

The CSRF Resurrections!

Starring the Unholy Trinity: Service Worker of PWA,
SameSite of HTTP Cookie, and Fetch

Dongsung Kim

About Me



Surname

- Dongsung (Donny) Kim
- IT-Security Expert @ Security Office
- @kid1ng / kiding#0001
- <https://kidi.ng>



Security Office

- Based in Sweden and South Korea
- Part of the Truesec Group
- <https://security-office.co.kr>



The Web Evolution

- New day, new browser features!
- Constantly being proposed, developed, tested, shipped, and standardized
- Not just aesthetic improvements, but also powerful APIs
 - WebUSB, WebGPU, Web NFC, Serial API, Sockets API, Sensor API, ...
 - ...and so much more!



The screenshot shows the Chrome Platform Status website. The main page lists 2161 features, with a 'Proposed' section visible. A detailed view of a feature is shown in a modal window.

Browser Platform Status Tracker

2022-07-01

Chromium

Changed Response.json()

Status | No active development → Enabled by default

Changed Block external protocol in sandboxed iframe

Status | In developer trial (Behind a flag) → Enabled by default
Flag | True → False

Added Custom highlight API pointer events

Status | No active development

Added Features: Remove window.webkitStorageInfo

Status | No active development

2022-06-30

Chromium

CSS Anchor Positioning

Anchor positioning allows a position to be constrained by the edges of the page.

Motivation

When building interactive components, you may want to render in a "top-layer", and wish to position an "anchor element". How the top-layer is constrained by the edges of the page.

CSS break-after, break-before, break-inside (Generic breaking)

Add support for the modern fragment identifier.

CSS color-mix function

color-mix takes two color specifications and returns a color that is a linear interpolation of the two.

Motivation

This new function will give designers a way to create variants of a single base color to use in different contexts related to background-color in the future. background-color from the rgb color space.

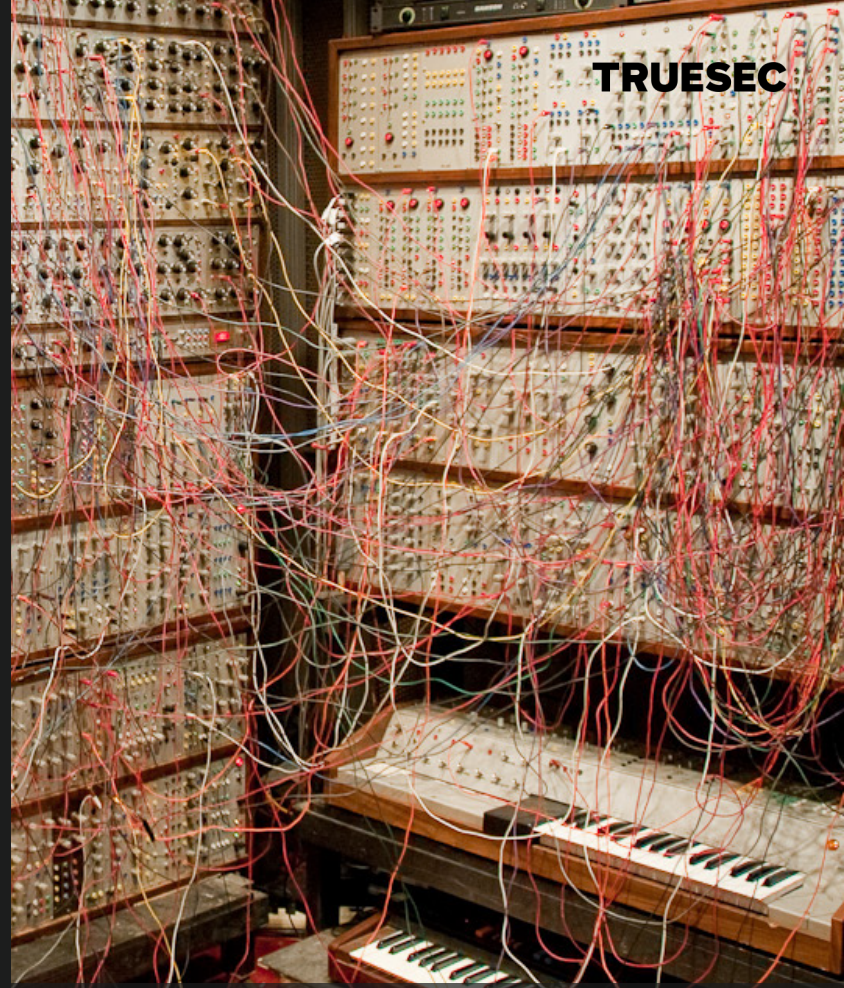
CSS default keyword

...and Security

- New features, new security problems!
- Intertwined and complex to maintain

Previous Works

- Franken et al. @ USENIX Security '18
 - PDF JS, AppCache, prerender, WebSocket, ...
 - Cookie policies can be bypassed
 - Fixed then some re-introduced
- Meltdown and Spectre
 - Rethinking of the web threat model
 - Powerful APIs disabled or reduced
 - performance, SharedArrayBuffer, ...



Chromium, "Post-Spectre Threat Model Re-Think"

Mozilla, "Mitigations landing for new class of timing attack"

Franken, et al., "Who Left Open the Cookie Jar? A Comprehensive Evaluation of (Third-Party) Cookie Policies." USENIX Security 18.

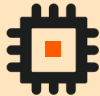
Joseph Paradiso, "The synth rig set up at the MIT Museum for its Jan-April 2012 installation"

Our Focus

HTTP Cookie
SameSite



Progressive Web App
Service Worker



Fetch

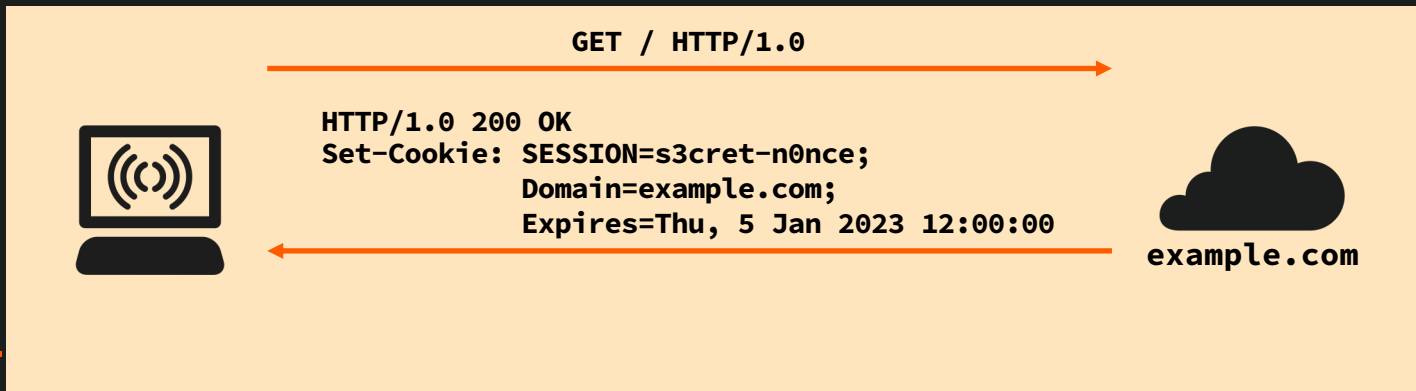


SameSite Cookie

The "Conclusive" Antidote to CSRF

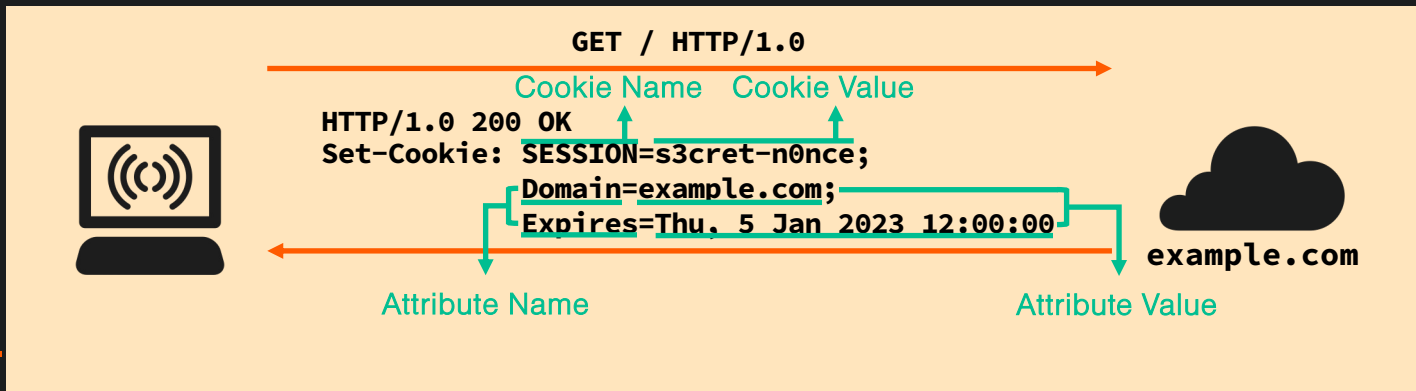
HTTP Cookie

- State management mechanism on top of the stateless HTTP
- Defines two HTTP headers: `Cookie` (Request), `Set-Cookie` (Response)
- Typically used to store server's "session identifier" in client storage
- User agents **always** send `Cookie` only with "matching" requests



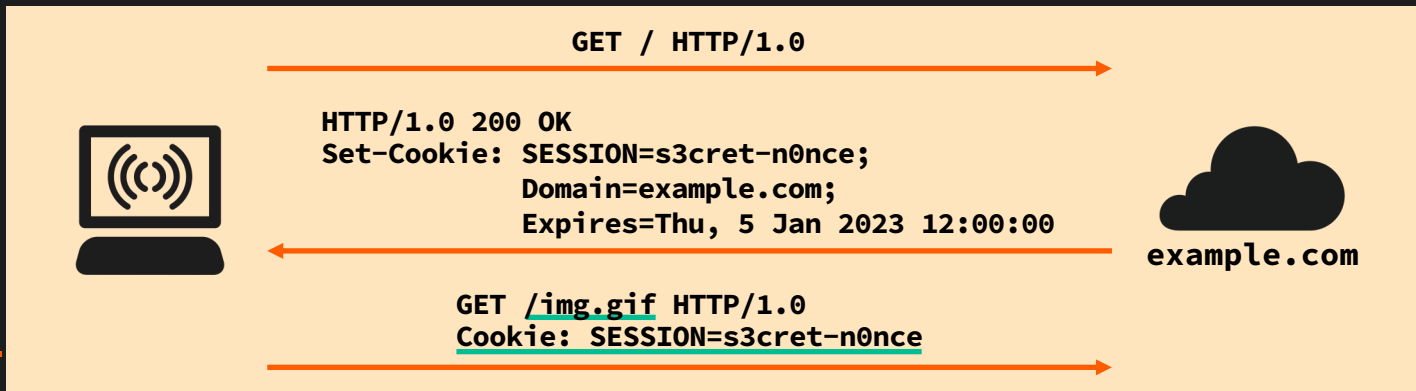
HTTP Cookie

- State management mechanism on top of the stateless HTTP
- Defines two HTTP headers: `Cookie` (Request), `Set-Cookie` (Response)
- Typically used to store server's "session identifier" in client storage
- User agents **always** send `Cookie` only with "matching" requests
Based on the attributes!



HTTP Cookie

- State management mechanism on top of the stateless HTTP
- Defines two HTTP headers: `Cookie` (Request), `Set-Cookie` (Response)
- Typically used to store server's "session identifier" in client storage
- User agents **always** send `Cookie` only with "matching" requests



HTTP Cookie

“For historical reasons, cookies contain a number of security and privacy infelicities.”

– *RFC6265bis*

- Third-party cookie
- Supercookie
- Eavesdropping
- Cross-site scripting
- Cross-site request forgery
- ...



"Cross-Site" Request

- `origin := (scheme, host, port)*`

URL

origin

`https://example.com:443/page`



`(https, example.com, 443)`

`https://sub.example.com:443/page`



`(https, sub.example.com, 443)`

`https://sub.example.com:8443/page`

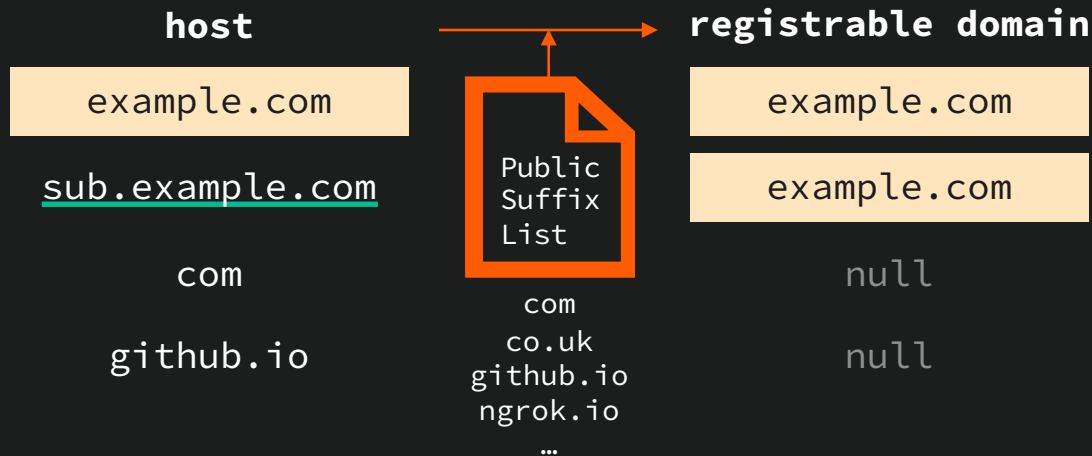


`(https, sub.example.com, 8443)`



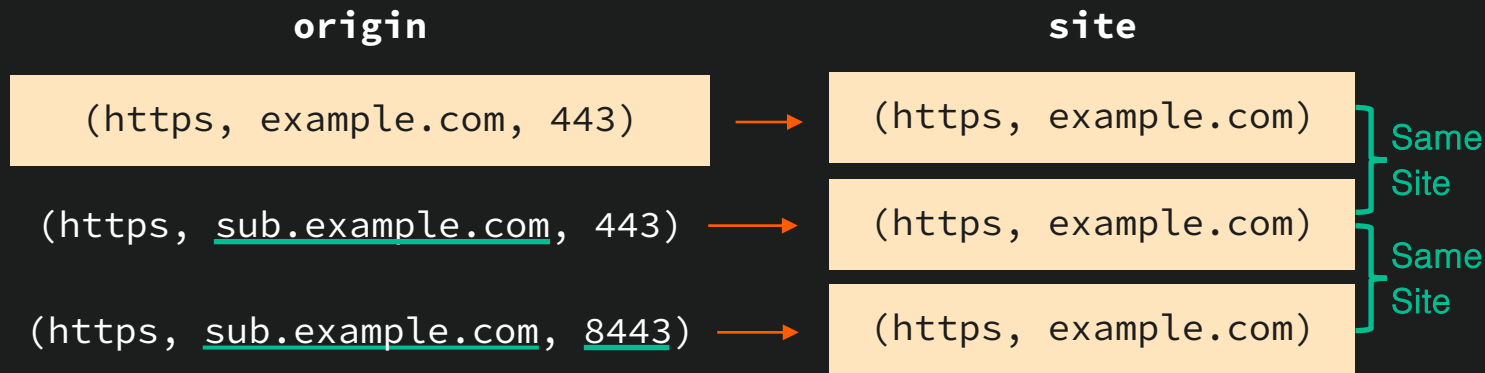
"Cross-Site" Request

- `origin := (scheme, host, port)`
- `registrable domain := Public suffix of host plus one additional label`



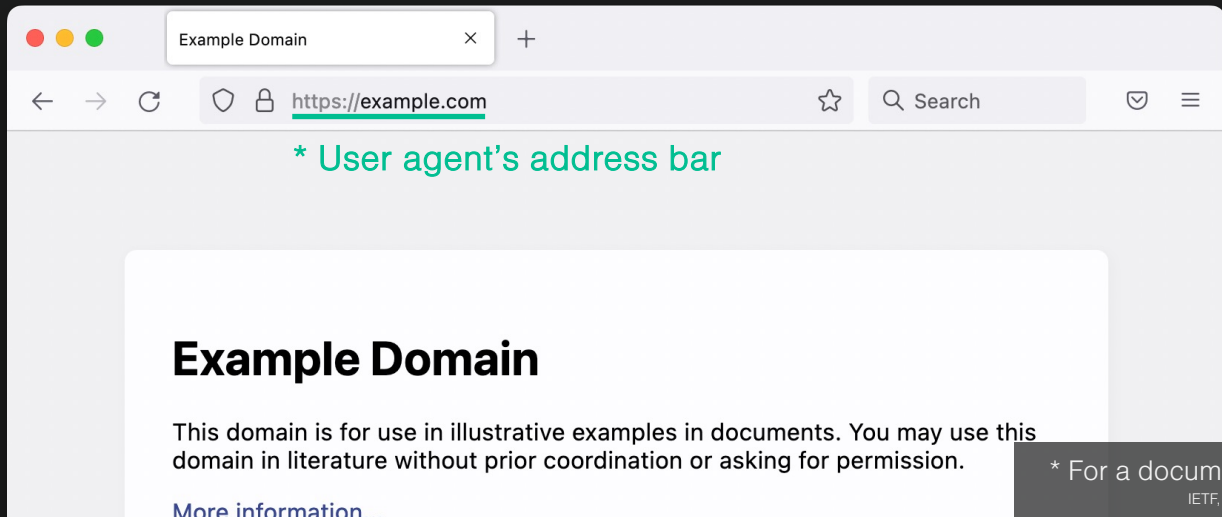
"Cross-Site" Request

- `origin := (scheme, host, port)`
- `registrable domain := Public suffix of host plus one additional label`
- `site := (scheme, registrable domain)`



"Cross-Site" Request

- `origin := (scheme, host, port)`
- `registrable domain :=` Public suffix of host plus one additional label
- `site := (scheme, registrable domain)`
- "site for cookies" := The document's top-level origin*



"Cross-Site" Request

- `origin := (scheme, host, port)`
- `registrable domain :=` Public suffix of host plus one additional label
- `site := (scheme, registrable domain)`
- `"site for cookies" :=` The document's top-level origin*
- `A request's client := {..., context, origin, ...}`



* For a document displayed in a top-level browsing context

"Cross-Site" Request

- `origin` := (scheme, host, port)
- `registrable domain` := Public suffix of host plus one additional label
- `site` := (scheme, registrable domain)
- “site for cookies” := The document’s top-level origin*
- A request’s `client` := {..., context, origin, ...}
- “Same-site” requests :=
 - The request’s URL’s origin is same-site with its `client`’s “site for cookies”
 - The request’s URL’s origin is same-site with every URL in its URL list (redirection chain)
 - The request is not triggered by user pressing a reload button
- “Cross-site” requests := Not “same-site”

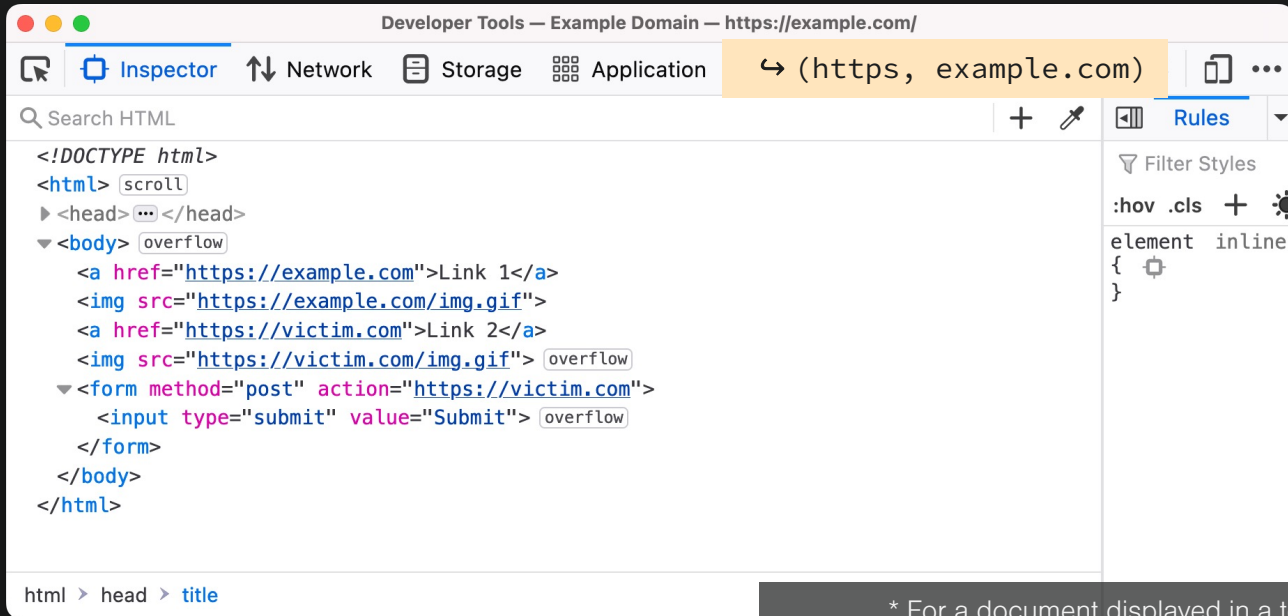
* For a document displayed in a top-level browsing context

"Cross-Site" Request

- "Same-site" requests :=

The request's URL's origin is same-site with its client's "site for cookies"

↔ (scheme, registrable domain) ↔ User agent's address bar*



* For a document displayed in a top-level browsing context

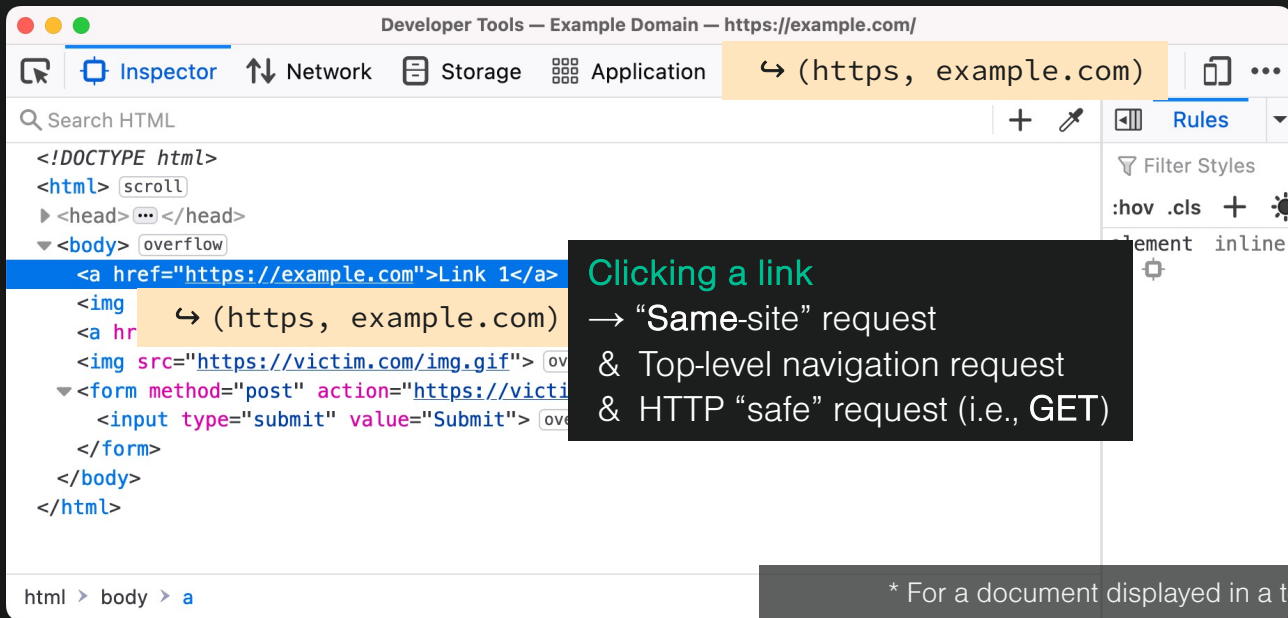
WHATWG, "Fetch Living Standard"

"Cross-Site" Request

- "Same-site" requests :=

The request's URL's origin is same-site with its client's "site for cookies"

↔ (scheme, registrable domain) ↔ User agent's address bar*

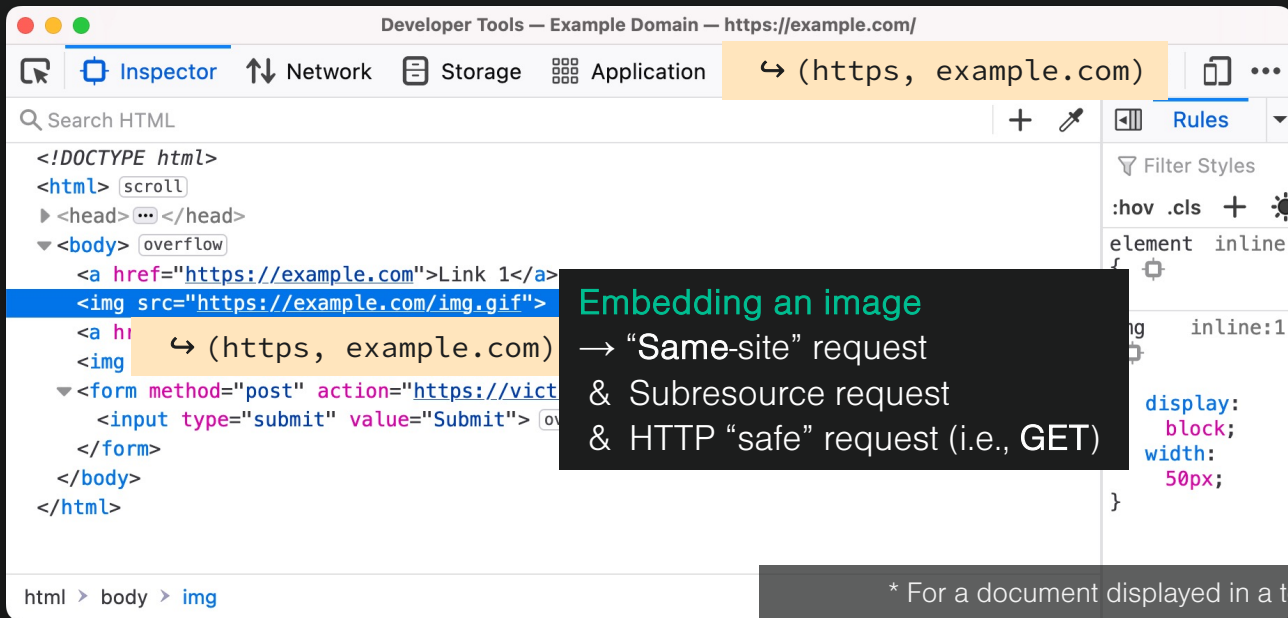


"Cross-Site" Request

- "Same-site" requests :=

The request's URL's origin is same-site with its client's "site for cookies"

↔ (scheme, registrable domain) ↔ User agent's address bar*



* For a document displayed in a top-level browsing context

"Cross-Site" Request

- "Same-site" requests :=

The request's URL's origin is same-site with its client's "site for cookies"

↔ (scheme, registrable domain) ↔ User agent's address bar*

Developer Tools — Example Domain — https://example.com/

Inspector | Network | Storage | Application | (https, example.com)

Search HTML

```
<!DOCTYPE html>
<html> scroll
  <head> ... </head>
  <body> overflow
    <a href="https://example.com">Link 1</a>
    
    <a href="https://victim.com">Link 2</a>
    <img
      <form
        <input type="submit" value="Submit">
      </form>
    </body>
  </html>
```

Clicking a link
→ "Cross-site" request
& Top-level navigation request
& HTTP "safe" request (i.e., GET)

html > body > a

* For a document displayed in a top-level browsing context

"Cross-Site" Request

- "Same-site" requests :=

The request's URL's origin is same-site with its client's "site for cookies"

↔ (scheme, registrable domain) ↔ User agent's address bar*

The screenshot shows a web browser's developer tools with the 'Inspector' tab active. The HTML document is from 'https://example.com/'. The DOM tree shows a form with a method of 'post' and an action of 'https://victim.com/'. A tooltip points to the action URL, indicating it is '(https, victim.com)'. A text box on the right explains that this is a 'Cross-site' request, which is also a top-level navigation request and an HTTP 'unsafe' request (specifically, a POST).

Developer Tools — Example Domain — https://example.com/

Inspector

Search HTML

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <a href="https://example.com">Link 1</a>
    
    <a href="https://victim.com">Link 2</a>
    
    <form method="post" action="https://victim.com">
      <input type="text">
    </form>
  </body>
</html>
```

Rules

Filter Styles

:hov .cls +

element inline { }

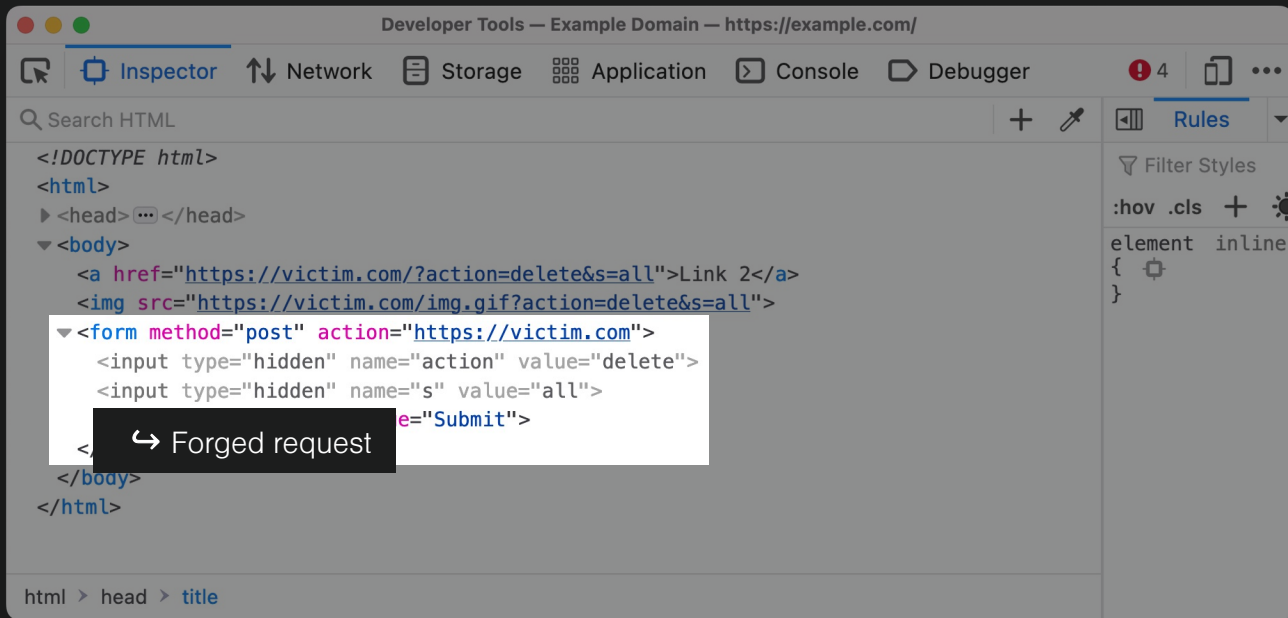
Submitting a form
→ "Cross-site" request
& Top-level navigation request
& HTTP "unsafe" request (i.e., POST)

html > body > form

* For a document displayed in a top-level browsing context

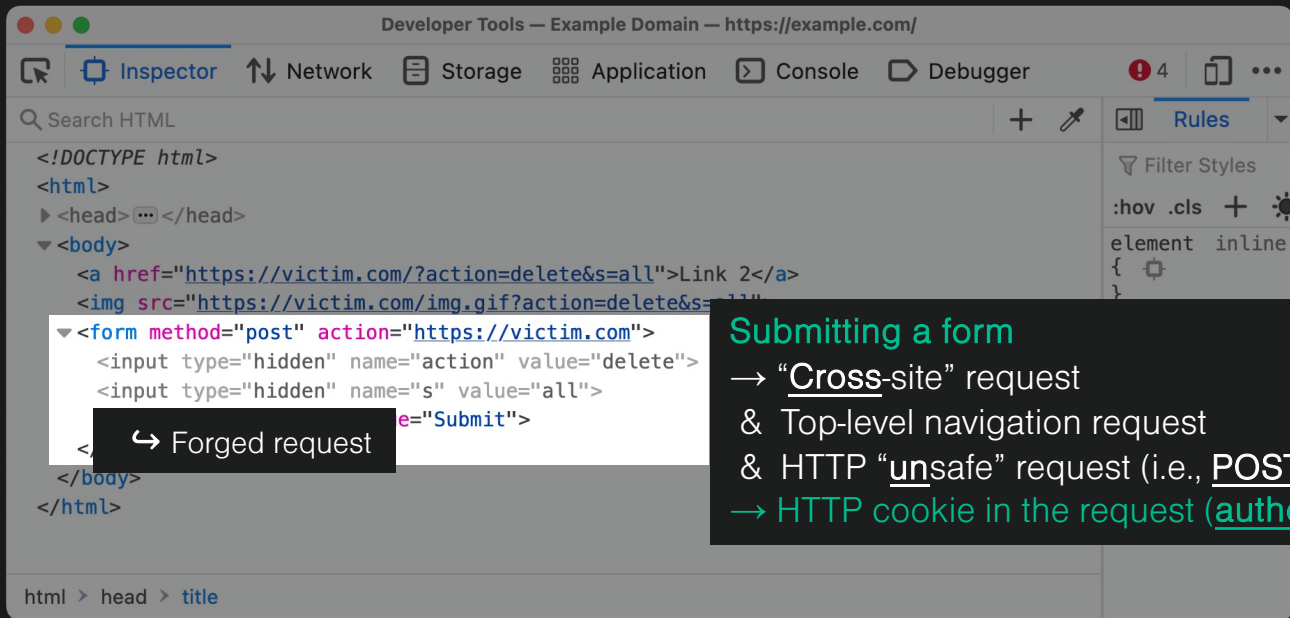
Cross-Site Request Forgery

- When an attacker forces a victim to execute forged “cross-site” requests
- User agents **always** send Cookie only with “matching” requests



Cross-Site Request Forgery

- When an attacker forces a victim to execute forged “cross-site” requests
- User agents **always** send Cookie only with “matching” requests



CSRF Mitigation

- Synchronizer token pattern
- Double submit cookie
- Cookie-to-header token
- Verifying referer/origin
- The SameSite attribute
- ...



The SameSite Attribute

- User agents send Cookie only but always with “matching” requests
An additional attribute to match!
- SameSite=Strict: Send the cookie only for “same-site” requests
❌ Clicking a link / ❌ Embedding an image / ❌ Submitting a form
- SameSite=Lax: “Same-site” + “cross-site” top-level navigation “safe” requests*
✅ Clicking a link / ❌ Embedding an image / ❌ Submitting a form
- SameSite=None: All requests†
✅ Clicking a link / ✅ Embedding an image / ✅ Submitting a form



HTTP/1.0 200 OK
Set-Cookie: SESSION=s3cret-n0nce;
Domain=example.com;
Expires=Thu, 5 Jan 2023 12:00:00;
SameSite=Strict



...and "Lax" by Default

- SameSite is widely supported, but the adoption has been slow
- Chrome 80+/86+ started treating cookies without SameSite as Lax by default
 - SameSite=Lax: "Same-site" + "cross-site" top-level navigation "safe" requests
- Cookie specification was also changed

CSRF

**Cross-Site Request
Forgery is dead!**

CSRF

CSRF is (really) dead

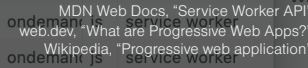
No. Same-site cookies alone are sufficient to protect against CSRF.



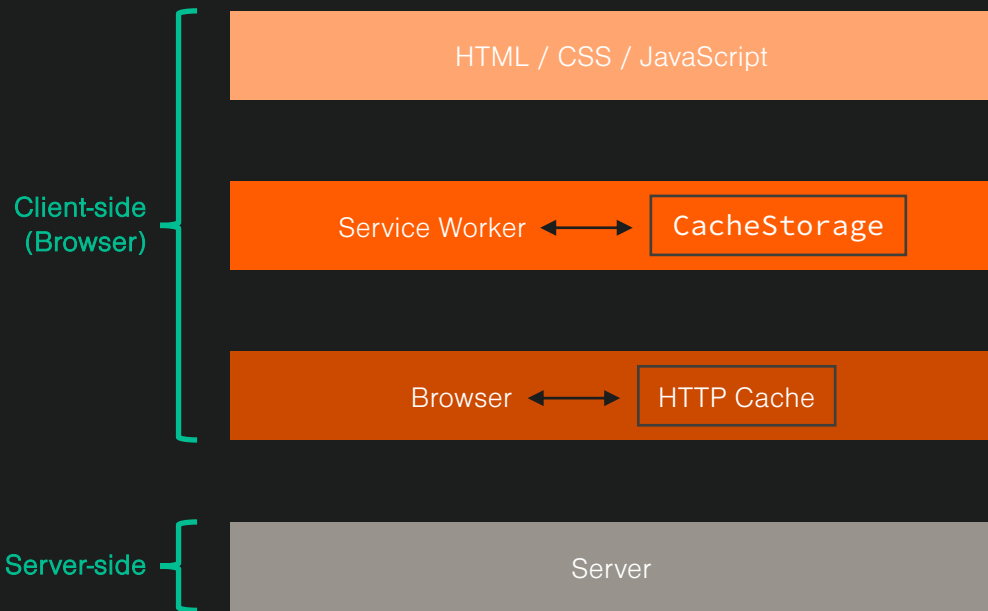
Service Worker

PWA's Network Layer

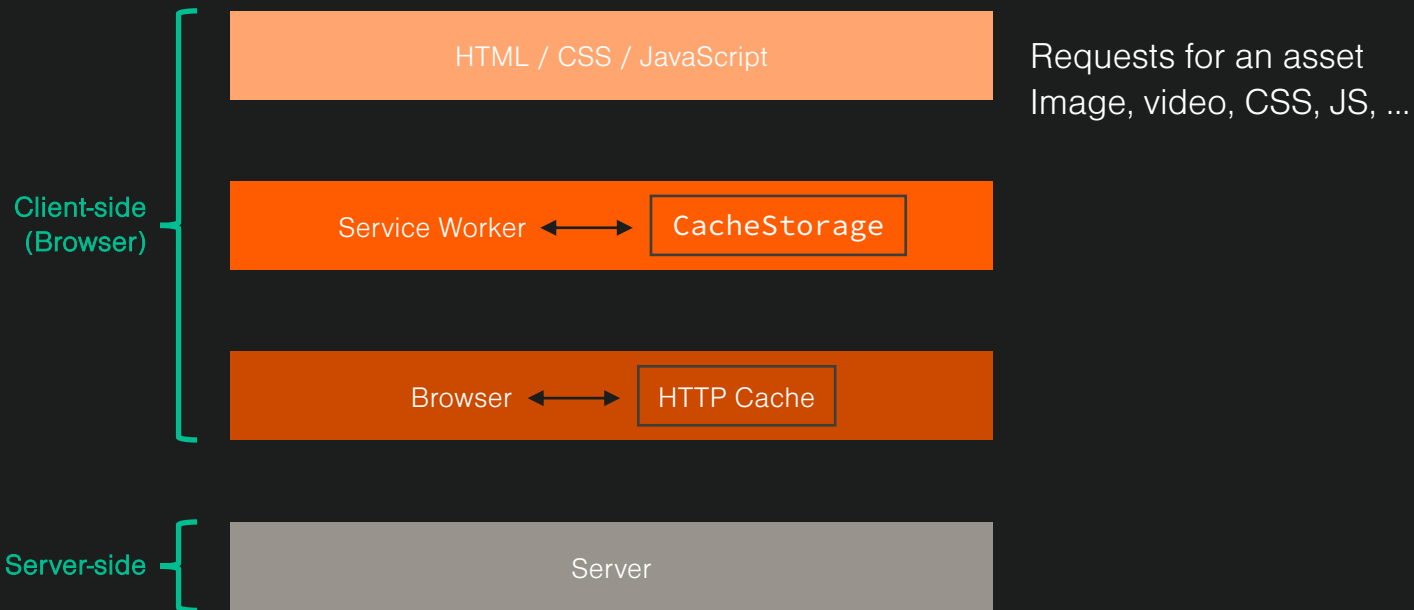
- Core technology of Progressive Web Apps
 - PWA: Native-like installable offline web app
- Runs in worker context; different thread
- Can act as a proxy server between web application, browser, and network
 - Intercepts network requests
 - Responds with custom responses
 - Together with `CacheStorage`
 - Enables offline fallback pages
- Mainly used for custom cache strategies



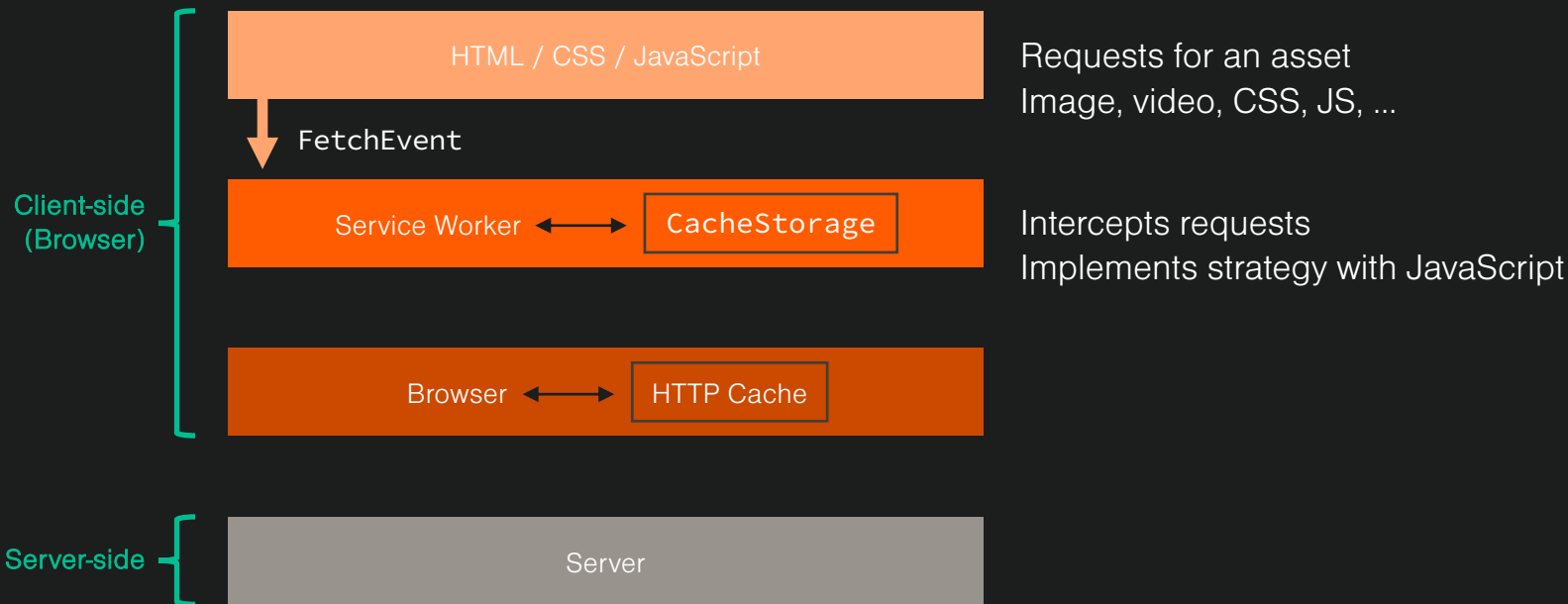
Service Worker Caching Flow



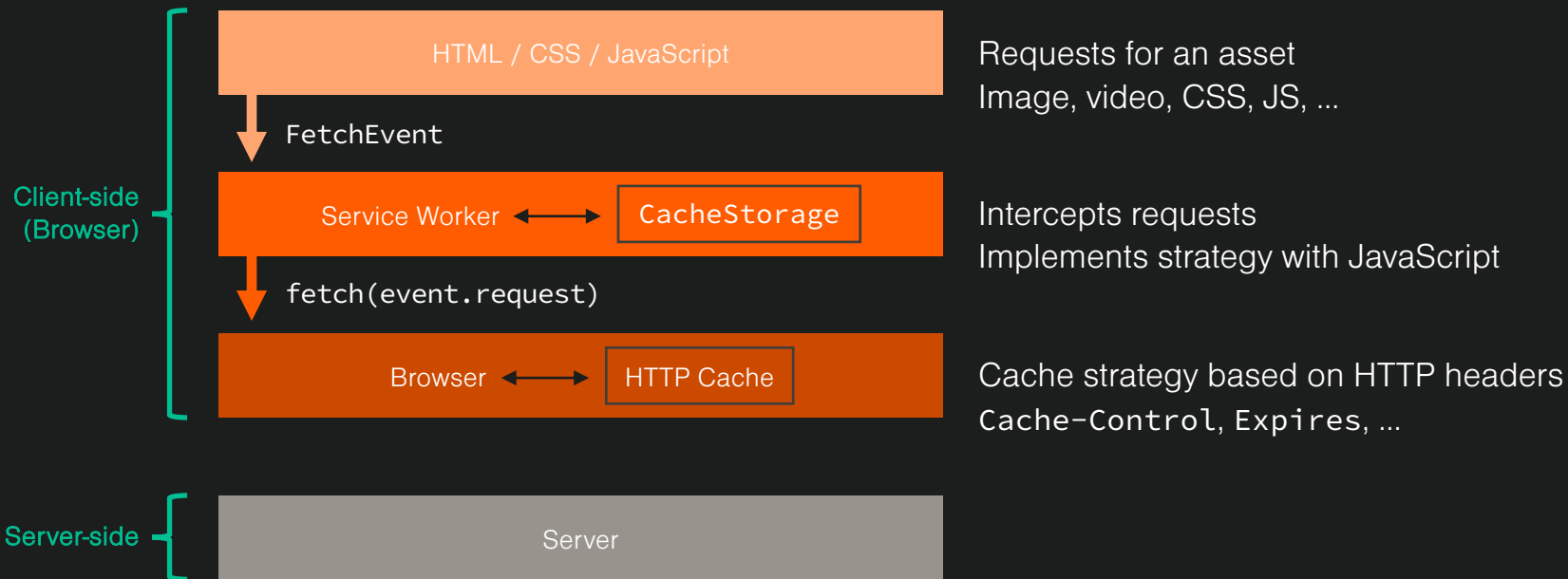
Service Worker Caching Flow



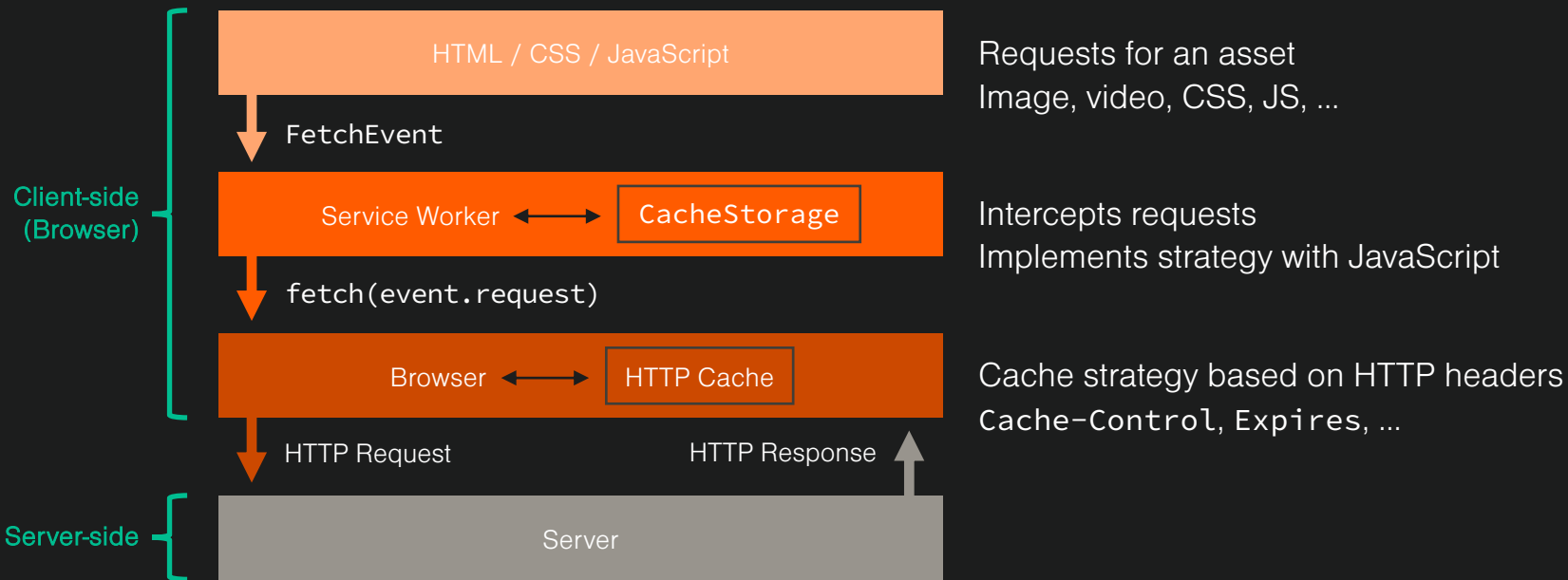
Service Worker Caching Flow



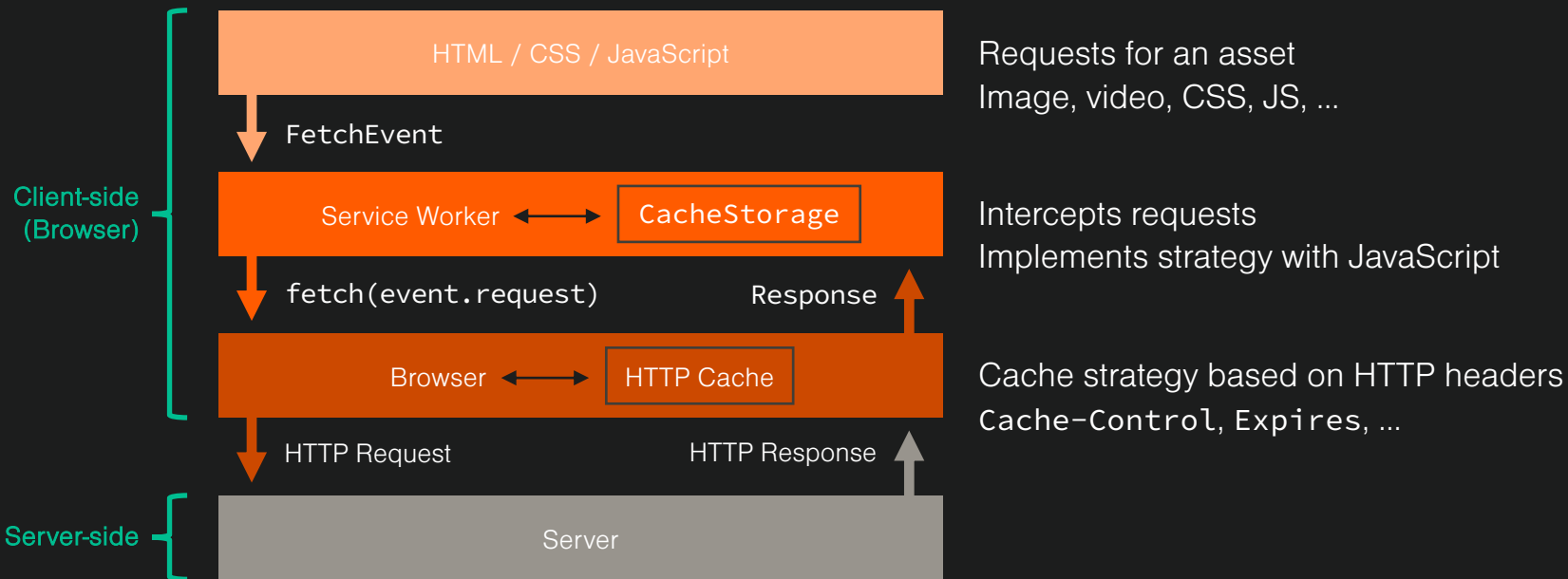
Service Worker Caching Flow



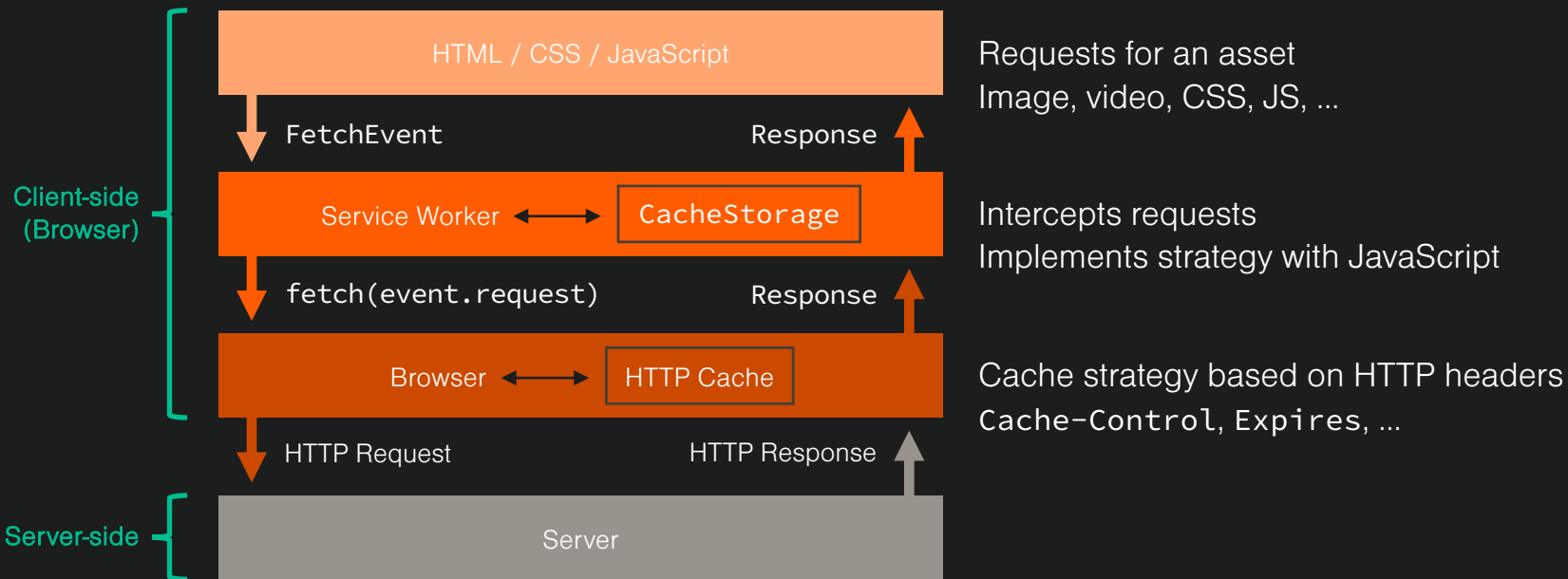
Service Worker Caching Flow



Service Worker Caching Flow



Service Worker Caching Flow



Service Worker Example

HTML
<html>

</html>

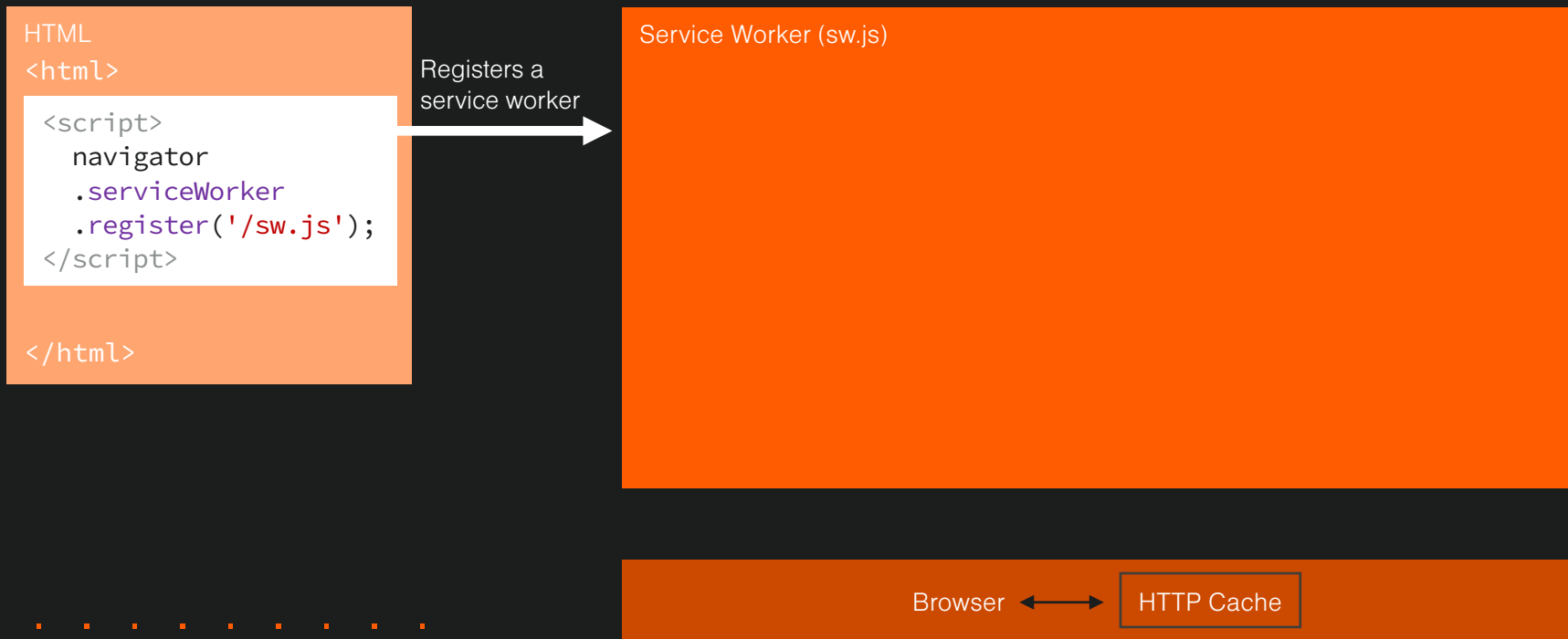
Service Worker (sw.js)

Browser



HTTP Cache

Service Worker Example



Service Worker Example

HTML

```
<html>
```

```
<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>
```

```
</html>
```

Registers a
service worker

Service Worker (sw.js)

```
// Intercepting all requests
self.addEventListener('fetch', (event) => {

});
```

Browser



HTTP Cache

<html>

```
<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>
```

Registers a service worker

Triggers FetchEvent

Service Worker (sw.js)

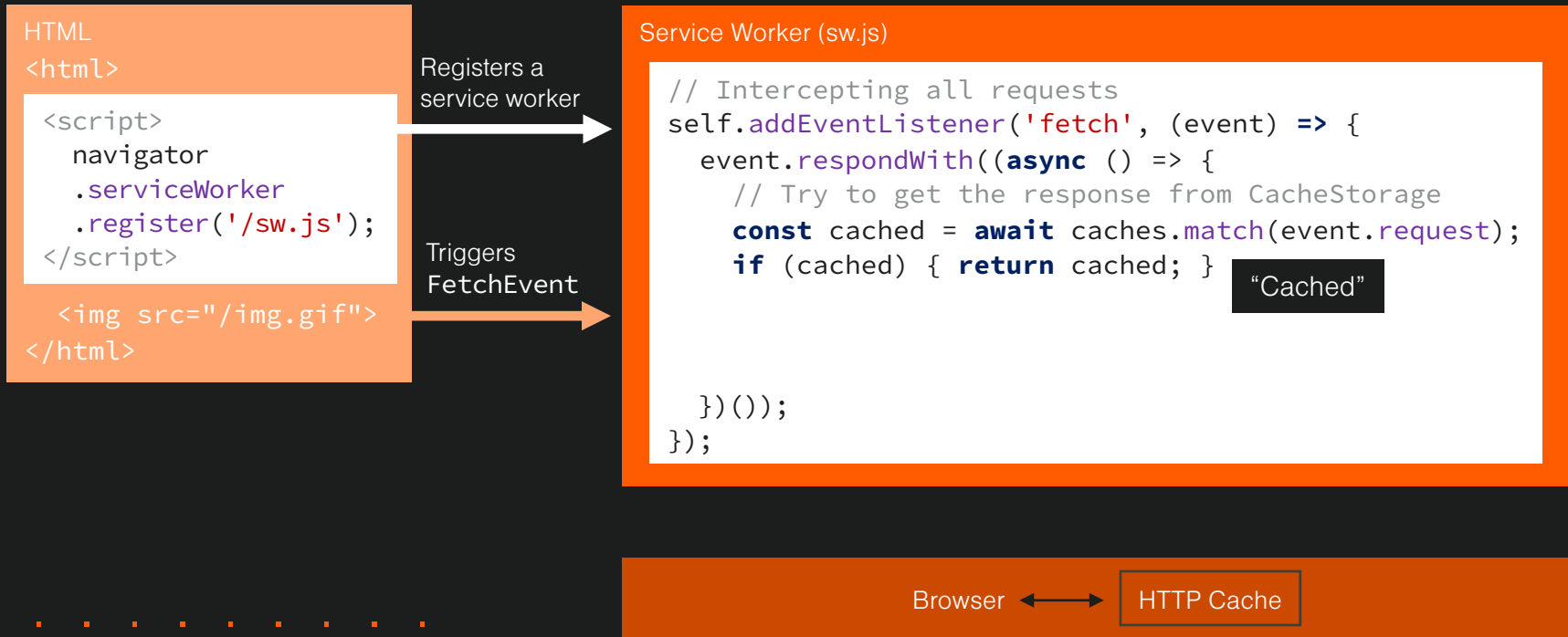
```
// Intercepting all requests
self.addEventListener('fetch', (event) => {

});
```

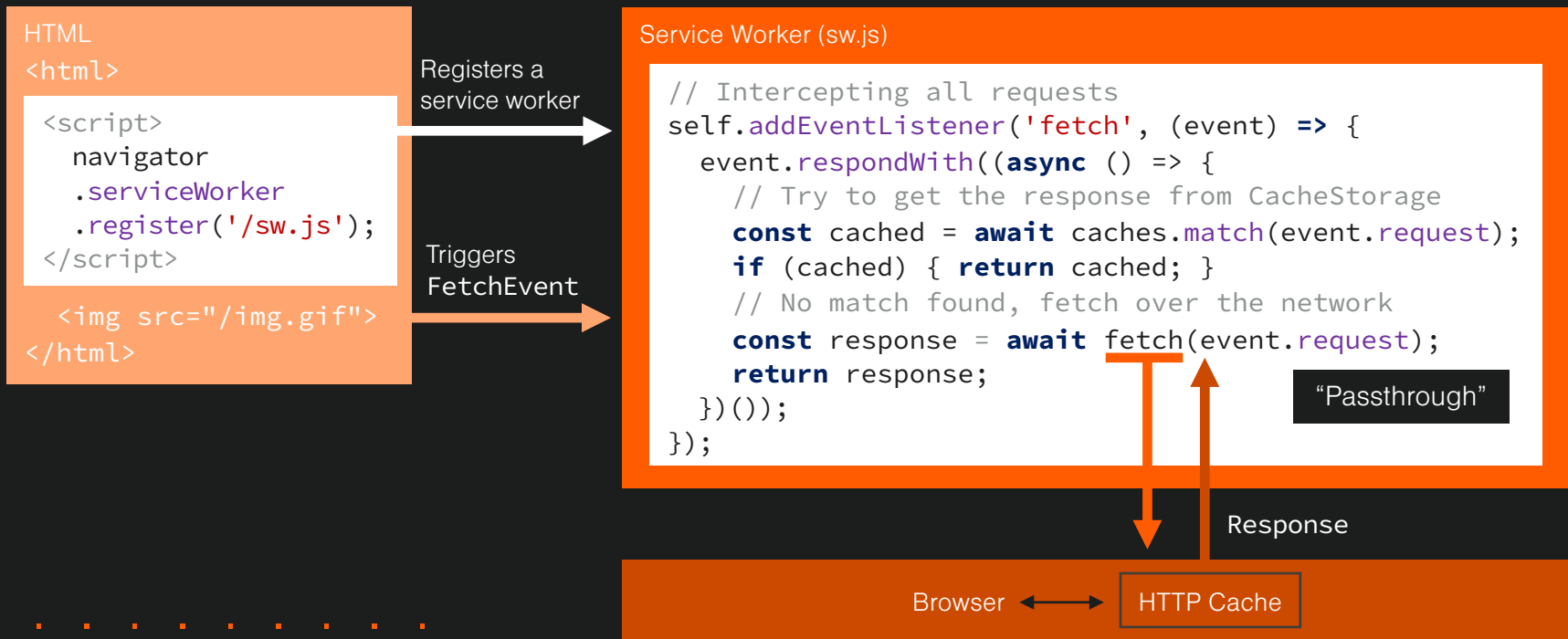
Browser

HTTP Cache

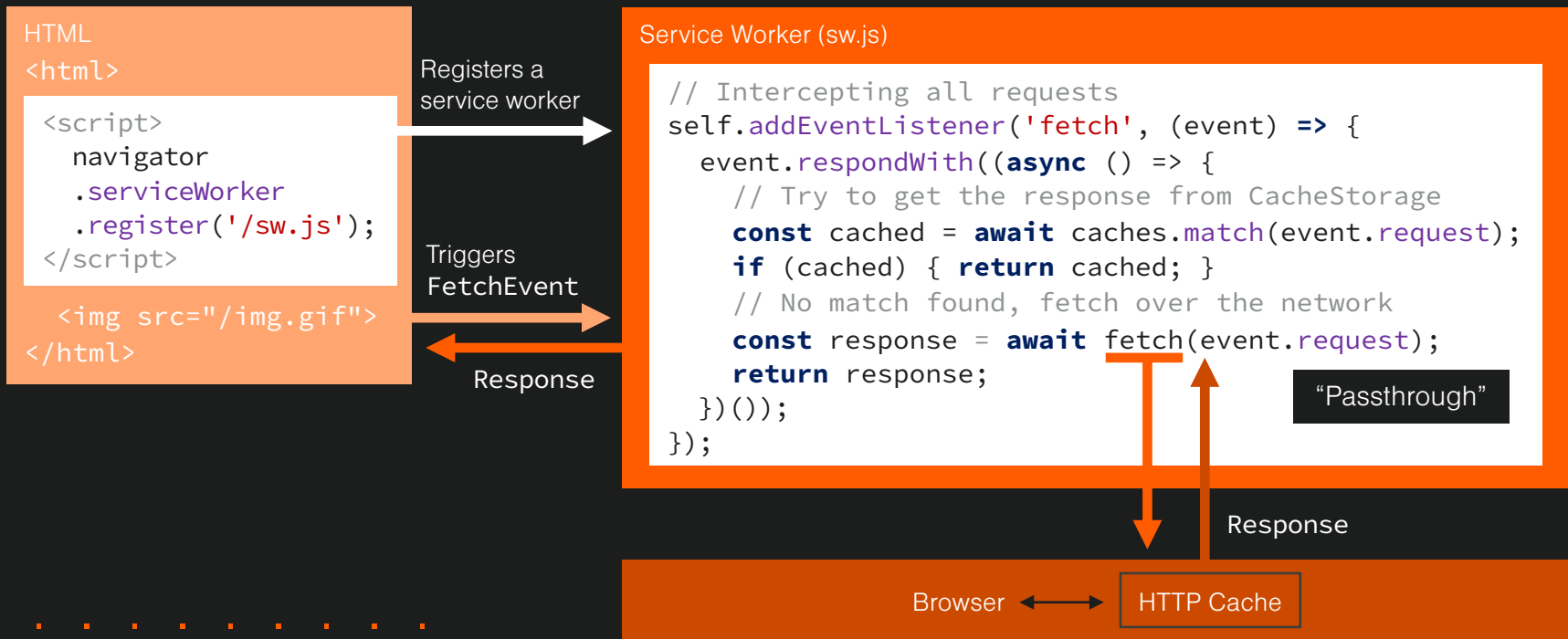
Service Worker Example



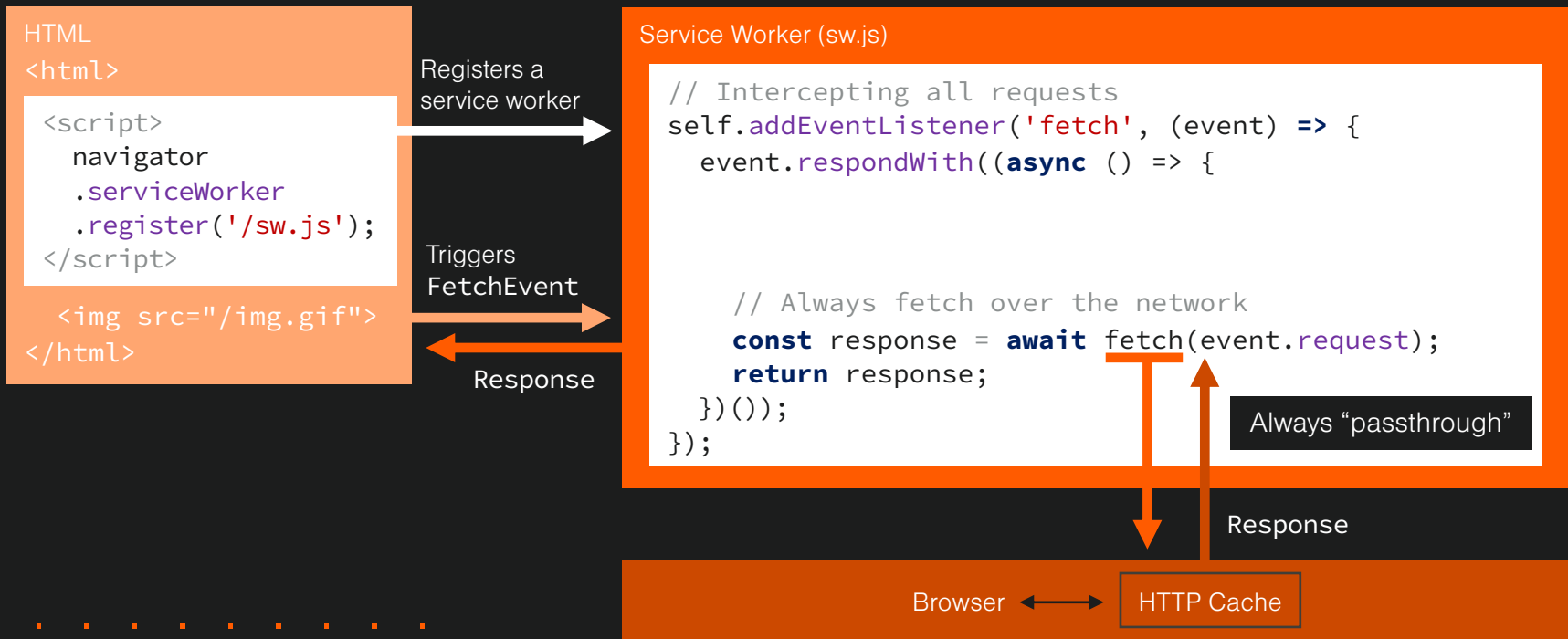
Service Worker Example



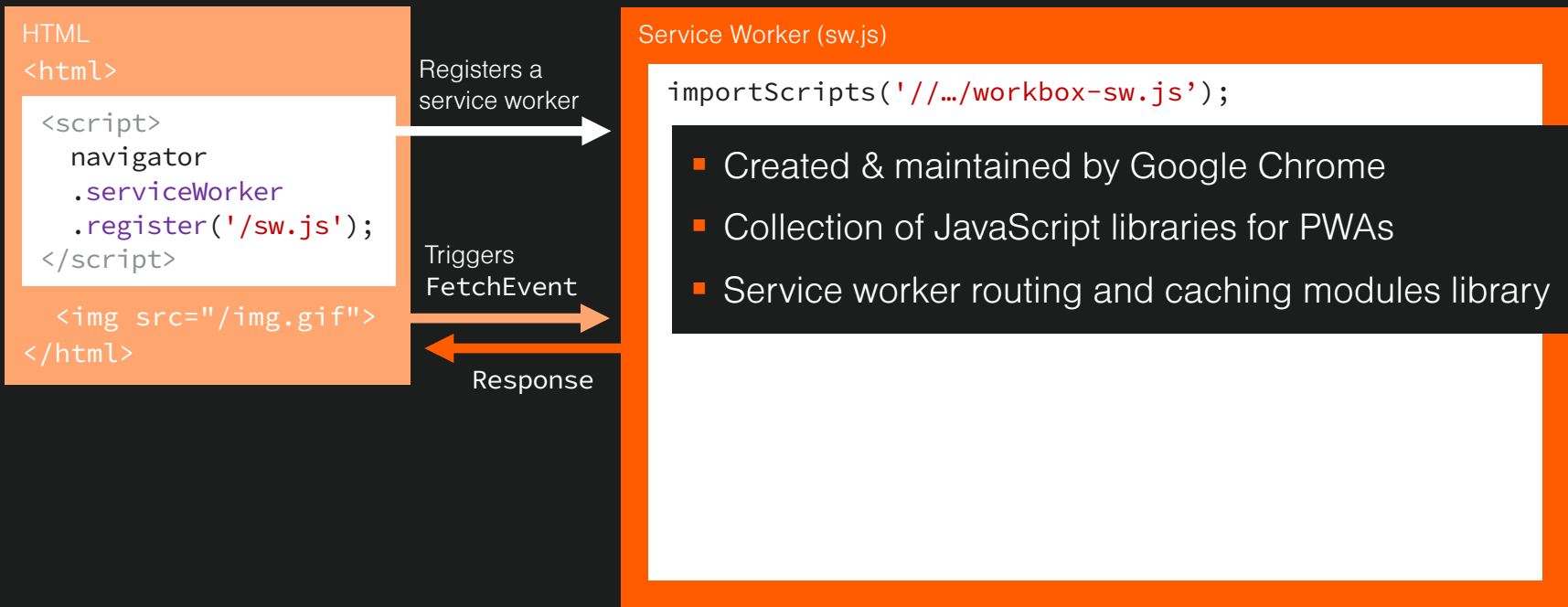
Service Worker Example



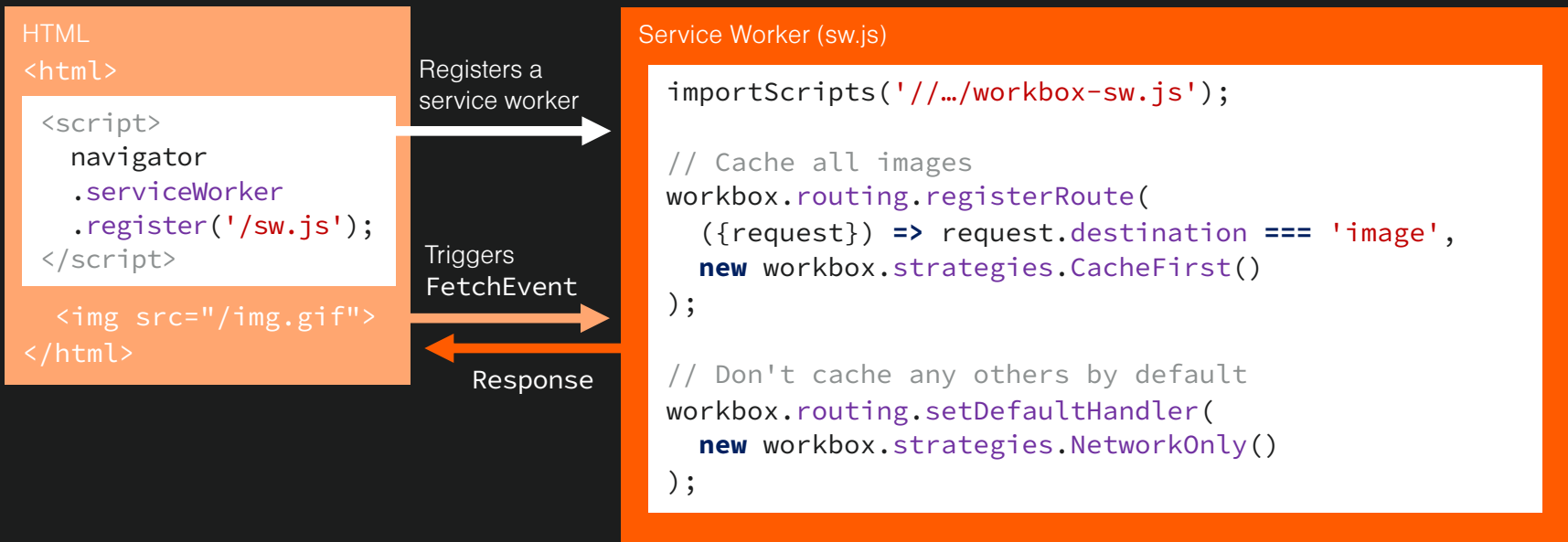
Service Worker Example



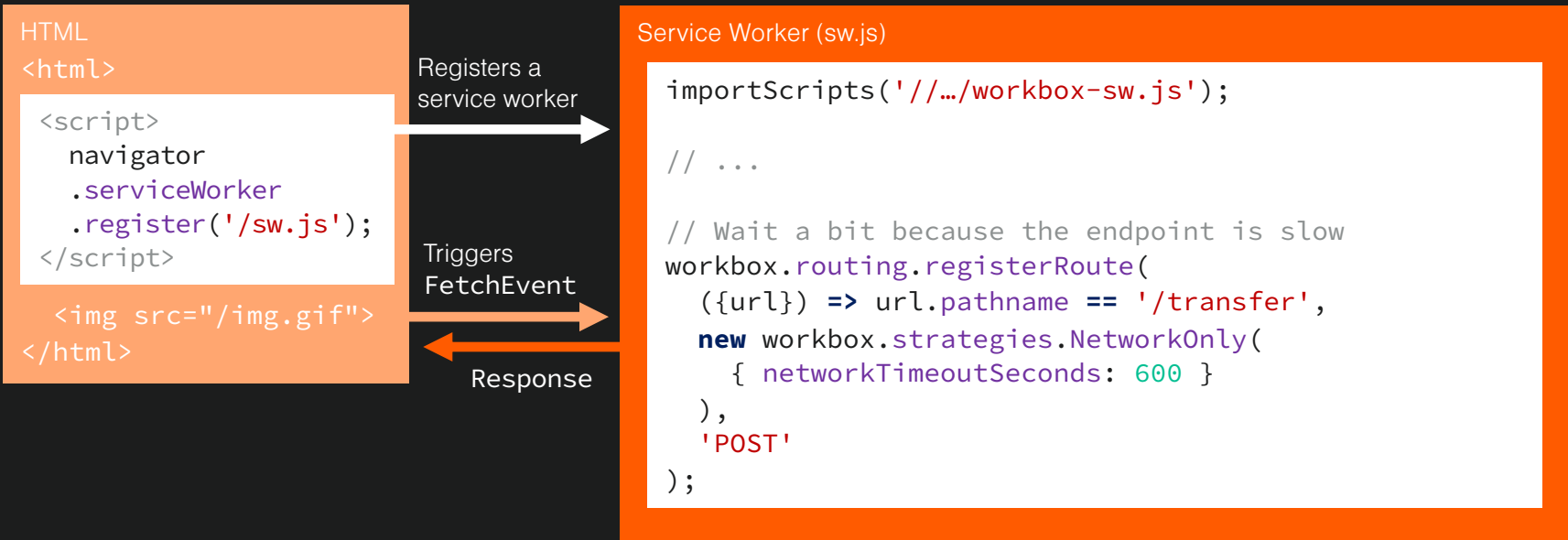
Example: Workbox



Example: Workbox - 1



Example: Workbox - 2



Example: Create React App

Template: cra-template-pwa

src/index.js

```
// ...  
  
serviceWorkerRegistration.unregister();  
  
// ...
```



Example: Create React App

Template: cra-template-pwa

src/index.js

```
// ...  
serviceWorkerRegistration.register();  
// ...
```

Registers a
service worker

src/service-worker.js

Based on Workbox
Customizable





The Collaboration

That Leads to CSRF

Code Spec Smells

HTTP Cookie
SameSite



Progressive Web App
Service Worker



Fetch



datatracker.ietf.org

[Search] [txt] [xml] [pdf] [bibtex] [Tracker] [WG] [Email]

Versions: (draft-thomson-http-omnomnom)
(draft-ietf-httpbis-cookie-alone)
(draft-ietf-httpbis-cookie-prefixes)
(draft-ietf-httpbis-cookie-same-site)
00 01 02 03 04 05 06 07 08 09 10

HTTP
Internet-Draft
Obsoletes: 6265 (if approved)
Intended status: Standards Track
Expires: October 22, 2020

Cookies: HTTP State Management Mechanism
draft-ietf-httpbis-rfc6265bis-06

Abstract

This document defines the HTTP Cookie and Set-Cookie. These header fields can be used by HTTP servers to store (called cookies) at HTTP user agents, letting the server maintain a stateful session over the mostly stateless HTTP protocol. Cookies have many historical infelicities that degraded

W3C Candidate Recommendation Draft

Service Workers

W3C Candidate Recommendation Draft, 12 July 2022

▼ More details about this document

This version:
<https://www.w3.org/TR/2022/CRD-service-workers-20220712/>

Latest published version:
<https://www.w3.org/TR/service-workers/>

Editor's Draft:
<https://w3c.github.io/ServiceWorker/>

History:
<https://www.w3.org/standards/history/service-workers>

web.archive.org

<https://fetch.spec.whatwg.org> Go

2,977 captures
14 Oct 2012 – 12 Jul 2022

NOV 30
2020 2021 2022 About this capture

Fetch

Living Standard — Last Updated 29 November 2021

Participate:

[GitHub whatwg/fetch \(new issue, open issues\)](#)
[Chat on Matrix](#)

Commits:

[GitHub whatwg/fetch/commits](#)
[Snapshot as of this commit](#)
[@fetchstandard](#)

Code Spec Smells

HTML

```
<html>
```

```
<script>
```

```
  navigator
```

```
    .serviceWorker
```

```
    .register('/sw.js');
```

```
</script>
```

```
  
```

```
</html>
```



Code Spec Smells

HTML

```
<html>
```

```
<script>
```

```
  navigator
```

```
    .serviceWorker
```

```
    .register('/sw.js');
```

```
</script>
```

```
  
```

```
</html>
```

WHATWG "HTML Living Standard"

4.8.4.3.5 Updating the image data

23. Fetch the image: Fetch request.



Code Spec Smells

HTML

<html>

```
<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>
```

```

</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

4. Fetching → 4.1. Main fetch → 4.2. Scheme fetch → 4.3. HTTP fetch

5.4. Set *response* to the result of invoking handle fetch for *requestForServiceWorker*.



Code Spec Smells

HTML

```
<html>
```

```
<script>
```

```
  navigator
```

```
    .serviceWorker
```

```
    .register('/sw.js');
```

```
</script>
```

```
  
```

```
</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Appendix A: Handle Fetch

24.3.1. Let *e* be the result of creating an event with FetchEvent.

24.3.12. Dispatch *e* at *activeWorker*'s global object.

ServiceWorker.



Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

```
// Intercepting all requests
self.addEventListener('fetch', (event) => {
  event.respondWith(async () => {
    // Always fetch over the network
    const response = await fetch(event.request);
    return response;
  })());
});
```

Always "passthrough"

Worker.



Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

5.6. Fetch method

The `fetch(input, init)` method steps are:

2. Let *requestObject* be the result of invoking the initial value of Request as constructor with input and *init* as arguments.

});

Worker.



Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

5.4. Request class

6.2. Set *request* to *input*'s request.

12. Set *request* to a new request with the following properties:

URL: *request*'s current URL.

method: *request*'s method.

header list: A copy of *request*'s header list.

client: This's {..., context, origin, ...}

origin: "client"

...

Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

5.4. Request class

6.2. Set *request* to *input*'s request.

12. Set *request* to a new request with the following properties:

URL: *request*'s current URL.

method: *request*'s method.

header list: A copy of *request*'s header list.

client: This's {..., context, origin, ...}

origin: "client"

...

Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

4. Fetching → ... → 4.6. HTTP-network-or-cache fetch

8.20.1.1. Let cookies be the result of running the "cookie-string" algorithm (see section 5.4 of [COOKIES]) with the user agent's cookie store and httpRequest's current URL.

header list: A copy of request's header list.

client: This's {..., context, origin, ...}

origin: "client"

...

Worker.

Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

IETF "Cookies: HTTP State Management Mechanism"

5.2.2.2. Service Workers

Requests which are initiated by the Service Worker itself (via a direct call to `fetch()`, for instance), on the other hand, will have a `client` which is a *ServiceWorkerGlobalScope*.

Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Code Spec Smells

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

IETF "Cookies: HTTP State Management Mechanism"

5.2.2.2. Service Workers

Requests which are initiated by the Service Worker itself (via a direct call to `fetch()`, for instance), on the other hand, will have a client which is a *ServiceWorkerGlobalScope*.

Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Code Spec Smells

HTML – victim.com

```
<html>
```

```
<script>
```

```
  navigator
```

```
  .serviceWorker
```

```
  .register('/sw.js');
```

```
</script>
```

```

```

```
</html>
```

HTML – attacker.com

```
<html>
```

```

```

```
</html>
```

WHATWG “HTML Living Standard”

WHATWG “Fetch Living Standard”

W3C “Service Workers 1”

Service Worker (<https://victim.com/sw.js>)

WHATWG “Fetch Living Standard”

WHATWG “Fetch Living Standard”

WHATWG “Fetch Living Standard”

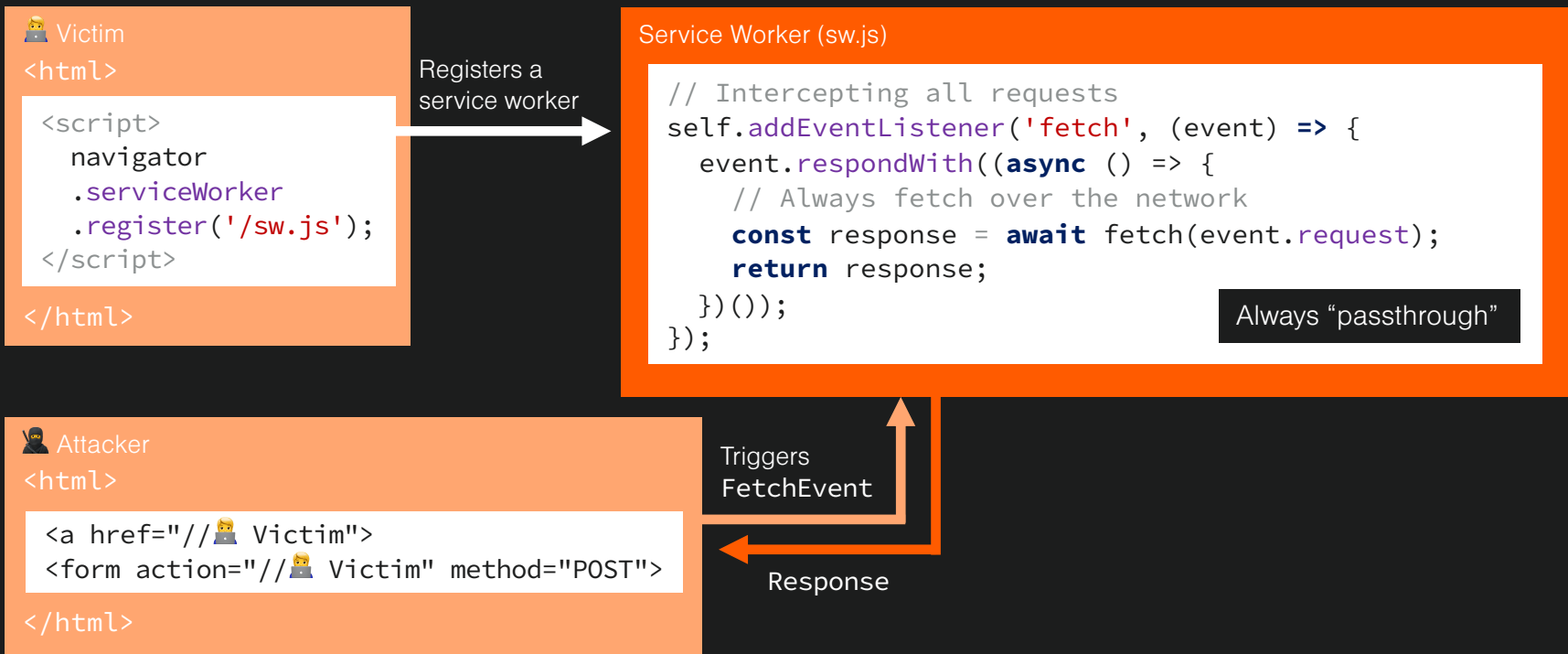
IETF “Cookies: HTTP State Management Mechanism”

5.2.2.2. Service Workers

Requests which are initiated by the Service Worker itself (via a direct call to `fetch()`, for instance), on the other hand, will have a `client` which is a *ServiceWorkerGlobalScope*.

Its “site for cookies” will be the registrable domain of the Service Worker’s URI.

CSRF is Back!



Pitfall: Workbox - 1

👤 Victim

```
<html>
```

```
<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>
```

```
</html>
```

Registers a
service worker

👤

Victim: Service Worker (sw.js)

```
importScripts('/.../workbox-sw.js');

// Cache all images
workbox.routing.registerRoute(
  ({request}) => request.destination === 'image',
  new workbox.strategies.CacheFirst()
);

// Don't cache any others by default*
workbox.routing.setDefaultHandler(
  new workbox.strategies.NetworkOnly()
);
```

👤 Attacker

```
<html>
```

```
<a href="//👤 Victim">
<form
  action="//👤 Victim/transfer"
  method="POST">
```

```
</html>
```

Triggers FetchEvent

Response

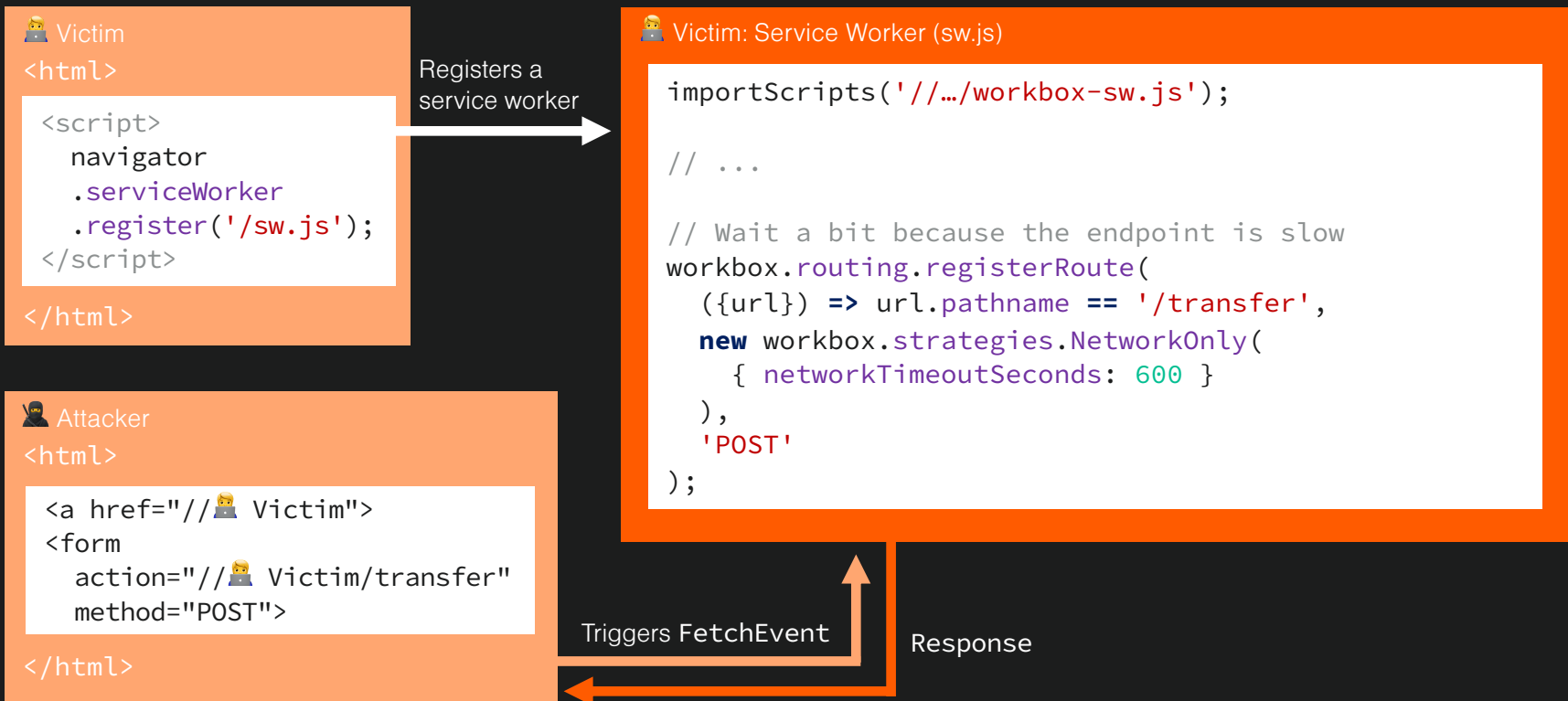
* <=v5 affects all HTTP methods. >=v6 only affects GET.

🚧 Demo C

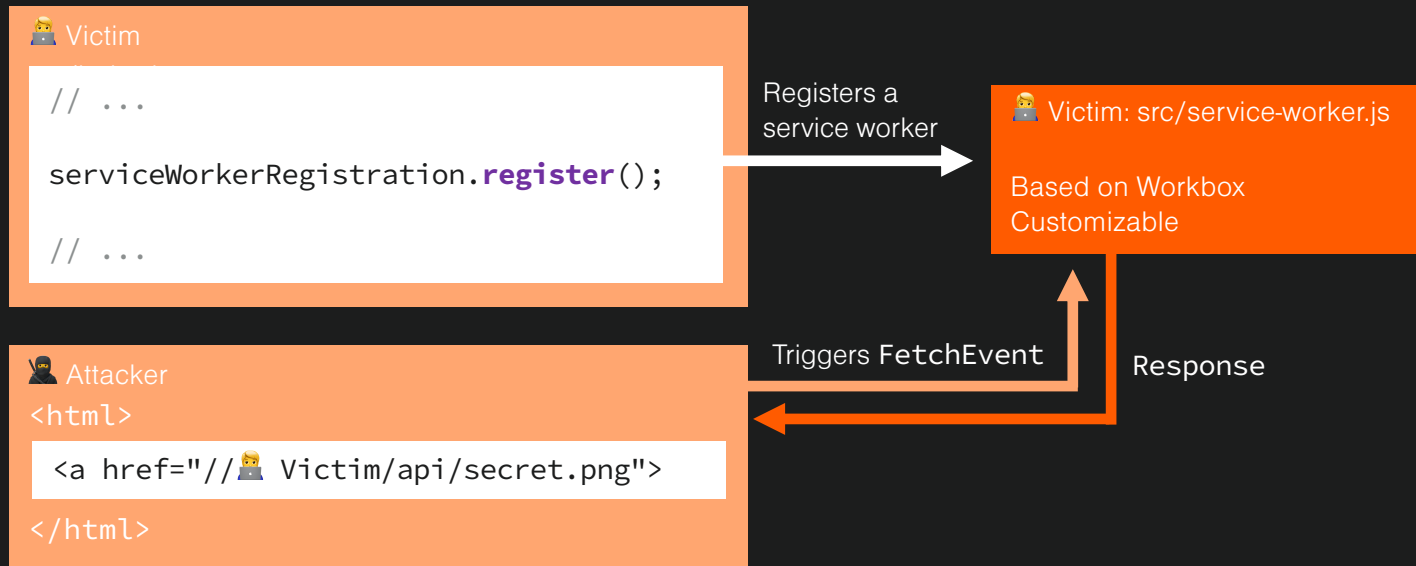
👤 Victim: <https://rogue-comfortable-loaf.glitch.me>

👤 Attacker: <https://acoustic-dusty-jodhpur.glitch.me>

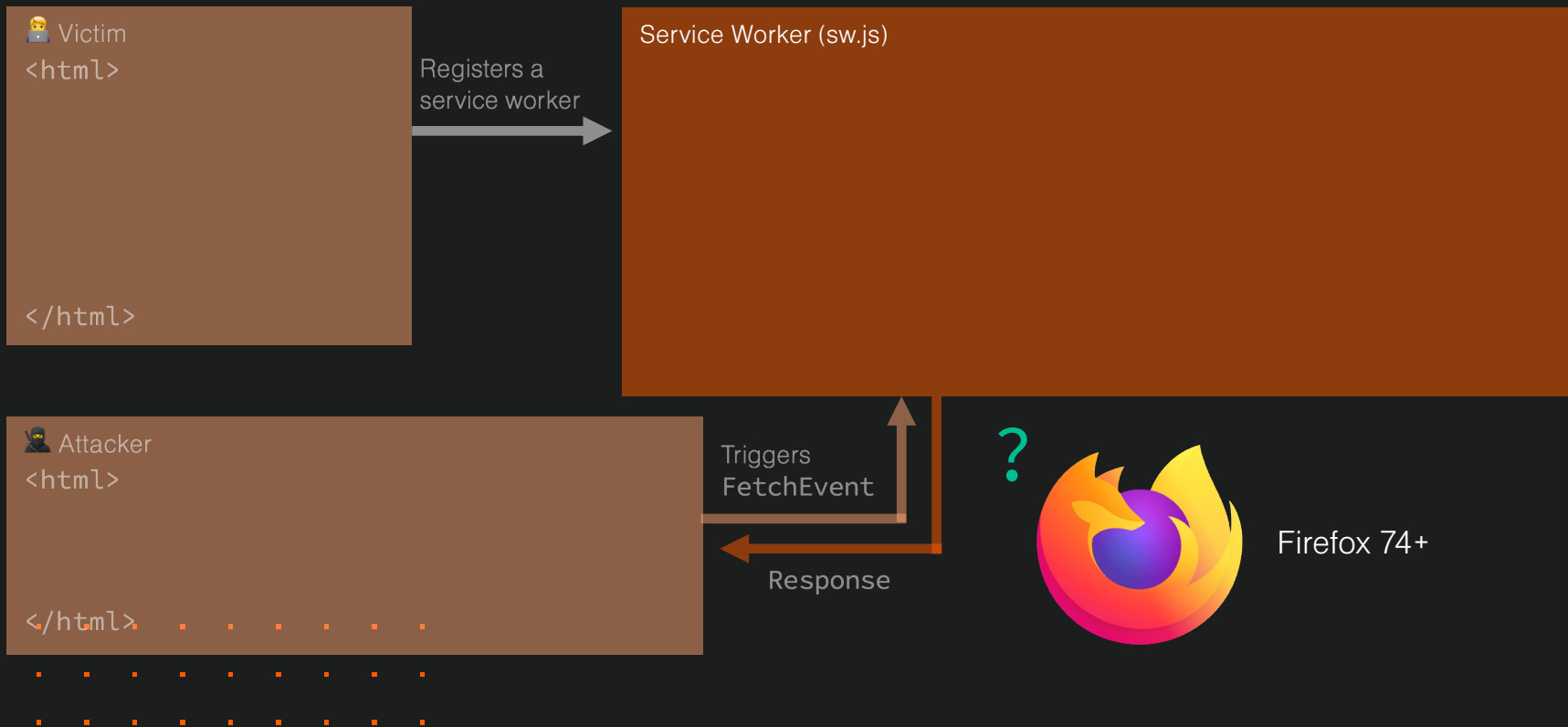
Pitfall: Workbox - 2



Pitfall: Create React App



...and Firefox Developer Tools



Patches, Updates

Yet Issues Still Remain

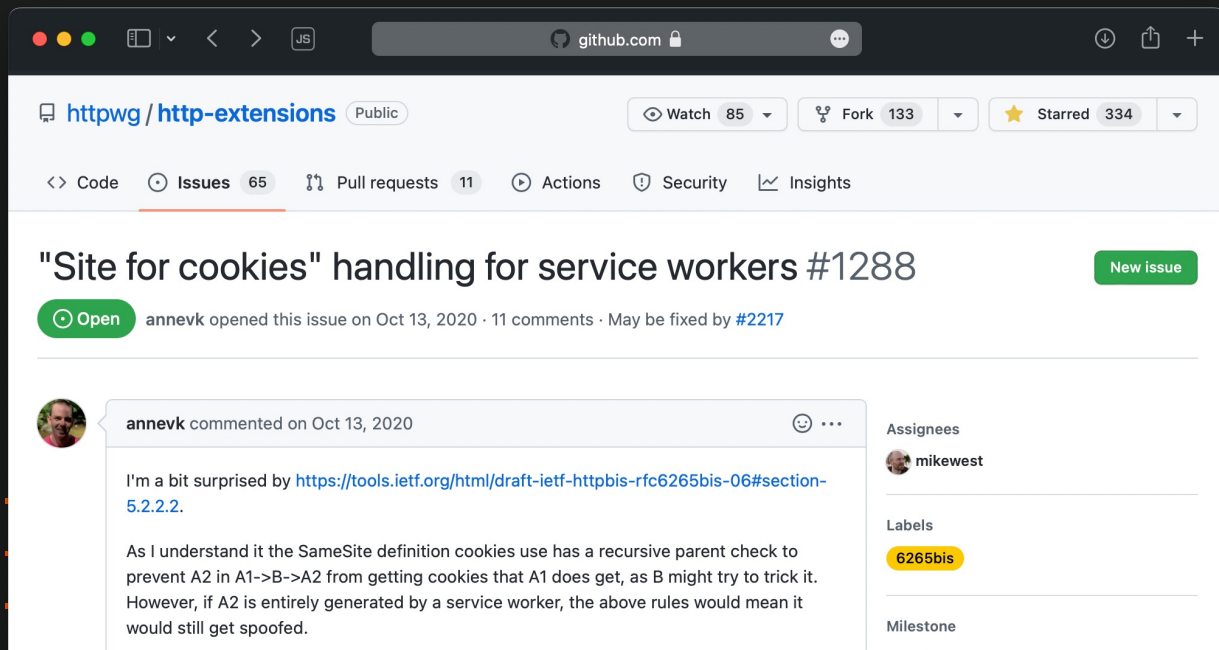
Disclosure Timeline

■ 2020-08-12: Reported to Mozilla and Google



Disclosure Timeline

- 2020-08-12: Reported to Mozilla and Google
- 2020-10-13: An issue on the Cookie spec was independently opened

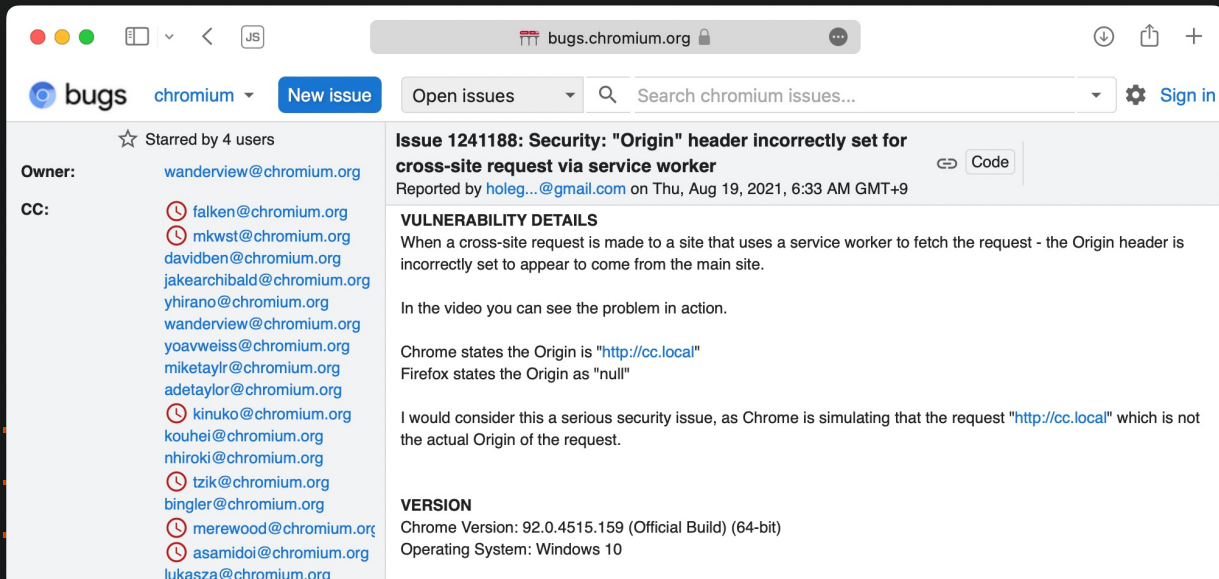


Disclosure Timeline

2020-08-12: Reported to Mozilla and Google

2020-10-13: An issue on the Cookie spec was independently opened

2021-08-19: A bug on the Origin header was independently reported



The screenshot shows the Chromium bug report page for Issue 1241188. The browser address bar shows 'bugs.chromium.org'. The page title is 'Issue 1241188: Security: "Origin" header incorrectly set for cross-site request via service worker'. The report was made by 'holeg...@gmail.com' on Thursday, August 19, 2021, at 6:33 AM GMT+9. The page includes a 'VULNERABILITY DETAILS' section with a description of the issue, a video link, and a 'VERSION' section with the Chrome version and operating system.

bugs chromium New issue Open issues Search chromium issues...

☆ Starred by 4 users

Owner: wanderview@chromium.org

CC: falken@chromium.org, mkwst@chromium.org, davidben@chromium.org, jakearchibald@chromium.org, yhirano@chromium.org, wanderview@chromium.org, yoavweiss@chromium.org, miketaylr@chromium.org, adetaylor@chromium.org, kinuko@chromium.org, kouhei@chromium.org, nhiroki@chromium.org, tzik@chromium.org, bingler@chromium.org, merewood@chromium.org, asamidoi@chromium.org, lukasza@chromium.org

Issue 1241188: Security: "Origin" header incorrectly set for cross-site request via service worker
Reported by holeg...@gmail.com on Thu, Aug 19, 2021, 6:33 AM GMT+9

VULNERABILITY DETAILS
When a cross-site request is made to a site that uses a service worker to fetch the request - the Origin header is incorrectly set to appear to come from the main site.

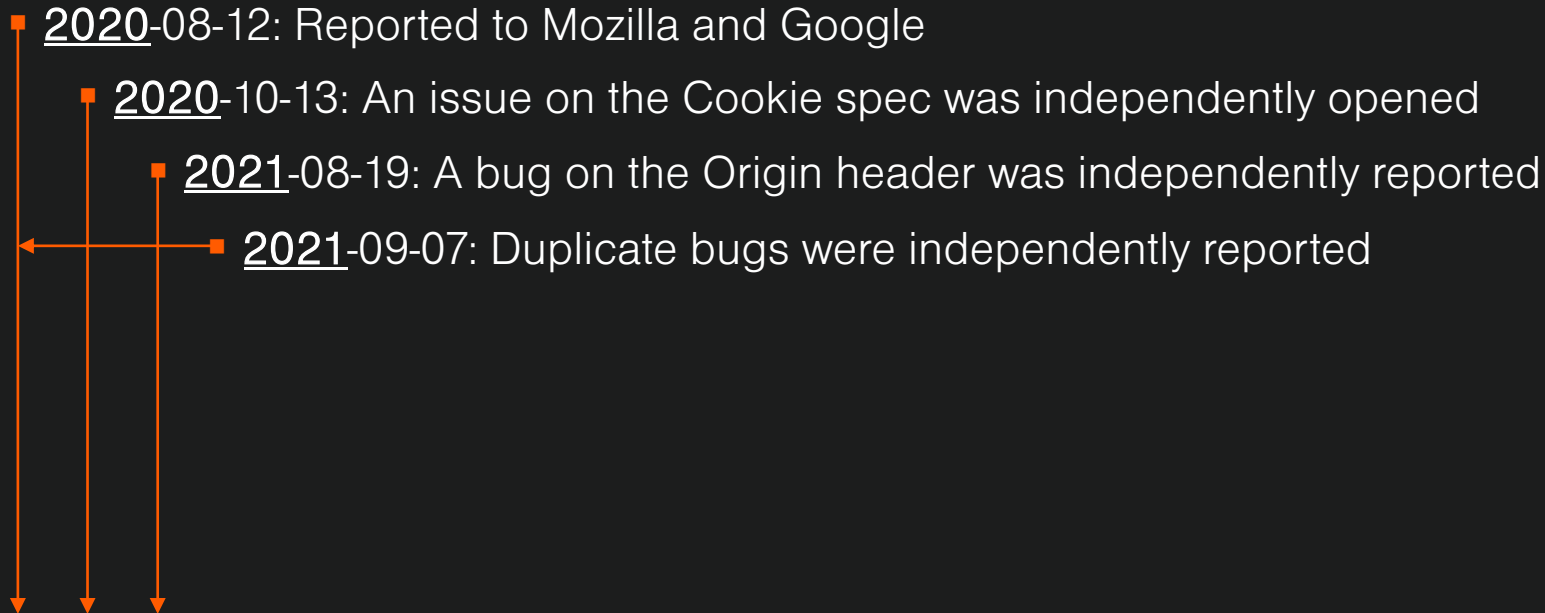
In the video you can see the problem in action.

Chrome states the Origin is "<http://cc.local>"
Firefox states the Origin as "null"

I would consider this a serious security issue, as Chrome is simulating that the request "<http://cc.local>" which is not the actual Origin of the request.

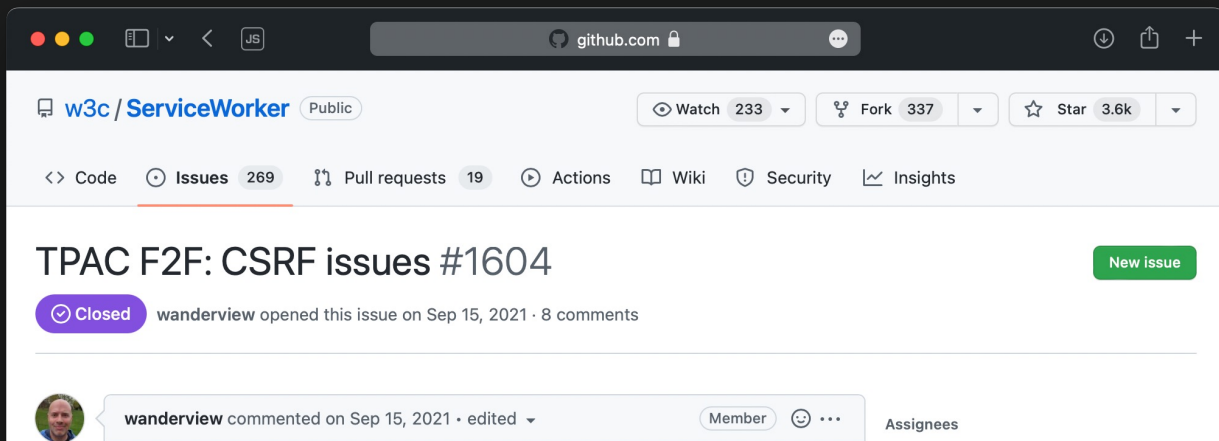
VERSION
Chrome Version: 92.0.4515.159 (Official Build) (64-bit)
Operating System: Windows 10

Disclosure Timeline



Disclosure Timeline

- 2020-08-12: Reported to Mozilla and Google
- 2020-10-13: An issue on the Cookie spec was independently opened
- 2021-08-19: A bug on the Origin header was independently reported
- 2021-09-07: Duplicate bugs were independently reported
- 2021-10-06: A Service Worker spec meeting was held



<https://github.com/w3c/ServiceWorker/issues/1604>
<https://github.com/w3c/ServiceWorker/issues/1606>
<https://github.com/w3c/ServiceWorker/issues/1607>
<https://github.com/w3c/ServiceWorker/issues/1608>
<https://github.com/w3c/ServiceWorker/pull/1629>
<https://github.com/w3c/ServiceWorker/pull/1630>
<https://github.com/w3c/ServiceWorker/pull/1632>
<https://github.com/whatwg/fetch/issues/1321>
<https://github.com/whatwg/fetch/issues/1322>
<https://github.com/whatwg/fetch/issues/1323>
<https://github.com/whatwg/fetch/issues/1327>
<https://github.com/whatwg/fetch/issues/1332>
<https://github.com/whatwg/fetch/issues/1335>
<https://github.com/privacypg/storage-partitioning/issues/25>

Disclosure Timeline

-
- 2020-08-12: Reported to Mozilla and Google
 - 2020-10-13: An issue on the Cookie spec was independently opened
 - 2021-08-19: A bug on the Origin header was independently reported
 - 2021-09-07: Duplicate bugs were independently reported
 - 2021-10-06: A Service Worker spec meeting was held
 - 2021-10-28: Chromium patches both bugs



Disclosure Timeline

-
- 2020-08-12: Reported to Mozilla and Google
 - 2020-10-13: An issue on the Cookie spec was independently opened
 - 2021-08-19: A bug on the Origin header was independently reported
 - 2021-09-07: Duplicate bugs were independently reported
 - 2021-10-06: A Service Worker spec meeting was held
 - 2021-10-28: Chromium patches both bugs
 - 2021-12-01: Fetch spec was amended

The Amended Fetch Spec

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```

WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

5.4. Request class

The new `Request(input, init)` constructor steps are:

12. Set *request* to a new request with the following properties:

URL: *request*'s current URL.

method: *request*'s method.

header list: A copy of *request*'s header list.

client: This's {..., context, origin, ...}

origin: "client"

...

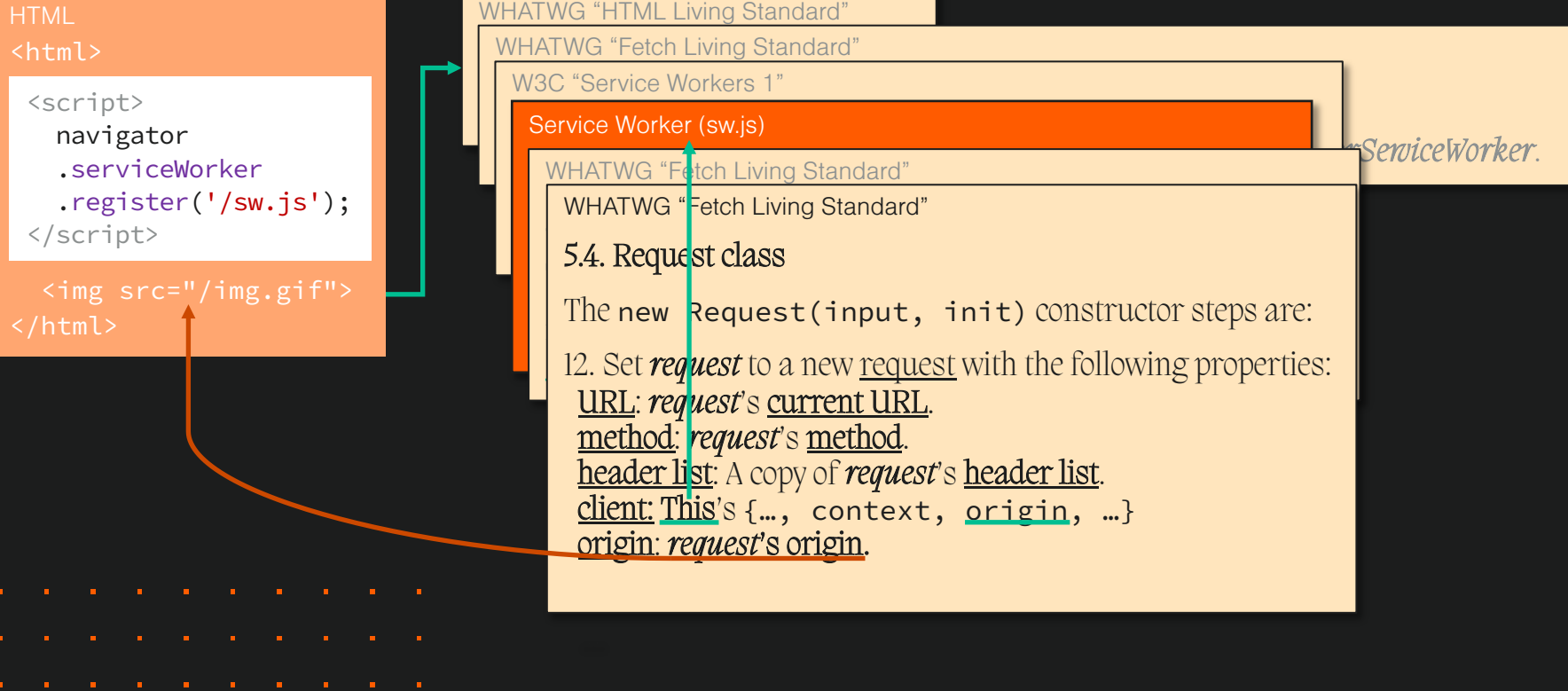
ServiceWorker.

The Amended Fetch Spec

```
HTML
<html>

<script>
  navigator
    .serviceWorker
    .register('/sw.js');
</script>


</html>
```



WHATWG "HTML Living Standard"

WHATWG "Fetch Living Standard"

W3C "Service Workers 1"

Service Worker (sw.js)

WHATWG "Fetch Living Standard"

WHATWG "Fetch Living Standard"

5.4. Request class

The new `Request(input, init)` constructor steps are:

12. Set *request* to a new request with the following properties:

URL: *request*'s current URL.

method: *request*'s method.

header list: A copy of *request*'s header list.

client: This's {..., context, origin, ...}

origin: *request*'s origin.

Disclosure Timeline

-
- 2020-08-12: Reported to Mozilla and Google
 - 2020-10-13: An issue on the Cookie spec was independently opened
 - 2021-08-19: A bug on the Origin header was independently reported
 - 2021-09-07: Duplicate bugs were independently reported
 - 2021-10-06: A Service Worker spec meeting was held
 - 2021-10-28: Chromium patches both bugs
 - 2021-12-01: Fetch spec was amended
 - 2022-01-04: Chrome 97 was released to the stable channel

Disclosure Timeline

- 2020-08-12: Reported to Mozilla and Google
- 2020-10-13: An issue on the Cookie spec was independently opened
- 2022-01-05: Google assigned CVE-2022-0117



Disclosure Timeline

- 2020-08-12: Reported to Mozilla and Google
 - 2020-10-13: An issue on the Cookie spec was independently opened
- 2022-01-05: Google assigned CVE-2022-0117
 - 2022-07-08: Cookie spec is amended (pending)



Disclosure Timeline

- 2020-08-12: Reported to Mozilla and Google
 - 2020-10-13: An issue on the Cookie spec was independently opened
- 2022-01-05: Google assigned CVE-2022-0117
 - 2022-07-08: Cookie spec is amended (pending)
- No update for Firefox available as of now

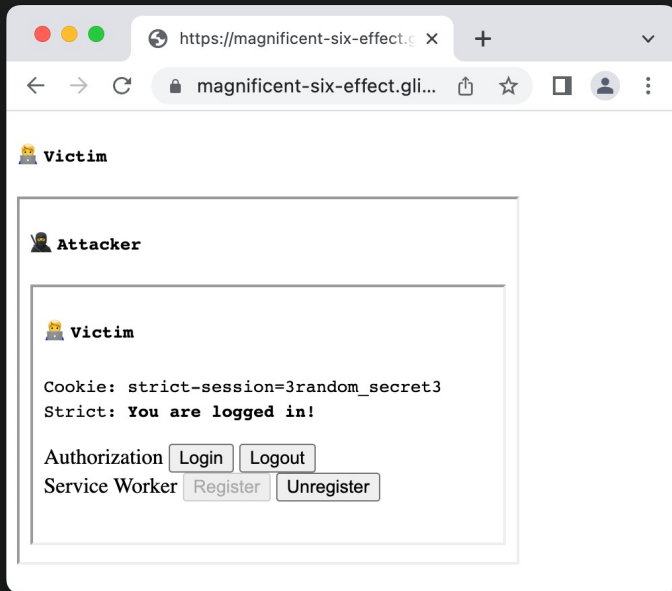
Affected Versions

- Google Chrome: ≥ 51 < 97
- Mozilla Firefox: ≥ 60 (Developer Tools bug: ≥ 74)



Remaining Issue: Nested Frame

- In case of Victim1 (news.com) → Attacker (ad.com) → Victim2 (news.com)
- To be addressed with Client-Side Storage Partitioning



Conclusions

Summary

- SameSite of HTTP Cookie
 - “Cross-Site” Requests and Forgery
 - The Attribute and Lax by Default
- Service Worker of PWA
 - Caching Flow with Service Worker
 - Code Examples
- The CSRF Resurrections
 - Deep Dive into the Specs
 - Proof of Concept with the Code Examples
- The Aftermath
 - Disclosure Timeline
 - Remaining Issue of Nested Frame



Takeaways

- Don't use SameSite cookies as the sole defense mechanism against CSRF
 - Should be considered as an additional layer of defense

“Developers are strongly encouraged to deploy the usual server-side defenses to mitigate the risk more fully.”

– *RFC6265bis*

- Double check network interactions with other tools whenever possible
 - Developer Tools might not actually depict reality
- A possible new attack vector until the issues are fully resolved
- Is there any other problem that has been overlooked?



Thank you!



www.truesec.com / www.security-office.co.kr



twitter.com/truesec



linkedin.com/company/truesec