

**Detailed Summary of "HTTP/2: The Sequel is Always Worse" by James Kettle**

**Introduction**

This paper explores new classes of vulnerabilities unique to HTTP/2, focusing on how its design and common implementation flaws introduce critical attack vectors—especially HTTP/2-exclusive desynchronization (desync) and request smuggling attacks. The research demonstrates real-world exploits against major web infrastructure, including cloud load balancers, CDNs, WAFs, and custom stacks used by large tech companies[[1]](#fn1)[[2]](#fn2)[[3]](#fn3).

**HTTP/2 Fundamentals and Key Differences**

* **Pseudo-Headers:**  
  HTTP/2 replaces the HTTP/1.1 request line with pseudo-headers (e.g., :method, :path, :authority, :scheme), which are parsed differently than standard headers.
* **Binary Protocol:**  
  Unlike HTTP/1.1’s text-based format, HTTP/2 is binary and uses frames with explicit length fields, reducing ambiguity in parsing message boundaries.
* **Message Length:**  
  In HTTP/2, message body length is determined by frame boundaries, not by Content-Length or Transfer-Encoding headers. However, these headers may still appear for backward compatibility, and can be mishandled during protocol downgrades[[1]](#fn1)[[2]](#fn2)[[4]](#fn4).

**HTTP/2 Desync and Request Smuggling Attacks**

**Core Problem: HTTP/2 to HTTP/1.1 Downgrades**

* Many front-end servers (e.g., proxies, CDNs) accept HTTP/2 from clients but downgrade requests to HTTP/1.1 before forwarding to back-end servers.
* This translation can reintroduce ambiguity, especially when HTTP/2 requests include headers (Content-Length, Transfer-Encoding) that are meaningless in HTTP/2 but critical in HTTP/1.1.
* If the downgrade process does not properly validate or sanitize these headers, attackers can exploit mismatches in request boundary parsing, leading to desync and request smuggling[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4).

**Attack Variants**

* [**H2.CL**](http://H2.CL) **(HTTP/2 Content-Length):**  
  Attackers include a Content-Length header in an HTTP/2 request. If the front-end downgrades this to HTTP/1.1 without validation, the back-end may misinterpret the body, allowing attackers to prepend data to subsequent requests.  
  *Case Study: Netflix*—A crafted HTTP/2 request with a misleading Content-Length allowed attackers to hijack responses, redirect users, and steal credentials. This was patched and tracked as CVE-2021-21295, with a $20,000 bounty[[1]](#fn1)[[2]](#fn2).
* **H2.TE (HTTP/2 Transfer-Encoding):**  
  Attackers send a Transfer-Encoding: chunked header in HTTP/2. If not filtered, the back-end may prioritize this over Content-Length, causing desync.  
  *Case Study: AWS Application Load Balancer and Verizon*—Allowed attackers to intercept OAuth tokens and credentials by redirecting users during login flows. Verizon awarded a $7,000–$10,000 bounty for these findings[[1]](#fn1)[[2]](#fn2).
* **H2.TE via Header Injection:**  
  HTTP/2’s binary format allows attackers to inject newline characters in header values, creating new headers during downgrade, enabling further smuggling vectors.  
  *Case Study: Netlify CDN*—Enabled persistent cache poisoning and full control over content on sites using Netlify[[1]](#fn1).
* **H2.X via Request Splitting:**  
  Exploiting HTTP/2 to inject multiple requests or responses, causing response queue poisoning where users receive responses intended for other users.  
  *Case Study: Atlassian Jira*—Attackers could cause users to receive each other's responses, including session cookies, leading to mass account compromise and forced global session expiry[[1]](#fn1).
* **Header Name and Request Line Injection:**  
  HTTP/2 allows colons and spaces in header names and request methods, enabling injection attacks that are impossible in HTTP/1.1.  
  *Case Study: Apache mod\_proxy*—Permitted bypassing of path restrictions and escaping subfolders via manipulated pseudo-headers[[1]](#fn1).

**Desync-Powered Request Tunneling**

* **Request Tunneling:**  
  When front-ends avoid reusing back-end connections, classic cross-user attacks are hard. However, attackers can still tunnel requests that bypass front-end security controls, leak internal headers, or poison caches.
* **Detection and Exploitation:**  
  New techniques for confirming tunneling vulnerabilities include using HEAD requests to manipulate response handling, and leveraging tools like Param Miner to guess or leak internal header names[[1]](#fn1).

**HTTP/2-Specific Exploit Primitives**

* **URL Prefix Injection:**  
  Manipulating the :scheme pseudo-header or multiple :path pseudo-headers to override routing or poison caches.
* **Header Name Splitting:**  
  Using colons or newlines in header names to split or tamper with headers, enabling host-header attacks or header injection.
* **Request Line Injection:**  
  Exploiting servers that tolerate spaces in :method to inject arbitrary request lines and bypass restrictions.
* **Header Tampering Wrap:**  
  Exploiting legacy line-folding in headers to tamper with internal headers, if the back-end supports it[[1]](#fn1).

**Tooling and Detection**

* **New Tools:**  
  The research introduces a custom open-source HTTP/2 stack integrated into Turbo Intruder, and updates to Burp Suite and HTTP Request Smuggler for automated detection of HTTP/2 desync and smuggling vulnerabilities[[1]](#fn1)[[5]](#fn5).
* **Testing Approaches:**  
  Automated scanning for hundreds of smuggling techniques, including malformed headers and pseudo-header manipulation, is essential for uncovering these flaws[[3]](#fn3).

**Defense and Mitigation**

* **End-to-End HTTP/2:**  
  Avoid protocol downgrading by using HTTP/2 throughout the entire request path.
* **Strict Validation:**  
  Enforce HTTP/1.1 charset and header restrictions during downgrade; reject requests with newlines in headers, colons in header names, or spaces in the method.
* **Awareness of RFC Gaps:**  
  Some vulnerabilities arise from ambiguities or omissions in the HTTP/2 RFC—implementers must go beyond the spec to ensure security.
* **Continuous Testing:**  
  Use updated tooling to regularly scan for new classes of HTTP/2-exclusive vulnerabilities[[1]](#fn1)[[3]](#fn3)[[2]](#fn2).

**Conclusion**

HTTP/2 introduces novel and severe security risks, primarily due to protocol downgrading and insufficient validation during translation to HTTP/1.1. These flaws enable attackers to perform powerful request smuggling and desync attacks, hijack sessions, poison caches, and bypass security controls on major web platforms. The paper demonstrates that HTTP/2’s complexity and flexibility create a broad and evolving attack surface, requiring new defenses, robust tooling, and ongoing research to secure modern web infrastructure[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[5]](#fn5).

**References:**  
[[1]](#fn1) http2whitepaper.pdf  
[[3]](#fn3) Akamai Blog  
[[2]](#fn2) PortSwigger Research  
[[5]](#fn5) DEF CON 29 - James Kettle - HTTP2: The Sequel is Always Worse  
[[4]](#fn4) Detectify Labs

⁂

1. http2whitepaper.pdf

1. <https://portswigger.net/research/http2>

1. <https://www.akamai.com/blog/security/http-2-request-smulggling>

1. <https://labs.detectify.com/how-to/set-up-docker-for-varnish-http-2-request-smuggling/>

1. <https://www.youtube.com/watch?v=rHxVVeM9R-M>