parse responses, potentially leading to new attack vectors[[2]](#fn2).
* **Pause-Based Smuggling:**  
  Relies on timing attacks to force desynchronization between servers[[2]](#fn2).

**Defense and Mitigation**

* **Consistent Parsing:**  
  Ensure all HTTP-handling components use the same logic for parsing requests and reject ambiguous or malformed requests[[6]](#fn6)[[7]](#fn7)[[8]](#fn8)[[9]](#fn9).
* **Strict Protocol Adherence:**  
  Remove or normalize conflicting headers before forwarding requests. Drop requests with both Content-Length and Transfer-Encoding headers[[7]](#fn7)[[8]](#fn8)[[9]](#fn9).
* **Adopt HTTP/2 End-to-End:**  
  Using HTTP/2 for all internal communication can eliminate many smuggling vectors[[2]](#fn2)[[6]](#fn6).
* **Disable Connection Reuse:**  
  Prevent persistent connections between front-end and back-end servers to reduce the attack surface, though this may impact performance[[6]](#fn6).
* **Update and Patch:**  
  Regularly update all HTTP-aware components and configure them securely to minimize risk[[6]](#fn6)[[7]](#fn7)[[8]](#fn8)[[9]](#fn9).

**Notable Case Studies**

* **PayPal:**  
  Attackers exploited request smuggling to hijack JavaScript files on the login page, eventually bypassing CSP protections and stealing user credentials, leading to significant bug bounty payouts and rapid security fixes[[1]](#fn1).
* **Multi-CDN/Proxy Chains:**  
  Desynchronization between multiple layers of proxies (even from the same vendor) can multiply the severity and reach of attacks[[1]](#fn1)[[5]](#fn5).

**Conclusion**

HTTP Desync Attacks are a critical and evolving threat to web infrastructure, enabling attackers to bypass security controls, steal data, and compromise users at scale. The complexity of modern web architectures and protocol ambiguities make rigorous, consistent HTTP parsing and layered defenses essential for mitigation[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4).

⁂

1. us-19-Kettle-HTTP-Desync-Attacks-Smashing-Into-The-Cell-Next-Door-wp.pdf

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

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

1. <https://blog.barracuda.com/2020/11/17/http-desync-attacks-a-variant-of-request-smuggling-attacks>

1. <https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn>

1. <https://www.cobalt.io/blog/a-pentesters-guide-to-http-request-smuggling>

1. <https://techdocs.broadcom.com/us/en/vmware-security-load-balancing/avi-load-balancer/avi-load-balancer/30-2/ddos-attack-mitigation/protection-from-http-desync-attack.html>

1. <https://techdocs.broadcom.com/us/en/vmware-security-load-balancing/avi-load-balancer/avi-load-balancer/30-1/vmware-avi-load-balancer-configuration-guide/ddos-attack-mitigation/protection-from-http-desync-attack.html>

1. <https://techdocs.broadcom.com/us/en/vmware-security-load-balancing/avi-load-balancer/avi-load-balancer/31-1/vmware-avi-load-balancer-configuration-guide/ddos-attack-mitigation/protection-from-http-desync-attack.html>