Intersection observer in fenced frames

shivanisha@chromium.org
Status: Complete
Visibility: Public
Aug 2022

References

Turtledove github issue: Posted here to continue the discussion with external folks on https://github.com/WICG/turtledove/issues/264

Introduction

This document aims to go into the problem and solution space for intersection observer API inside fenced frames.

The problem here is the communication side channel that can open up between the embedding page and the fenced frame via the intersection observer events. The embedding page could move the fenced frame programmatically in such a way that the bits transferred via the IO events can be used to transfer user identifying information from the embedding page to the fenced frame. Since the fenced frame has unrestricted navigation capability on user activation, this information can then be exfiltrated out. In the short term it can also be exfiltrated via unrestricted network access and event level reporting but those channels would be removed in the long term.

There are 2 parts of the solution:

1. **Intersection observer events flow:** This refers to the information flow of the events as they happen in real time. The various solutions that have been thought of require the events to either flow to the fenced frame (see API gates section below) or to a standalone worklet/storage that can be later read by a worklet.
2. **Reporting:** This refers to the reporting of this information to a server. The ads use cases using fenced frames, like FLEDGE, currently support event-level reporting but it will be replaced with aggregate reporting in the future. There have been discussions about whether we want to have an intersection observer solution for the temporary event-level reporting phase or not and the decision there is if the solution does not lend itself to be used by both event-level and aggregate reporting approaches seamlessly then we only create a new solution for long-term aggregate reporting.

Note that currently intersection observer events are available inside fenced frames as they would in an iframe. This is done in the short-term so that dependent use cases are not broken.

Solution space

This section goes into the various solution spaces in no particular order. The respective sections discuss their feasibility and trade-offs in detail.

Gating the Intersection Observer events

This approach includes keeping the intersection observer registration and events flow the same as it happens in iframes but additionally ensuring that there are heuristics in place such that programmatically moving fenced frames on the screen cannot be used as a communication channel from the embedder.

Some of the heuristics can be:

1. Only allow user scrolling/user activation gated intersection observer events. (from Intersection Observer owners: this is feasible implementation-wise).
    a. This is not sufficient since there are valid programmatically initiated events as well such as when the fenced frame goes out of viewport as more content gets added to the page. Can these be rate-limited to only be fired if the %-visible moves by 'x'?
2. Send updates during enter/exit transitions. Perhaps by sending during initial resize, deletion, and the first two CSS animations (from visible to invisible and vice versa). **con:** this would miss all updates due to reflow/relayout in the embedding document.
3. Send updates only after layout quiescence. I.e., stable layout after <X>ms.
4. Any more heuristics to be added?

Pros

- The information flow does not change from what it is currently. The intersection observer events still flow to the fenced frame as they would in an iframe so it is low effort for the ad tech vendors. This makes the solution more amenable for viewability vendors.
- Aligns with the fact that there may be multiple third parties inside an ad creative that are independently processing the intersection observer events, e.g. for independent verification.
- Lends itself seamlessly to whatever reporting is currently being used (event level or aggregate)

Cons

- The heuristics are based on the underlying principle that to be used as a communication channel, the embedder needs to do user visible actions and that will be a disincentive for sites to some extent. But there can be workarounds e.g. having a small enough fenced frame that it doesn't lead to a user visible effect, having a transparent FF (opacity:0) etc. which makes it harder to guarantee the underlying principle.
- Requires careful consideration of heuristics that they are sufficient for all use cases of ads e.g. billing, ad-spam detection etc. Since this approach requires some IO events to not be fired, it loses out on accuracy that IO guarantees.

- There could be legitimate solutions where a publisher would auto scroll for the user to different parts of the page. In that scenario it would be hard to differentiate that from a publisher trying to encode data maliciously.
- Another consideration would be to evaluate if this impacts the MRC (Media Rating Council) accreditation. The rules in the accreditation do not gate on user activation so they will likely be impacted by this approach.

Summary of feasibility
Given the various cons, especially the fact that it will be a solution where it is hard to guarantee privacy as well as hard to guarantee accuracy, it seems to be a non-starter.

Browser API based solution

[Update 4/17/2024]: This was actually asked as a feature request in https://github.com/WICG/turtledove/issues/826
This refers to a solution where there is a new API that a script in the fenced frame can invoke e.g. `MRCViewabilityEventsRegister` (name TBD) that only fires an event when the viewability aligns with MRC's definition of viewability. The information would be simple and granular enough to not be able to be used as a communication channel. MRC (Media Rating Council) accreditation defines it as >=50% visibility for 1 sec (2 sec for video ads). The number of events will also need to be gated and the time of the event might need to be randomly fuzzed.

Pros
- Simple to implement and guarantee an information flow between the embedder and the fenced frame which is private.

Cons
- Only caters to a subset of use cases of intersection observer and only to ads based use cases.
- The time at which an event gets fired can be intentionally chosen by the embedder to communicate arbitrary bits of information via the timestamp. The time at which the IO event gets triggered will therefore need to be randomly fuzzed.
- It will still require gating the maximum number of events and the open question is how many events are sufficient to satisfy the use cases. Without a maximum, even with the limited rate of information flow (a max of 1 event per second) a unique ID could be easily transferred to a fenced frame (e.g. 32 seconds for a 32 bit ID).
- Moves the ownership of the standard's implementation to the browser and puts it on the browser to update it as the standard evolves.
- Might be an issue with non-MRC accredited vendors and will require all of ad-tech to move to this standard definition of viewability.

Summary of feasibility

This is not a generic solution or an alternative to intersection observer which makes it hard to gauge its feasibility but I am leaning towards this being a non-starter as well given that it doesn't provide all the data that intersection observer would. It requires more inputs from ad-tech to determine if it's good enough for an initial solution. It definitely doesn't give all the flexibility or covers all of the use cases so I can see ad-tech not being in favor of it but the alternative of a worklet based solution(see below) will require more changes in their workflow.

Aggregate report based solution

This refers to a solution space where ad-tech gets to run its script in a worklet environment and process events locally. It requires a persistence mechanism where the browser could store the events and those could then be later processed by an aggregate reporting worklet. This could be accomplished using a shared storage worklet because of the accompanying storage access or piggyback off of the aggregate attribution reporting. It would require the following steps:

1. Multiple parties within the ad creative register the events they are interested in along with their origin so that the browser can create the worklets of the respective origins later for aggregate reporting.
2. On those events being fired, the browser will save them to shared storage or another aggregate reporting storage.
3. At some point, during the aggregate reporting timeline (approx. 24 hrs), the browser creates those worklets and they can read the events from shared storage to process them and send aggregate reports.

Pros

- Since the worklet is not required to always be alive for the duration of the ad since shared storage gates the output to only be aggregated and that doesn't need real-time processing, it is not a big concern for performance as opposed to worklets that are not backed by a storage mechanism. It still requires consideration for when the worklets should be created to generate the aggregate reports.

Cons

- A new information flow for intersection observer API that requires multiple origin worklets to be created.
- Any IO use cases that rely on real-time IO events to make decisions e.g. scripted animations will break in fenced frames. Another use case that will break is using IntersectionObserver V2 for clickjack protection (IntersectionObserver V2 features are currently used on ~4% of all page loads).
- Much more implementation intensive than the above approach as this will require new sets of worklets to be created, considerations on when they should be created and information to be flowing from the browser process to the worklet process in addition to the fenced frame's renderer process. There will be some interesting implementation considerations since intersection observer events require both the embedder's as well as

the frame's renderer processes to create the resultant event, so that will imply an additional hop for that resultant event to be dispatched to the worklet process.

Summary of feasibility
Looks feasible, is design/implementation-intensive but provides privacy and accuracy guarantees.

Aggregate report based solution - no additional worklets

This is a variant of the solution defined above in which the vendor does not get a chance to run a worklet locally and it's the browser that stores the events and sends an aggregated report.
It would require the following steps:

1. Multiple parties within the ad creative register the events they are interested in along with their reporting destination origin.
2. On those events being fired, the browser will save them to aggregate reporting storage.
3. At some point, during the aggregate reporting timeline (approx. 24 hrs), the browser reads the events from the storage and sends aggregate reports.

Pros

- No extra worklets needed to process the intersection observer events, implying less complexity in the browser's implementation as well as developer's workflows.

Cons

- Similar to above, any IO use cases that rely on real-time IO events to make decisions will break in fenced frames.

Summary of feasibility
Overall feasible and provides privacy and accuracy guarantees. But see below for the hybrid approach.

Registration based solution

This requires the FLEDGE worklet to register upfront in the events it is interested in reporting and the browser will be responsible to match the IO events with the ones registered and send reports based on those.

Pros

- Does not require a worklet to be running after the initial registration is done.

Cons

- Requires the browser to process the events and see if they match the ones registered by the worklet. Also requires the browser to process those events, per the specifications, so the events that would be sent to the ad tech provider would only be the ones that they are interested in.
- the declarative syntax is very complex (e.g. AMP)

Summary of feasibility
Feasible in the aggregate reporting scenario. Would be open to hearing from more ad-tech partners.

Hybrid of aggregate and IO events based solution

This approach requires the browser to send some IO events that would be helpful to detect clickjacking in addition to the aggregate reporting approach. Specifically, this will require the browser to fire the IO events when a click happens which date back to 'n' milliseconds before the click. Additionally, it will save the rest of the events (non-user activation triggered) in an aggregate storage which can then be reported via an aggregate report (the latter part is the same as this section).

Pros

- No extra worklets needed to process the intersection observer events, implying less complexity in the browser's implementation as well as developer's workflows.
- Allows clickjacking to be detected in real-time.

Cons

- Other real-time use cases like scripted animations still won't be solved but maybe that's ok to lose that functionality in fenced frames.

Summary of feasibility
Overall feasible and provides privacy and accuracy guarantees.