DASHLANE

Cross-Origin Resource Sharing For The Web (Extension)

Tim

Senior Front-End Engineer @ Dashlane

twitter.com/tpillard

timtech.blog

Table of contents

Cross-Origin Resource Sharing ?

Cross-Origin ~~Resource Sharing~~

~~Cross~~-Origin

Origin

“the origin of a URI serialized into a string”

“often used as the scope of authority or privilege by user agents”

“as an HTTP header field, indicates which origins are associated with an HTTP request”

— IETF RFC6454 (slightly adpated)

“The Origin request header indicates where a fetch originates from.”

— Origin HTTP Header, MDN

Origin

- Originally defined in *IETF RFC6454: The Web Origin Concept*
- `Origin` HTTP Header definition then supplanted in WHATWG Fetch Standard

Origin

- `http://localhost:3000/foo`
- `https://app.company.test/login`
- `https://www.company.test/create-account`
- `https://api.company.test/authentication`
- `chrome-extension://fdjamakpfbbddfjaooikfcpdsjohcfmg/bar`

Origin

- `http://localhost:3000/foo`
- `https://app.company.test/login`
- `https://www.company.test/create-account`
- `https://api.company.test/authentication`
- `chrome-extension://fdjamakpfbbddfjaooikfcpdsjohcfmg/bar`

Origin

URL

- `http://localhost:3000/foo`
- `https://api.company.test/authentication`
- `chrome-extension://fdjamakpfbbddfjaooikfcpdsjohcfmg/bar`

Scheme / Protocol	Host	Port
`http:`	`localhost`	**3000**
`https:`	`api.company.test`	*None specified*
`chrome-extension:`	`fdjamakp...ohcfmg`	*None specified*

Cross-Origin Resource Sharing !?

Cross-Origin Resource Sharing?

Same-Origin

Same-Origin

Same-Origin Policy

The same-origin policy (SOP) is a **critical security mechanism** that restricts how a document or script loaded from one origin can **interact with a resource from another origin**.
It helps **isolate** potentially malicious documents, reducing possible **attack vectors**.

“There is no single same-origin policy.”

— W3C

“Although the same-origin policy **differs between APIs**, the **overarching intent** is to let users visit **untrusted web sites** without those web sites **interfering with the user's session** with **honest web sites.**”

— W3C

Same-Origin Policy

So, what **can** we **typically** do

- Everything you want from and to the same origin. ✓
- Cross-Origin Embeds ⓘ
  - `<script src=”https://app.dashlane.com/carbon.js”></script>`
- Cross-Origin Writes ⚠
  - `<form method=”POST” action=”https://api.dashlane.com/” />`
- Cross-Origin Reads ✗
  - `fetch(‘https://api.dashlane.com/’).then((response) => ...)`

“In other words, Same-Origin policy **typically** allows (some) Cross-Origin Embeds & Writes **by default for backwards compatibility**, but restricts Cross-Origin Reads.”

— Tim, after reading some stuff on the Internet

“Therefore, you need to **explicitly configure & check anything & everything** that could be used in a cross-origin context, whether you want to allow, disallow, or restrict it.”

— Tim, after enough bad experiences with Cross-Origin things

But Why?

But Why?

- Same-origin policy:
  - is a “critical security mechanism”
  - it “reduces possible attack vectors”

But Why?

- Same-origin policy:
  - is a “critical security mechanism”
  - it “reduces possible attack vectors”

- Ok. But why is it “critical” ? Which “possible attack vectors” ? What attacks?

But Why?

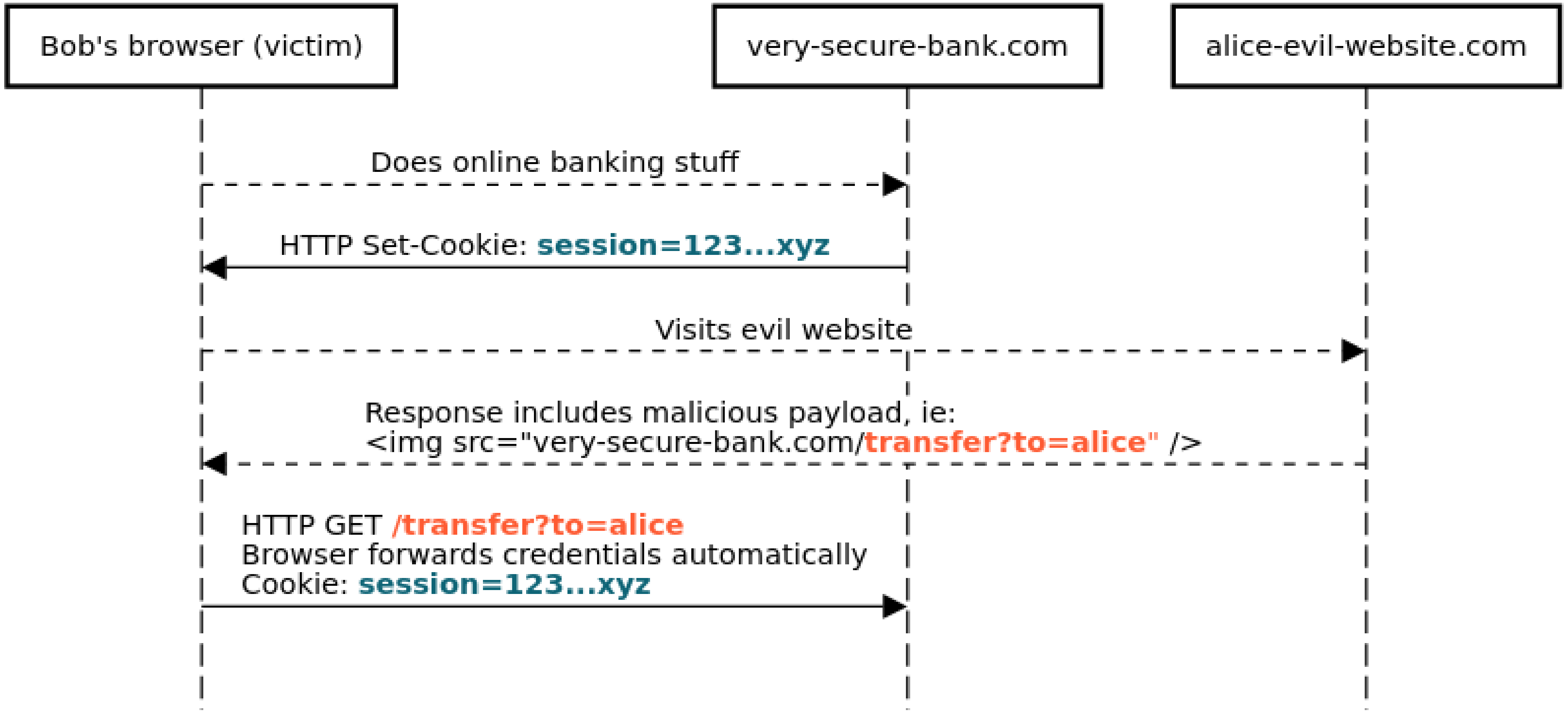
- Same-origin policy:
  - is a “critical security mechanism”
  - it “reduces possible attack vectors”

- Ok. But why is it “critical” ? Which “possible attack vectors” ? What attacks?

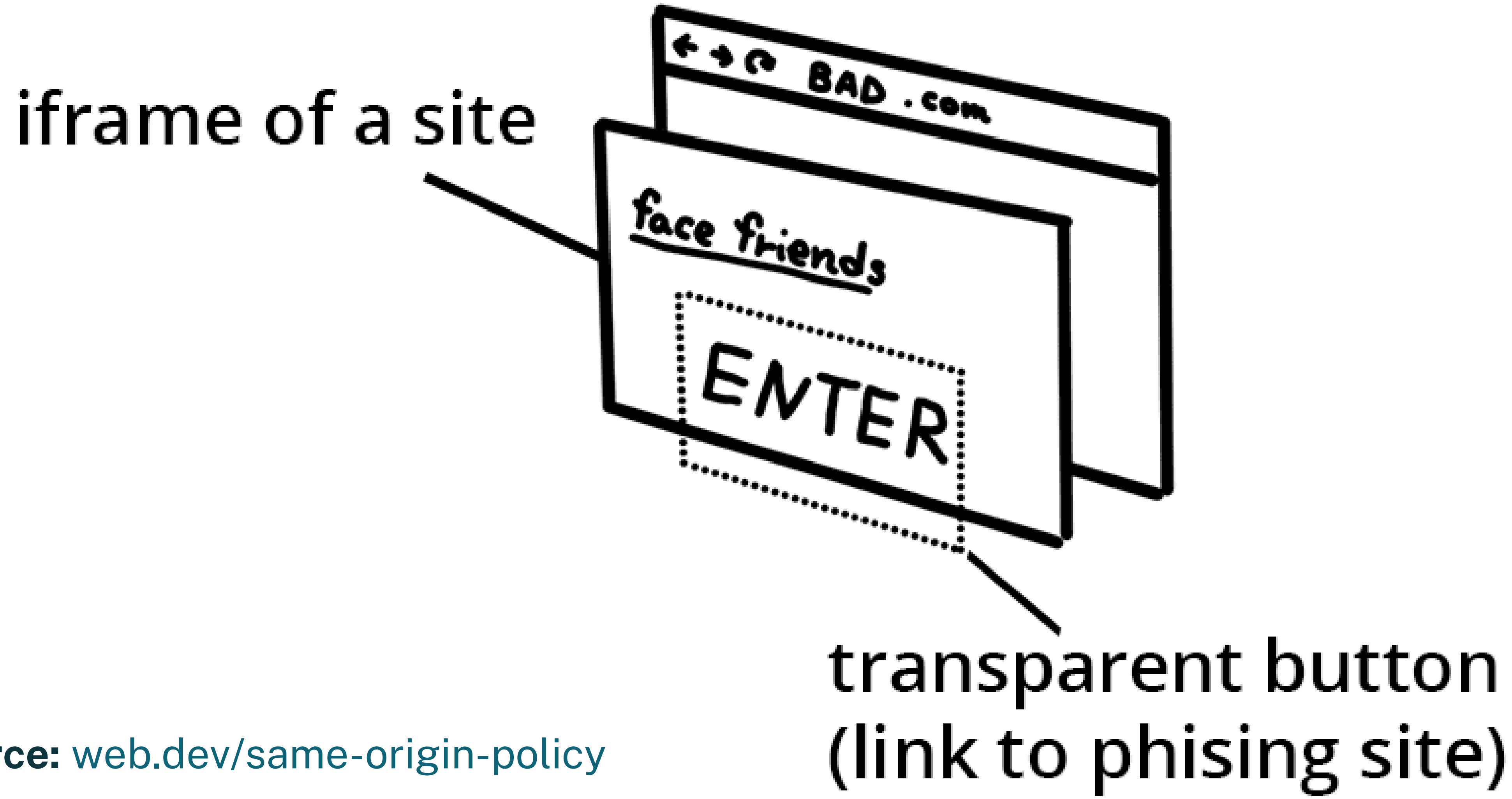
- There are mainly two contenders:
  - Clickjacking / Cross-Origin Framing
  - Cross Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF)


Bob's browser (victim)
very-secure-bank.com
alice-evil-website.com
Does online banking stuff
HTTP Set-Cookie: session=123...xyz
Visits evil website
Response includes malicious payload, ie:
<img src="very-secure-bank.com/transfer?to=alice" />
HTTP GET /transfer?to=alice
Browser forwards credentials automatically
Cookie: session=123...xyz

Clickjacking / Cross-Origin Framing



Source: web.dev/same-origin-policy

Cross-Origin Resource Sharing.

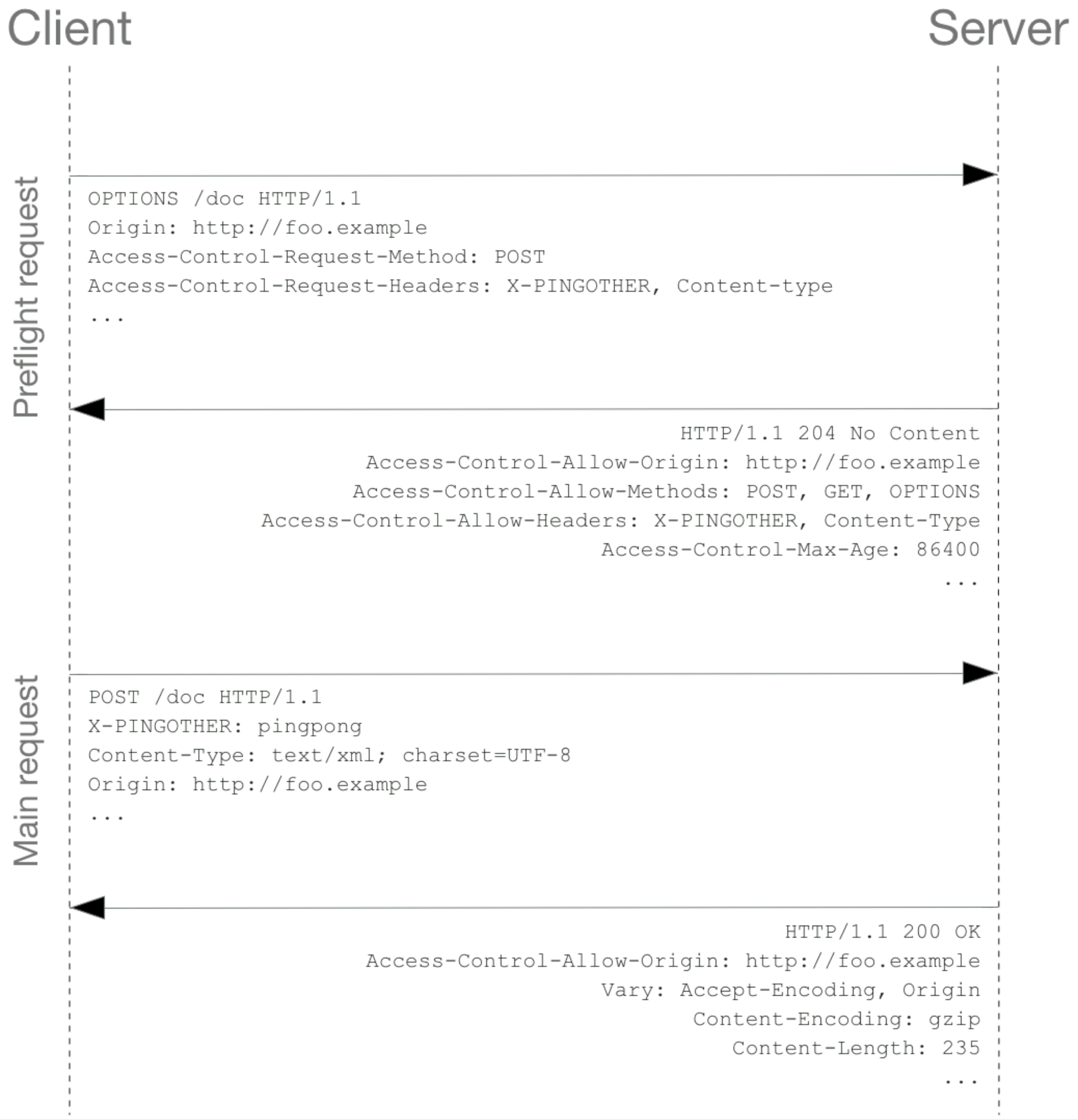
Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate **any other origins than its own** from which **a browser should permit loading of resources**.

OPTIONS CORS Preflight Requests

Unlike “simple requests”, for "preflighted" requests the browser **first sends an HTTP request using the OPTIONS method** to the resource on the other origin, in order to determine **if the actual request is safe to send**.

OPTIONS CORS Preflight Request

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
// Setting a custom header
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
// Setting a “non-simple request compliant” / “unsafe” Content-type header value
xhr.setRequestHeader('Content-Type', 'application/xml');
...
xhr.send('<person><name>Arun</name></person>');
```



```
OPTIONS /doc HTTP/1.1
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-type
...
```

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
...
```

```
POST /doc HTTP/1.1
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Origin: http://foo.example
...
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://foo.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
...
```

Source: developer.mozilla.org/docs/Web/HTTP/CORS

CORS For The Web

A few tips & tricks to avoid problems

CORS For The Web

Generally prefer same-origin communication

- Avoid 3rd party / cross-origin if you don't have a very good reason
  - Host 3rd party assets yourself (fonts, scripts, images…)
  - Expose & selectively 3rd party / cross-origin HTTP APIs & endpoints
  - Be wary of 3rd party analytics, trackers, ad systems, service providers…

CORS For The Web

Same-Origin Policy: handle with care

- Double-check any “simple requests” or cross-origin operations that are normally permitted by Same-Origin Policy.
  - What is permitted today, mostly is for backwards compatibility.
  - Tomorrow, everything will be different, and your web product will break.

CORS For The Web

Actually, pretend Same-Origin Policy doesn’t allow anything

- Going further, you can pretend Same-Origin Policy doesn’t allow anything that is cross-origin, and always rely on CORS & necessary security mechanisms instead.

CORS For The Web

The ~~cake~~ spec is a lie

- The spec is the first source of truth.
- Browsers work very hard to keep users secure, but they:
  - sometimes only partially implement, divert from, or go beyond the spec.
  - don't always share the same vision.
  - have bugs & regressions all the time.
  - regularly run experiments on real users.
- Monitor & analyze your traffic.
- Test recklessly.
- Read the changelogs, documentations, bugtrackers.

CORS For The Web (Extension)

The tale of an Extension production incident.

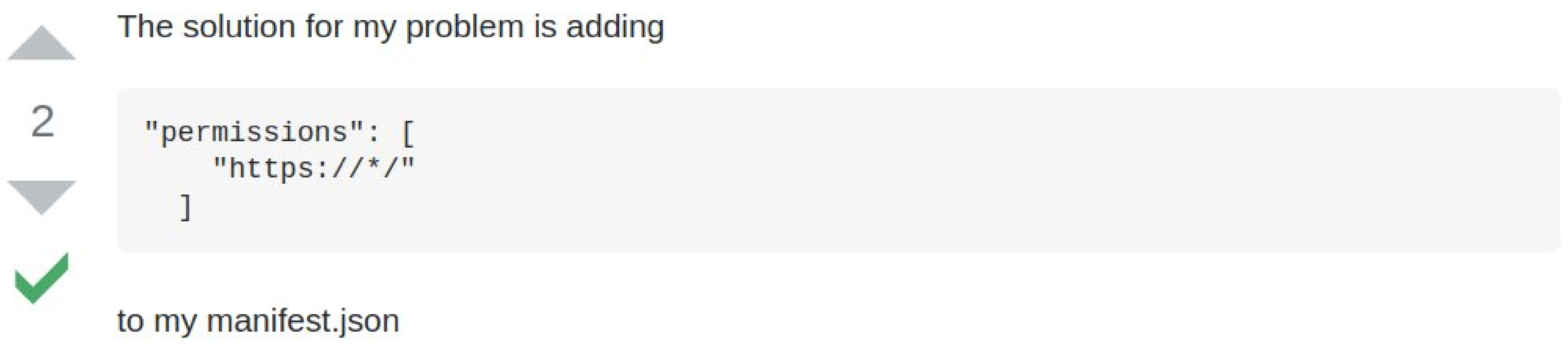
CORS For The Web (Extension)

The tale of an Extension production incident

Technical Storytelling (Twitter)

CORS For The Web (Extension)

The tale of an Extension production incident



2

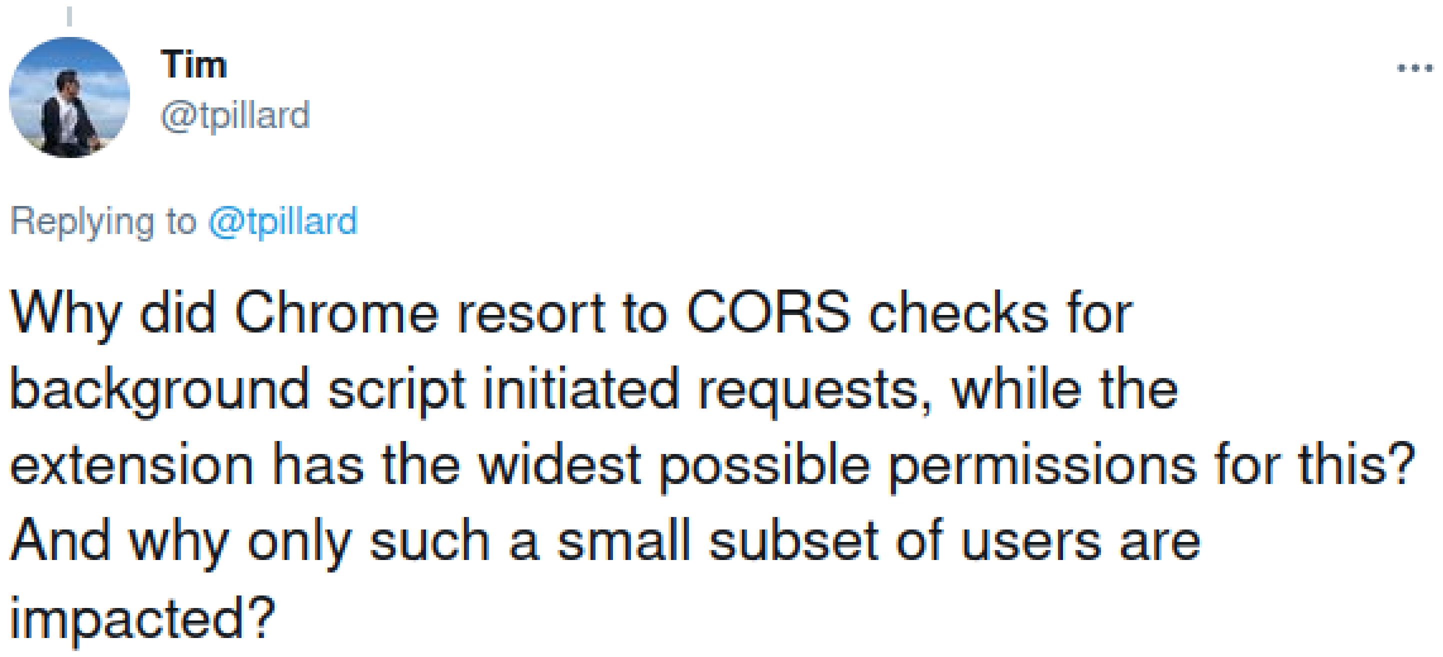
The solution for my problem is adding

```
"permissions": [
    "https://*/"
  ]
```

to my manifest.json

CORS For The Web (Extension)

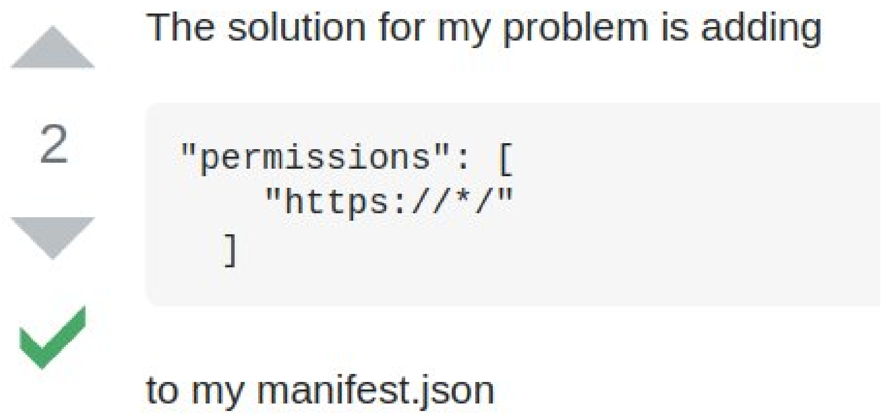
The tale of an Extension production incident



Tim
@tpillard

Replying to @tpillard

Why did Chrome resort to CORS checks for background script initiated requests, while the extension has the widest possible permissions for this? And why only such a small subset of users are impacted?



2

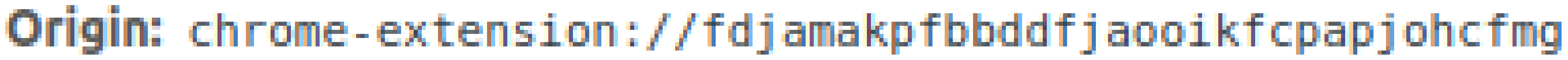
The solution for my problem is adding

```
"permissions": [
    "https://*/"
  ]
```

to my manifest.json

CORS For The Web (Extension)

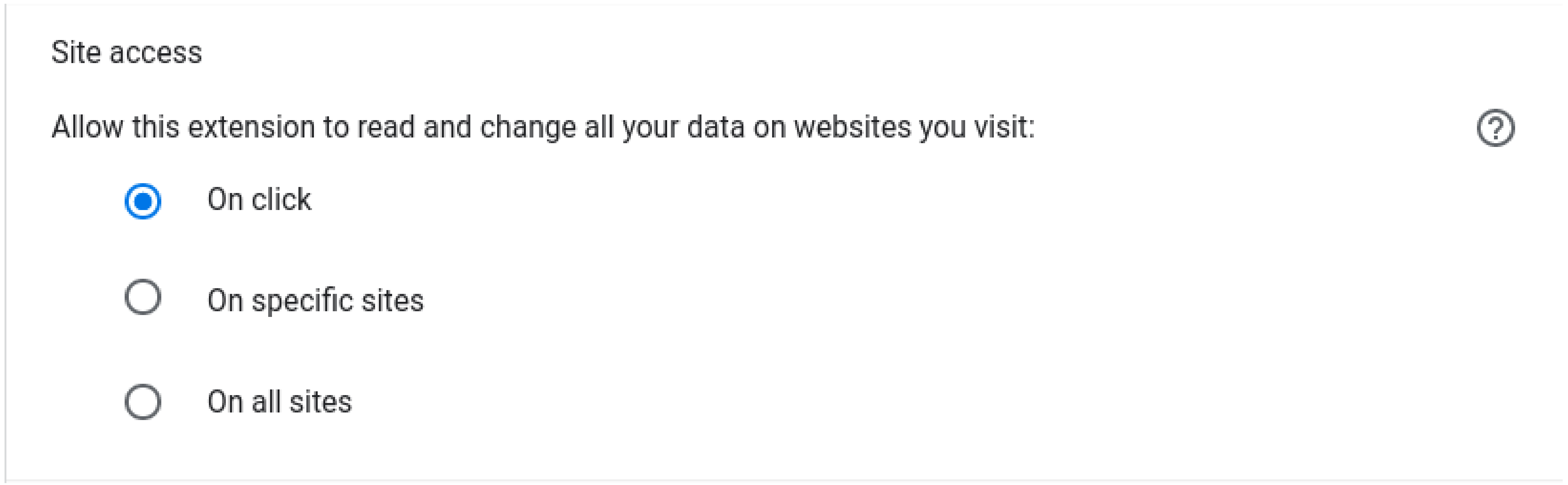
The tale of an Extension production incident



Origin: `chrome-extension://fdjamakpfbbddfjaooikfcpapjohcfmg`

CORS For The Web (Extension)

The tale of a Dashlane Extension production incident



Site access

Allow this extension to read and change all your data on websites you visit:

[✓ On click]

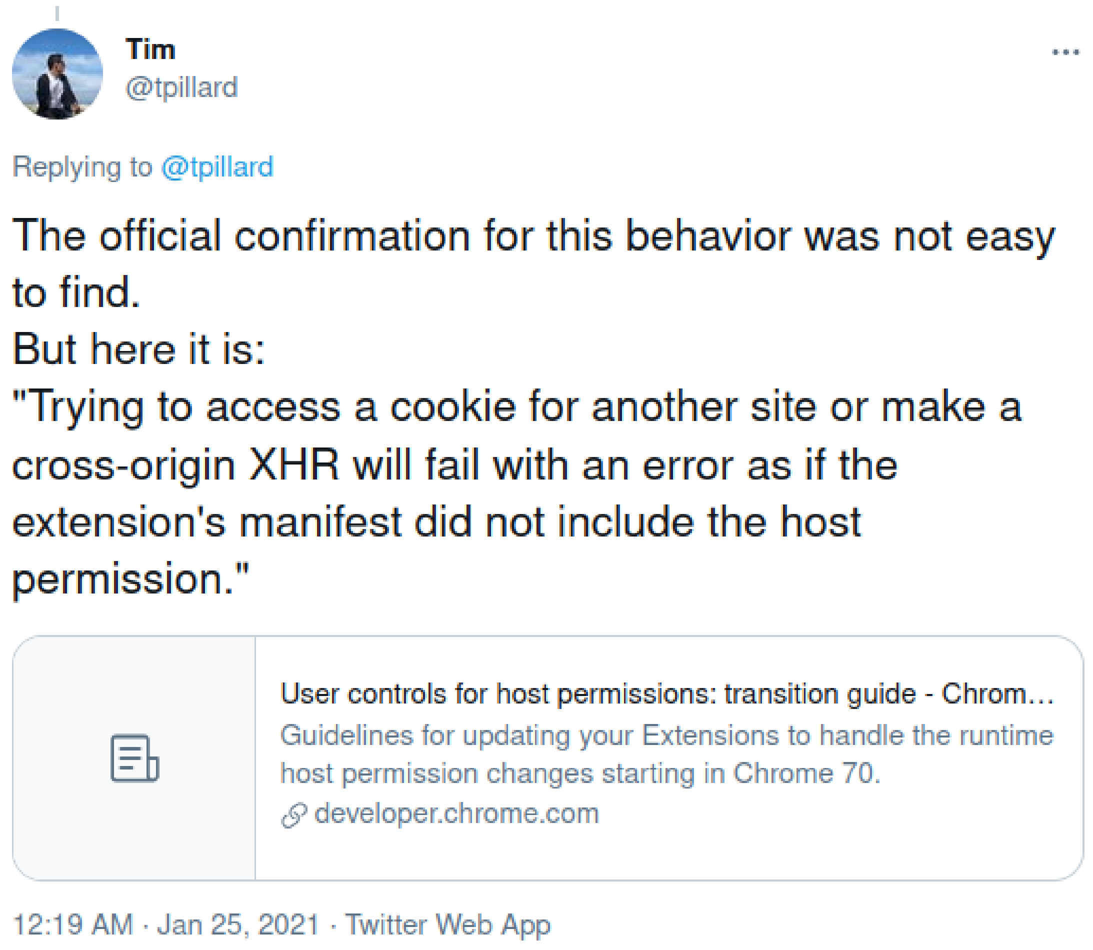
On specific sites

On all sites

Origin: `chrome-extension://fdjamakpfbbddfjaooikfcpapjohcfmg`

CORS For The Web (Extension)

The tale of an Extension production incident



Tim
@tpillard

Replying to @tpillard

The official confirmation for this behavior was not easy to find.
But here it is:
"Trying to access a cookie for another site or make a cross-origin XHR will fail with an error as if the extension's manifest did not include the host permission."

User controls for host permissions: transition guide - Chrom…
Guidelines for updating your Extensions to handle the runtime host permission changes starting in Chrome 70.
developer.chrome.com

12:19 AM · Jan 25, 2021 · Twitter Web App

Web Security in a Post-Spectre World

NOT
TODAY

Web Security in a Post-Spectre World

A girl has many names

- Cross-Origin Read Blocking (CORB)
- Opaque-Response Blocking (ORB)
- Out-Of-Process iFrames (OOPIF)
- Cross-Origin Resource Policy (CORP)
- Resource Isolation Policy
- Site Isolation
- Project Fission
- And more!

See: w3c.github.io/webappsec-post-spectre-webdev

DASHLANE