HTTP
Encrypted
Information can be
Stolen through
TCP-windows

by
Mathy Vanhoef & Tom Van Goethem
Agenda

- Technical background
  - Same-Origin Policy
  - Compression-based attacks
  - SSL/TLS & TCP
- Nitty gritty HEIST details
- Demo
- Countermeasures
Same-Origin Policy

GET /vault

Mr. Sniffles

https://bunnehbank.com
Same-Origin Policy

GET /vault

Mr. Sniffles

https://bunnehbank.com
the World Wide Web

Mr. Sniffles

https://bunnehbank.com
Mr. Sniffles

https://bunnehbank.com
the World Wide Web

GET /vault

https://bunnehbank.com
the World Wide Web

GET /vault

https://bunnehbank.com
the World Wide Web

GET /vault

Mr. Sniffles

https://bunnehbank.com
the World Wide Web

GET /vault

https://bunnehbank.com
the World Wide Web

GET /vault

https://bunnehbank.com
the World Wide Web

Mr. Sniffles

GET /vault

https://bunnehbank.com
Agenda

- Technical background
  - Same-Origin Policy
  - **Compression-based attacks**
    - SSL/TLS & TCP
- Nitty gritty HEIST details
- Demo
- Countermeasures
You requested:
/vault

dev-secret=carrots4life
→ 51 bytes

You requested:
/vault
_secret=carrots4life
→ 47 bytes
You requested: `/vault?secret=a`

→ 50 bytes

You requested: `/vault?secret=c`

→ 49 bytes
You requested: 

```
/vault?secret=a
```

→ 50 bytes

```
/vault?secret=c
```

→ 49 bytes

49 bytes < 50 bytes → 'c' is a correct guess
You requested: `/vault?secret=ca`
_ rrots4life
→ 49 bytes

You requested: `/vault?secret=cb`
_ arrots4life
→ 50 bytes
You requested: `/vault?secret=ca`

→ 49 bytes

You requested: `/vault?secret=cb`

→ 50 bytes

49 bytes < 50 bytes → 'ca' is a correct guess
Compression-based Attacks

- Compression and Information Leakage of Plaintext [FSE'02]
  - Chosen plaintext + compression = plaintext leakage
- CRIME [ekoparty'12]
  - Exploits SSL compression
- BREACH [Black Hat USA'13]
  - Exploits HTTP compression
Agenda

- Technical background
  - Same-Origin Policy
  - Compression-based attacks
  - SSL/TLS & TCP
- Nitty gritty HEIST details
- Demo
- Countermeasures
GET /vault

crypt(
    GET /vault HTTP/1.1
    Cookie: user=mr.sniffles
    Host: bunnehb ank.com
)

1 TCP data packet
encrypt( ) = 29 TCP data packets
encrypt() = 29 TCP data packets

TCP packet 1
TCP packet 2
...
TCP packet 10

initcwnd = 10
encrypt( ) = 29 TCP data packets

TCP packet 1
TCP packet 2
...
TCP packet 10
10 ACKs

initcwnd = 10
encrypt(             ) = 29 TCP data packets

TCP packet 1
TCP packet 2
...  
TCP packet 10

10 ACKs

initcwnd = 10

cwnd = 20
encrypt( ) = 29 TCP data packets

TCP packet 1
TCP packet 2
...
TCP packet 10
10 ACKs
TCP packet 11
...
TCP packet 29

initcwnd = 10
cwnd = 20

HEIST
HEIST

- A set of techniques that allow attacker to determine the exact size of a network response

- ... purely in the browser

- Can be used to perform compression-based attacks, such as CRIME and BREACH, in the browser
Browser Side-channels

• Send authenticated request to /vault resource

```javascript
fetch('https://bunnehbank.com/vault',
  {mode: "no-cors", credentials: "include"})
```

• Returns a Promise, which resolves as soon as browser receives the first byte of the response

```javascript
performance.getEntries()[−1].responseEnd
```

• Returns time when response was completely downloaded
• Step 1: find out if response fits in a single TCP window
Fetching small resource: T2 - T1 is very small

- SSL handshake complete
- GET /vault
- fetch('...')
- initial TCP window sent
- initial TCP window received
- Promise resolves
- first byte received
- responseEnd
- TCP handshake complete
- initial TCP window sent
- Promise resolves
- first byte received
- responseEnd
- TCP handshake complete
Fetching large resource: T2 - T1 is round-trip time

TCP handshake complete
GET /vault
fetch('...')
SSL handshake complete

first byte received
initial TCP window received

T1

initial TCP window sent
Promise resolves

T2

ACK sent
second TCP window received

second TCP window sent
responseEnd
HEIST

- Step 1: find out if response fits in a single TCP window
- Step 2: discover exact response size
Discover Exact Response Size

initcwnd

second TCP window

Resource size: ?? bytes

Reflected content: x bytes
Discover Exact Response Size

initcwnd

second TCP window

Resource size: ?? bytes

Reflected content: x/2 bytes
Discover Exact Response Size

- initcwnd
- second TCP window

Resource size: ?? bytes
Reflected content: x/2 + x/4 bytes
After $\log(n)$ checks, we find:

- $y$ bytes of reflected content = 1 TCP window
- $y+1$ bytes of reflected content = 2 TCP windows

→ resource size = $\text{initcwnd} - y$ bytes
HEIST

• Step 1: find out if response fits in a single TCP window
• Step 2: discover exact response size
• Step 3: do the same for large responses (> initcwnd)
Determine size of large responses

• Large response = bigger than initial TCP window

• initcwnd is typically set to 10 TCP packets
  • ~14kB

• TCP windows grow as packets are acknowledged

• We can arbitrarily increase window size
GET /foo
10 TCP packets
10 ACKs

GET /vault
19 TCP packets
19 ACKs

CWND = 10
CWND = 20
sent in single TCP window
HEIST

• Step 1: find out if response fits in a single TCP window
• Step 2: discover exact response size
• Step 3: do the same for large responses (> initcwnd)
• Step 4: if available, leverage HTTP/2
Leveraging HTTP/2

• HTTP/2 is the new HTTP version
  • Preserves the semantics of HTTP
• Main changes are on the network level
  • Only a single TCP connection is used for parallel requests
Leveraging HTTP/2

• Determine exact response size *without* reflected content in the same response

• Use (reflected) content in other responses on the same server
  • Note that BREACH still requires (a few bytes of) reflective content in the same resource
GET /reflect?x=... = 6 TCP packets

GET /vault = 6 TCP packets

contains both /reflect and /vault

Promise resolves

responseEnd

CWND = 10

GET /reflect?x=... = 3 TCP packets

9 TCP packets

9 ACKs
GET /reflect?x=... = 5 TCP packets
GET /vault = 6 TCP packets

CWND = 10
CWND = 20

contains both /reflect and part of /vault
DEMO
Other targets

• Compression-based attacks
  • gzip compression is used by virtually every website

• Size-exposing attacks
  • Uncover victim's demographics from popular social networks
  • Reveal victim's health conditions from online health websites
  • ....

• Hard to find sites that are not vulnerable
Countermeasures

• Browser layer
  • Prevent side-channel leak (*infeasible*)
  • **Disable third-party cookies** (*complete*)

• HTTP layer
  • Block illicit requests (*inadequate*)
  • Disable compression (*incomplete*)

• Network layer
  • Randomize TCP congestion window (*inadequate*)
  • Apply random padding (*inadequate*)
Conclusion

• Collection of techniques to discover network response size in the browser, for all authenticated cross-origin resources

• Side-channel originates from subtle interplay between multiple layers

• Allows for compression-based and size-exposing attacks

• HTTP/2 makes exploitation easier

• Many countermeasures, few that actually work
Questions?

Mathy Vanhoef
@vanhoefm
mathy.vanhoef@cs.kuleuven.be

Tom Van Goethem
@tomvangoethem
tom.vangoethem@cs.kuleuven.be