To CORS!

Goals and Objectives

- <== YOU ARE HERE
- Introductions
- Web Applications in 2019
- What is the Same-Origin Policy?
- What is Cross-Origin Resource Sharing?
- Common Security Concerns w/CORS
- CORS Exploitation Framework
- Final Thoughts

Tim Tomes

- @lanmaster53
- Jesus Freak
- Husband, Father, and Veteran
- Coder, Breaker, Teacher, Entrepreneur, and Sharer
  - PortSwigger Preferred Training Partner
  - https://www.lanmaster53.com
- "Burp Suite master and king of making HTTP requests tremble."

Kevin Cody

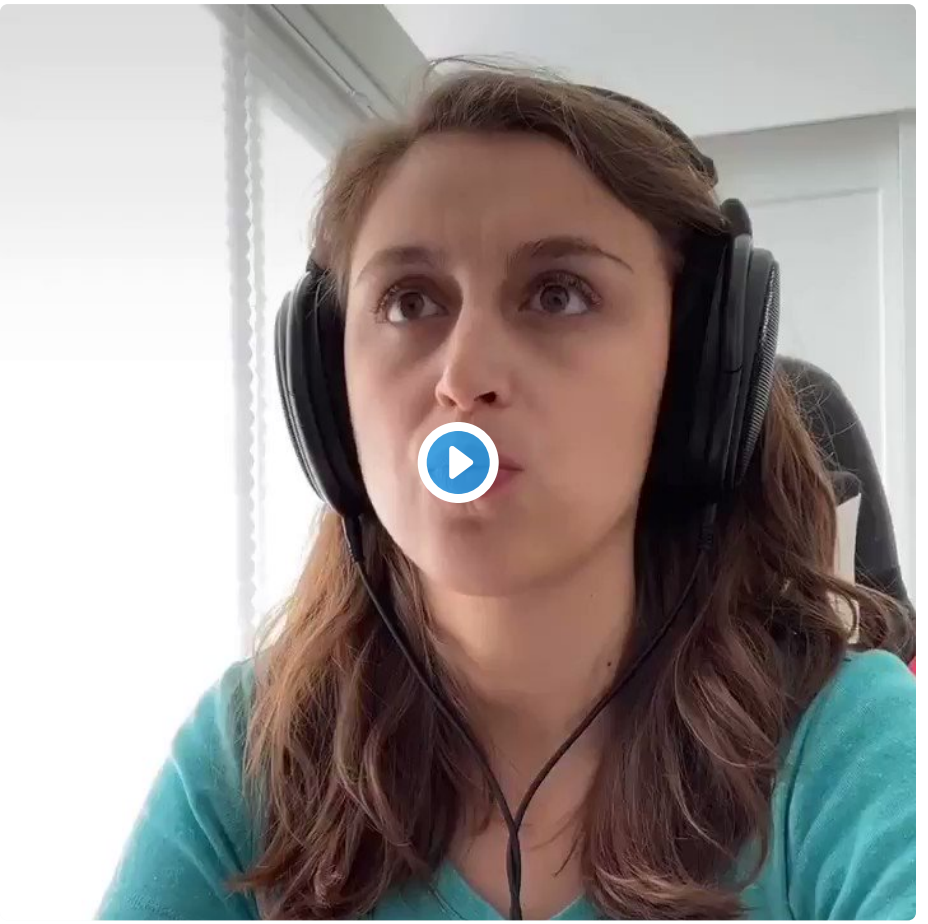
- @kevcody
- Pittsburgh, PA
- Autonomous Vehicle Hacker
- Vulnerability Stumble-uponer-er
- OWASP PGH Chapter Leader
- Husband/Dad/Hacker/Lock Picker/DIYer (via Youtube)/Fisherman/ADD (flavor-of-the-moment)

State of Web Applications in 2019

- Single-Page Applications
  - JavaScript and client-side heavy.
  - Sometimes client-side only!
- Emerging asynchronous communications.
  - Bayeux
  - CometD
  - Long Polling
  - WebSockets
- Microservices with sub or super domains.

Cassidy Williams
@cassidoo

Of cors this would happen



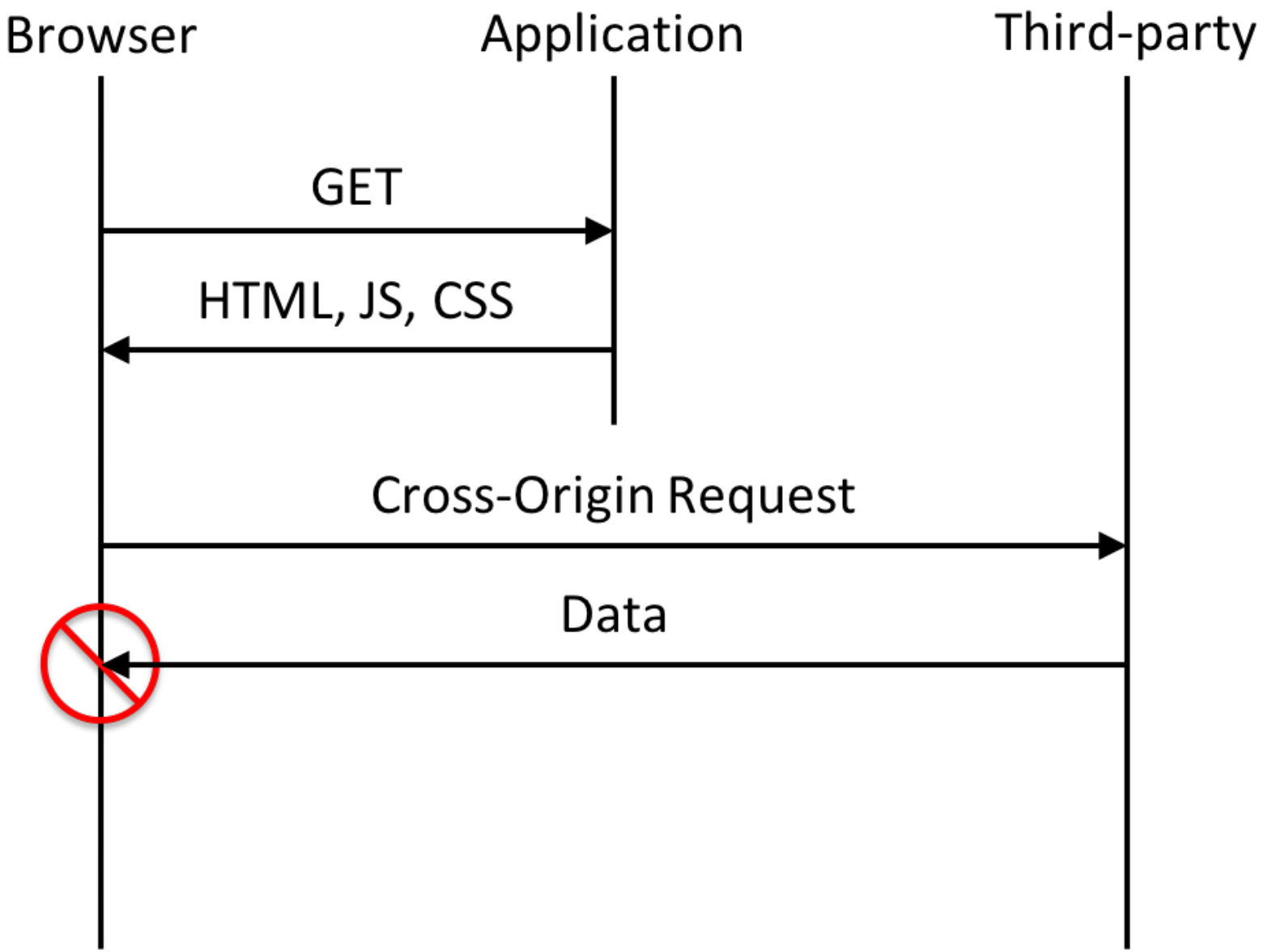
♡ 10.6K 8:29 PM - Aug 5, 2019

2,264 people are talking about this

Same-Origin Policy (SOP)

- Means same protocol, host, and port.
  - `[http]://[api.pwnedhub.com]:[80]`
  - vs. `https://api.pwnedhub.com`
  - vs. `http://pwnedhub.com`
  - vs. `http://api.pwnedhub.com:8080`
- Enforced by the browser.
  - Applies to services supporting browser-based web applications.
- Restricts the impact of XSS.
  - But not CSRF!

Same-Origin Policy Applied



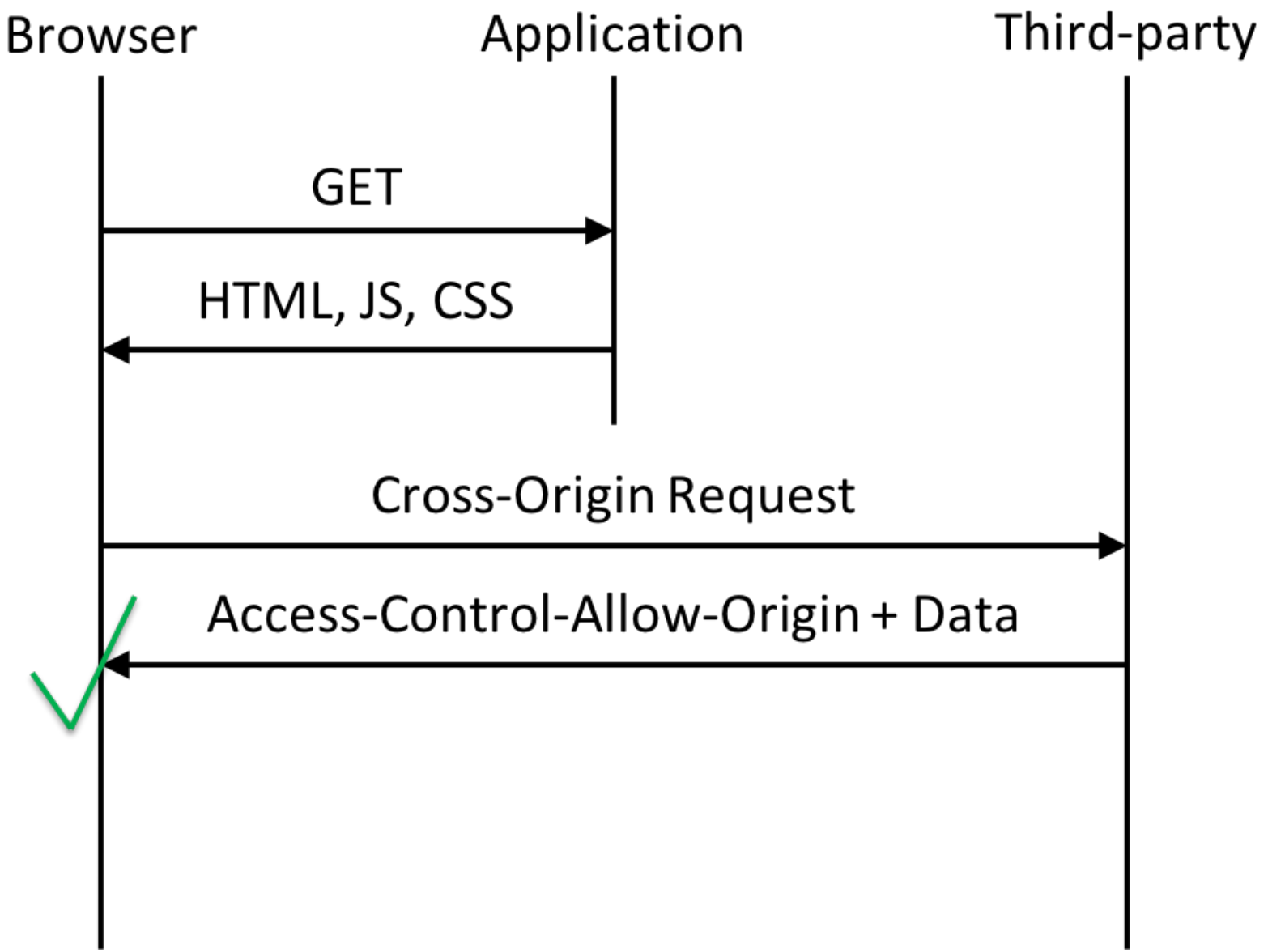
Wait a second...

Apps do this all the time!

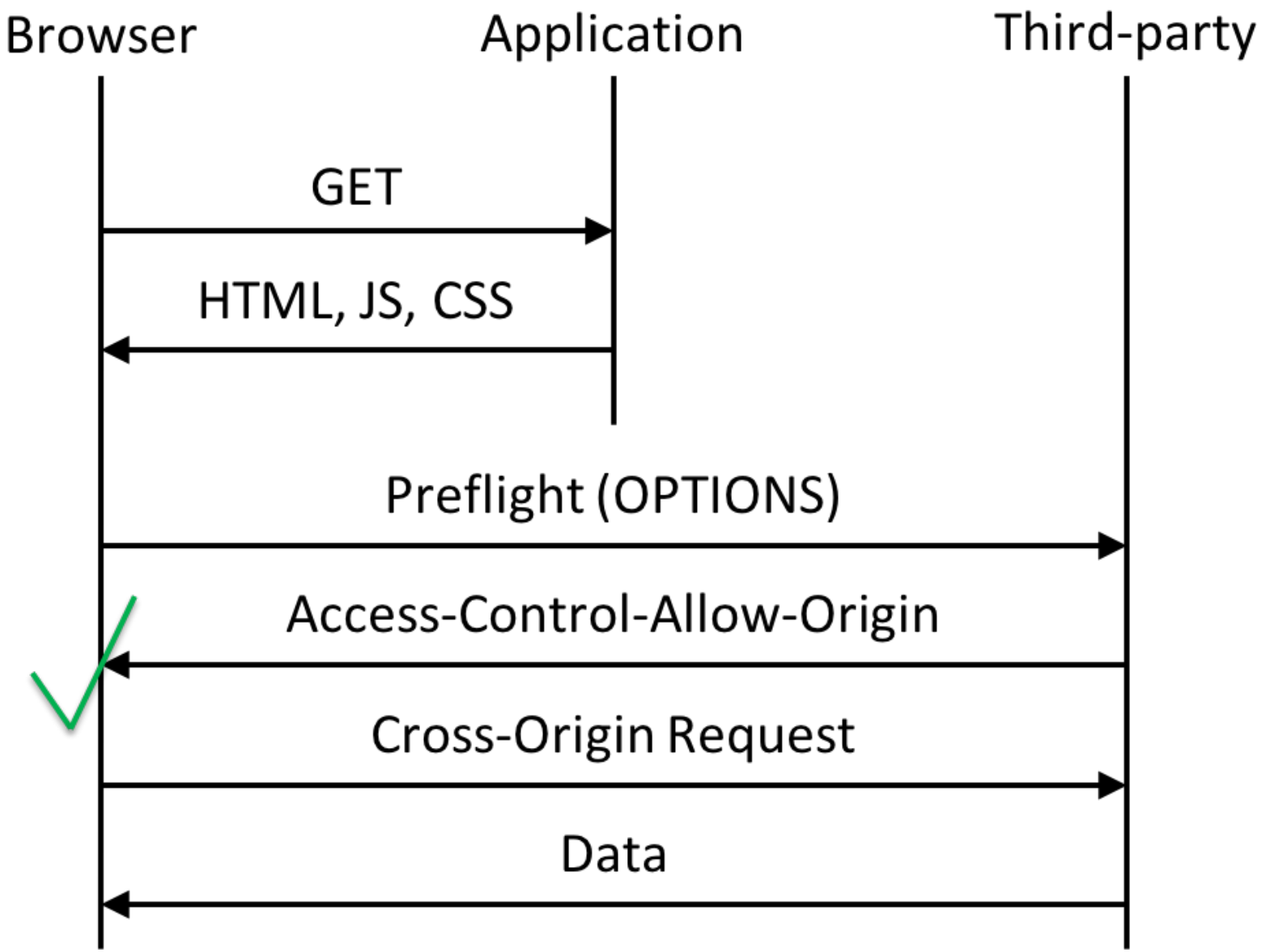
Cross-Origin Resource Sharing (CORS)

- Server-controlled mechanism to bypass Same-Origin Policy.
  - SOP is the default cross-origin behavior.
- Removes the need for server-side mashups.
  - Allows client-side web apps to call cross-origin APIs.
- Two types of cross-origin requests:
  - Simple
  - Preflighted

Simple Cross-Origin Request



Preflighted Cross-Origin Request



CORS Concerns

- Problem!
  - SOP Roadblock
- CORS Solutions?
  - Wildcard Origin
  - Arbitrary Reflected Origin

What we want to do...

```
POST /checkin HTTP/1.1
Host: derbycon.com
User-Agent: American Online
Content-Type: application/json
Origin: hackertracker.info
Content-Length: 349
Connection: close

{"Speaker":"Kevin Cody"}
```

What the browser does...

```
OPTIONS /checkin HTTP/1.1
Host: derbycon.com
User-Agent: American Online
Content-Type: */*
Origin: hackertracker.info
Connection: close
```

What the server says...

```
HTTP/1.1 200 OK
Allow: POST, OPTIONS
Content-Type: text/html; charset=utf-8
Date: Fri, 30 Mar 2019 23:56:44 GMT
Content-Length: 0
Connection: Close
```

...and nothing works.

SOP Roadblock

- The default condition.
- Functionality is kind of important.
- Broken apps == no $$
- This is likely what causes insecure configurations.
- The developer makes some server changes...

What the browser does (no change)...

```
OPTIONS /checkin HTTP/1.1
Host: derbycon.com
User-Agent: American Online
Content-Type: */*
Origin: hackertracker.info
Connection: close
```

What the server says now...

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Allow: POST, OPTIONS
Content-Type: text/html; charset=utf-8
Date: Fri, 30 Mar 2019 23:56:44 GMT
Content-Length: 0
Connection: Close
```

...and it works, kinda.

- The resource requires authentication.
- The developer makes more server changes...

What the server says now...

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: True
Allow: POST, OPTIONS
Content-Type: text/html; charset=utf-8
Date: Fri, 30 Mar 2019 23:56:44 GMT
Content-Length: 0
Connection: Close
```

What happens?

NOTHING! ...again!

Wildcard Origin

- The new normal.
- Considered safe.
- Doesn't allow authenticated requests.
  - Unsafe requests that change state.
- The developer makes more server changes...

What the browser does (again)...

```
OPTIONS /checkin HTTP/1.1
Host: derbycon.com
User-Agent: American Online
Content-Type: */*
Origin: hackertracker.info
Connection: close
```

What the server says now...

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: hackertracker.info
Access-Control-Allow-Credentials: True
Allow: POST, OPTIONS
Content-Type: text/html; charset=utf-8
Date: Fri, 30 Mar 2019 23:56:44 GMT
Content-Length: 0
Connection: Close
```

...and it works, with authentication.

- Yuss! Functionality! Mission accomplished.
- The developer moves on to the next task.

But what if we try...

```
OPTIONS /checkin HTTP/1.1
Host: derbycon.com
User-Agent: American Online
Content-Type: */*
Origin: somerandomorigin.com
Connection: close
```

...and the server says...

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: somerandomorigin.com
Access-Control-Allow-Credentials: True
Allow: POST, OPTIONS
Content-Type: text/html; charset=utf-8
Date: Fri, 30 Mar 2019 23:56:44 GMT
Content-Length: 0
Connection: Close
```

Arbitrary Reflected Origin

- Super bad
  - Cookie-based credentials can be included!
- Actually quite common
  - rs/cors - Go
  - cyu/rack-cors - Ruby
  - *flask-cors - Python
  - *go-chi/cors - Go
- These are only the libraries we've stumbled across.
- Additional research by Evan Johnson.
  - https://ejj.io/misconfigured-cors
  - https://gist.github.com/ejcx/74c7d7727767ca3aafa6

What's the difference between CSRF and a CORS Arbitrary Reflected Origin?

The attacker can access the response!

Mitigation

- Arbitrary Reflected Origin Attacks
  - Origin whitelisting
  - *Avoid cookie-based authentication.
- Wildcard Origin Issues
  - There are none! ... Or are there?
  - Wildcards (*) look dangerous.
  - There must be edge cases or methods to abuse.

We can send unauthenticated requests from any user to a vulnerable service via XHR/AJAX *and* access the results!

What to do?

- Worked up a small POC.
- Thought I was crazy.
- Called up Tim :)

CORS Exploitation Framework (CEF)

- Always looking for an excuse to write code.
- Proof-of-concept
- Exploits permissive CORS policy.
- Uses a distributed attack infrastructure.
  - Because... permissive policy
- Deployed via XSS or embedded JS.
- A BeEF module just waiting to happen.
  - Internal network mapping attacks, etc.

CEF Demo

http://cef.derbycon.rip

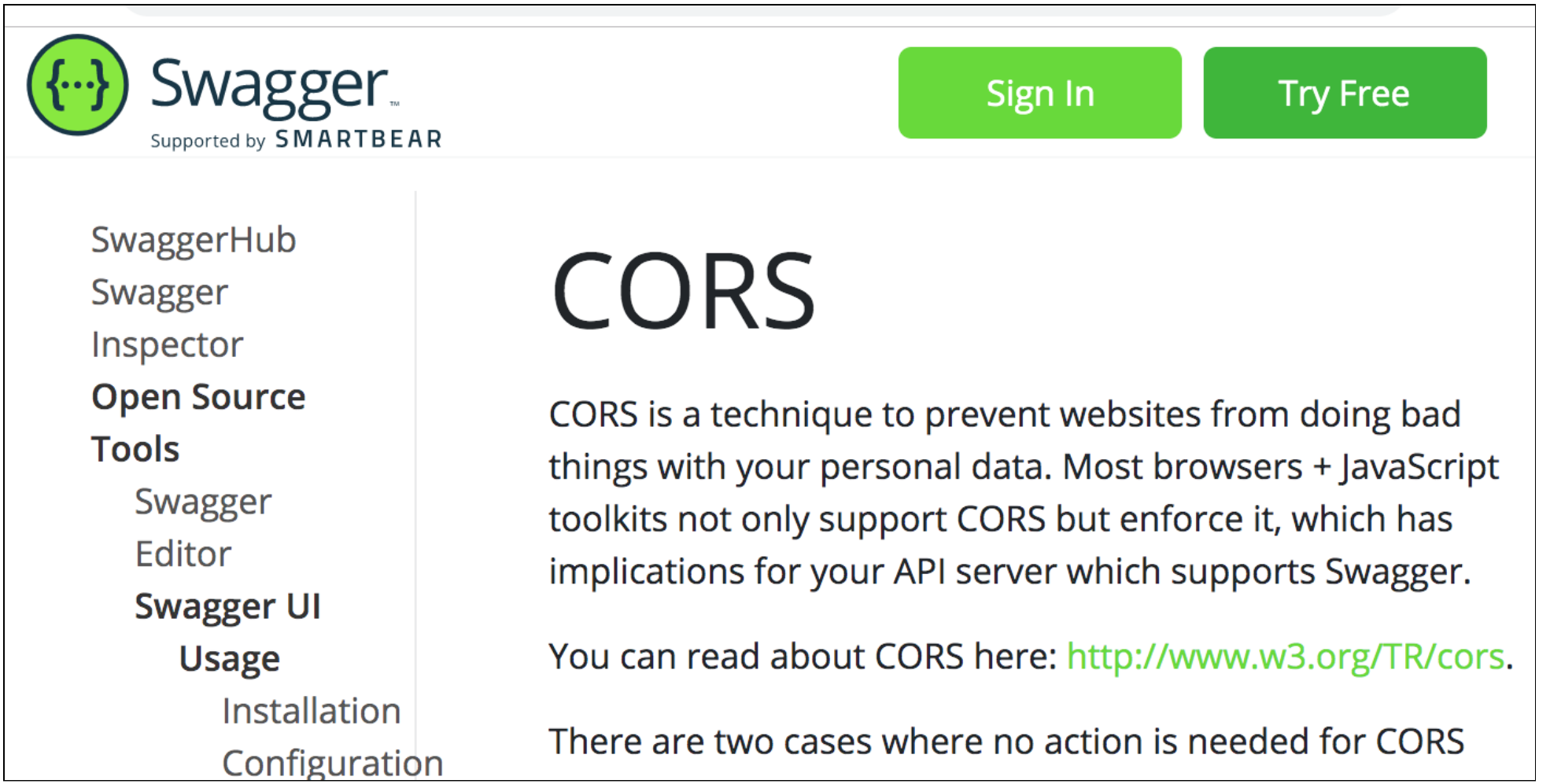
More Mitigation

- Synchronized Token Pattern
  - Just like CSRF.
- But, on the authentication call?
  - Not likely.
  - If so, rarely enforced.

Final Thoughts

- Bad recommendations everywhere.
  - Vulnerable CORS implemented ONLY to support self-documenting APIs.
  - enable-cors.org literally recommends wide open implementations.
- Default state of solutions.
  - Most extensions default to wild card.
  - Other extensions default to arbitrary origin.
  - NONE default to a completely secure implementation.
- Wild card without credentials CAN still be dangerous.
  - You just saw why.
- CORS is widely misunderstood...

Use swagger they said. It will be better they said.



The Goods

- https://github.com/lanmaster53/cef

Other Resources

- https://www.lanmaster53.com/
- https://kcody.co/