Listen to the whispers

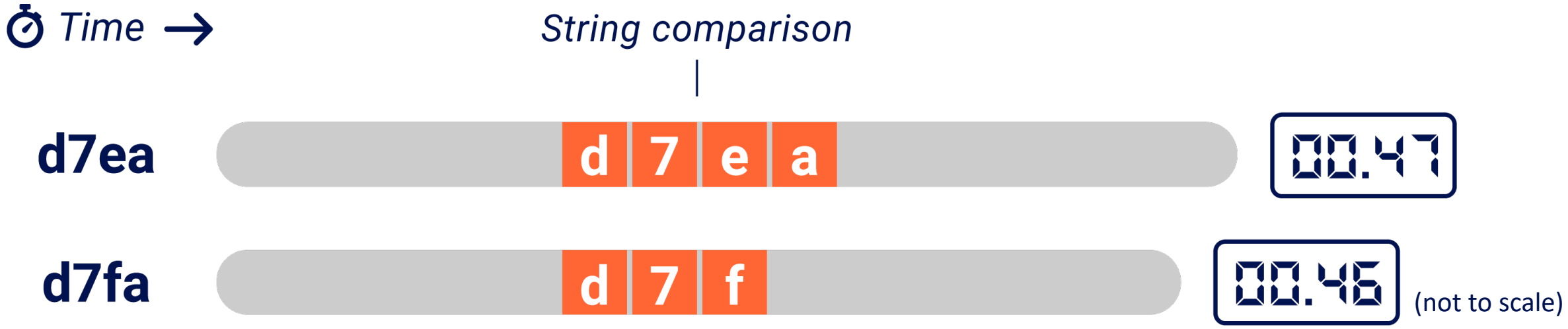
web timing attacks that actually work

James Kettle

PortSwigger Research

The timing trap

Does the database contain a password reset token starting with d7e?




```
def strcmp(s1, s2):
    for c1, c2 in zip(s1, s2):
        if c1 != c2:
            return False
        time.sleep(0.01)
    return True
```

The timing divide

Attacks I've used

Delay

- **1,300ms** — Is there a bug report containing 'API-KEY: XYZ'
- **200ms** — Does requesting a password reset for carlos trigger an email?
- **30ms** — Does the website lock on this row?
- **7µs** — Does the compressed PostgreSQL database contain a token with "xuz"

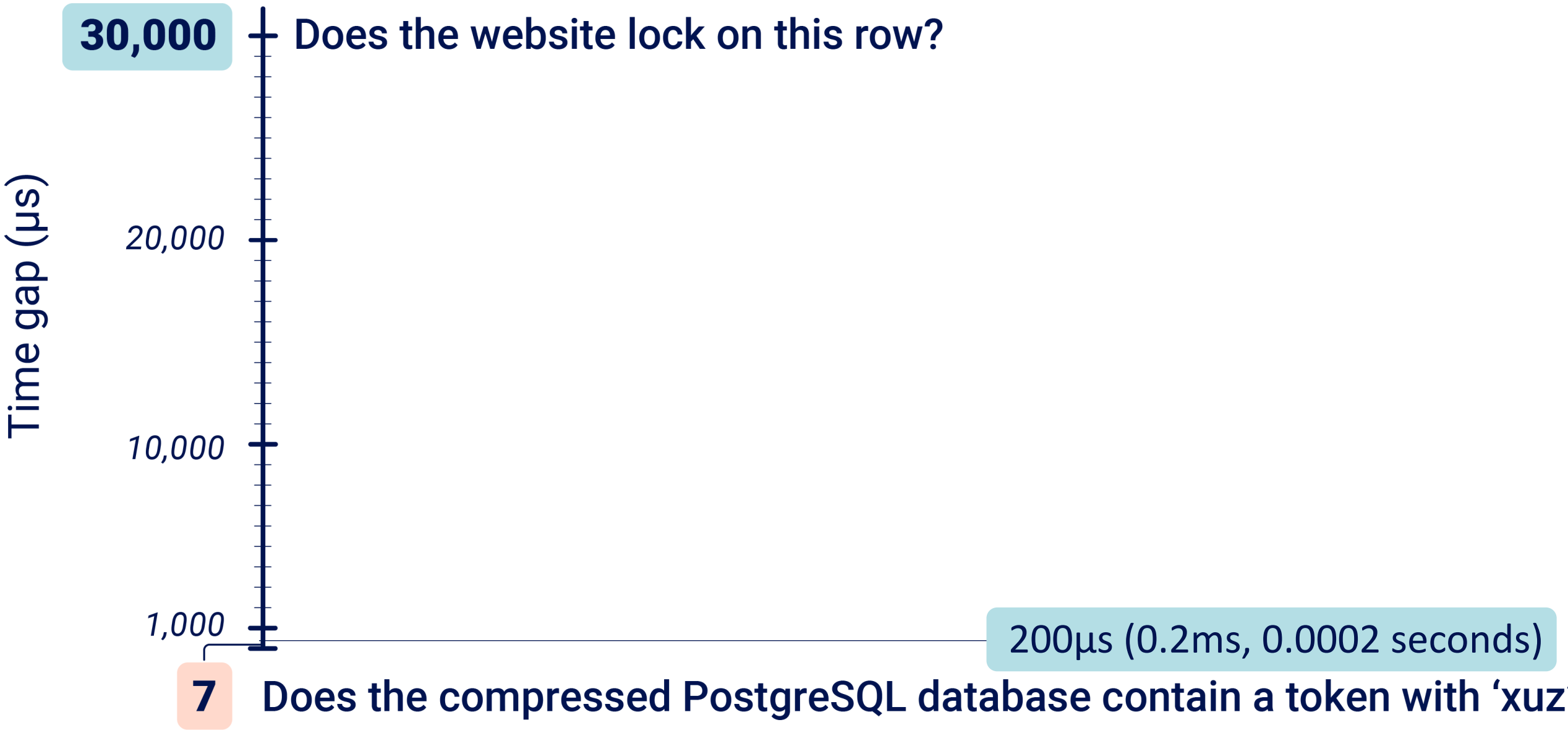
Lab-proven

- **5ns?** — Does the database contain a password reset token starting with d7e?

Theoretical

Attacks I've read about

The timing divide



Outline

Making timing attacks that work everywhere

Listening to whispers:

- Hidden attack-surface
- Server-side injection
- Reverse proxy misconfigurations

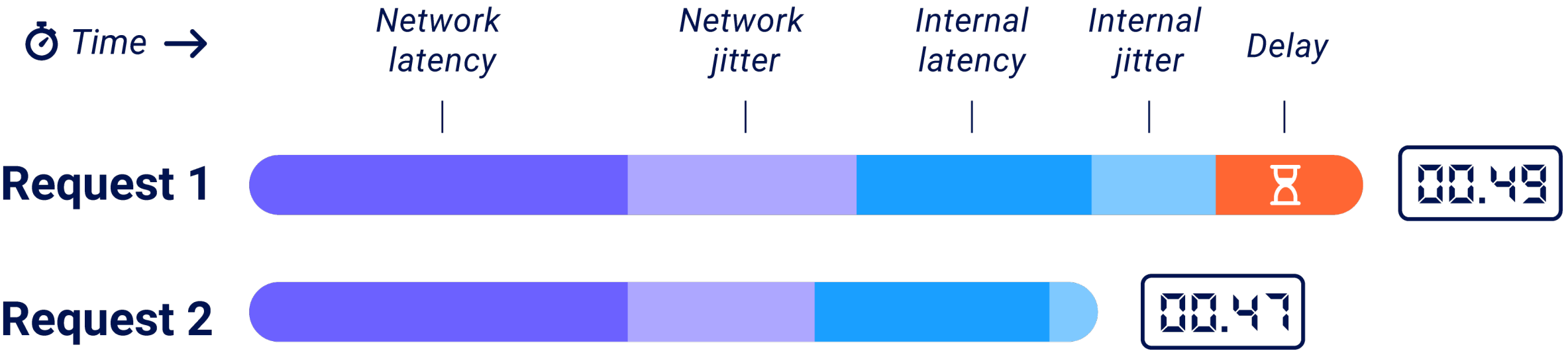
Defense / Takeaways / Questions

PortSwigger/param-miner

Making timing attacks that work everywhere

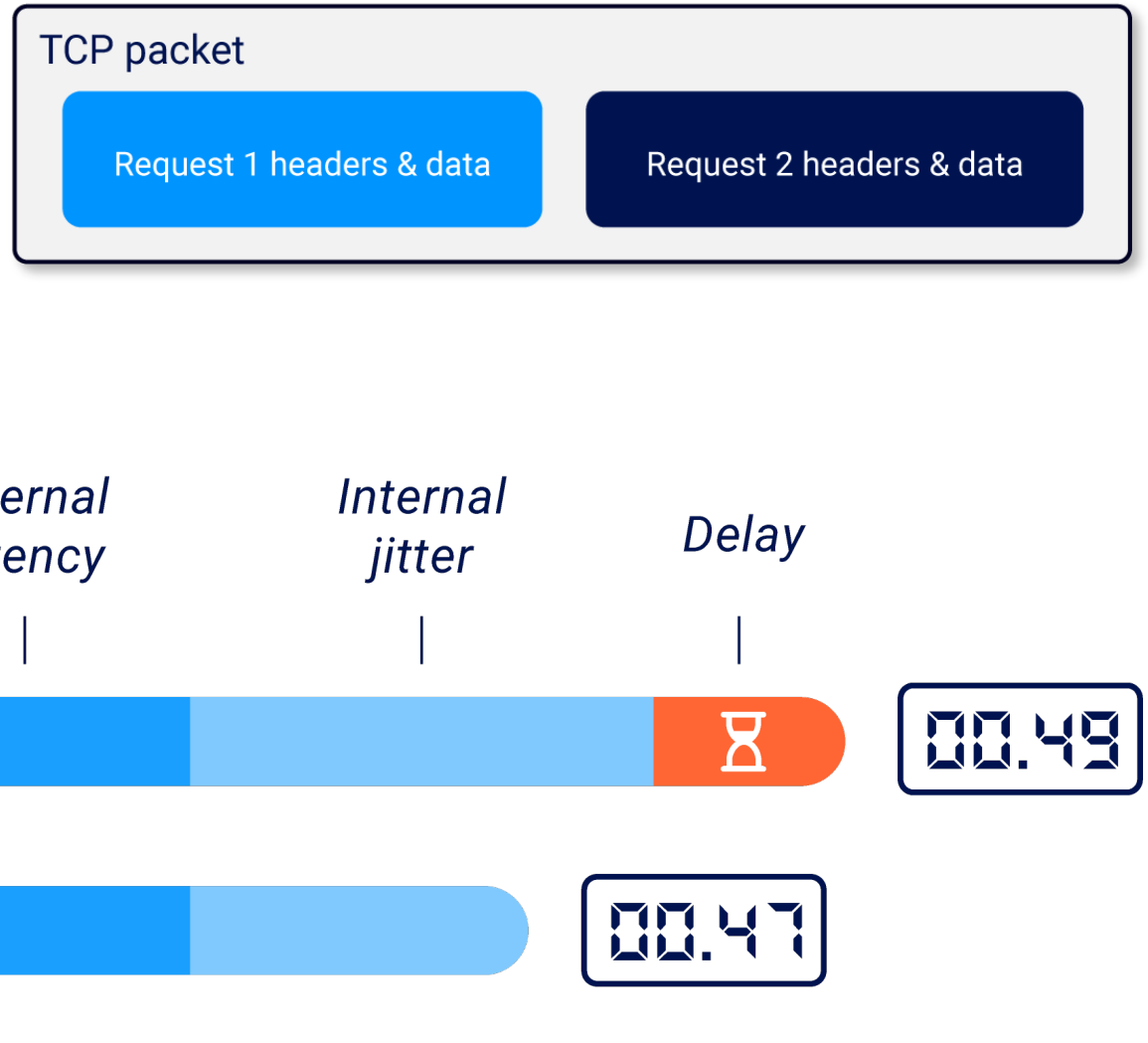
The equation for timing attack success

$$\text{success} = \frac{\text{signal}}{\text{noise}}$$

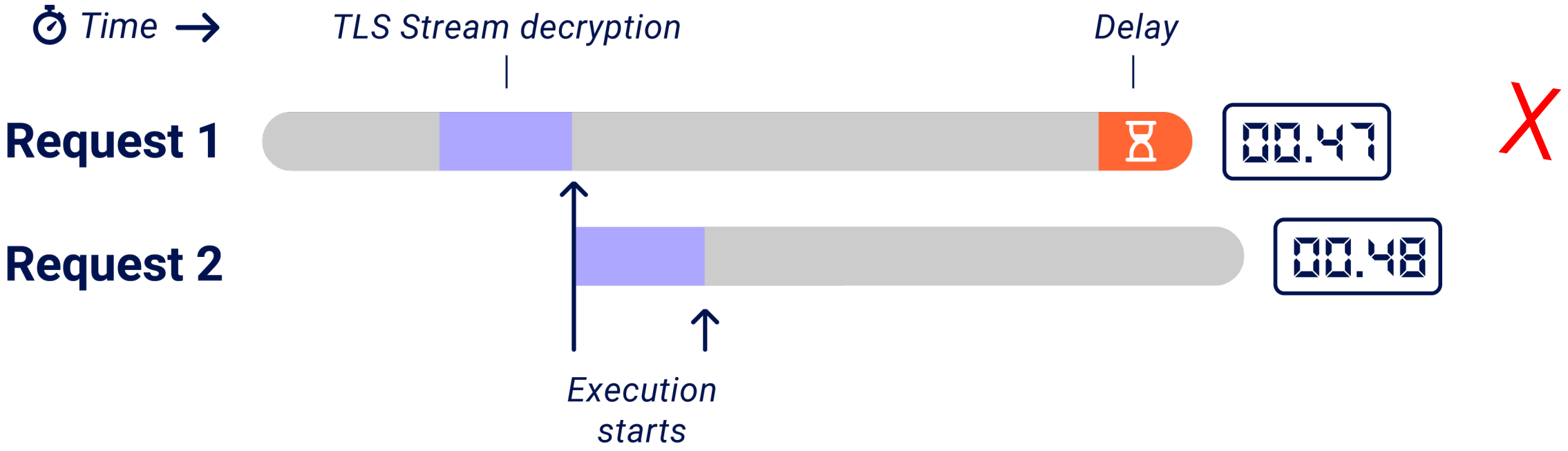


Making timing attacks 'local'

Timeless Timing Attacks (2020)



The sticky ordering problem



Solution #1: resynchronize with dummy parameters on first request

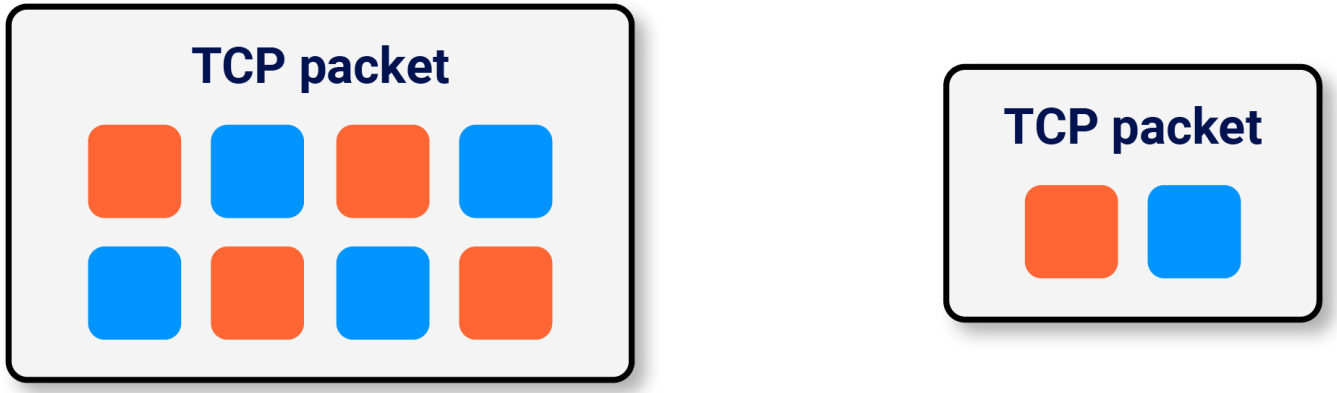
- Requires per-target configuration
- Fails outright on some targets
- Amplifies internal noise

Making timing attacks universal: single-packet attack

SPA v1 (2023)

Some servers start processing here :(

SPA v2



Enhancing the single-packet attack

disable TCP_NODELAY
send a ping frame
for each request with no body:
    send the headers
    withhold an empty data frame

for each request with a body:
    send the headers, and the body except the final byte
    withhold a data frame containing the final byte

wait for 100ms
send a ping frame
send the final frames

>200% accuracy enhancement on nginx

github.com/nxenon/h2spacex

Burp Suite Pro/Community 2024.5

Making timing attacks feasible

$$\text{success} = \frac{\text{signal}}{\text{noise}}$$

Amplify the signal

- Longest split code path
- Think DoS

```
GET / HTTP/1.1
X-U: a
{255}
X-U256: a
```

-> 256 times easier to detect

Minimize noise

- Embrace performance features
- Shortest shared code path

Add

```
GET / HTTP/1.1
Cookie: sid=d83a
DNT: 1
```

Remove

Hidden attack-surface

Guess params

does the application support a query parameter called 'exec'?

Discovery overload

Payload	Response	Response time
foo: x	HTTP/1.1 200 OK	50ms
commonconfig: x	HTTP/1.1 200 OK	55ms
commonconfig: {}	HTTP/1.1 200 OK	50ms

foo: x	--connection closed--	30ms
authorization: x	--connection closed--	50ms
GET /?id=random	HTTP/1.1 200 OK	310ms
GET /?foo=random	HTTP/1.1 200 OK	22ms

In cache key

Cache miss

Not in cache key

Cache hit

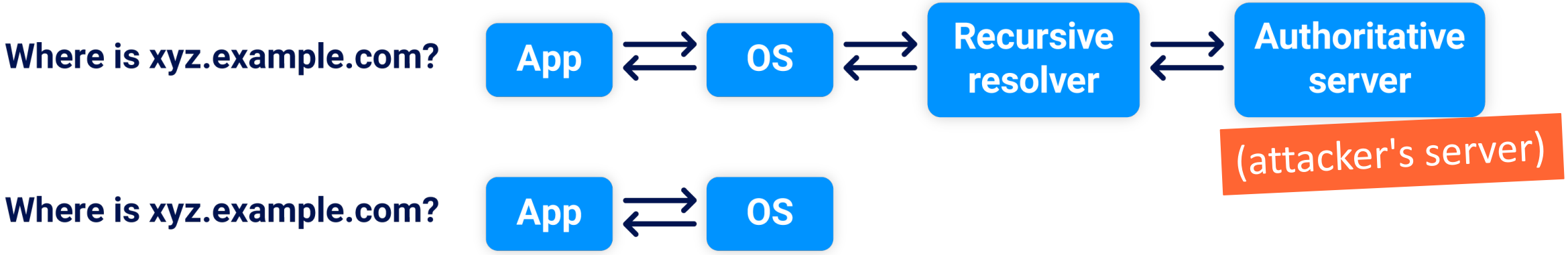
The hardest problem:

time analysis is *too powerful*

nowafpls

Zooming in: IP address spoofing via HTTP header

`Random-header: xyz.example.com`	65ms
`True-Client-IP: xyz.example.com`	70ms
`True-Client-IP: xyz.example.com`	65ms



375 vulnerable targets

217 with audible DNS caching

206 of which also cause a DNS pingback

Zooming further

`True-Client-IP: x.psres.net`	`90ms`
`True-Client-IP: 1.1.1.1`	`170ms`

Time	Browser	IP	Location
5 minutes ago	Chrome on Windows	1.1.1.1	Cloudflare

-> Timing analysis reveals control flow changes – like exceptions

Server-side injection

Detect server-side injection

SQLi with a classic payload

Payload	Response	Response time
`GET /api/alert?mic='`	`{}`	`162ms`
`GET /api/alert?mic=''`	`{}`	`170ms`

Alternate discovery path: `'||sleep(5)||'`

-> For sleep-capable bugs, use advanced timing for WAF evasion

-> What about other injections? JSON, XML, CSV, URL, HTTP, SMTP…

Blind server-side JSON injection

Invalid JSON speeds the response up by 0.2ms

`key=aa\"bb`	`"error": { "message": "Invalid Key: aa\"bb" }`	24.3ms
`key=a"\bb`	`"error": { "message": "Invalid Key: a"\bb" }`	24.1ms
`key=aaa…a"bbb`	`"error": { "message": "Invalid Key: ****bbb" }`	24.3ms

…unless the invalid syntax is redacted

->something is parsing the response server-side!

Blind server-side parameter pollution

/path?objectId=57%23	Can't parse parameter	180ms
/path?objectId=57%21	Can't parse parameter	430ms

```
Hypothesis: /backend?objectId=57#important-param=X
```

You need to know what to expect

Bug-doppelgangers

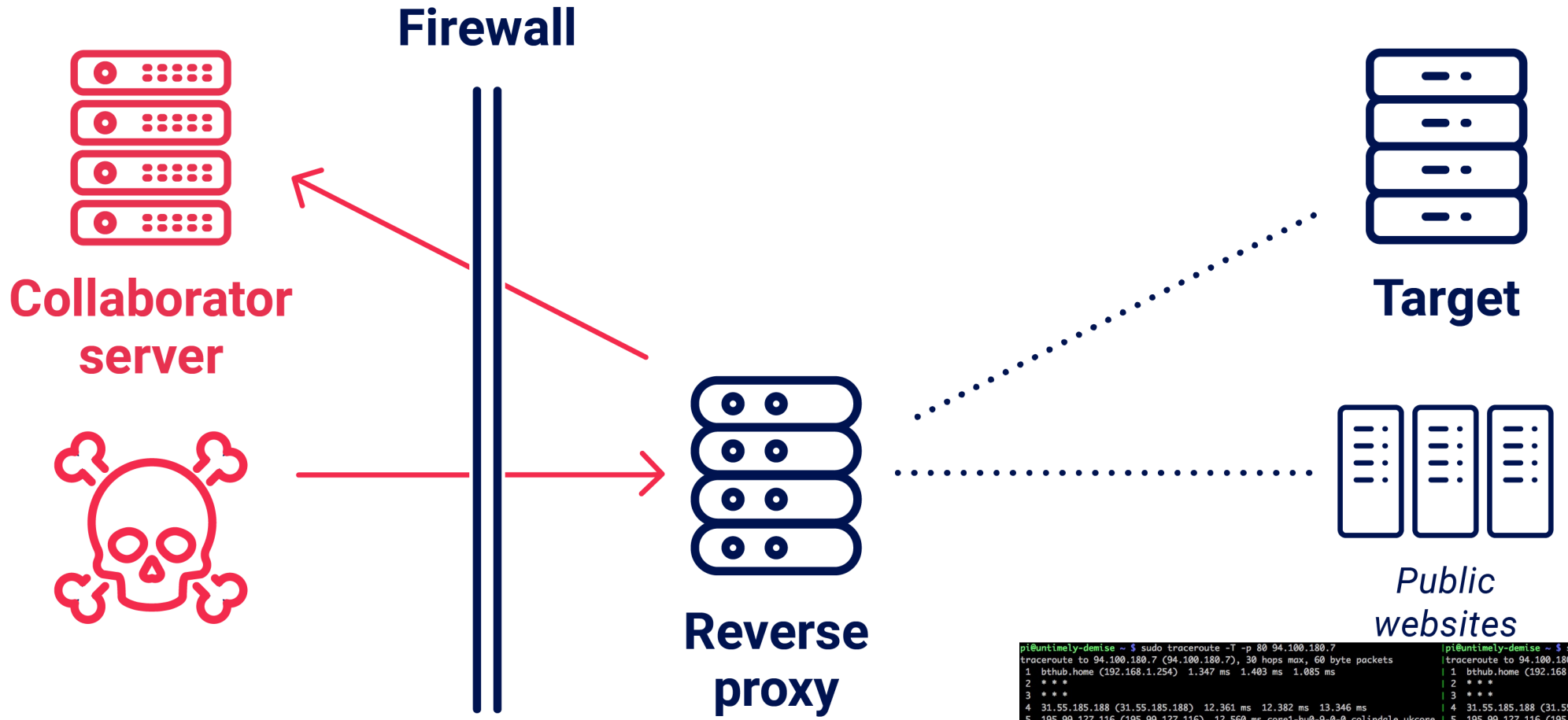
Equivalent but non-blind vulnerabilities useful for developing the understanding required for a successful timing-based exploit

Reverse Proxy Misconfigurations

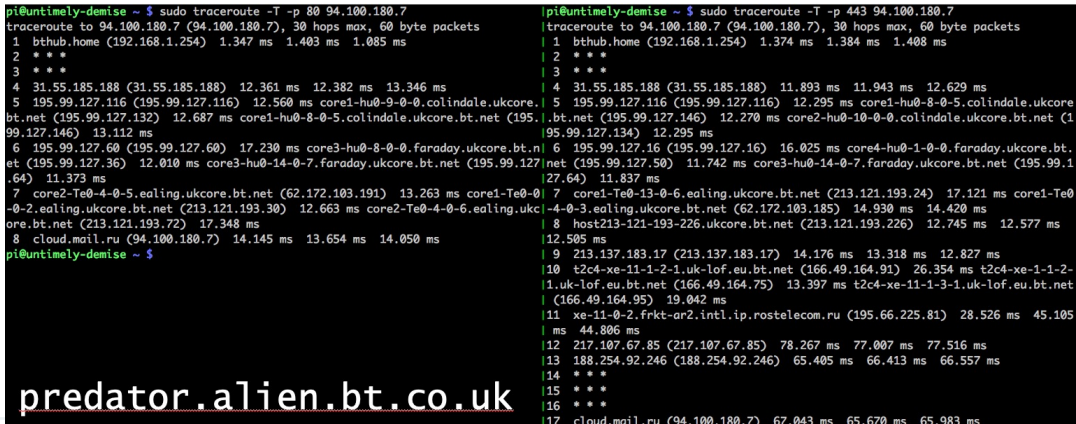
Detect scoped-SSRF
Find internal targets

will the front-end proxy to arbitrary subdomains?

SSRF via open reverse proxy




```
GET / HTTP/1.1
Host: xyz.burpcollaborator.net
```



The scoped-SSRF blind spot

Scoped SSRF: SSRF restricted to *.example.com

Caused by:

- Restricted server listener
- Internal-only DNS server
- Input validation

Payload in host header	Full SSRF	Scoped-SSRF
abc.example.com	404 Not Found	404 Not Found
abc.notexample.com	404 Not Found	403 Forbidden

-> Scoped-SSRF is invisible to DNS-pingback detection

Detecting scoped-SSRF

[↗] Detect scoped-SSRF

Host header	Response	Time
foo.example.com	404 Not Found	25ms
foo.random.com	403 Forbidden	20ms

abc.example.com	404 Not Found	25ms
abc.example.com	404 Not Found	20ms

Second response is faster due to DNS caching

aaa…{62}.example.com	404 Not Found	25ms
aaa…{63}.example.com	404 Not Found	20ms

Faster due to invalid DNS label length

[↗] Find internal targets

Subdomain sources:

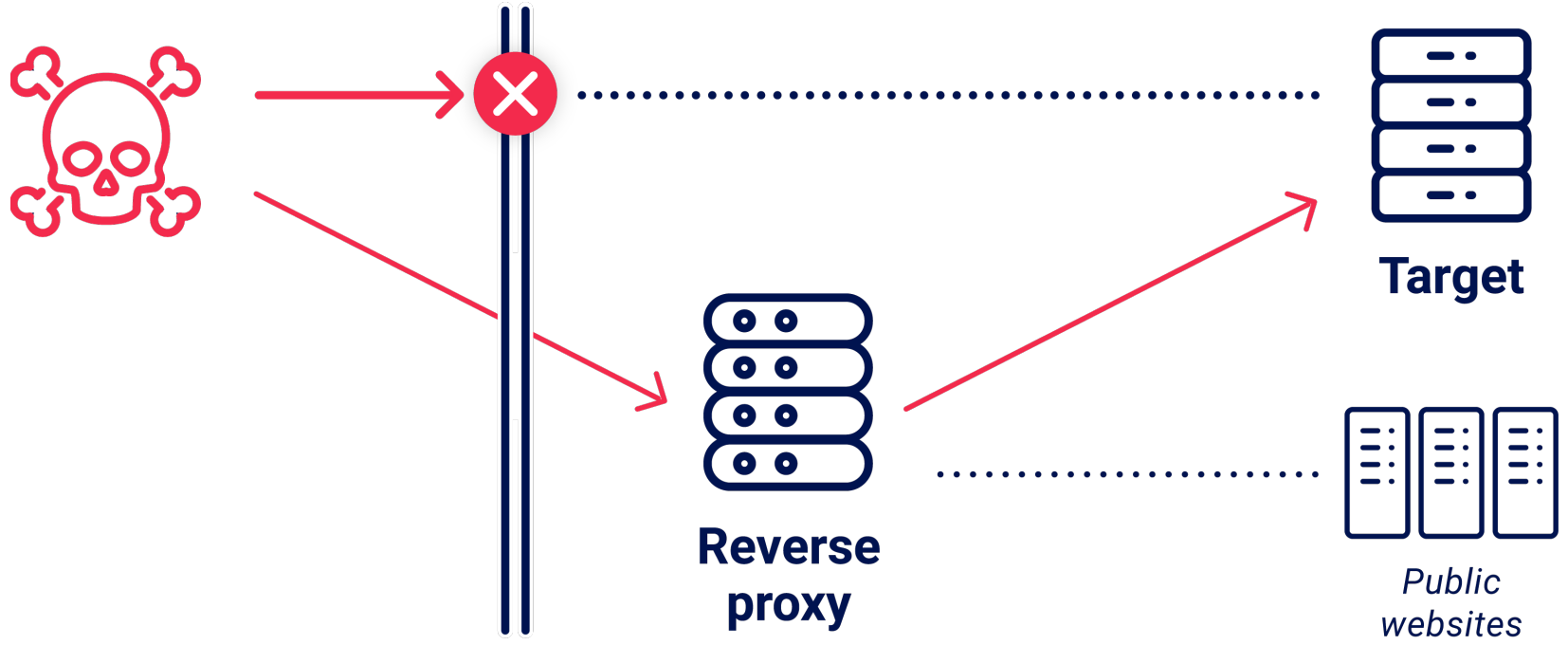
- 'fdns' DNS database from Rapid7 Project Sonar (58gb!)
- Online services: columbus.elmasy.com & dns.projectdiscovery.io

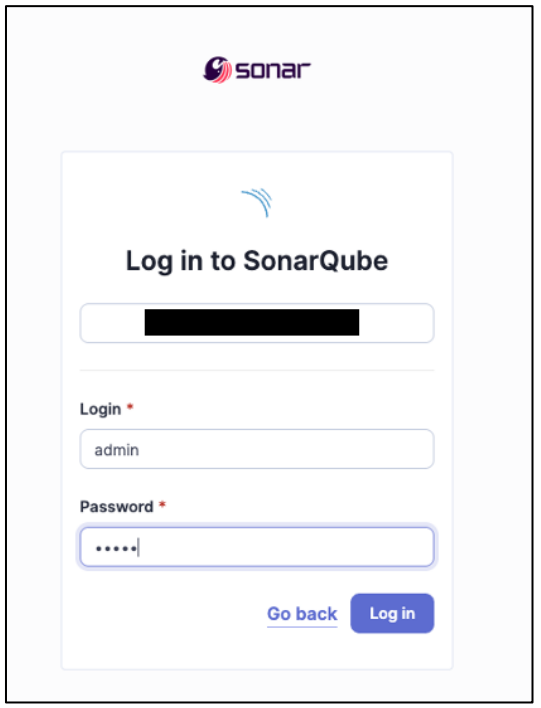
Entry point	Host header	Result

`mail.example.com`	`mail.example.com`	`HTTP/1.1 302 Found`
		`Set-Cookie: sid=abc`
		`X-Cache: miss`

`proxy.example.com`	`mail.example.com`	`HTTP/1.1 302 Found` `Set-Cookie: sid=def`

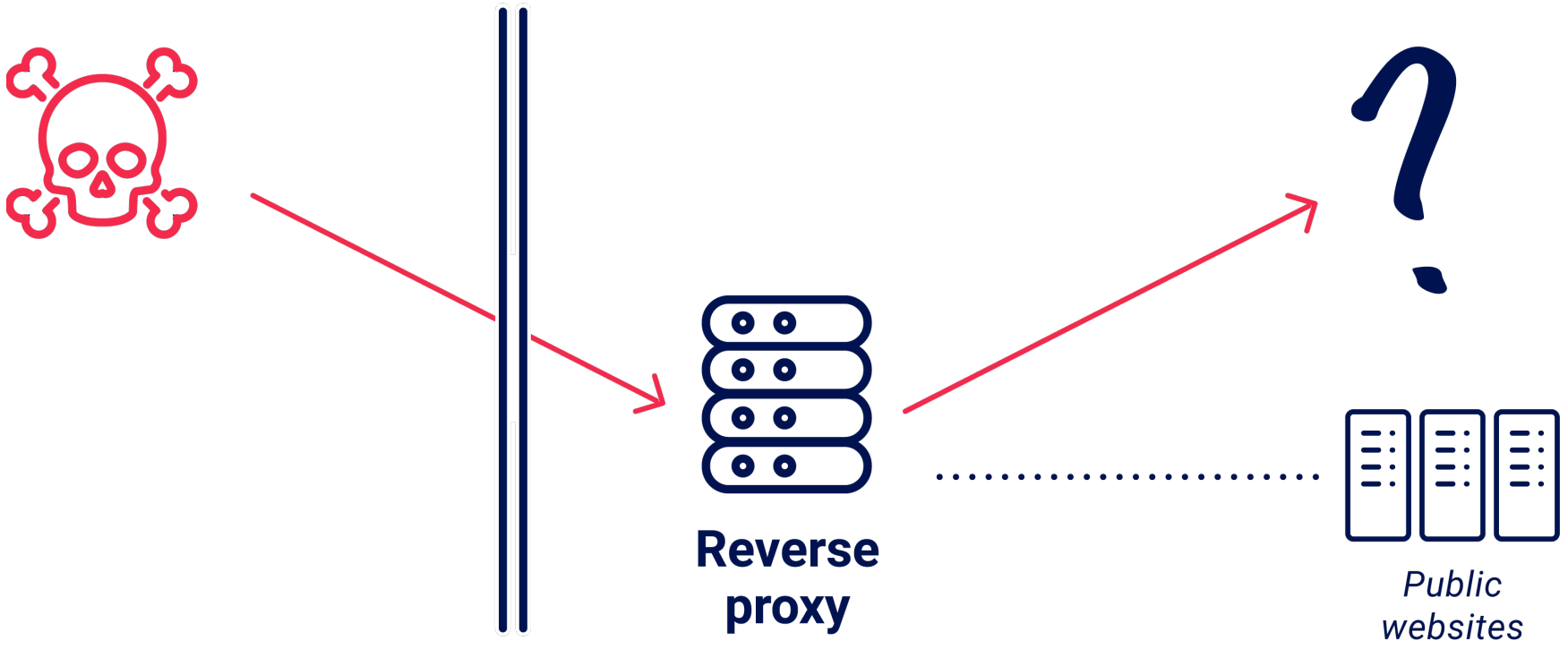
Firewall bypass





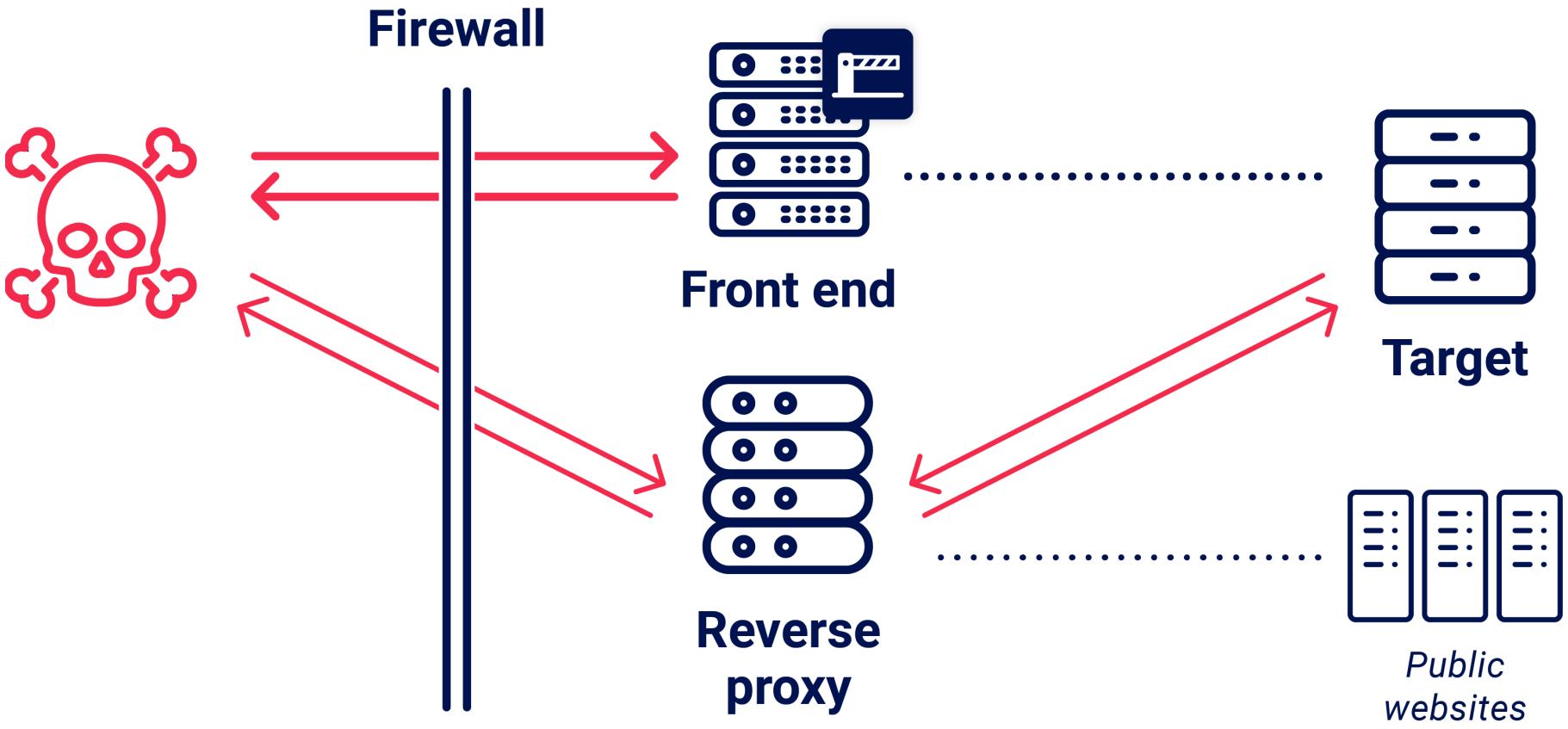
Entry point	Host header	Result
sonarqube.redacted	sonarqube.redacted	-reset-
app.redacted (proxy)	sonarqube.redacted	200 OK

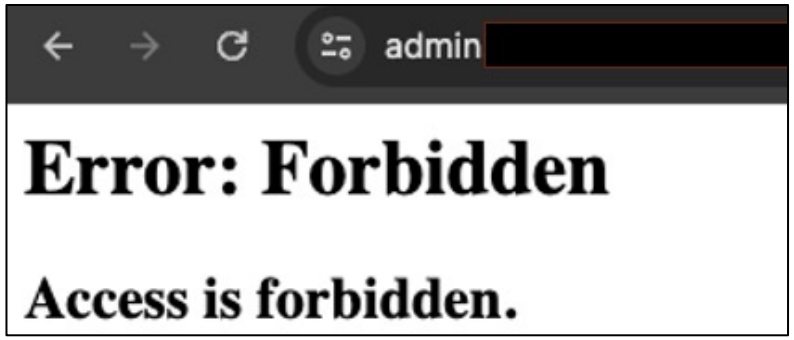
Firewall bypass – invisible route variant



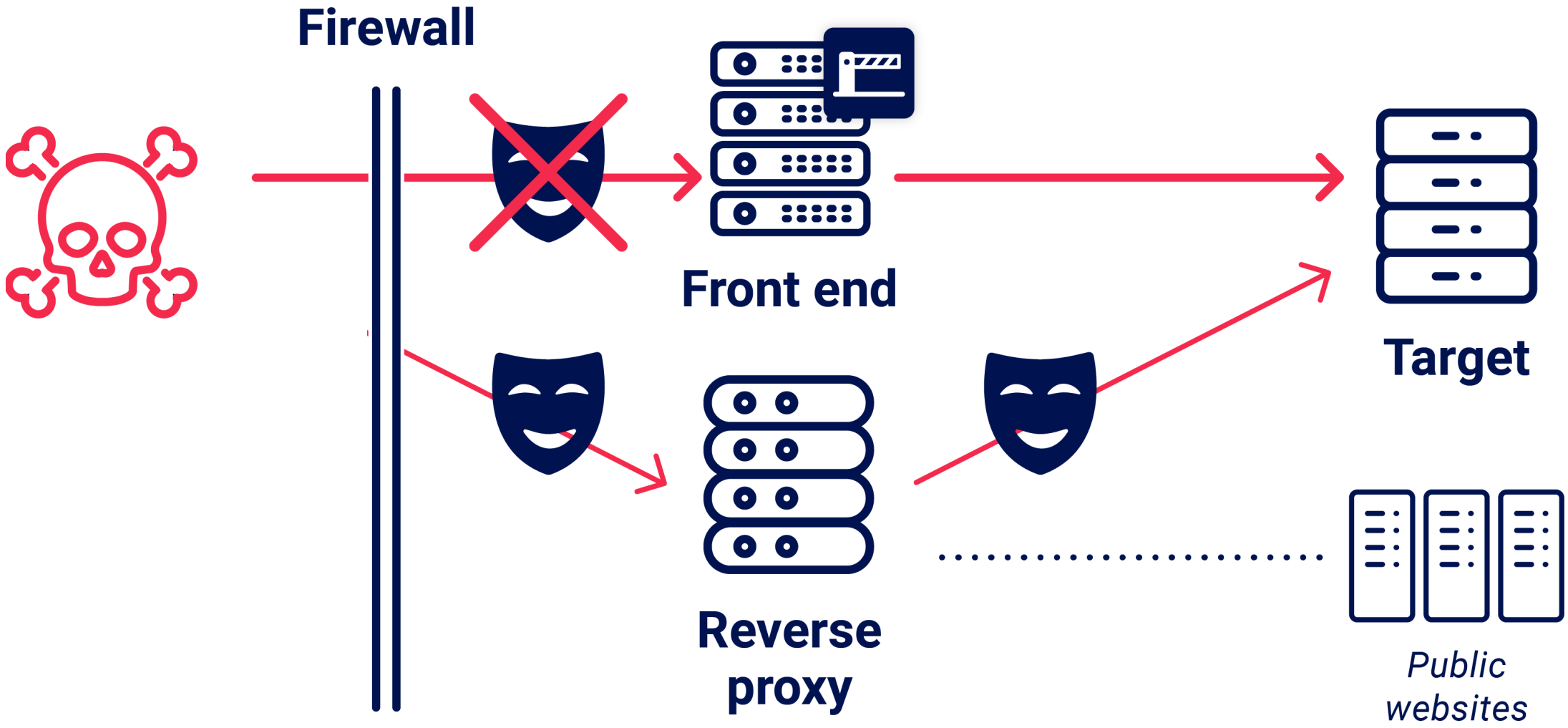
Entry point	Host header	Result
admin.redacted.gov	N/A	DNS probe fail
www.redacted.gov	admin.redacted.gov	200 OK

Front-end rule bypass





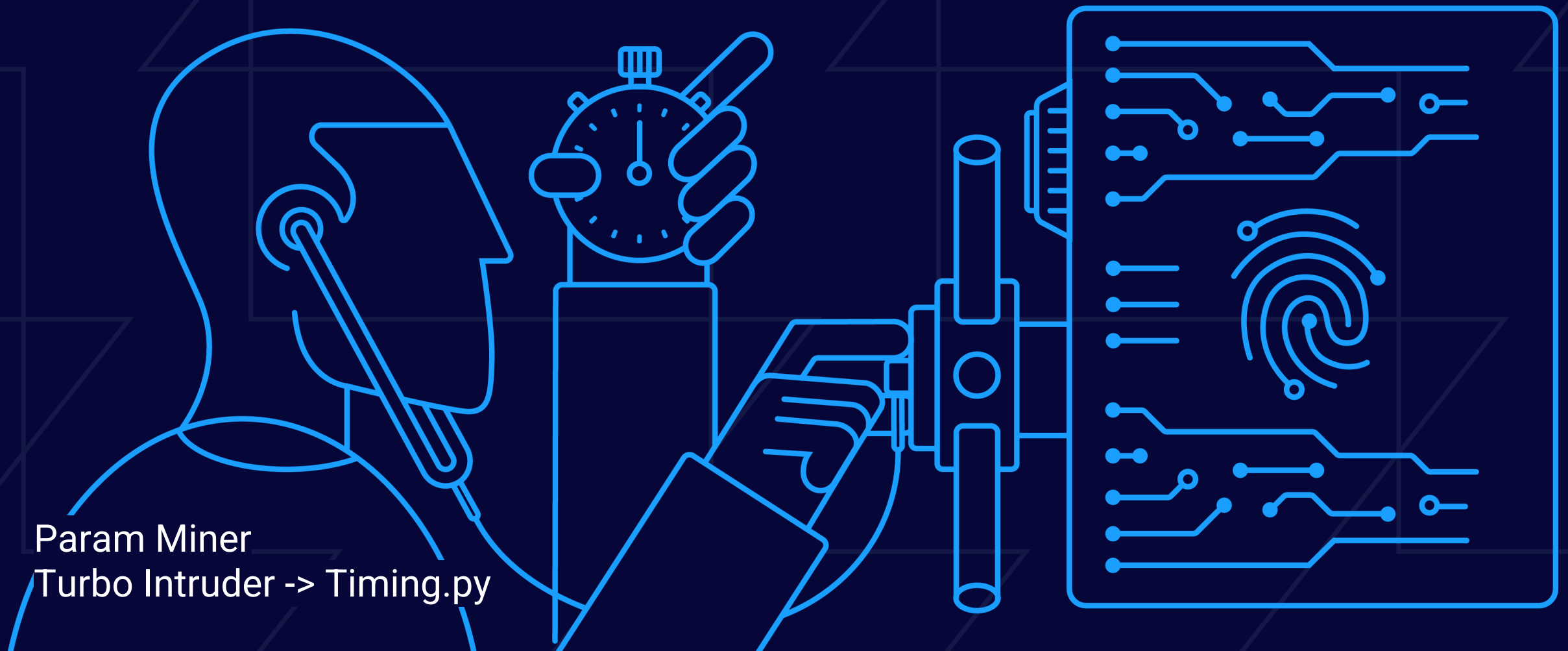
Front-end impersonation




```
Service-Gateway-Is-Newrelic-Admin: true
Service-Gateway-Account-Id: 934454
```

CTF

https://listentothewhispers.net/



Param Miner
Turbo Intruder -> Timing.py

Upcoming tool enhancements

- Enhance param-detection accuracy with single-packet-attack
- Enhance stealth & speed with t-test
- Your requests!

Feature requests here: https://github.com/portswigger/param-miner

Defense

Assume attackers have full execution flow visibility

Always request a PoC…

…but patch just in case
your security should not rely on noise

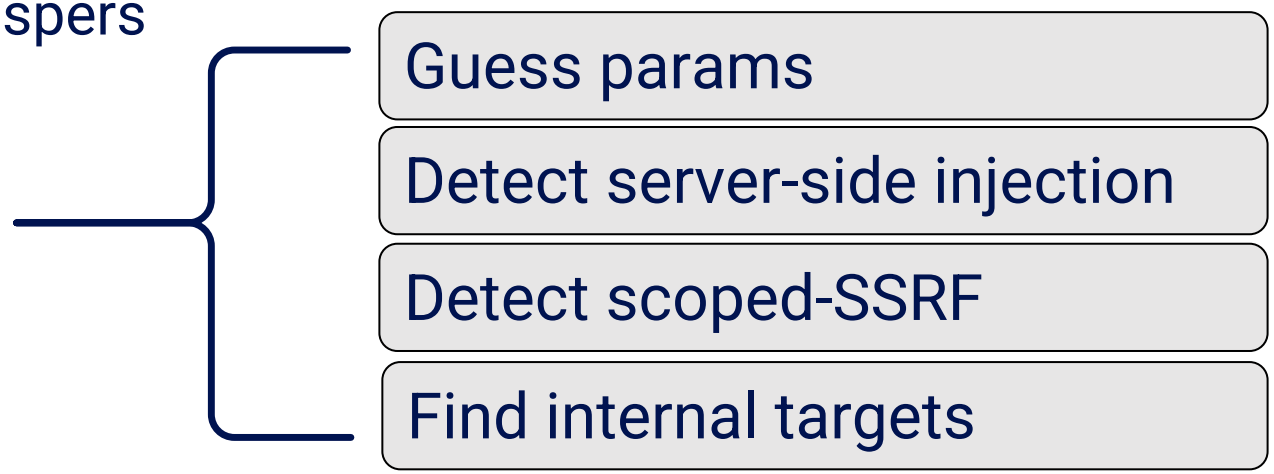
Break the single-packet attack

- WAF: stagger the packets
- Webserver: throttle to 1 request per 1-5ms per IP

References & further reading

Whitepaper, slides & CTF
portswigger.net/research/listen-to-the-whispers

Source code
github.com/PortSwigger/param-miner
github.com/PortSwigger/turbo-intruder



References & further reading:
martinschwarzl.at/media/files/compression.pdf
usenix.org/conference/usenixsecurity20/presentation/van-goethem
portswigger.net/research/the-single-packet-attack-making-remote-race-conditions-local
soatok.blog/2021/08/20/lobste-rs-password-reset-vulnerability/
www.ezequiel.tech/p/10k-host-header.html
www.youtube.com/watch?v=hWmXEAi9z5w
opendata.rapid7.com/sonar.fdns_v2/
portswigger.net/research/cracking-the-lens-targeting-https-hidden-attack-surface

Takeaways

Web timing attacks answer difficult questions

The single-packet attack makes them 'local', universal, and feasible

The murmurs are always there… waiting for you to listen

@albinowax

Email: james.kettle@portswigger.net

Paper: https://portswigger.net/research

PortSwigger