Enabling same-site cross-origin prerendering

Domenic Denicola , 2022-07-26
PUBLIC: world-readable / chromium.org commentable

Background

Right now prerendering is restricted to same-origin targets: any cross-origin target or redirect will cause the prerender to be canceled. We are interested in expanding this to allow **cross-origin, but still same-site** prerendering. For example, https://a.example.com/ prerendering https://b.example.com/. We are *not* yet looking to allow cross-site prerendering.

Technically the boundary in question is actually the network partition key / NetworkIsolationKey. For top-level documents this is the site, but due to storage partitioning it differs for subframes. This distinction is not really important for prerendering since we do not allow prerendering inside a subframe, so we will mostly talk about sites in this document.

Side note: prefetching is allowed cross-origin. It has separate paths depending on the network partition key (see spec). Thus, it does not distinguish between same-site cross-origin and same-site same-origin; they both go down the "partitioned prefetch" path. In this way this project can be seen as part of aligning prefetching and prerendering into a more consistent story.

Security concerns vs. privacy concerns

Site / network partition key is the web's privacy boundary. So as long as we stay within that, we are fine from a privacy perspective.

Origin is the web's security boundary. So allowing cross-origin prerendering mostly needs to be considered from a security perspective. In particular:

- Can a referrer at origin A attack a prerendered document at origin B?
- Can a prerendered document at origin B attack its referrer at origin A?

Here "attack" generally means things like:

- Read or write the contents of the DOM or any sensitive data stored in JavaScript
- Trigger actions on the other origin that cannot be easily triggered otherwise (i.e. cannot be triggered via fetch() / <form> submission / an <iframe> hosting that origin).
- Tell what is going on in the other origin's window (e.g. session history, loading of resources, …)
- Use privileges granted by the other origin (e.g., hardware access or some other powerful feature) to abuse the user.

In contrast, we are *not* concerned about privacy leakages between such cross-origin same-site documents. In particular, it is fine if they are able to cooperate in such a way as to communicate data about the user's activities; they can already do that with cookies (even after storage partitioning).

Analysis

Concern: Side effects and opt-in

Referrer A prerendering origin B can trigger various side effects just by loading origin B: e.g. modifying origin-scoped storage or sending fetches with the authority of origin B. To be clear, origin A does not *control* origin B, but it triggers *whatever B normally does on load*, which could be undesirable for B.

Thus, probably origin B needs to opt in with a header (such as Supports-Loading-Mode). If the header is not seen then we must discard the prerender. This makes prerenders more safe than iframes, where referrer A can iframe origin B and origin B must opt-*out* using X-Frame-Options or CSP frame-ancestors.

Alternatives:

- Use an opt-out model, by having the server detect requests with `Sec-Purpose: prefetch;prerender` headers. This is dangerous as cross-origin prerendering is then introducing a new attack which sites must update to avoid.
- Use an opt-out model, but also respect `X-Frame-Options` and CSP frame-ancestors. Basically, if a site is OK being iframed, then we say it is OK being prerendered. (In this case we probably then need to build an extra opt-in for sites which don't want to be iframed, but do want to be prerendered…) This avoids introducing a new attack, but is semantically a bit strange.

Recommendation: an opt-in is best. If we were designing the web today, iframes would be opt-in, and we might one day move iframes to opt-in. Opt-in via Supports-Loading-Mode has always been our plan for cross-origin prerender due to the issues with requiring a storage access upgrade, and most of the reasoning still holds here.

Concern: process isolation

We should ensure that origin B is put in its own isolated process, not the same process as origin A. In particular, if document A and document B have different cross-origin isolation status, we *must* ensure this, to avoid Spectre attacks and maintain web-exposed security invariants.

This is a potential concern because normally, if origin A *iframes* origin B, they are put in the same process (in the same-site cross-origin case we are considering here).

Fortunately, our implementation does not treat prerendered pages like iframes. Instead it treats them like no-opener popups. This gives the exactly desired process isolation properties, so there is no more work to do here. See the full analysis in Appendix: full process isolation analysis.

Concern: activationStart timing leakage

activationStart is a performance timing API introduced with prerendering. Generally, we take great care not to leak timing information across origins. (See, e.g., this discussion.) So one might be concerned that we should censor this in some way for cross-origin same-site prerendering.

However, this is not a real concern. activationStart is just an ergonomic way of recording the time at which the prerenderingchange event happens; it does not contain any extra information that is not already available.

Another way of thinking about this is via the threat model. It is clear the referrer cannot attack the prerendered page by sending it this extra information. And since the prerendered page is not the one initiating the prerender, we cannot characterize receiving activationStart as any kind of attack from the prerendered page on the referrer.

Otherwise: we're good

Because of the basically-nonexistant interface between the referrer document and the prerendered document, all other attacks are cut off.

There's no way a referrer document can reach into the DOM or JS APIs of the prerendered document (not even limited ones like is possible with cross-origin iframes or COOP: restrict-properties windows). There's no way it can read state, even state that often leaks across iframe boundaries like history length or navigation success. The permissions state and grants of the origins are completely disconnected, and in particular the prerendered document remains unable to use any permission-requiring APIs before activation.

(Note that BroadcastChannel and Shared Workers, often used as tricky communications channels, cannot communicate between cross-origin same-site documents.)

Note: referrer policy

The default referrer policy on the web is strict-origin-when-cross-origin. This means that if https://a.example.com/1.html prerenders https://a.example.com/2.html, the full referrer will be sent. But if the same referrer document prerenders https://b.example.com/2.html, only the origin (https://a.example.com/) will be sent, losing the 1.html path.

This is fine and everything should work automatically. It's just worth noting since there will be a different end result between same-site same-origin and same-site cross-origin, and developers might need to compensate. (For example, if they are using the Referer + Sec-Purpose headers on the server and are used to getting full URLs in Referer.)

This does mean we'll need to be sure to implement the spec's list of sufficiently strict speculative navigation referrer policies, since this is the first time they might matter.

Note: redirect handling

To spell out our policy for redirect handling: if a redirect would take us to another network partition key (site), then we cancel the prerender. This is necessary because any information passed to another site is a potential privacy leak.

This includes cases like: Referrer https://a.example.com/ prerenders https://b.example.com/ which redirects to https://other.example/ which redirects back to https://b.example.com/. Even though we end up back on a same-site destination, we must cancel the moment we see the Location: header sending us to https://other.example/, and never send a network request to that destination.

Work items

Opt-in design

The opt-in will be via a new value of the Supports-Loading-Mode header: credentialed-prerender. This value only applies in the same-site cross-origin case:

- In the same-origin case, credentialed prerender does not need an opt-in, so this header is harmless.
- In the cross-site case, credentialed prerender is not allowed, even with this opt in.

We could try for a more explicit name, e.g. credentialed-same-site-prerender, if we think that would avoid confusion.

Redirects complicate things. The total processing model is:

- The initial request must be to a same-site URL.
- If any response, including the initial response, is cross-site, then we must immediately cancel prerendering.
- If the final response is cross-origin same-site, then it must have this header set.

We only check the final response for the header, because checking intermediate redirects does not gain anything; once the request has been issued, any server-side behavior has already happened, and we will not process any client-side behavior for redirect responses. So,

https://example.com/ → https://a.example.com/ without the header → https://b.example.com/ with the header succeeds.

We do require that all URLs be same site; e.g., https://example.com/ → https://example.other/ → https://a.example.com/ fails since there is a cross-site page in the middle.

Specification and explainers

The explainers at WICG/nav-speculation would need updating. In general they are getting fairly stale, so we might need to do some other updates first. But they are currently very clear about the boundary for prerendering being origin, not site. Also the above analysis and discussion should be included there.

The spec is… confusing? It has a concept of "uncredentialed-prerender" which is used for cross-origin navigations, but this is not really used? It doesn't seem to reflect what we're shipping, which is to cancel on any cross-origin navigations… We should first update the spec to match what we're shipping, then expand it to cover the cross-origin same-site case.

We'll need to do work on the opt-in header. It already has an explainer, but it's currently focused on uncredentialed-prerender, whereas our case is credentialed. It doesn't have any spec text yet.

Implementation

See:

- https://chromium-review.googlesource.com/c/chromium/src/+/3849090
- https://chromium-review.googlesource.com/c/chromium/src/+/3858984

Appendix: full process isolation analysis

Background

How strict do we need to be about process-isolating origin A (referrer) from origin B (prerender target)? What about if we hit the process limit? What about on low-memory Android devices that don't even enable site isolation?

The main concern is high-resolution timers via SharedArrayBuffer, and other powerful features, which are currently guarded by cross-origin isolation. **So far in Chromium we have the following invariant: document A and document B can only be in the same process if either (1) neither document has access to COI-requiring powerful features; or (2) both documents have opted in to the risk of being in a process with such a dangerous other origin.**

Today enforcing this invariant is all about iframes and popups. In particular, if the embedder document has access to COI-guarded features, then it must have used the COEP and COOP headers. These enforce that (1) any embedded iframe must opt-in to being embedded by such a powerful embedder, using CORP headers; otherwise that iframe fails to load; (2) any opened popup windows do not have opener access.[1] (1) allows browsers to put the iframes into the same process, if necessary. In particular, this is necessary for browsers like Safari, Android WebView, and low-memory Android Chromium, where site isolation is not available for iframes. (2) enables opened popup windows to be put into a new process since there are no sync access APIs. This is done even on browsers without site isolation.

See Process model for COOP+COEP by Camille Lamy for more.

Implications for prerendering

Prerendering introduces a new relationship where document A could initiate a load of document B. So we need to think about how to maintain this invariant.

In particular, we currently have no enforcement mechanism that says that if document A has access to COI-gated features, then document B must opt in to being prerendered. Similarly, if document B has access to COI-gated features, we have no mechanism to ensure document A is OK with that.

Until now, this was not a problem: document A and document B were always same-origin. But with cross-origin prerendering we need to be concerned.

The solution

Always put into a new process: basically, treat prerendered documents like popups. If document A/B is cross-origin isolated and document B/A is not, document B *must* go into a new process. If both document A and document B are cross-origin isolated, and they are cross-origin, then document B *must* go into a new process.

Thankfully, this is what is already implemented! Prerendered pages get a brand new BrowsingInstance, which the process model treats as a new tab. Those almost always get a new tab, and definitely always get a new tab when needed to maintain the cross origin isolation invariants. (The latter is enforced by ProcessLocks.)

[1] There are slight variants of these also available, such as COEP: credentialless and COOP: restrict-properties. They take somewhat different approaches. COEP: credentialless essentially ensures that the embedee has no valuable data to steal, even if it does get put into the same process. COOP: restrict-properties gives a very restricted opener that can still be put into a separate process.

Backup solutions

These solutions are not necessary in Chromium's implementation. They might be interesting for other browsers or to see the reasoning process.

Require process isolation: if we find out that document B will not be able to be put into a separate process from document A, e.g. due to being on Android WebView or low-memory Android, cancel the prerender immediately. (Note that currently Android WebView and low-memory Android cannot trigger prerendering.)

Conditionally require process isolation: if we find out that document B will not be able to be put into a separate process from document A, *and* we know that either document A or document B will be cross-origin isolated, cancel the prerender immediately. Since cross-origin isolation status is determined by headers, we can know this ahead of time for document A, and upon receiving the headers for document B.

This will generally give superior coverage versus "Require process isolation", at the cost of slight additional implementation work.

Require opt-in: Try to figure out some way of treating prerendered documents like iframes, instead of like popups. This would likely require some kind of header dance, either the existing COEP/CORP one, or some new one that does not use the word "embedder". Then, we could allow web developers to set enough headers so that document B can be put into the same process as document A, despite mismatched cross-origin isolation statuses.

Domenic Denicola does not recommend this. For iframes, the header configuration has turned out to be quite fiddly and difficult to adopt, and the design and implementation work for this would be large. Camille Lamy also advises against this.

Hybrid approach: If we are concerned about excess memory consumption on low-memory devices, do "Always put into a new process" on most devices, but "Conditionally require process isolation" on low-memory devices.

See also

- Process model for COOP+COEP
- Fenced frames process isolation public explainer
- Fenced Frame Process Isolation design doc, especially the "Aligning the process model with the explainer" section
- Prerendering process model

Appendix: summary of prefetching

(This is not really needed to understand the document but it took me a while to figure out so let's keep it in case it's useful.)

Per spec, prefetching is allowed cross-origin, using roughly this algorithm:

- Determine if the to-be-navigated URL is in the same network partition key (same site) as the referrer URL.
- If yes, perform a partitioned prefetch:
    - If any redirect takes us outside of the original network partition key, then cancel the prefetch.
- Otherwise, perform an uncredentialed prefetch:
    - If any partition keys encountered via the response or any intermediate redirects have cookies already, then cancel the prefetch.
    - In particular, if a redirect takes us back to the original network partition key, then cancel the prefetch.
    - Store any cookie headers encountered along the way, so that we can promote them to real cookies upon activation.
- (Also the prefetch might be canceled if requires-anonymous-client-ip is not satisfiable.)