Smashing the state machine

the true potential of web race conditions

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/smashing-the-state-machine

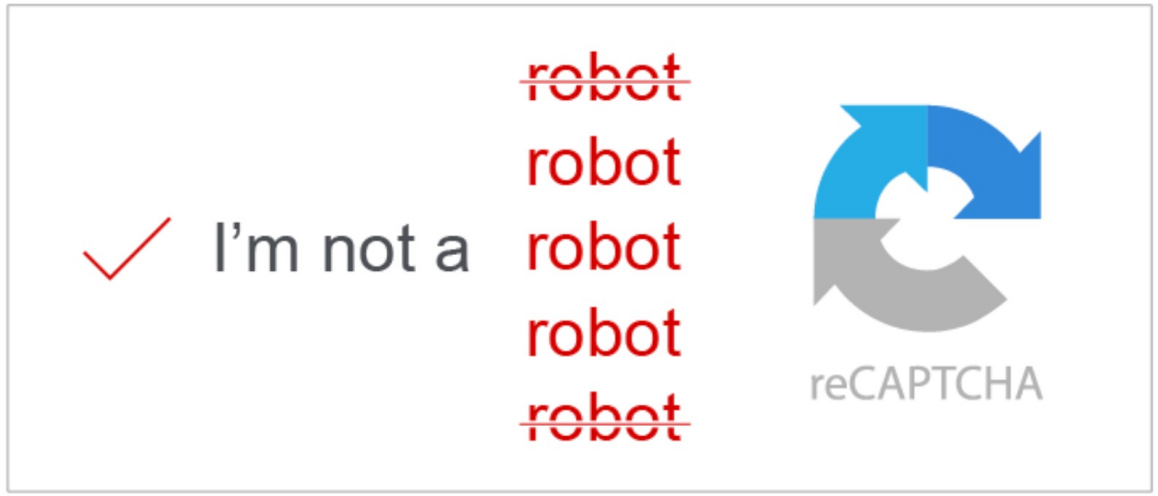
If it's not uploaded yet, you can get notified when it's ready by following me at https://twitter.com/albinowax

- albinowax

The known potential of race conditions

What have you seen?

[transfer/withdraw, redeem voucher, apply discount, review/rate, login]



Virtually all **limit-overrun:**

```
if (i < limit):
    i++
    do_action()
```

Exception: *Race conditions on the web*, by Josip Franjković

/confirmemail.php?e=user@gmail.com&c=13475&code=84751

Outline

The true potential

- Single-packet attack
- Strategy

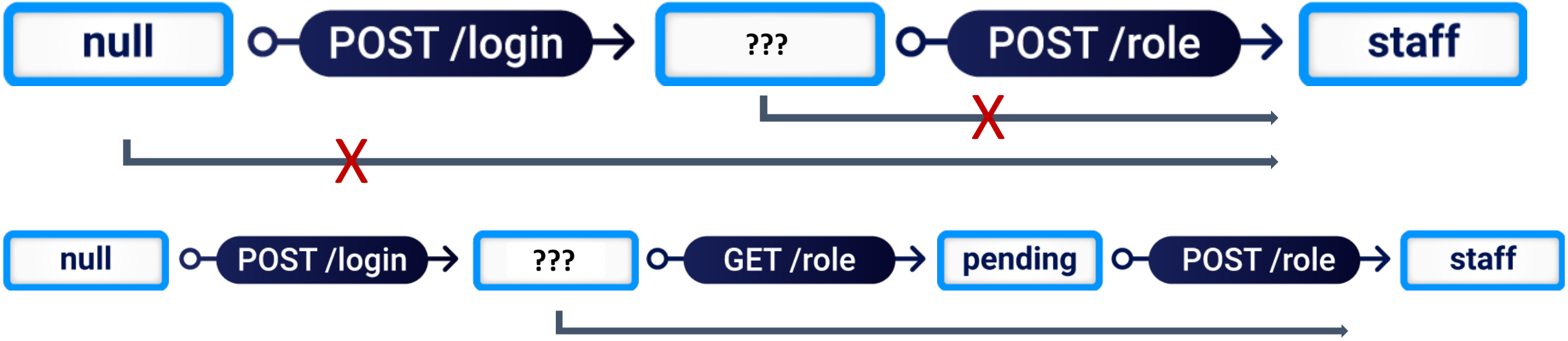
Case studies / Demo

Future research

Defense / Takeaways / Questions

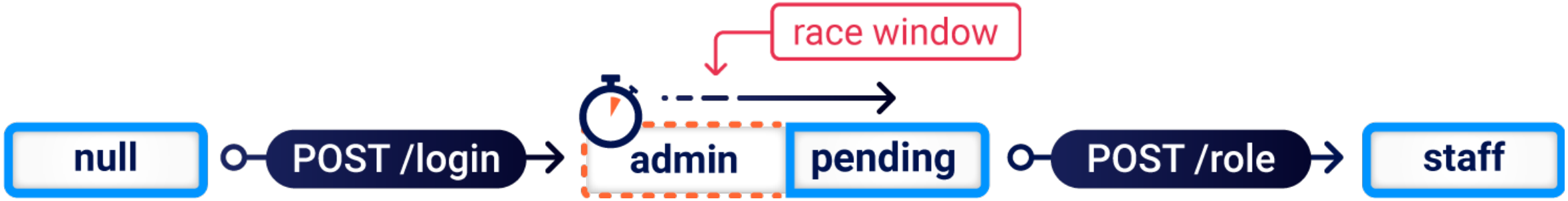
The true potential of race conditions

POST	/login	302 Found
GET	/role	200 OK
POST	/role	302 Found



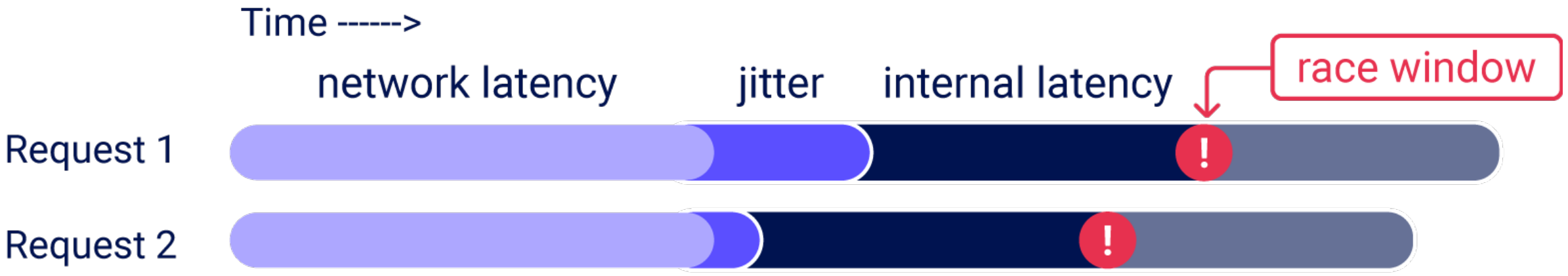
The true potential of race conditions

POST	/login	302 Found
GET	/role	200 OK
POST	/role	302 Found

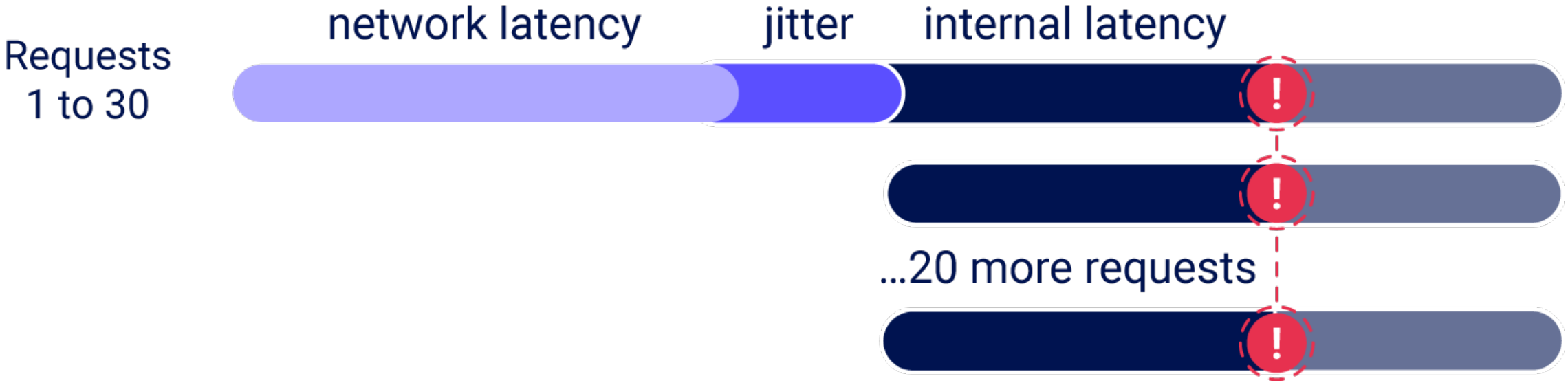


with race conditions, everything is multi-step

Making race conditions reliable: Single-packet attack

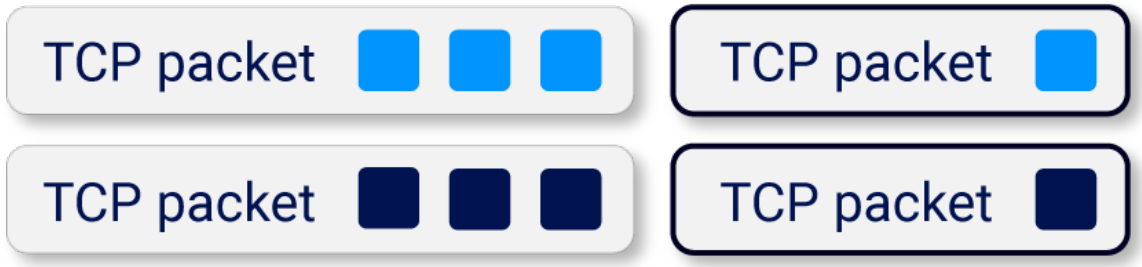


Single-packet attack

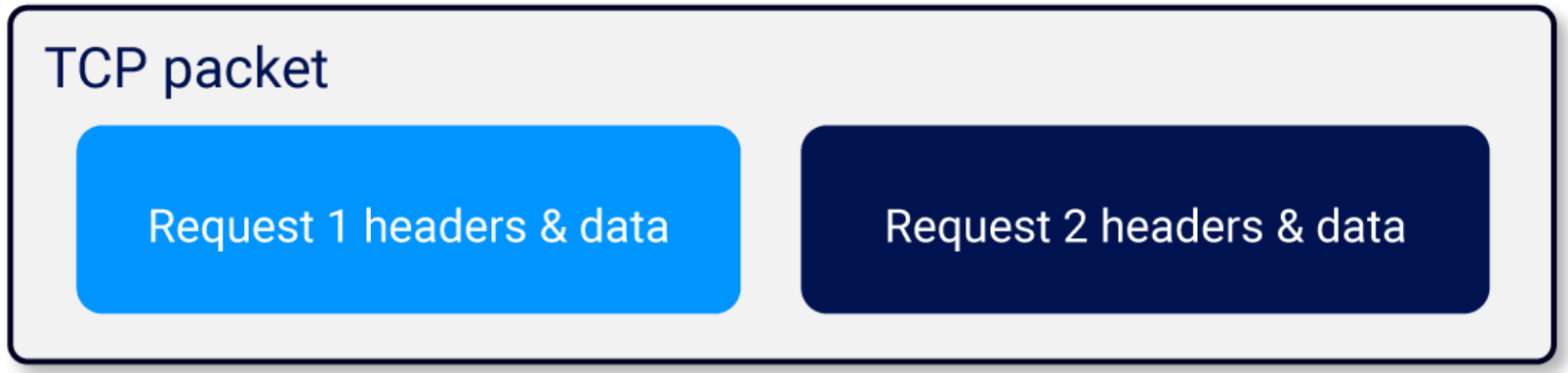


Single-packet attack: under the hood

Last-byte sync



Timeless timing attack



Single-packet attack

TURBO INTRUDER

Single-packet attack: the recipe

disable TCP_NODELAY // make the OS buffer packets

for each request with no body:
    send the headers
    withhold an empty data frame

// some servers process a request early if they see $content-length bytes
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 // the OS doesn't buffer the first frame after a delay
send the final frames
// reference implementation: https://github.com/portswigger/turbo-intruder

benchmark

20 requests ->

Melbourne ⟶ Dublin

17,208km

Last-byte sync:
Median spread: 4ms
Standard deviation: 3ms

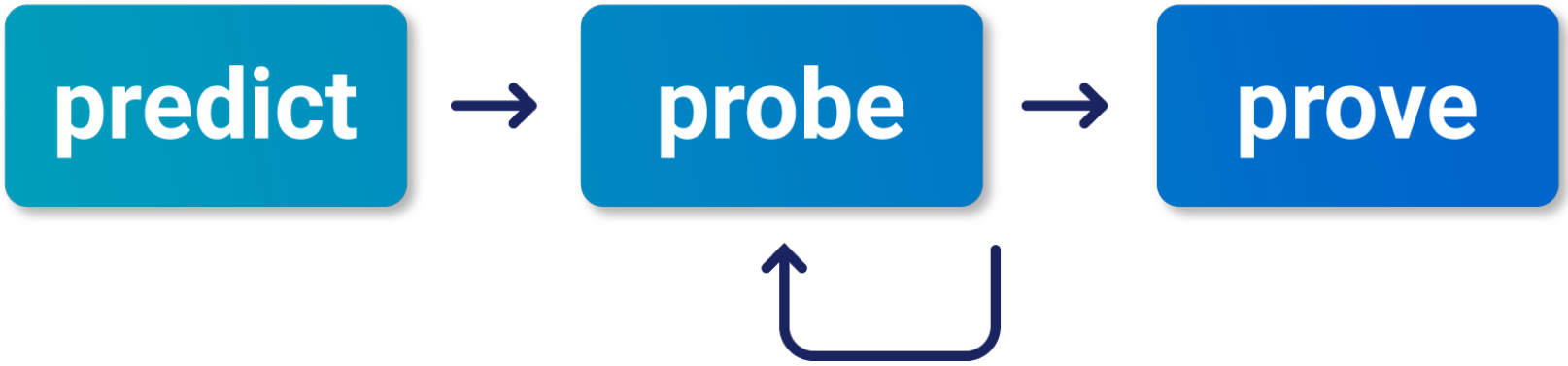
Single-packet attack:
Median spread: 1ms
Standard deviation : 0.3ms

▶ *4 to 10 times more effective*
30 seconds vs 2+ hours of attempts

The single-packet attack makes remote races local

https://github.com/portswigger/turbo-intruder/benchmark.py

Methodology



Predict potential collisions

Probe for clues

Prove the concept

Predict potential collisions

predict → probe → prove

Identify stateful objects/systems & map endpoints

- Users, sessions, orders...

Edit vs Append

- Does password reset invalidate previous reset links?

Will our requests affect the same record?

```
session=b94, userid=hacker
session=b94, userid=victim
```

userid	token
hacker	5623ea2acfc0d8
victim	677717aa16a917

```
session=b94, userid=hacker
session=b94, userid=victim
```

sessionid	userid	token
b94...	???	???

Probe for clues

predict → probe → prove

Craft chaotic blend of conflicting requests

Benchmark expected behavior

- Send request blend **in sequence**
- Analyze responses, timing, emails, side-effects…

Probe for clues

- Send request blend **in parallel**
- Look for anomalies
- No anomalies? Tune timing to tighten execution spread

Prove the concept

predict → probe → prove

Understand & clean

- Trim superfluous requests
- Tune the timing
- Automate retries

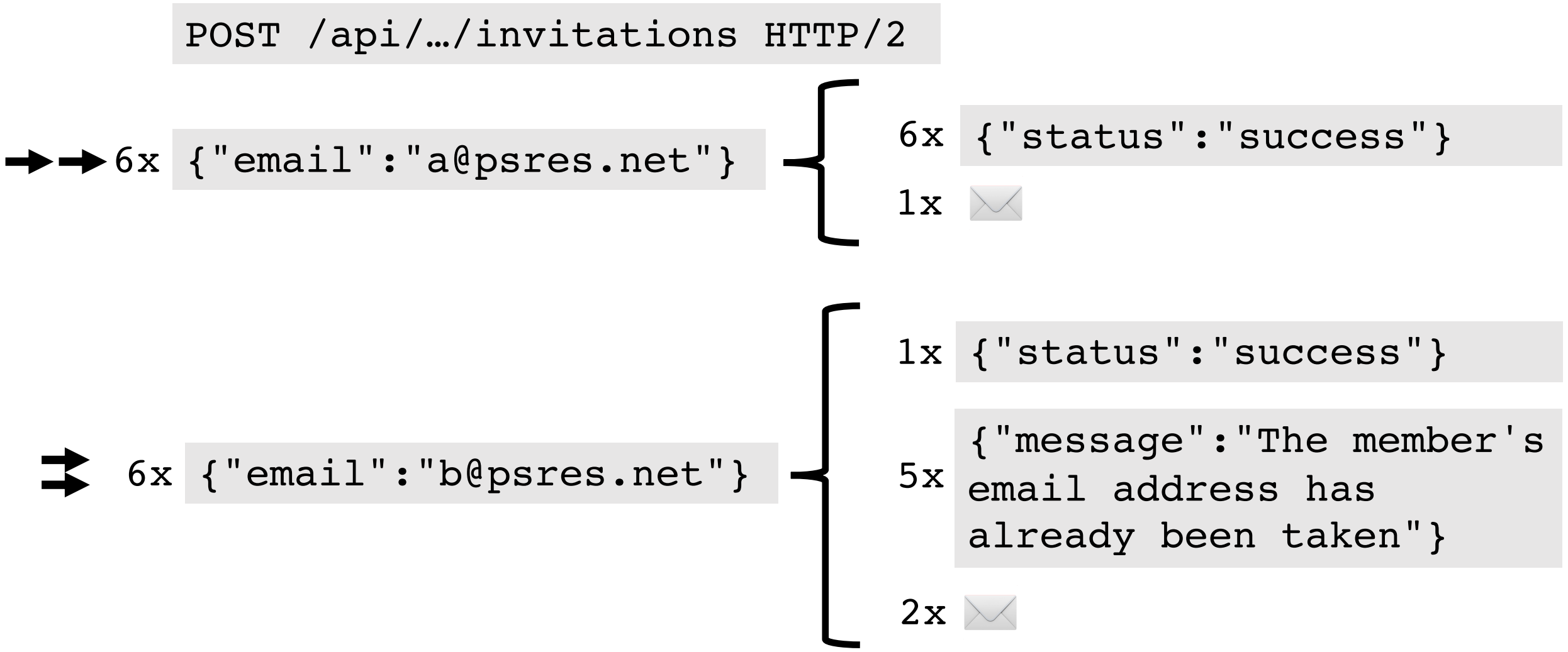
Explore impact

- Think of it as a structural weakness
- Look for chains & variations
- Don't stop at the first exploit

-$5,000

Case studies

Object-masking via limit-overrun



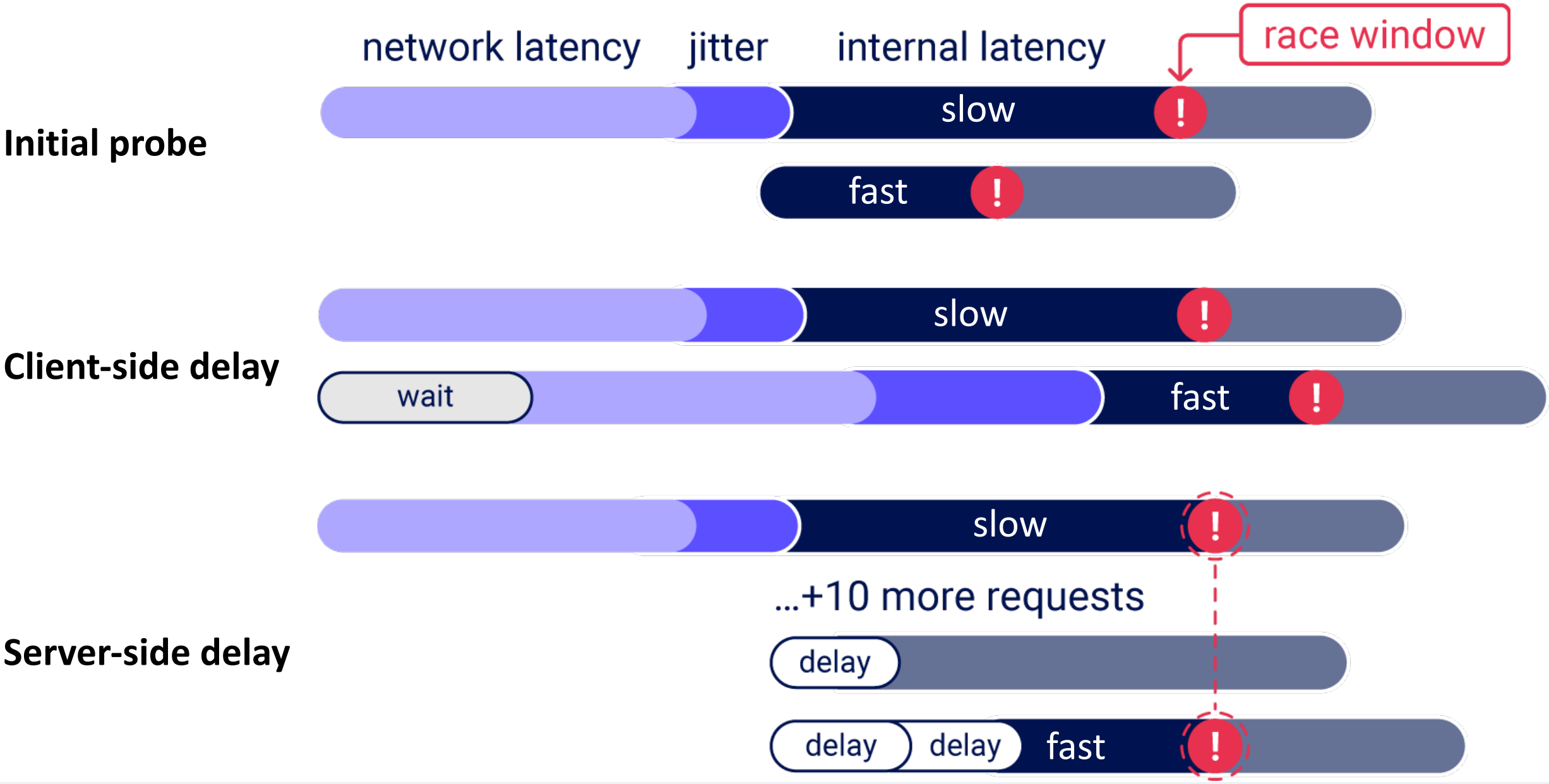
“User was successfully removed from project”

A multi-endpoint collision

Add to basket during checkout:

Gitlab email verification:

Multi-endpoint collisions: handling internal latency



Multi-endpoint collisions: handling internal latency

```
POST /-/profile HTTP/2

user[email]=x2@psres.net
```

```
GET /users/conf?token=vsz… HTTP/2
```

```
To: x2@psres.net
Subject: confirmation
…
x2@psres.net, confirm
your email address
```

```
POST /-/profile HTTP/2

user[email]=x2@psres.net
```

90ms

```
GET /users/conf?token=vsz… HTTP/2
```

```
To: x2@psres.net
Subject: confirmation
…
x1@psres.net, confirm
your email address
```

demo: single-endpoint collision!

https://gitlab.com/albinowax1

Single-endpoint collision code analysis

```
self.unconfirmed_email = self.email // from 'email' parameter
...
self.confirmation_token = @raw_confirmation_token = Devise.friendly_token
...
// this spins off a different thread to render & send the email (hint 1)
send_devise_notification(:confirmation_instructions,
                         @raw_confirmation_token,
                         { to: unconfirmed_email } )
```

```
To: unconfirmed_email
```

```
// template engine reads the variables back from the database
- confirmation_link = confirmation_url(confirmation_token: @token)
#content
  = email_default_heading(@resource.unconfirmed_email) // hint 2
  %p= _('Click the link below to confirm your email address.')
  #cta
    = link_to _('Confirm your email address'), confirmation_link
```

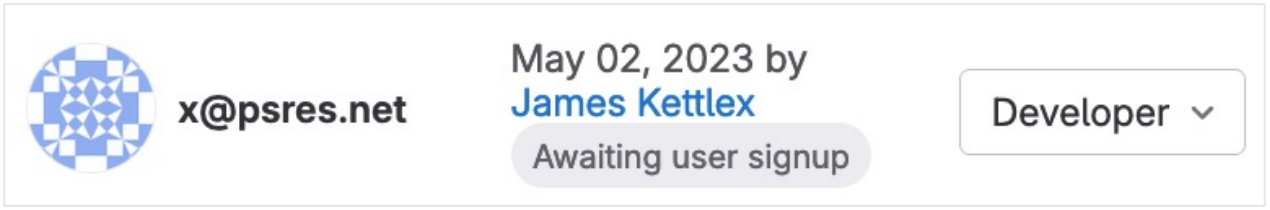
Impact

Gitlab

Attack #1: Invitation hijack

Attack #2: 'Sign in with Gitlab'

Patched in 15.7.2 on 4th Jan 2023



Devise - *"far and away the most popular authentication system for Rails"*

Reported to 4 addresses 200+ days ago. No patch.

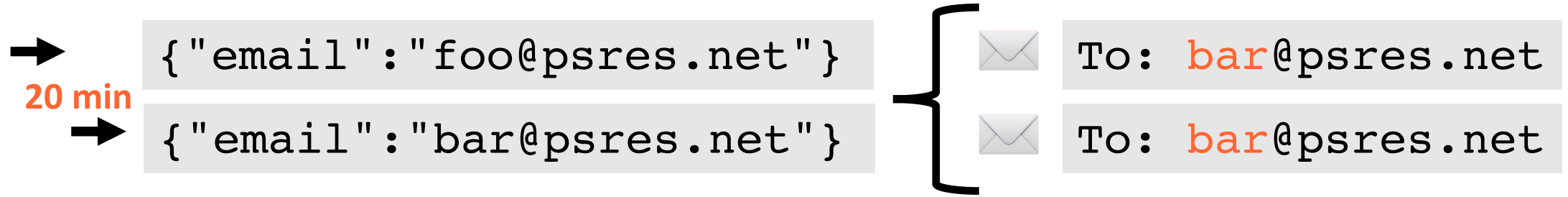
Easily detected via /users/confirmation

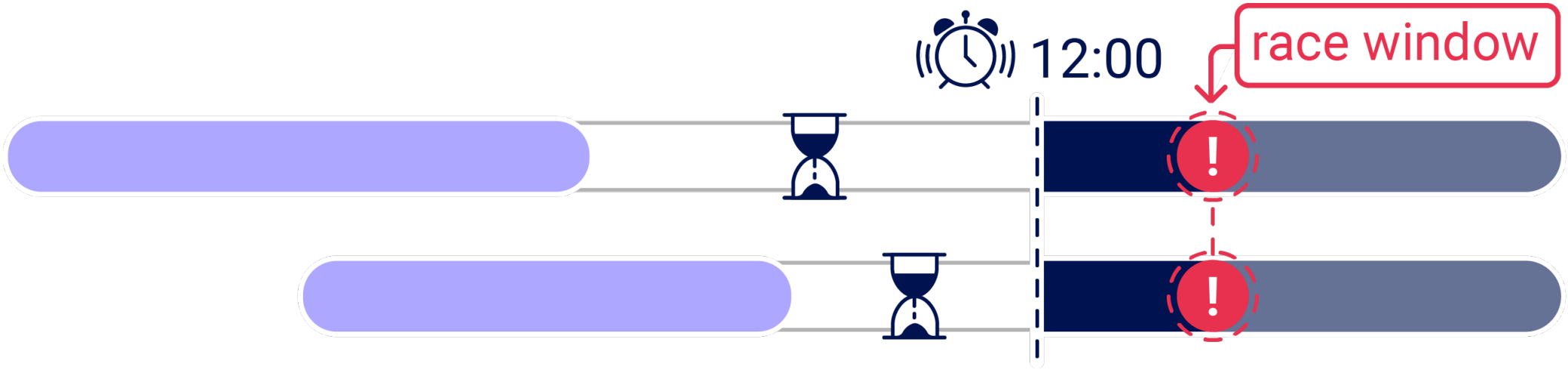
Case study highlights:

- Visible locking
- No-hint scenario
- Hidden endpoint

Deferred collisions

20 min





Timing is irrelevant, so volume is critical

Second-order clues are extremely valuable

Further research

Partial construction attacks

Object creation may contain a race window:

```
datastore.set(sessionid, 'user', user)
datastore.set(sessionid, 'token', rand(32))
```

Requirement 1: uninitialized value/state doesn't trigger exceptions

Requirement 2: Attacker can provide a matching value

```
[no token parameter]
token
token=
token=null
token[]=
{"token":null}
```

https://bugs.chromium.org/p/project-zero/issues/detail?id=2085

Data-structures and race-condition defenses

Locking

Seen in: PHP native sessions, database transactions

- Masks races in other layers

Batching

Seen in: most major session handlers and ORMs

- Entire record is read in, cached, and written back afterwards
- Internally consistent during request lifecycle
- Inconsistent across parallel requests, and background threads

No defence

Seen in: databases, custom session-handlers

- Not consistent during request lifecycle!

What if the session handler has no defence?

```
# Bypass code-based password reset
session['reset_username'] = username
session['reset_code'] = randomCode()
Exploit: Synced reset for $victim and $attacker
```

```
# Bypass 2FA
session['user'] = username
if 2fa_enabled:
        session['require2fa'] = true
Exploit: Synced login and sensitive page fetch
```

```
# Session-swap
session['user'] = username
set_auth_cookies_for(session['user'])
Exploit: Force session cookie on victim, then sync login
```

Improving the single-packet attack

Breaking the 30-request barrier

- Achievable with custom TCP/TLS stack via fake dropped packets
- Simpler/easier strategies may exist

Developing server-side precision

- Micro-delays to counteract TLS decryption time
- Longer delays for staggered attacks
- Generic techniques especially valuable

Defense

- Avoid sub-states
- Avoid mixing data sources
- Use datastore consistency features
  - Transactions
  - Atomic operations
  - Uniqueness constraints
- Know your session handler

References & further reading

Whitepaper, slides & academy topic

//portswigger.net/research/smashing-the-state-machine

//portswigger.net/web-security/race-conditions

Practice labs

Limit-overrun
Rate-limit bypass
Multi-endpoint
Single-endpoint
Partial construction

Source code

//github.com/PortSwigger/turbo-intruder

Templates

single-packet-attack
multi-endpoint
email-extraction
benchmark

References & further reading:

//josipfranjkovic.com/blog/race-conditions-on-web

//usenix.org/conference/usenixsecurity20/presentation/van-goethem

//aaltodoc.aalto.fi/bitstream/handle/123456789/47110/master_Papli_Kaspar_2020.pdf

//googleprojectzero.blogspot.com/2021/01/the-state-of-state-machines.html

//soroush.me/downloadable/common-security-issues-in-financially-orientated-web-applications.pdf

//portswigger.net/research/how-I-choose-a-security-research-topic

Takeaways

The single-packet attack makes race conditions reliable[ish]

With race conditions, everything is multi-step

Predict, probe, prove

PortSwigger

@albinowax
Email: james.kettle@portswigger.net
Paper: https://portswigger.net/research