Breaking privacy and security by abusing cross-origin resource size

by Tom Van Goethem
Overview

• Introduction
  • Web 101; same-origin policy

• Exposing cross-origin resource size
  • Browser-based timing attacks
  • Browser cache
  • TCP windows

• Defence mechanisms
Introduction
• What happens when I open https://twitter.com/?
  • DNS resolution of twitter.com
  • TCP connection to 199.16.156.198:443
  • set up SSL connection
  • send GET / request with headers (User-Agent, Cookie, ...)
  • receive response for /
  • parse & render HTML
  • fetch other resources (JS, IMG, CSS, ...), possibly from other origins
  • cache resources
  • ???
I DON’T KNOW. WHAT DO I KNOW ABOUT IT? ALL I KNOW IS WHAT’S ON THE INTERNET.
• What happens when I open https://attacker.com/?

• DNS resolution of attacker.com
• TCP connection to 13.33.33.37:443
• set up SSL connection
• send GET / request with headers (User-Agent, Cookie, ...)
• receive response for /
• parse & render HTML
• fetch other resources (JS, IMG, CSS, ...), possibly from other origins
• cache resources
• ???
GET / HTTP/1.1
Host: foo.com
User-Agent: Victim-browser
Cookie: foo_session=bar_42

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 6720

<html><head><title>...
// using <img>
let i = new Image();
i.src = 'https://foo.com/';

// using <video>
let v = document.createElement('video');
v.src = 'https://foo.com/'
// using Fetch API
let opts = {
  "mode": "no-cors",  // don't use CORS
  "credentials": "include"  // attach cookies
};
fetch('https://foo.com/', opts).then(function(resp) {
  console.log('yay! a response!');
});
Can not access content of cross-origin resources
GET / HTTP/1.1
Host: foo.com
User-Agent: Victim-browser
Cookie: foo_session=bar_42

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 6720

<html>
  <head>
    <title>Welcome, Mr. Smith</title>
  </head>
  ...

John Smith
GET /search?q=delete+emails HTTP/1.1
Host: clinton-mail.com
User-Agent: Hillary
Cookie: sess=3727c5a4c0a97e98

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 536720

<html>
<head>
  <title>8410 results</title>
</head>
...
GET /search?q=email+security HTTP/1.1
Host: clinton-mail.com
User-Agent: Hillary
Cookie: sess=3727c5a4c0a97e98

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 29154

<html>
<head>
  <title>5 results</title>
</head>
...
Exposing cross-origin resource size

Timing attacks
GET /batman/
GET /joker/
GET /joker/
Classic Cross-site Timing Attacks

- Classic timing attacks have several limitations
  - Network irregularities
  - gzip compression
  - Round-trip for each measurement
  - Rate-limiting
Browser-based Timing Attacks

- Timing attacks in browsers overcome these limitations
  - Timing measurement starts *after* resource is downloaded
  - Measurements are more accurate
  - For some attacks: resource is only downloaded once
  - Obtain multiple measurements in short interval
Exposing cross-origin resource size

Browser-based timing attacks
GET /batman/
Browser-based Timing Attacks

- Side-channels allow measuring time to process resource
  - Parse as specific format (~ CPU processing time)
  - Retrieve from cache (~ disk read time)
  - Store in cache (~ disk write time)
let video = document.createElement('video');

// suspend => download complete
video.addEventListener('suspend', function() {
    start = window.performance.now();
});

// error => parsing complete
video.addEventListener('error', function() {
    end = window.performance.now();
});

video.src = 'https://example.org/resource';
let video = document.createElement('video');

// suspend => download complete
video.addEventListener('suspend', function()
{
    start = window.performance.now();
});

// error => parsing complete
video.addEventListener('error', function()
{
    end = window.performance.now();
});

video.src = 'https://example.org/resource';
let url = 'https://example.org/resource';
let opts = {credentials: "include", mode: "no-cors"};
let request = new Request(url, opts);
let bogusReq = new Request('/bogus');
fetch(request).then(function(resp) {
    // Resource download complete
    start = window.performance.now();
    return cache.put(foo, resp.clone());
}).then(function() {
    // Resource stored in cache
    end = window.performance.now();
});
The graph compares the average time to perform timing attacks under different conditions, with the x-axis representing the difference in file size (kB) and the y-axis showing the average time in milliseconds (ms). The graph includes the following types of attacks:

- Classic timing attack
- Video parsing attack
- Cache storing attack
- Cache + video parsing

The chart illustrates how the average time to perform the attacks changes with varying file size differences, with distinct markers for each type of attack.
Demo
Limit Visibility of this Post

Choose who can see your post on Facebook based on their demographic. For example, if you enter "Spanish" below, only people who have Spanish set as their language on Facebook or list Spanish as one of their languages on their Profile will be eligible to see your post on your Page, in News Feed and in Search. Learn more.

Locations
Enter country

- Everywhere
- By State/Province
- By City

Gender
- All
- Men
- Women

Age
13 - 65+

Languages
Enter language

Save Post Settings  Cancel
Age-discovery Attack

1. Create Facebook posts, each targeted to users of a specific age

2. Discover age-range of the user
   - Fetch corresponding resources
   - Obtain timing measurements
   - Determine age-range according to the value of timing measurements

3. Discover exact age of the user
   - Repeat (2) but for posts targeted to specific age
Timing Attack: Detect Facebook Age

Status: Start downloading resources
Time elapsed: 0.52s
Timing Attack: Detect Facebook Age

Status: Obtaining measurements
Time elapsed: 0.865s
Timing Attack: Detect Facebook Age

Status: Start downloading resources
Time elapsed: 11.102s
Discovered age-range: 23-32
https://labs.tom.vg/
Moar Attacks

- Facebook: demographics
- LinkedIn: connections, ...
- Twitter: following, identity, ...
- Google: search history
- Amazon: shopping history
- Gmail: inbox search
- ...

VULNERABILITIES EVERYWHERE

VULNERABILITIES
Exposing cross-origin resource size

Browser cache
Browser Storage Side-Channel Attacks

• Leverage browser's Cache API
  • Programmable cache
  • Store any (including cross-origin) resources in a cache

• Available space is limited per site

• Discovered 3 different attack techniques
  • Per-site quota, global quota, Quota Management/Storage APIs
Per-site quota
Per-site quota

http://h4x.com

@MrBunnsy

http://h4x.com
Per-site quota

Twitter

h4x.com

@MrBunnsy

h4x.com
Per-site quota
Per-site quota

- h4x.com
- @MrBunnsy

x
Per-site quota
Per-site quota
Per-site quota

\[ x - y = 172,046 \text{ bytes} \]
Quota Management/Storage APIs

@MrBunnsy

h4x.com

@MrBunnsy

x
Quota Management/Storage APIs

generateEstimate()
Quota Management/Storage APIs
Quota Management/Storage APIs

@MrBunnsy

h4x.com

x
Quota Management/Storage APIs
Quota Management/Storage APIs

getEstimate() → y bytes

@MrBunnsy

h4x.com
Quota Management/Storage APIs

\[ y - x = 172,046 \text{ bytes} \]
Real-world consequences

- User identification
  - e.g. by Twitter username

- Revealing private information
  - e.g. discover disease entered on WebMD

- Search-oriented information leakage
  - e.g. GMail search [Gelernter: CCS'15]

- Infer cross-origin cache operations
  - e.g. detect group membership on Telegram
DEMO
Exposing cross-origin resource size

TCP windows
GET /vault

**TCP handshake**

SYN

SYN, ACK

ACK

**SSL handshake**

Client Hello

Server Hello

Pre-Master Secret
encrypt(
    GET /vault HTTP/1.1
    Cookie: user=mr.sniffles
    Host: bunnehbinkbank.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

• 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
HEIST

• Step 1: find out if response fits in a single TCP window
Fetching small resource: T2 - T1 is very small

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

TCP handshake complete

GET /vault

SSL handshake complete

fetch('...')

initial TCP window sent

first byte received

initial TCP window received

Promise resolves

T1

ACK sent

second TCP window received

T2

second TCP window sent

responseEnd

T1 - T2 is round-trip time
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

Resource size: ?? bytes

second TCP window

Reflected content: x/2 bytes
Discover Exact Response Size

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
$\rightarrow$ 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

= 19 TCP data packets

CWND = 10

CWND = 20
GET /foo
10 TCP packets
10 ACKs
CWND = 10

GET /vault
19 TCP packets
19 ACKs
sent in single TCP window
CWND = 20
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 = 9 TCP packets

contains both /reflect and /vault

Promise resolves

responseEnd

CWND = 10

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

GET /vault = 6 TCP packets

GET /reflect?x=... = 6 TCP packets

Promise resolves

10 TCP packets

10 ACKs

1 TCP packet

1 ACK

CWND = 10

CWND = 20

contains both /reflect and part of /vault

responseEnd
Defence mechanisms
• The size of resources can leak at various layers
  • → Defence layers can be applied at various layers

• Very few defences work properly

• Often a tradeoff between performance/usability and security

• What “security grade” do we want?
  • Does a rough estimation of the resource size already leak information?
- Network layer
  - Add random padding
    - Not resilient against statistical attacks
    - Increases bandwidth
  - Add random delays
    - Affects performance
  - Randomize TCP window size
    - Is the possible variability sufficient?
• HTTP layer
  • Block requests triggered by attacker.com
  • Hard to determine originator of the request
  • Disable compression
    • Only prevents compression-based attacks
    • Affects network bandwidth
    • Only disable compression for secret/private information?
• **Browser layer**
  - Add random padding to cached Response objects
    - Work in progress (~ 9 months, and counting)
    - Reduces accuracy of exposed resource size
  - Disable third-party cookies
    - Breaks (a small part of) the web :-(
  - SameSite cookies
    - Cookies only included in same-site requests
    - Promising feature (when adopted)
Conclusion

- Resource size can leak sensitive information

- Various techniques exist that can reveal the size of cross-origin resources
  - Browser-based, network-based

- Variety of defence methods, few that work properly
  - Disable third-party cookies by default?
Questions?

@tomvangoethem

tom.vangoethem@cs.kuleuven.be