Fenced frames
Local unpartitioned data access
One year update

TPAC 2024

Shivani Sharma, Andrew Verge (Google Privacy Sandbox)

9/25/2024

Topics and Goals

- Provide a one-year update on the unpartitioned data access functionality in fenced frames
- Goals: Communicate the progress and demo the functionality

Notes will be taken here: https://pad.w3.org/p/FencedFramesBreakoutNotes

Please sign yourself in the notes, add in the queue heading for questions

Github session issue: https://github.com/w3c/tpac2024-breakouts/issues/40

Meeting details: https://www.w3.org/events/meetings/26d68797-a5ab-45f4-b8fa-d0b3fa5a50f5/

Overview of progress

- Last TPAC's presentation talked about the goals and high level design
- Followed by publishing the explainer in Q4 2023
- Intent to prototype on blink-dev
- Implementation behind a disabled flag close to completion
- We will be presenting a demo in today's session

Fenced frames: Problem & Vision

- **Problem:**
  - ↳ In a web that has its cookies and storage partitioned by top-frame site, there are occasions when it would be useful to display content from different partitions in the same page.
  - ↳ This can only be allowed if the documents that contain data from different partitions are isolated from each other such that they're visually composed on the page, but unable to communicate with each other, arbitrarily.
  - ↳ Iframes do not suit this purpose since they have many communication channels with their embedding frame (e.g., postMessage, programmatic focus etc.).

Fenced frames: Problem & Vision

- **Vision:**
  - ↳ A new HTML element, fenced frames, that allows embedding documents on a page, that explicitly prevents arbitrary communication between the embedder and the frame.

Fenced frames and privacy information flows

- **Fenced frames' privacy information flows are determined by the consumer APIs'/ use cases privacy flows**
  - ↳ They provide an isolated context which has limited/controlled ways to communicate with the embedding context,
    - → E.g., no postMessage
    - → Unique, ephemeral partitioned storage, cookies
    - → window.top points to the FF root and not the primary top-level page, etc.
    - → Explainer goes into them in detail
  - ↳ Additionally, fenced frames provide the primitives, e.g. to disallow network access, opaque source URL, and the consumer APIs determine their privacy information flow and which primitives best suit their privacy guarantees.

Fenced frames and privacy information flows

- FencedFrameConfig provides a way to make certain properties of the document loaded in a FF, opaque to the embedding context
  - ↳ Notably, the source URL of the document, e.g. return value of runAdAuction or selectURL

- FencedFrameConfig could also be created by the embedding context providing the source URL, in which case the source is not opaque
  - ↳ Example use case: personalized payment button

Fenced frames and privacy information flows

- Do not allow unrestricted network access if the FF has access to both embedding page's data as well as cross-site data,
  - ↳ Such a use case needs to invoke `window.fence.disableUntrustedNetwork()` primitive before accessing cross-site data

End-to-end flow for unpartitioned access use case: personalized payment buttons

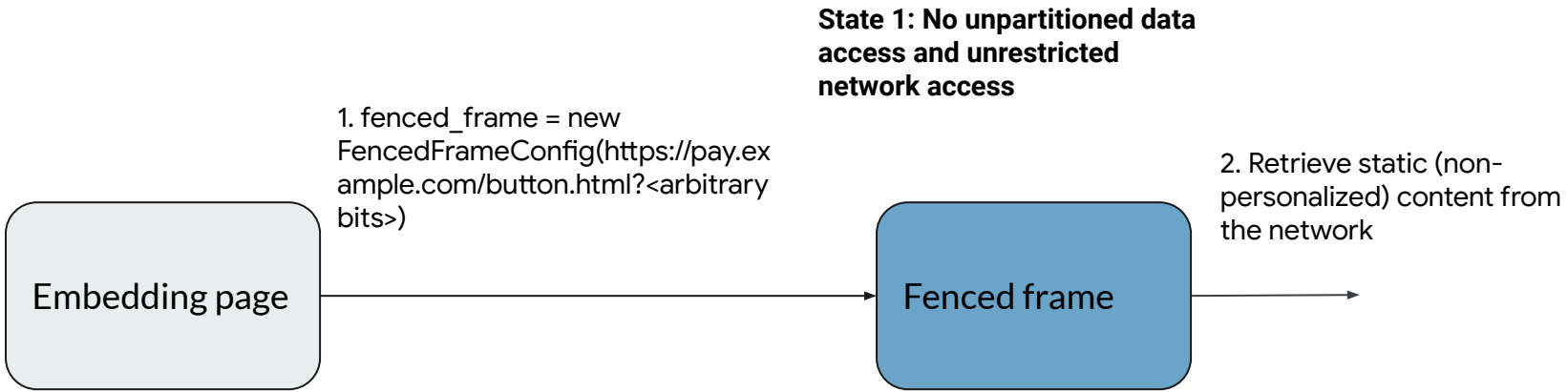
- Guard: Note that this will be gated behind a UX setting which the user can disable this functionality with.
- Same as today: User visits the payment provider's site as a first party and enter their payment details
  - ↳ Unrelated to the personalized button
- New: The payment provider decides what all is needed to render in the personalized button and writes it to unpartitioned storage (Shared Storage)

  window.sharedStorage.set("last-4-digits", value)

End-to-end flow

- Same as today: User visits the merchant's site and payment provider's script decides to create a button
  - ↳ This script runs in the merchant's page
- New: The payment provider script creates a fenced frame instead of an iframe

End-to-end flow

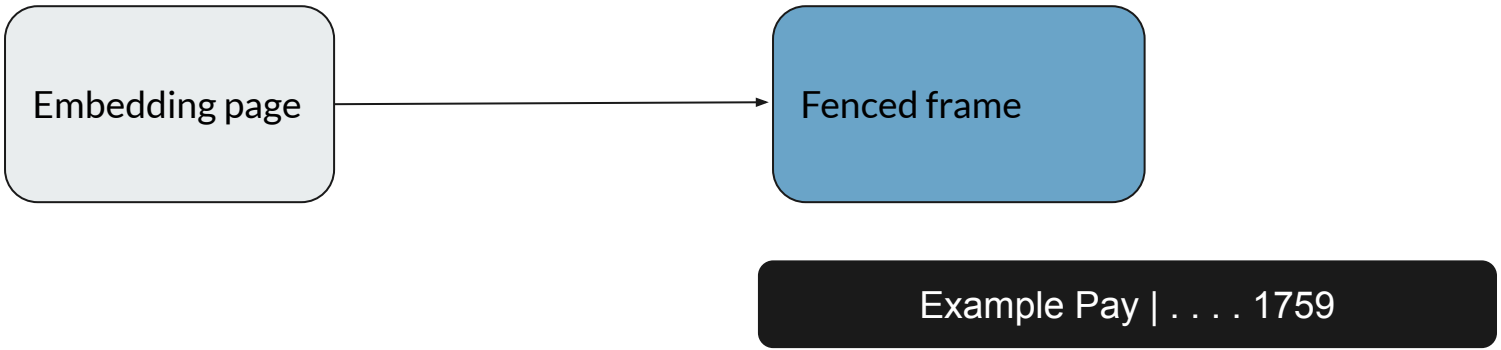


End-to-end flow

3. window.fence.disableUntrustedNetwork()

State 2: Network access revoked, allow unpartitioned data access (via Shared Storage)

4. sharedStorage.get(“last-4-digits”)
Display the data



End-to-end flow

- Same as today: User clicks on the button
  - New: Click listener on the payment provider’s script listens for the click
  - It opens up a PaymentHandler context or a top-level page and the transaction proceeds

New API surfaces

- Constructor FencedFrameConfig(src url)
- This is a scenario where the src url is not opaque to the embedding context
- The src url could contain any arbitrary bits
  - ↳ That's ok for privacy because the FF either does not have cross-site data to join it with (state 1)
  - ↳ It can be joined with cross-site data in state 2, but at that point there is no exfiltration (network is revoked)

window.fence.disableUntrustedNetwork()

- Fenced frame can voluntarily give up its network access, which includes revoking
  - ↳ Subresource fetches
  - ↳ Initiating any navigations, including for top-level context or new tabs etc.
  - ↳ Pre*: Prefetch, preload …
  - ↳ Alternate network APIs like WebSockets, Direct Sockets, and WebRTC etc.
  - ↳ Fetches from workers
  - ↳ Others…

window.fence.disableUntrustedNetwork()

- Chromium implementation makes use of the Network Isolation Key's nonce
  - ↳ Already uniquely identifies all network requests from a given fenced frame tree
  - ↳ The original purpose of the nonce was to create a ephemeral and unique network partition
  - ↳ Being reused now for network revocation as well
- Requests that don't carry that nonce e.g. top-level navigations are blocked separately

window.fence.disableUntrustedNetwork()

- Nested iframes
  - ↳ Any network is also revoked in all the nested iframes
- Nested fenced frames
  - ↳ The call will not resolve until all nested fenced frames have also revoked their network by invoking window.fence.disableUntrustedNetwork()
- In progress requests
  - ↳ All in-progress network requests would be cancelled (except the top-level navigation that have already been initiated)

window.fence.disableUntrustedNetwork()

- Why “untrusted”
  - ↳ Some forms of network that cannot exfiltrate cross-site data can be allowed, e.g.
  - ↳ Private Aggregation API which allows aggregated data to be sent out on the network as that inherently disallows any arbitrary data exfiltration.

window.sharedStorage.get()

- Why Shared Storage
  - ↳ API simplicity: no new API, existing get() will need to be exposed in a new context
  - ↳ Shared Storage is, by definition, unpartitioned data, vs cookies/local storage
  - ↳ Javascript only, vs. cookies which also go with network requests
  - ↳ Defined in terms of output gates in addition to origin vs. cookies/local storage that are defined in terms of only hostname/origin

window.sharedStorage.get()

- Guards:
  - ↳ disableUntrustedNetwork() successfully resolved
  - ↳ Permissions Policy
  - ↳ User has not disabled the feature via UX
  - ↳ Privacy Sandbox's attestation

window.fence.notifyEvent(Event e)

- Called by JS running in the fenced frame, ideally within an event listener for `e`.
- The argument `e` must:
  - Be a DOM Event object
  - have `isTrusted = true`
  - have `eventPhase != NONE` (currently dispatching)
  - Currently, have `type = “click”`

- When called with a valid event object, a corresponding event w/ type `fencedtreeclick` fires in the embedding document.
  - `fencedtreeclick` event objects contain no contextual info like mouse coordinates or timestamps

- The fenced frame must have transient activation, which will be consumed and applied to the embedder instead.
  - Allows embedder to open new windows, use the Payment Request API, etc.

Sample code

- Consider ecommerce site `fancystore.com`, and payments provider `examplepay.com`
- First, the user visits `examplepay.com` in a first-party context to register their card info:

```
async function registerCard() {
  // Prepare HTTP request with user-provided card infomation.
  let request = createCardRegistrationRequest({number: 'XXXX XXXX XXXX 1234',
    expDate: 'MM/YY', ...});

  // Register the card information with examplepay.com.
  let response = await fetch(request);

  // If the card was registered successfully, write the last 4 digits of the
  // card number to Shared Storage for origin "examplepay.com." The data to
  // write could come from the response body, a 1p cookie in a response header,
  // or somewhere else.
  if (response.status === 200) {
    let body = await response.json()
    await window.sharedStorage.set('last4', body.last4);
    console.assert(body.last4 === '1234');
  }
}
```

Sample code cont.

- Then, the user visits `fancystore.com` to buy something.
  - At checkout, the `examplepay.com` API loads the payment button in a fenced frame.

```
let example_pay_button = examplePayAPI.createButton();
document.body.appendChild(example_pay_button);
```

- In `examplePayAPI.createButton()`:

```
function createButton() {
  let fenced_frame = document.createElement('fencedframe');
  // Create a fenced frame config using a URL directly instead of a config-generating
  // API like Protected Audience or sharedStorage.selectURL(). Note that the URL is
  // same-origin to the site where the card was first registered.
  fenced_frame.config = new FencedFrameConfig('https://examplepay.com/make_button');

  // Registering a "fencedtreeclick" event handler on the fenced frame element allows
  // it to respond to a "click" event that fires inside the frame's content.
  fenced_frame.addEventListener('fencedtreeclick', () => {
    startPaymentFlow();
  });
  return fenced_frame;
}
```

Sample code cont.

- Finally, inside the `examplepay.com/make_button` fenced frame:

```
function personalizeButton () {
  // By waiting for the page to finish loading, we can ensure that there's
  // no additional JS waiting to execute before revoking network.
  window.onload = async () => {
    // First, disable untrusted network access in the fenced frame.
    await window.fence.disableUntrustedNetwork();

    // Read the last four digits of the card from Shared Storage
    // and render them in a button.
    b = document.createElement('button');
    b.textContent = await window.sharedStorage.get('last4');

    // Tell the embedder that the button was clicked, so that the payment flow can be
    // initiated. This will fire a "fencedtreeclick" event at the fenced frame element
    // in the embedder, which we previously registered a handler for.
    b.addEventListener('click', (e) => {
      window.fence.notifyEvent(e);
    });

    document.body.appendChild(b);
  }
}
```

Demo

- Payments provider: https://demo-payments-provider.glitch.me
- Merchant: https://demo-merchant.glitch.me

- To try for yourself, run Chromium/Chrome with the following flags:
  - `--enable-features=FencedFramesDefaultMode,FencedFramesLocalUnpartitionedDataAccess`
  - `--disable-features=EnforcePrivacySandboxAttestations`
- Works on Chrome Canary and Beta

Privacy Considerations

- Clicks as a communication vector from fenced frame -> embedder.

- Scenarios
  - A single click on one frame: 1 bit
  - Multiple clicks on one frame: >1 bit
  - Loading *n* coordinating fenced frames, but only clicking on one: Log2(n) bits
  - Clicks across different browsing contexts over time

- Mitigation: Rate Limiting
  - Only allow *n* related (same-origin/same-site/?) frames per page load
  - Discussion ongoing as to specific details

Thank you!

Questions?