W3C WebRTC WG Meeting

December 14, 2020
8:00 AM - 9:30 AM Pacific Time

Chairs: Bernard Aboba
Harald Alvestrand
Jan-Ivar Bruaroey

W3C WG IPR Policy

- This group abides by the W3C Patent Policy https://www.w3.org/Consortium/Patent-Policy/
- Only people and companies listed at https://www.w3.org/2004/01/pp-impl/47318/status are allowed to make substantive contributions to the WebRTC specs

Welcome!

- Welcome to the 2nd December interim meeting of the W3C WebRTC WG!
  - During this meeting, we will talk about getCurrentBrowsingContextMedia and HTML capture, Capabilities and Testing.
- We promise that this is the very last virtual interim of 2020!
  - So enjoy the Holidays, and stay safe!

One last request...

- Please respond to the Doodle Poll for a Virtual Interim during the week of January 18-22, 2021:
  - https://doodle.com/poll/5xzvpsp3iw6dwfyu
- Poll closes today (December 14, 2020).

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/December 14_2020
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - https://w3c.github.io/mediacapture-image/
  - https://w3c.github.io/mediacapture-output/
  - https://w3c.github.io/mediacapture-screen-share/
  - https://w3c.github.io/mediacapture-record/
  - https://w3c.github.io/webrtc-pc/
  - https://w3c.github.io/webrtc-extensions/
  - https://w3c.github.io/webrtc-stats/
  - https://w3c.github.io/mst-content-hint/
  - https://w3c.github.io/webrtc-priority/
  - https://w3c.github.io/webrtc-nv-use-cases/
  - https://w3c.github.io/webrtc-dscp-exp/
  - https://github.com/w3c/webrtc-insertable-streams
  - https://github.com/w3c/webrtc-svc
  - https://github.com/w3c/webrtc-ice
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is being recorded. The recording will be public.

Issues for Discussion Today

- Capture This Tab (Elad)
- First Class HTML Capture (Jan-Ivar)
- WebRTC Capabilities
  - PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)
- Testing (Dr. Alex)
- WebRTC Extensions
  - Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)
- Media-Capture Main
  - PR 742: Define the fitness distance for unexposed constraints (Jan-Ivar)

getCurrentBrowsingContextMedia

(Elad Alon)

Save users from accidentally sharing their tentative job-search during their 1:1 with their current manager.

State of the art:

- getDisplayMedia allows the app to call upon the user to pick a share-source.
- The source cannot be restricted by the app; i.e. the app cannot specify that it's interested in tab-sharing.
- Some risk of user sharing wrong thing; e.g. whole screen or a different tab .

Proposal:

- Add getBrowserContextMedia - allow app to prompt the user for permission to share the current tab.

Benefits:

- Apps can reduce the risk of the user sharing the wrong thing. Avoid users accidentally sharing their job-search with their manager.
- The app can intelligently process captured content, based on intimate knowledge with the placement of elements in the captured stream - cropping, annotations, etc.

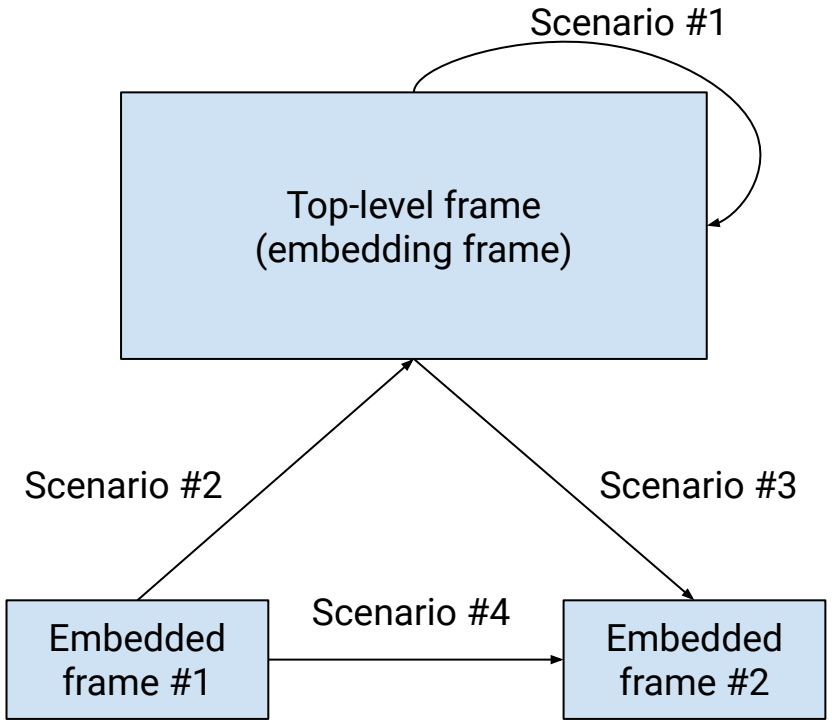
Use cases:

- Sharing of a document (e.g. Google Slides) to a meeting can be done via a button-click from the page, with a browser prompt to allow/disallow, rather than to choose the current tab.
    - **Also**, with getBrowserContextMedia, an app like Google Slides needs not worry about:
        i. Detecting if the user chose another tab by mistake (technically difficult).
        ii. Explaining the mistake to the user and asking him to try again.
- Recording a [video-conferencing / gaming / etc.] session to a local file and/or streaming it (e.g. Twitch).
- Useful for embedding defect-reporting mechanisms in an application - can capture a video of the bug.

The main security issue is the ability of a page to capture output from other sites, breaking the origin-isolation model of Web security.

- Ancient Scandinavian proverb

Risks


Scenario #1
Top-level frame
(embedding frame)
Scenario #2
Scenario #3
Scenario #4
Embedded
frame #1
Embedded
frame #2

1. Browsing context capturing itself

Browsers attempt to prevent applications from spying on the user. For example, to prevent applications from deducing whether a user has visited certain other websites, Chrome intentionally mis-reports to the application the color of links (unless they have not been set to a non-default color by the application itself); this prevents the application from finding out if links are purple, meaning that they have been visited.

Screen-capture allows an application to circumvent these protections - anything that's visible as purple to the user, will be seen as purple by the capture, including links.

When examining this particular example, some ways to preserve the privacy of the user's history come to mind, including:

- Forcing links' color to the default color if a screen-capture is active.
- Denying new screen-captures, and breaking off active screen-captures, if a default-color link is ever visible on the page.

Even when considering just this one example (link-purpling), we cannot think of any cure which would not be more bitter than the disease. We conclude that the only reasonable mitigation would be to make the user acutely aware of the risks of permitting screen-capture. The confirmation-dialog will remain our only mitigation of these risks as a whole, but individual issues could be given special attention as they come up. For example, if link-purpling is ever considered sufficiently dangerous, we could apply one of the aforementioned mitigations that are specific to that issue. We don't, however, have a general, scalable solution for concealing the user's history/identity from the application.

2. Embedded resource capturing the embedding resource

Risk

An embedded resource (e.g. iframe) might capture information from the embedding resource (e.g. top-level application).

Mitigation

The display-capture feature-policy solves this issue for getDisplayMedia, and can do so for getCurrentBrowsingContextMedia as well. Namely, code from a cross-origin resources is not allowed to call either of these functions, unless the iframe-tag includes the relevant allow attribute, specifying allow="display-capture”.

3. Embedding resource capturing an embedded resource

Risk

An embedding resource (e.g. top-level application) might capture information from an embedded resource (e.g. iframe).

Mitigation's Limitations

1. Common and desirable scenario. The mitigation must allow approved captures to proceed uninterrupted - there should be no break in the capture while determining whether the capture is allowed. This rules out hypothetical mitigations that would pause the capture until the embedded resource fully/partially loads, and leaves us with a feature-policy communicated via an HTTP header as our only option.
2. Should not be so restrictive as to prevent real usage of the feature. An opt-in mechanism would be prohibitively restrictive - complex applications would require too many web-servers to be configured (and configuration maintained). We are forced to consider only mechanisms that default to allowing the capture. (Also helpful if applying the mitigation to getDisplayMedia.)
3. Existing mechanisms would be problematic to use, because they would tie together different functionalities that the embedded resource might not be interested in combining. We are compelled to introduce a new feature-policy.

Mitigation

Define a new feature-policy, allow-capture-by-embedder, settable via an HTTP response header, but not via an iframe's allow attribute. A resource served with this HTTP response header may only be captured by allowlisted embedders.

allow-capture-by-embedder

When allow-capture-by-embedder is specified for a resource:

1. Attempts to capture the display by the embedder are only allowed if the embedder is allowlisted (or is same-origin).
2. Any running display-capture by the embedder is broken off.

The policies supported by allow-capture-by-embedder will be:

- '*': Capturing is allowed by any embedder. This is the default, and applies either if the policy is not specified at all, or if it's specified without an explicit policy value.
- 'self': Capturing is allowed by embedders from the same origin only. (Note that same-origin embedders may nevertheless transfer this permission to resources from other origins.)
- 'none': No embedders are allowed to capture (until they stop embedding this resource).
- '<origin(s)>: This policy can be added to 'self'. It specifies specific origins which are still allowed to capture the current resource. (This permission is still transitive.)

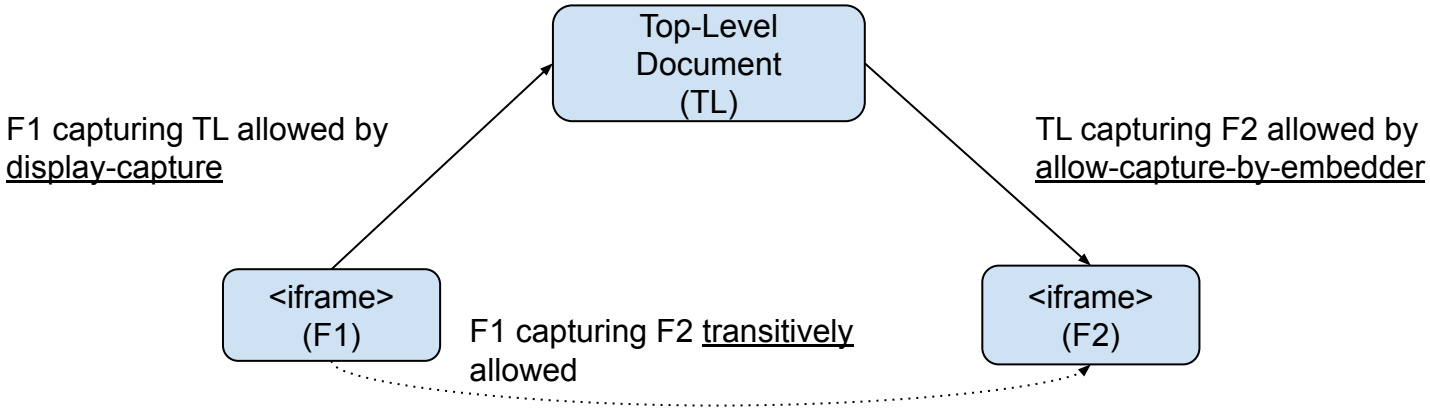
4. One embedded resource capturing another embedded resource

Risk

If multiple resources are embedded, any one of them could end up capturing any of the others.

Mitigation

We define the mitigation of scenario #2 as transitive - any embedded resource that allows its embedder to capture it, assumes this permission to be inherited by any resource that is allowed to capture the embedder itself.



That is, assume TL is the top-level document, and it embeds to iframes, F1 and F2. If F2 allows itself to be captured by TL (managed by allow-capture-by-embedder), and TL allows itself to be captured by F1 (using display-capture), then F2 transitively allows itself to be captured by F1.

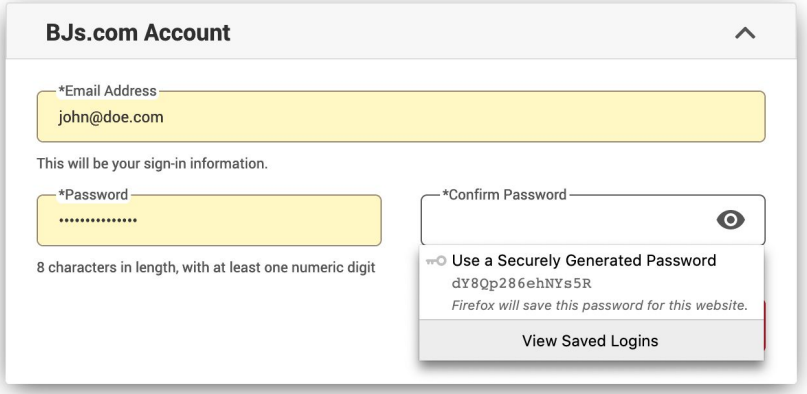
getTabMedia() secured by COEP

(Jan-Ivar - Mozilla)

HTML Capture Security Risks

1. **Cross-origin resource capture** (rendered with user's cookies from other site) violates CORS; active attacks on personal info from multiple sites.

2. **User information harvesting** from same-origin and cross-origins:
   a. Link purpling (browser history)
   b. Form autofill (address, credit card info)
   c. Web Extensions (e.g. LastPass)
   d. Font size (vision, age), coloring, and Other preferences
   e. File input element may contain private info



getTabMedia scope: Support highly motivated use cases

getTabMedia Goals:

1. Solve prohibitive UX flow for *“Present Google Slides”* & “*record this meeting”*.
2. Promote web over native; Present safely with integrated HTML capture.

Stream thyself!

Google’s neat idea: Google Slides streaming itself into an ongoing meeting using existing tech like RTCPeerConnection.

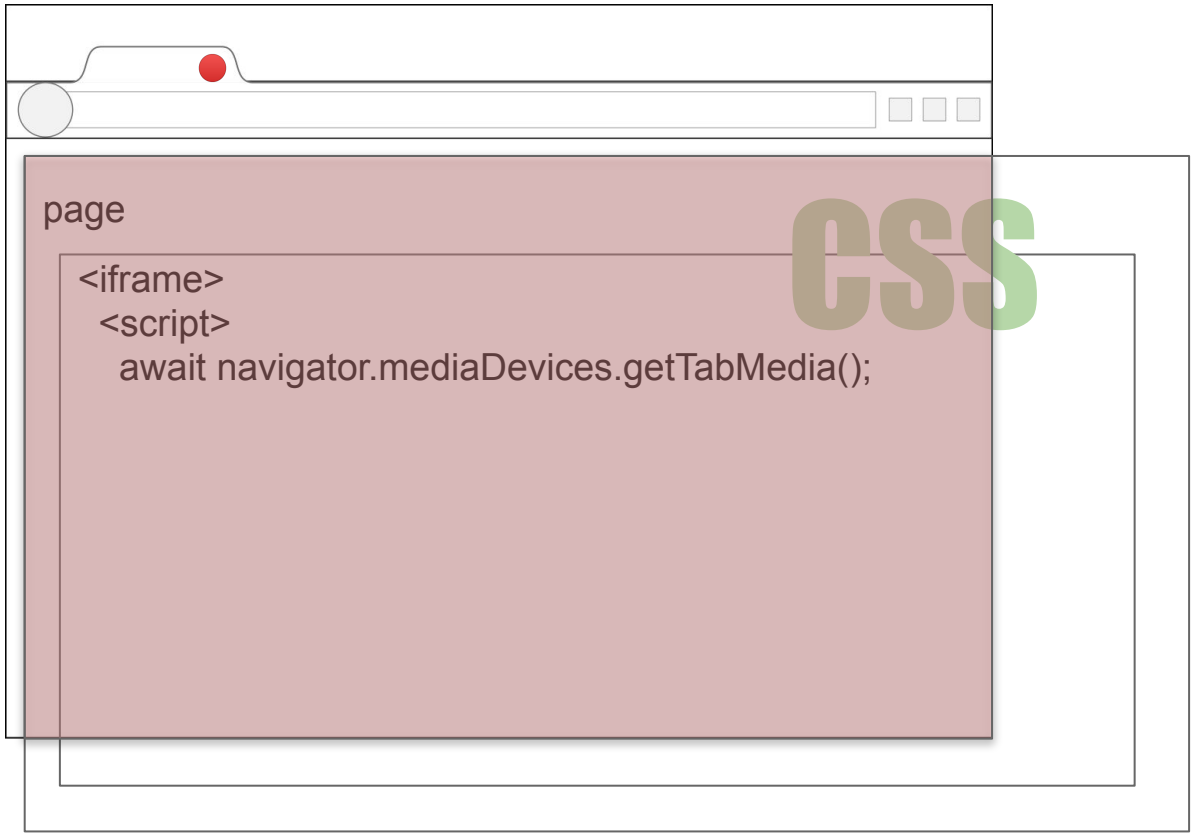
Stream thyself!

Record web conference in progress from client perspective, including web layout.

Appeal: A page only needs to *capture itself*. Buy-in ensured by *requiring code in a highly motivated target*. No threat of capture from outside.

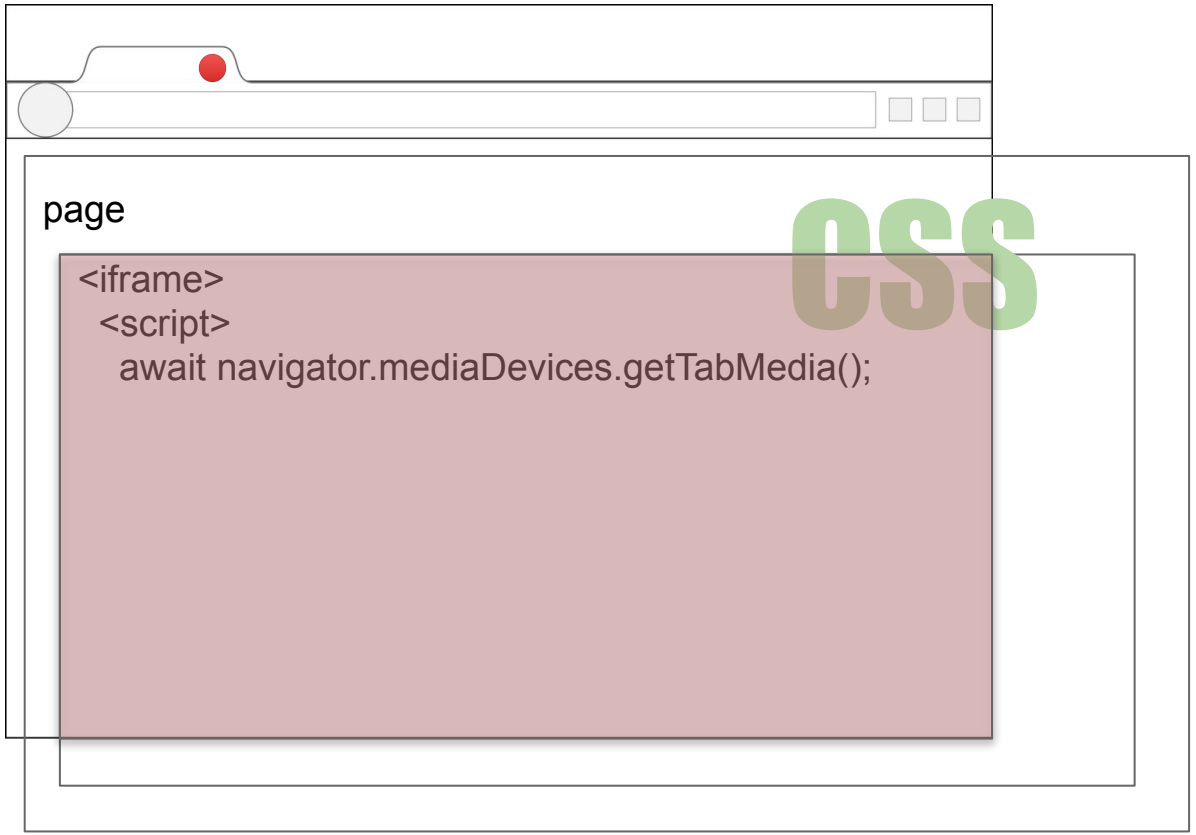
Defining “Stream thyself!”

Proposal A: Capture top-level browsing context’s active document’s ***viewport*.**



Defining “Stream thyself!”

Proposal B: Intersection of TLBC’s active document’s & iframe’s ***viewports***.



Finer control

Solves cropping

Testing (Dr. Alex)

- Simulcast testing with external SFU

- Browsers vendors do not oppose
  (with explicit no commitment about when will they act on it)

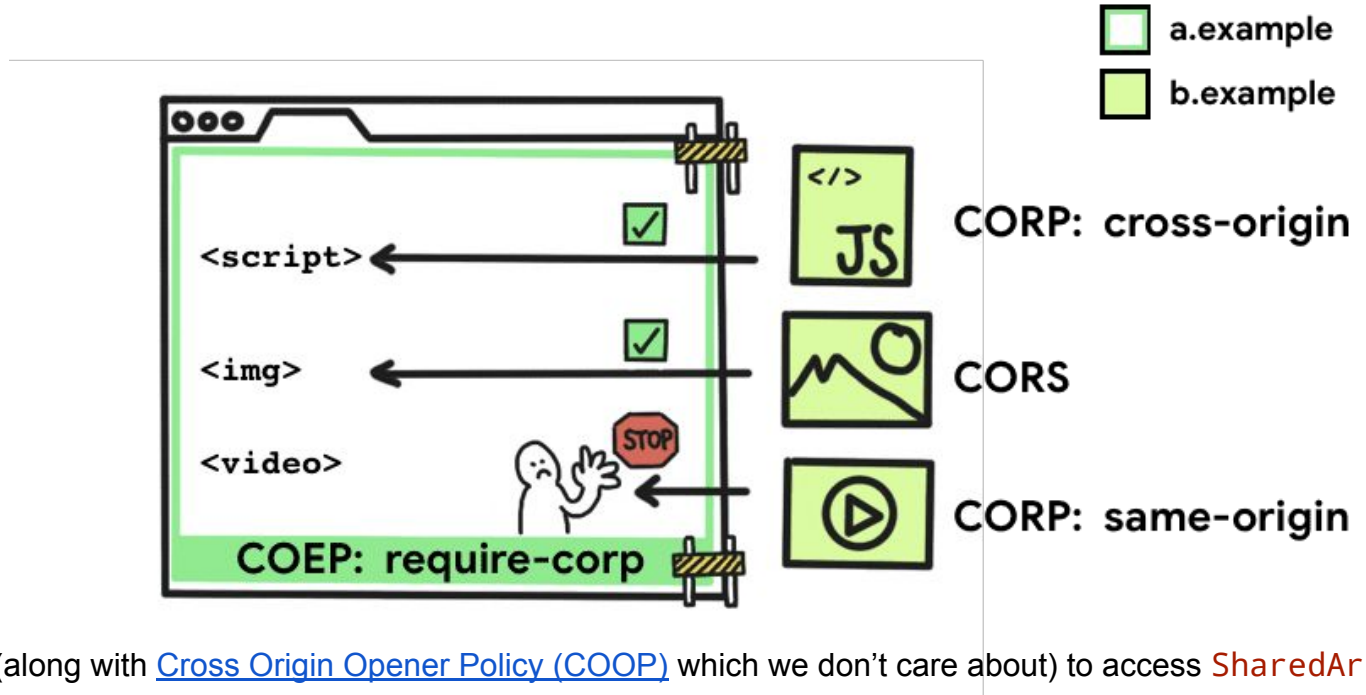
- Candidate SFUs : Medooze / MediaSoup / aioRTC
  - Given the past 2 years experience, Medooze passes.
  - aioquic used for QUIC, included in WPT. *No answer to contact on linkedIn.*
  - MediaSoup tech lead volunteers.

- Next steps
  - Harald is checking with WPT
  - Inaki on stand-by

COEP Explainer

From https://web.dev/why-coop-coep/ *(**TL;DR:** "If the web could be designed from scratch")*

Cross Origin Embedder Policy (COEP) prevents a document from loading any cross-origin resources that don't explicitly grant the document permission (using CORP or CORS). With this feature, you can declare that a document cannot load such resources.



It's ***required*** (along with Cross Origin Opener Policy (COOP) which we don't care about) to access `SharedArrayBuffer`.

COEP Explainer

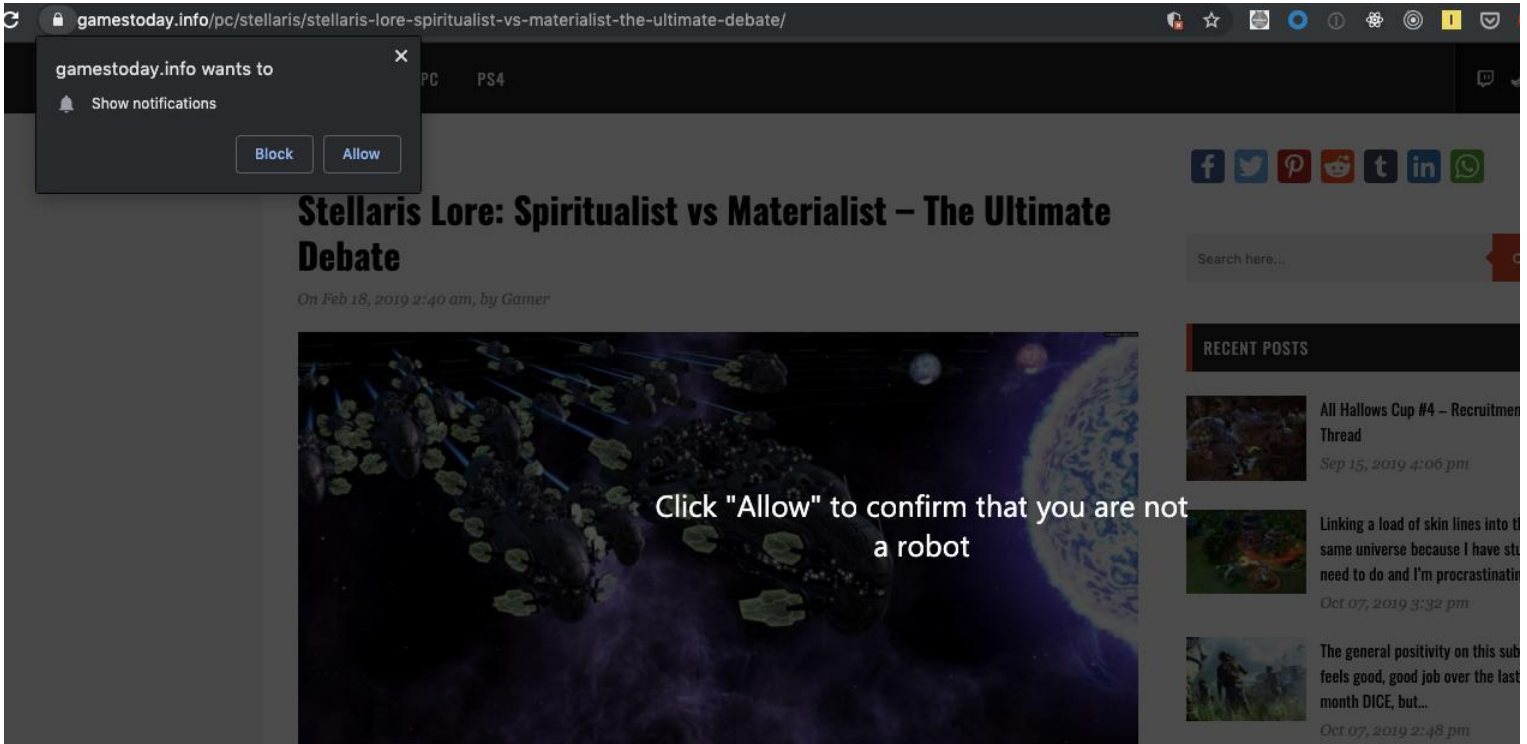
From https://web.dev/why-coop-coep/

> “The web is built on the same-origin policy: a security feature that restricts how documents and scripts can interact with resources from another origin. …
>
> For a long time, the combination of CORS and opaque resources was enough to make browsers safe. …
>
> This all changed with Spectre, which makes any data that is loaded to the same browsing context group as your code potentially readable. ***If `evil.com` embeds a cross-origin image, they can use a Spectre attack to read its pixel data***, which makes protections relying on "opaqueness" ineffective.”

Who's in charge of screen-sharing?

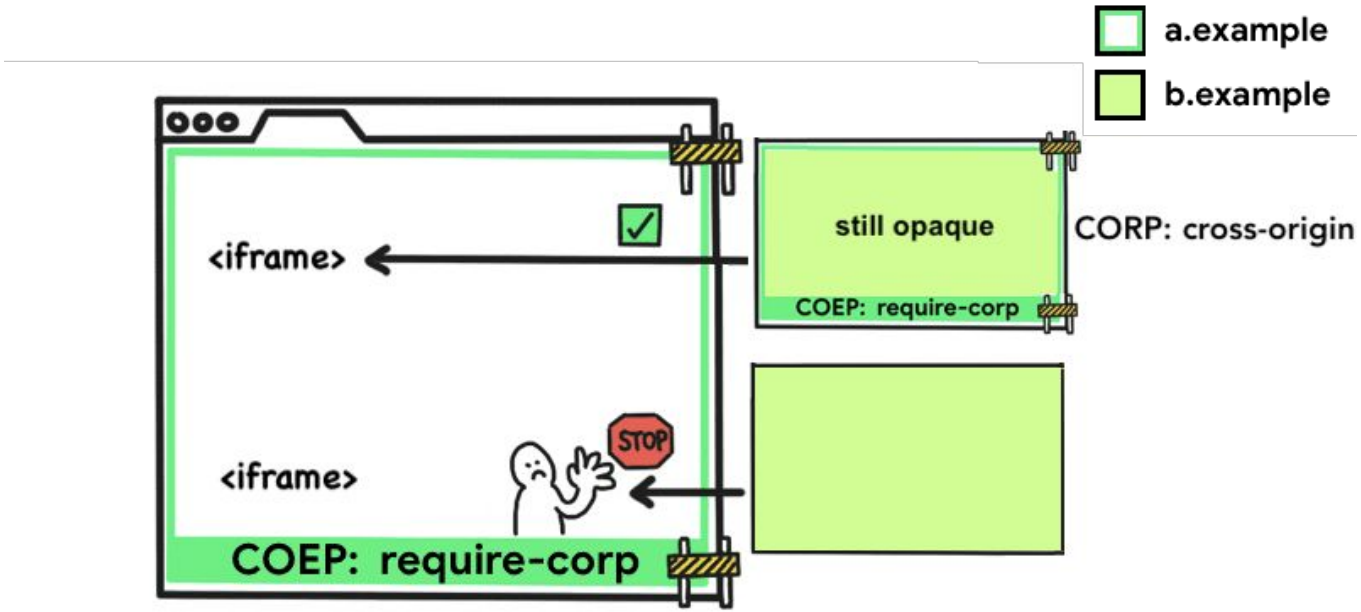
`getDisplayMedia`: The user

`getTabMedia`: The site



COEP Explainer on iframes

Web.Dev article doesn't elaborate on it, but COEP applies to iframes as well.
COEP merely blocks loading, i.e. cross-origin iframes are still opaque.

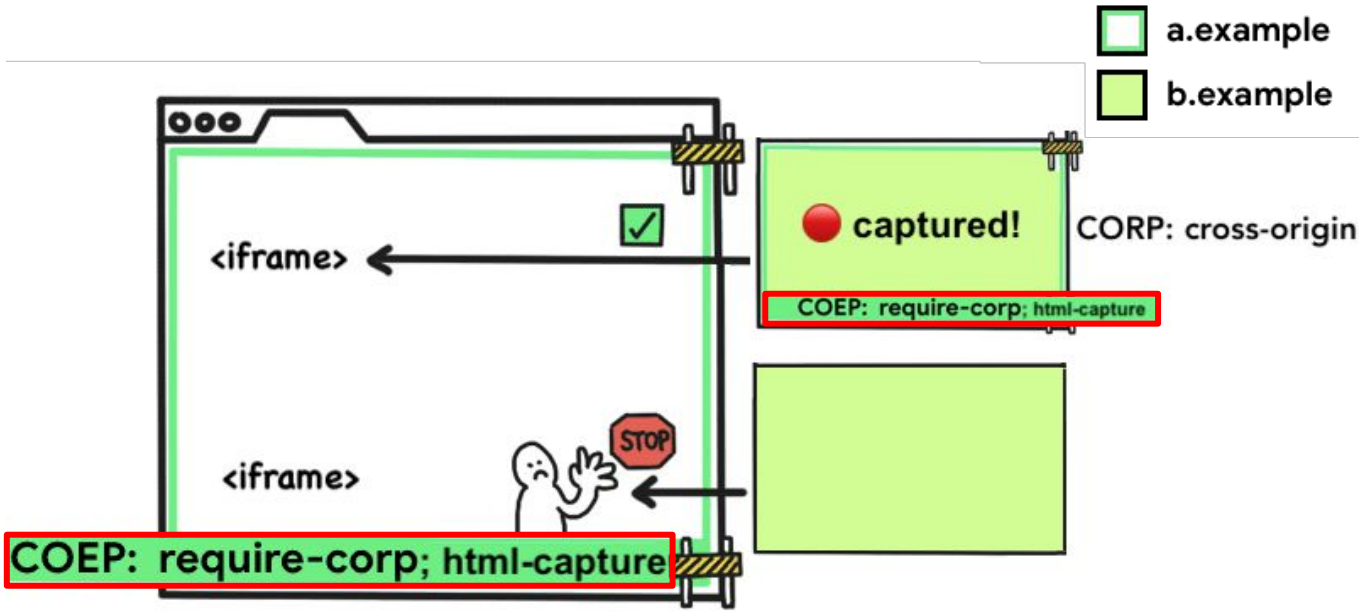


Compat: opting in today doesn't mean you're opting into HTML capture. Need flag.

COEP Explainer on iframes

Proposal: Add a new “html-capture” modifier to COEP: require-corp

```
Cross-Origin-Embedder-Policy: require-corp; html-capture
```



Require this for `getTabMedia()` to not reject with `SecurityError`.

Mitigate User information harvesting

- Require (a subset of) existing **mediacapture-screen-share** protections:
  - MUST require User Gesture (transient activation)
  - MUST follow Privacy Indicator Requirements
  - MUST NOT store a "granted" permission entry

- Require additional protections:
  - MUST require permission to use (*“Allow site to record its rendering?”*)
  - MUST attempt to explain the risk to the user (*“This may fingerprint you”*)
  - MUST mute/pause while `document.visibilityState != “visible”`
  - MUST require current global object == relevant global object, to reject `iframe.contentWindow.navigator.mediaDevices.getDocumentMedia()`

MAY allow User Agents to further mitigate information leakage in this COEP mode (but not to the point of making it an Ad-blocker blocker mode!)

Bikeshed on API

Good idea to define API under **mediacapture-screen-share**, to leverage existing protections there like the Privacy Indicator Requirements. We can still bikeshed:

```
await navigator.mediaDevices.getTabMedia();
```

or

```
await navigator.mediaDevices.getDocumentMedia();
```

or

```
await navigator.mediaDevices.getHTMLMedia();
```

or

```
await document.captureStream();
```

WebIDL

```
partial interface Document {
  [SecureContext] Promise<MediaStream> captureStream();
};
```

Bikeshed on Permissions policy

Sites use "display-capture" today to enable the highly user-driven `getDisplayMedia` picker API: 🥏

```
<iframe src="https://B.com/index.html" allow="display-capture">
```

Given the security risks are slightly different, should we define a new policy? 🗿

```
<iframe src="https://B.com/index.html" allow="tab-capture">
```

or

```
<iframe src="https://B.com/index.html" allow="html-capture">
```

Being specific avoids permission escalation.

Issues for Discussion Today

- WebRTC Capabilities
  - PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)

PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)

Problem: It's not clear what the requirements are of getCapabilities().

1. As previously noted, for privacy reasons *"browsers can consider mitigations such as reporting only a common subset of capabilities"*.
2. getCapabilities() is a synchronous API, but querying HW encoder/decoder support might block JS.

Intended use case (example):

- Check codec capabilities and do RTCRtpTransceiver.setCodecPreferences() *before or during* O/A exchange.

Is our intended use-case fulfilled if only a subset is reported?

- Only if getCapabilities() and createOffer()/createAnswer() match…
- Would an implementation where they don't match be spec-compliant?

PR 2597: Allow piggybacking getCapabilities on most recent Offer or Answer (Henrik)

Proposal 1:

- Clarify that if a capability was exposed in createOffer() or createAnswer() it must also be exposed in getCapabilities() if later called in the same document.
  - ✔️ Prevents having to parse SDP to obtain the full set of capabilities.
  - ❌ Does NOT prevent getCapabilities() from being HW-agnostic prior to createOffer()/createAnswer(). **Proposal 2:** ✔️ Require this to also match?
- Clarify that if a capability was exposed in getCapabilities() it MUST be returned by future getCapabilities() in the same document.
  - ✔️ Prevents setCodecPreferences() from throwing InvalidModificationError due to codec not being in the capability list.

Hopefully no implementation needs to be updated to fulfill these requirements.

- Revisit async API in an extension spec.

Issues for Discussion Today

- WebRTC Extensions
  - Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)
- Media-Capture Main
  - PR 742: Define the fitness distance for unexposed constraints (Jan-Ivar)

Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)

- **<content goes here>**

PR 742: Define the fitness distance for unexposed constraints (Jan-Ivar)

- **<content goes here>**

For extra credit

Name that Bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird

Appendix A: No mitigation for x-origin video & image capture

Simple HTML header doesn't protect non-iframe cross-origin resources at all 👎

```
// A.com/index.html
// Headers: none
<video src="https://C.com/video1234567890.webm"> // 🔴 captures video from C!
<iframe src="https://B.com/iframe.html" allow="html-capture">

// B.com/iframe.html
// Headers: Allow-Capture-By-Embedder: *
<video src="https://C.com/video1234567890.webm"> // 🔴 captures video from C!


// C.com/video1234567890.webm
// Headers: none
```

Affects both videos and images (<img>)

Appendix B: Poor mitigation for Cross-origin iframe capture

Simple HTML header easily defeated by wrapping an iframe in another iframe 👎

```
// A.com/index.html
// Headers: none
<iframe src="https://B.com/wrapper_exploit.html" allow="html-capture">

// B.com/wrapper_exploit.html
// Headers: Allow-Capture-By-Embedder: *
<iframe src="https://C.com/index.html" allow="html-capture"> // 🔴 captures C!


// C.com/index.html
// Headers: none
```

...unless we continuously check headers all the way down, but we're chasing DOM changes (iframes added, iframe navigation, redirects etc. 😮) = Poor man's COEP.

Appendix C: Mitigate x-origin video & image capture w/COEP

COEP already requires videos & images be either CORP or CORS 👍

```
// A.com/index.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
<video src="https://C.com/video1234567890.webm"> // C not loaded
<iframe src="https://B.com/iframe.html" allow="html-capture">

// B.com/iframe.html
// Headers: Cross-Origin-Embedder-Policy: require-corp
//          Cross-Origin-Resource-Policy: capture
<iframe src="https://C.com/index.html" allow="html-capture"> // C not loaded


// C.com/video1234567890.webm
// Headers: none

// D.com/publicVideo.webm
// Headers: Cross-Origin-Resource-Policy: cross-origin (or Access-Control-Allow-Origin: *)
```

Appendix D: Mitigate Cross-origin iframe capture with COEP

Leverages Cross-Origin-Embedder-Policy to opt-in iframes to capture 👍

```
// A.com/index.html
// Headers: Cross-Origin-Embedder-Policy: require-corp; html-capture
<iframe src="https://B.com/wrapper_exploit.html" allow="html-capture">

// B.com/wrapper_exploit.html
// Headers: Cross-Origin-Embedder-Policy: require-corp; html-capture
//          Cross-Origin-Resource-Policy: cross-origin; html-capture
<iframe src="https://C.com/index.html" allow="html-capture"> // C not loaded


// C.com/index.html
// Headers: none
```

COEP ensures iframes are origin clean or (mockup) explicitly opted-in to capture.

This being a powerful feature, a higher bar of entry = a good thing.