A deep overview

HTTP Request Smuggling

PhiloCyber 2024

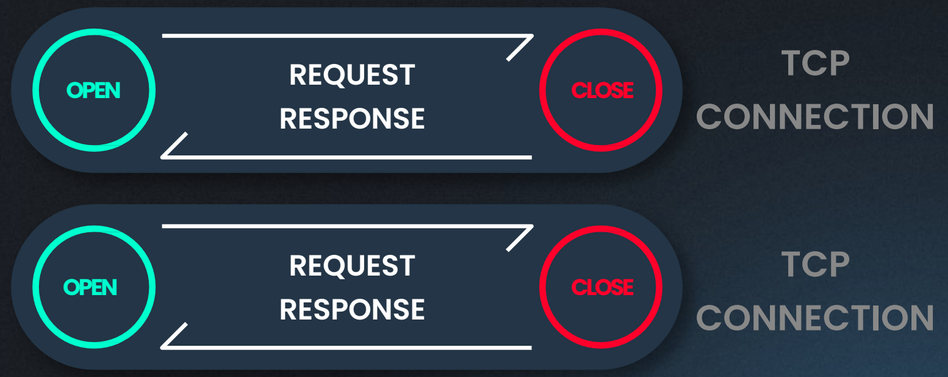
Index

1. Introduction

A bit of history about HTTP going from 0.9 to 2.0

HTTP 0.9

- HTTP (HyperText Transfer Protocol), created in 1991.
- Extremely Simple.
- One TCP connection per request.
- Only one method (GET) and a header request that only point the path.

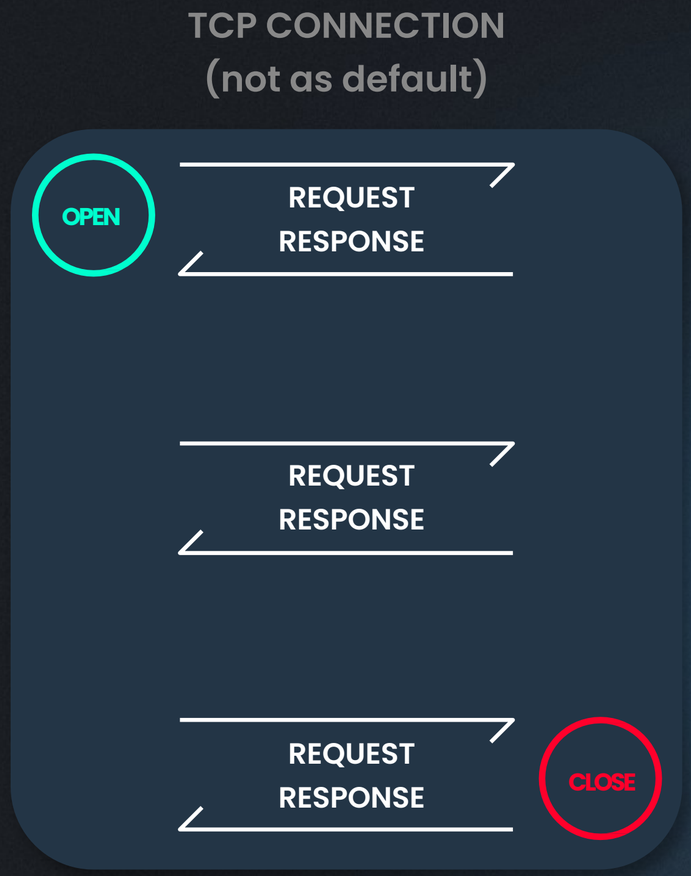



```
# The full URL wasn't included as the protocol, server, and
# port weren't necessary once connected to the server.
GET /mypage.html

# The response was extremely simple too and only in HTML:
# it only consisted of the file itself, not even response
# headers (one TCP request per connection witho).
<html>
  A very simple HTML page
</html>
```

HTTP 1.0

- Introduced it in 1996.
- Strong transition from simple documents retrieval to more dynamic web applications.
- More HTTP methods like POST, HEAD, PUT and DELETE.
- Multiple HTTP requests under the same TCP connection.
- Adding metadata and more context in several new headers in order to facilitate more specialised types of interactions.



New Headers in HTTP 1.0

Content-Length

Indicating the size of the entity-body in bytes.

Connection

In HTTP/1.0, this header could include 'keep-alive' to request a persistent connection.

Date

Indicating the date and time at which the message was originated.

Content-Type

Specifying the media type of the entity-body sent to the recipient. EG, text/html, application/json, etc.

Content-Encoding

Specifying the encoding applied to the entity-body, allowing compression or encryption.

New Headers in HTTP 1.0

Content-Language

Describing the natural language(s) of the intended audience for the entity-body.

Host

Specifies the domain name of the server.

Accept

Informs the server about the types of content the client can understand.

User-Agent

Provides information about the client making the request, such as the type of browser or user agent.

Authorization

Used for sending credentials (e.g. username and password) to access protected resources.

HTTP 1.1

- Just one year after the previous version, 1997.
- More focused in efficiency improvements.
- More HTTP Methods
- Keep-alive as default
- Pipelining
- Range Request

New Headers and changes in HTTP 1.1

Transfer-Encoding
Specifies the form of encoding used to safely transfer the payload body to the user.

Host
*Changed it to **mandatory** to distinguish between different virtual hosts that share the same IP address.*

Cache-Control
Allowing sophisticate cache control with directives like "no-cache, no-store, max-age, etc".

ETag
Providing a unique identifier for a specific version of a resource, aiding in cache validation.

Ranged
Enabled clients to request specific portions of resources, as we mentioned above.

New Headers and changes in HTTP 1.1

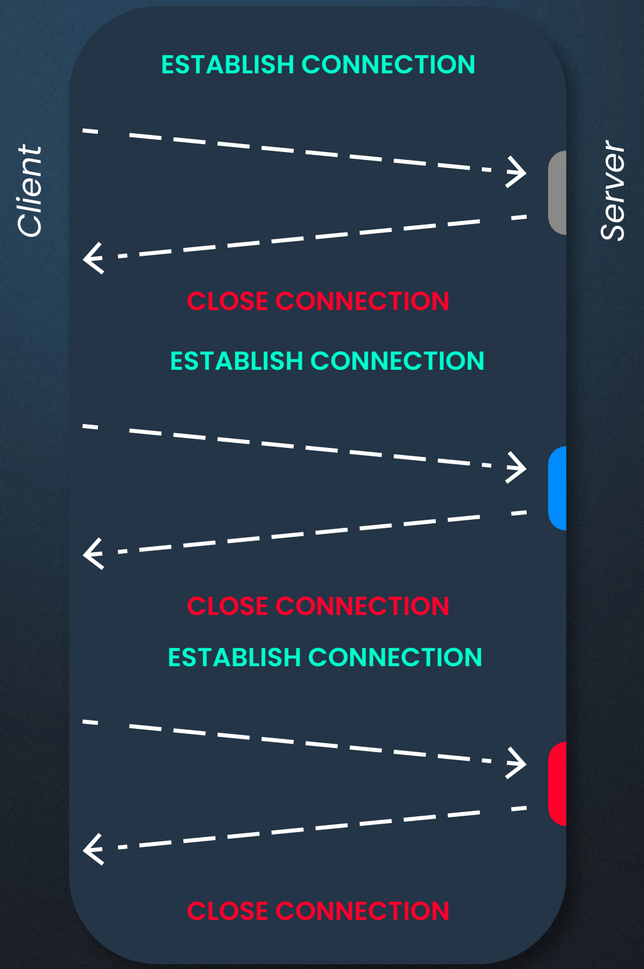
If-Range
Allowing conditional request based on a resources ETag, enabling clients to retrieve the resource only if it had been modified.

Content-Range
Sent by the server in partial responses to specify the range of bytes being returned.

Age
Informed clients about the time a resource had been cached in a proxy.

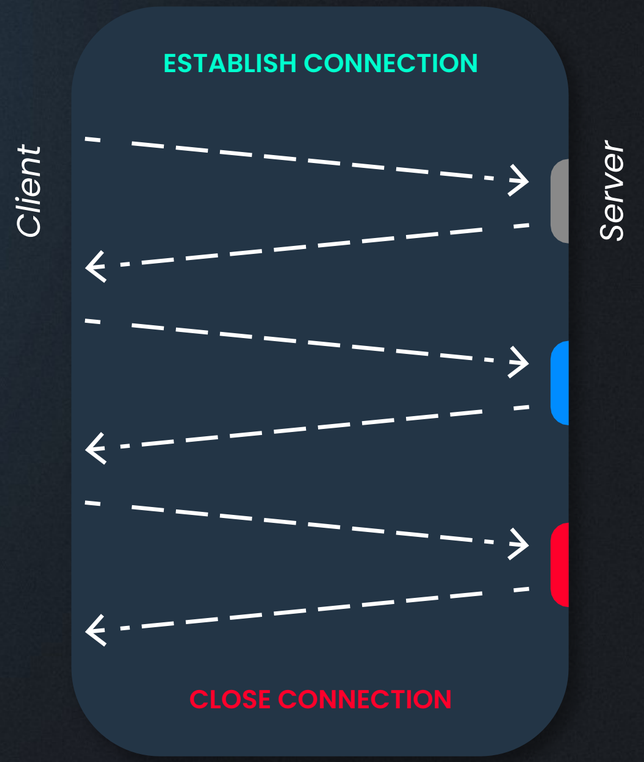
HTTP/0.9

Short-lived Connections



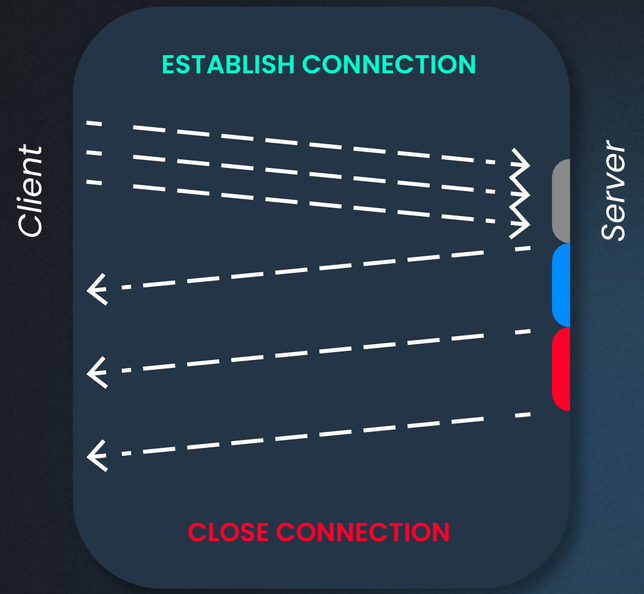
HTTP/1.0-1.1

Persistent Connections



HTTP/1.1

HTTP Pipelining



Pipelining

We are able to send different HTTP request in the same connection without having to wait for the first response of the first request (for this example, red).
Besides, we can see how we can delimit our payload length by using two different headers,
Content-Length and Transfer-Encoding.

```
POST / HTTP/1.1
Host: example.local
Content-Length: 5

HELLOGET /index HTTP/1.1
Host: example.local
Content-Length: 23

This is another exampleGET /admin HTTP/1.1
Host: example.local
Transfer-Encoding: chunked

5
HELLO
0
```

For more information about this please refer to:

https://www.rfc-editor.org/rfc/rfc2616#section-4.4

Transfer-Encoding over Content-Length

*"If a message is received with both a Transfer-Encoding header field and a Content-Length header field, the latter **must** be ignored"*

HTTP/2.0… Now what?

Introduced in 2015, it aimed to reduce latency and improve the performance of HTTP traffic.

Multiplexing

Major update, enabling multiple concurrent requests and responses within a single TCP connection*, allowing resources to be sent and received asynchronously, improving data transfer efficiency and reducing latency.

Binary Protocol and Header Compression

It was adopted a binary framing layer that replaced the plain text format used in HTTP/1.1. Improved parsing efficiency, leading to faster processing and reduced bandwidth usage.

Server Push

This new version introduced the server push feature. Allowing servers to proactively send resources to the client's cache before they are requested*. This optimized the loading of web pages by pre-emptively providing necessary resources, reducing the need for subsequent requests.

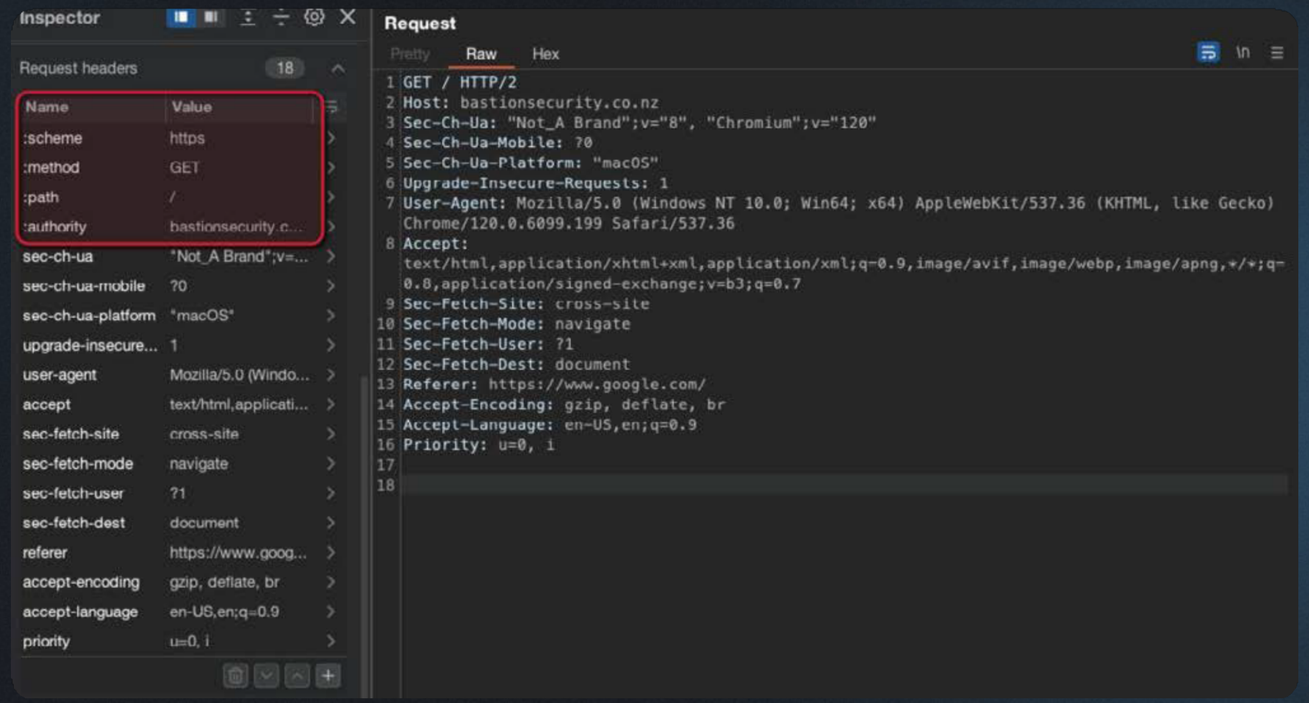
Stream Prioritisation and Dependency

Allowing prioritisation of streams, *enabling critical resources to be prioritised for faster delivery*, improving the overall user experience. Dependency information between different resources helped optimise data

Newer version, newer ways to handle data

The following pseudo-headers are defined in an HTTP/2 request:

```
:method GET
:path /index.php
:authority bastionsecurity.co.nz
:scheme https
```



Full compatibility with the previous versions (HTTP methods, headers, and query paths still exist but data is formatted differently in transit).

From clear text to binary.

Transfer-Encoding no longer supported.

2. What is HTTP Desync?

HTTP Request Smuggling

Request Smuggling (RS)

Desync Attack A.K.A Request Smuggling (RS) is considered an advance attack vector that exploits the discrepancy between frontend and a backend systems in the parsing of incoming HTTP request, by forcing a disagreement in the request boundaries between the two systems, thus causing a ***desynchronization.***

How it works

Stage 1

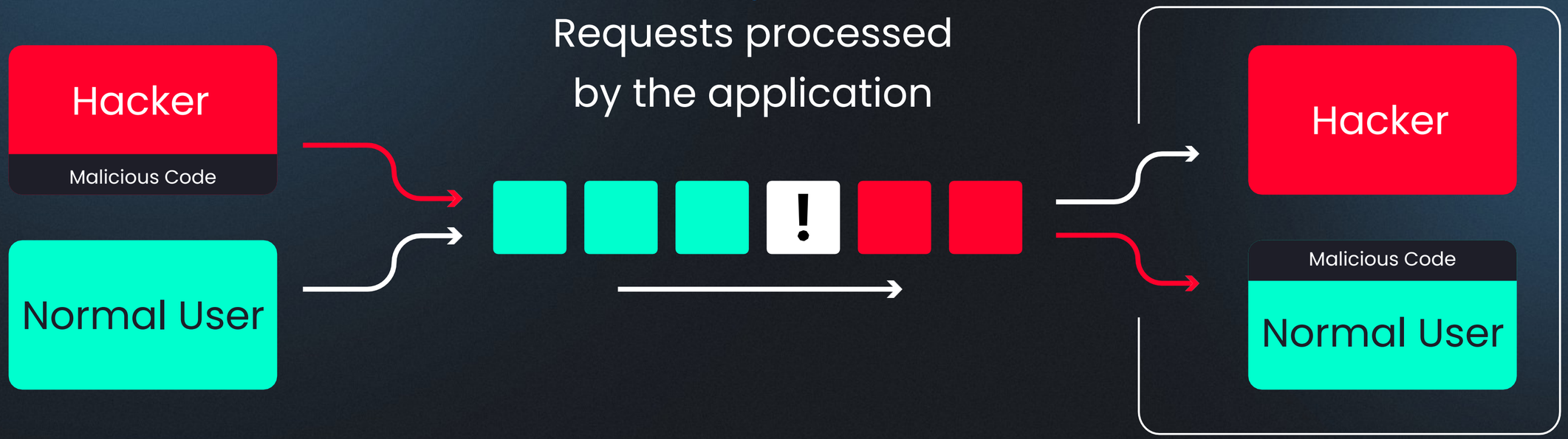
User requests sent to server

Stage 2

Requests processed by the application

Stage 3

Response sent back to users



Vulnerability Impact

Depending on the specific type of disagreement between the systems, this vulnerability can have a different impact, including mass exploitation of XSS, stealing of other users' data, and even WAF bypasses.
For more details on HTTP request smuggling attacks, have a look at this great blog post by James Kettle.

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

3. Types of Desync Attacks

Live examples

CL.TE

- Most common type of desync attack.
- The Reverse Proxy takes the content-length as reference for the HTTP request boundary (not supporting chunked encoding).
- Instead, the Web Application Server uses Transfer-Encoding for delimitation.

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 10
Transfer-Encoding: chunked

0
HELLO
```

Reverse Proxy interpretation

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 10
Transfer-Encoding: chunked

0
HELLO
```

Web Server interpretation

HACKTHEBOX

Identification and Exploitation

DEMO CL-TE

TE.TE

- Overlooked by several automate tools.
- There's a difference in how the comments follow the RSA standard; but both still support the Transfer-Encoding header.
- We need to force the misinterpretation by manipulating the header as:

Description	Header
Substring Match	Transfer-Encoding: **test chunked**
Space in Header name	Transfer-Encoding: **chunked**
Horizontal Tab Separator	Transfer-Encoding: [\x**09**]**chunked**
Vertical Tab Separator	Transfer-Encoding: [\x**0b**]**chunked**
Leading Space	Transfer-Encoding: **chunked**

It’s about how the different systems are interpreting the TE header, might only check the presence of “chunked”, while the other is checking an exact match.

Note: The sequences [\x09] and [\x0b] are not the literal character sequences used in the obfuscation. Rather they denote the horizontal tab character (ASCII 0x09) and vertical tab character (ASCII 0x0b).

Identification and Exploitation

DEMO TE-TE

TE.CL

- The most complex one to manually identify and exploit.
- The Reverse Proxy takes the transfer-encoding as reference for the HTTP request boundary (not supporting CL).
- Instead, the Web Application Server uses the content-length for delimitation.

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 3
Transfer-Encoding: chunked

5
HELLO
0
```

Reverse Proxy interpretation

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 3
Transfer-Encoding: chunked

5
HELLO
0
```

Web Server interpretation

HACKTHEBOX

Identification and Exploitation

DEMO TE-CL

4. Automation

Is it worth it?

Scenarios

- Automation may be worthwhile during extensive scope testing, but the availability of free GitHub tools for such testing is limited.
- In terms of tools/automation the best one currently is the HTTP Request Smuggler created by James Kettler, but can return false positives.
- We should try to avoid relaying in automate tools, even more if we know that there is not mature tool available like nmap, sqlmap, etc.

5. Preventions

Possible mitigations

Mitigating HTTP request smuggling attacks can be challenging since the vulnerabilities causing them typically reside within the web server software, making prevention from within the web application impossible.

Prevention

- Ensure that web server and reverse proxy software are kept up-to-date such that patches for security issues are installed as soon as possible.
- Ensure that client-side vulnerabilities that might seem unexploitable on their own are still patched, as they might become exploitable in an HTTP request smuggling scenario.
- Ensure that the default behaviour of the web server is to close TCP connections if any exception or error occurs on the web server level during request handling or request parsing.
- If possible, configure HTTP/2 usage between the client and web server and ensure that lower HTTP versions are disabled.

6. HTTP/2.0, Game Over?

Let’s keep track of the new changes

HTTP/2.0 features

- Chunked encoding is not longer supported.
- HTTP/2 transmits the request body in a binary format consisting of data frames.
- There is no explicit length field required to determine the length of the request body.
- The data frames contain a built-in length field that any system can use to calculate the request body's length.
- So good news... HTTP desync attacks are almost impossible if HTTP/2 is used correctly in a deployment setting!

...Wait a minute, what about HTTP/2 Downgrading?

Thank you for your time

Ricardo Prieto

Penetration Tester at

BASTION

@PhilocyberWithRichie

https://philocyber.com

PhiloCyber 2024