Subresource prefetching+loading via Signed HTTP Exchange

This document is public

horo@, jyasskin@ (last update: May/22/2019)

Spec issue: https://github.com/WICG/webpackage/issues/347
- Signed Exchange subresource substitution
- Signed Exchange alternate link

Crbug: https://crbug.com/935267
TAG review: https://github.com/w3ctag/design-reviews/issues/352
Example: https://github.com/horo-t/sub-sxg
Chrome status: https://www.chromestatus.com/features/5126805474246656
Intent to Implement:
https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/zdvJLcditVA
Design Doc:
https://docs.google.com/document/d/1wsK0pQYSrB_ETYPvdVr3_KjYpdAPf1le4S5LKSU9sQM/edit

Summary

Support signed exchange subresource prefetching and loading by extending the HTTP link header.

Motivation

Signed Exchanges allow a distributor to distribute content signed by a publisher and have the UA show the publisher's URL, give scripts access to the publisher's local storage, etc. Notably, the distributor cannot change anything in the signed content, which means that if they distribute a signed top-level HTML file, subresources keep the publisher's URLs, so they can't take advantage of any signed versions that might be prefetched the distributor's origin. To allow the subresources to be prefetched from the same distributor as the top-level page, the publisher would need to change the subresource URLs in the HTML to point to the distributor's URL, which means they'd need to sign a separate top-level HTML file for each distributor. This seems likely to have the unwanted effect of causing publishers to only sign versions for a few very-centralized distributors.

We want to allow publishers to create a single signed top-level HTML file that allows its subresources to be prefetched from any distributor.

Bundles of signed resources satisfy the same use case, but they don't let the browser use its knowledge of its cache to decide what to request. The Link-based system here also seems useful for CDNs to serve cross-origin resources.

Proposal

Allow the distributor of a prefetched signed exchange (SXG) containing a top-level HTML page to use Link HTTP headers to declare that the SXG's preloaded subresources are available at particular distributor URLs so that the UA can prefetch them from the distributor.

- Use the "alternate" link relation to allow the distributor of a top-level resource to offer a URL for a signed version of one of its subresources. Specifically,

  Link: <https://distributor.example/path/signed_subresource.sxg>;
  rel="alternate";
  type="application/signed-exchange;v=b3";
  anchor="https://publisher.example/original_subresource.image"

  This header appears in the outer HTTP response of the main resource signed exchange to declare the location of the subresource signed exchange URL.
- Introduce a new "allowed-alt-sxg" link relation to declare which subresources a distributor can substitute with their signed version. This header appears in the signed inner response of the main resource. This link relation includes the SHA-256 hash of the header information from the substitutable signed exchange, both to prevent the distributor from encoding messages to the publisher in the selection of substituted resources and to prevent the distributor from selecting a vulnerable mix of subresources. For example:

  Link: <https://publisher.example/image>;
  rel="allowed-alt-sxg";
  header-integrity="sha256-MEUCID..."
- When a prefetched main resource signed exchange has a signed preload link to one of the allowed-alt-sxg subresources, and the outer response declared a replacement for that subresource, the replacement gets prefetched instead of the original.
- When the user clicks a link and navigate to the main resource signed exchange, the UA use the prefetched matching subresource signed exchanges and doesn't fetch the original subresource URL. If the sha256 hash of headerBytes of the subresource signed exchange is not same as the “header-integrity” attribute, the UA will fall back to load the original subresource URL.

Simple example: prefetch a single subresource SXG

1. The user opens https://aggregator.example/index.html.
2. The UA detects a link tag.

<link rel="prefetch" href="https://distributor.example/publisher.example/article_1.html.sxg">

3. The UA prefetches the sxg.
4. The server returns the sxg:
    - In **unsigned** HTTP response from distributor.example:
        - **content-type**: application/signed-exchange
        - **link**: <https://distributor.example/publisher.example/**script.js.sxg**>;rel="**alternate**";**type="application/signed-exchange[;v=...]"**;**anchor**="https://publisher.example/script.js";
    - In **signed** response header of SXG:
        - **link**: <https://publisher.example/script.js>;rel="**allowed-alt-sxg**";header-integrity="sha256-MEUCIA..."
        - **link**: <https://publisher.example/script.js>;rel="**preload**";as="script"
5. When the UA detects the **preload** link header of script.js in the signed response header, the UA finds the matching **allowed-alt-sxg** link header in the signed response header and the matching **alternate** link header in the unsigned HTTP response header. If there is a matching link headers, the UA prefetches the matching sxg (script.js.sxg). So both article_1.html.sxg and script.js.sxg (and certURL of SXGs) will be stored to the cache.
6. When the user clicks the link of article_1.html.sxg, the UA loads the sxg from the cache and detects the **preload** link header of script.js, the matching **allowed-alt-sxg** link header and the matching **alternate** link header. So the UA loads **script.js.sxg** instead of script.js. While loading **script.js.sxg**, the UA checks that the request URL in signed field of the SXG is same as the URL of **script.js**, and that the sha256 hash of headerBytes of the SXG is same as the header-integrity field of **allowed-alt-sxg** link header. If it doesn't match, the UA stop loading **script.js.sxg**, and fall back to load the original script.js instead.

Demo: https://sub-sxg.appspot.com/sxg/amptestnocdn_js_preload.sxg (screenshot)
Error case demo:
https://sub-sxg.appspot.com/sxg/amptestnocdn_js_preload_error.sxg (screenshot)
The header-integrity field of **allowed-alt-sxg** link header of v0.sxg is different from the actual sha256 hash.

Multiple subresource SXG

If there are multiple matching subresource SXGs (example: **script.js.sxg** and **image.jpg.sxg**), the UA must check that there is no error in the all SXGs (eg: sig matching, URL matching, Merkle Integrity error) before processing the content of the SXGs. This is intended to prevent the distributor from encoding a user ID into the set of subresources it prefetches.

This means that the UA can use the subresource SXGs only when they are defined in the header.
To prevent distributors from sending tracking IDs to the publisher's page, Chrome will load **all** the resources from the publisher if **any** integrity check failed.

Demo: https://sub-sxg.appspot.com/sxg/amptestnocdn_js_img_preload.sxg
Error case demo:
https://sub-sxg.appspot.com/sxg/amptestnocdn_js_img_preload_error.sxg (screenshot)
The header-integrity of allowed-alt-sxg link header of **v0.js.sxg** is same as the actual sha256 hash. But the header-integrity of **allowed-alt-sxg** link header of nikko_640_jpg.sxg is different from the actual sha256 hash. In this case, Chrome will fetch both original subresources (v0.js and nikko_640.jpg) to avoid tracking ID injection.

Responsive (multi-source) images

Chrome supports imagesrcset and imagesizes attributes of preload link header to preload responsive (multi-source) images (crbug/813452). To preload SXGs of the appropriate image, the SXG response from the distributor would be like this:

- In **unsigned** HTTP response from distributor.example:
  - **content-type**: application/signed-exchange
  - **link**: <https://distributor.example/publisher.example/**wide.jpg.sxg**>;rel="**alternate**";type="application/signed-exchange[;v=...]";**anchor**="https://publisher.example/**wide**.jpg";
  - **link**: <https://distributor.example/publisher.example/**narrow.jpg.sxg**>;rel="**alternate**";type="application/signed-exchange[;v=...]";**anchor**="https://publisher.example/**narrow**.jpg";
- In **signed** response header of SXG:
  - **link**: <https://publisher.example/**wide**.jpg>;rel="**allowed-alt-sxg**";header-integrity="sha256-MEUCIB..."
  - **link**: <https://publisher.example/**narrow**.jpg>;rel="**allowed-alt-sxg**";header-integrity="sha256-MEUCIC..."
  - **link**: <https://publisher.example/**wide**.jpg>; rel=**preload**; as=image; **imagesrcset**="https://publisher.example/**wide**.jpg 640w, https://publisher.example/**narrow**.jpg 320w"; **imagesizes**="(max-width: 640px) 100vw, 640px"

Demo: https://sub-sxg.appspot.com/sxg/amptestnocdn_js_img_preload.sxg
Note: the proof of concept CL does not have the right matching algorithm yet.

Content negotiation using Variants and Variant-Key

The **alternate** link header and **allowed-alt-sxg** link headers can have variants and variant-key attributes to support content negotiation (eg: WebP support).

- In **unsigned** HTTP response from distributor.example:
  - **content-type**: application/signed-exchange
  - **link**: <https://distributor.example/publisher.example/**image_jpeg.sxg**>;rel="**alternate**";type="application/signed-exchange[;v=...]";**variants-05**="accept;image/jpeg;image/webp";**variant-key-05**="image/**jpeg**";**anchor**="https://publisher.example/image";
  - **link**: <https://distributor.example/publisher.example/**image_webp.sxg**>;rel="**alternate**";type="application/signed-exchange[;v=...]";**variants-05**="accept;image/jpeg;image/webp";**variant-key-05**="image/**webp**";**anchor**="https://publisher.example/image";
- In **signed** response header of SXG:
  - **link**: <https://publisher.example/image>;rel="**allowed-alt-sxg**";**variants-05**="accept;image/jpeg;image/webp";**variant-key-05**="image/**jpeg**";header-integrity="sha256-MEUCID..."
  - **link**: <https://publisher.example/image>;rel="**allowed-alt-sxg**";**variants-05**="accept;image/jpeg;image/webp";**variant-key-05**="image/**webp**";header-integrity="sha256-MEUCIE..."
  - **link**: <https://publisher.example/image>; rel=**preload**; as=image;

If a UA supports WebP, the UA must load **image_webp.sxg** which content is WebP format. Otherwise the UA must load **image_jpeg.sxg** which content is JPEG format.

Demo:

> https://sub-sxg.appspot.com/sxg/amptestnocdn_js_img_vary_preload.sxg (screenshot)
> "nikko_640.jpg" is served in WebP format which is more size efficient than JPEG format.

Why we can't use the SRI's integrity instead of signature?

This is because SRI's integrity can be used only for verifying the integrity of the content body. So if the UA use the SRI's integrity value in **allowed-alt-sxg** link header, we can use the subresource SXGs to track the users by changing content-type and detecting the image loading failure.

SXG's header-integrity can verify the integrity of both the response header and the content body, because SXG's header must have digest header.

Can't we merge allowed-alt-sxg to preload header?

It becomes complicated when supporting the imagesrcset attribute.

Subsetting the list of subresources

If there is a missing rel="alternate" link outer header, UA should ignore the all subresource signed exchanges to avoid user tracking. So if a distributor want the user to prefetch the subset of sxg subresources, the publisher must generate multiple main signed exchange. (For example: article.html.no-subsxg.sxg, article.html.js-only-subsxg.sxg, article.html.js-and-img-subsxg.sxg)

Origin of the main and subresources SXGs

To limit the complexity, we are restricting the main SXG and the subresources SXGs to be served from the same origin. If we allow it, we need to consider the cross origin information leak issues.

- Spec discussion: https://github.com/WICG/webpackage/issues/347#issuecomment-455494121