User Activation v2: Browser-side states for OOPIFs

Incremental plan to move current UAv2 implementation to browser side

mustaq@chromium.org, alexmos@chromium.org, dcheng@chromium.org
Implementation bug: crbug.com/780556 (a part of crbug.com/696617)
May 3 2018

Background

This document gives an implementation plan for adding OOPIF support to the core User Activation v2 design.

This document ***replaces*** our previous plan w/OOPIFs for three reasons:

- The possibility of a compromised renderer was not considered in the previous design. There have been two proposals to fix v1 code (based on UserGestureTokens) for this problems. With v2 core implementation mostly done, investing in v1 seemed like a waste of effort.
- The previous design couldn't resolve a race condition around two consecutive popup requests from different renderers. Browser-side activation tracking seems to be the only feasible solution.
- We had plans to move UAv2 activation detection to browser side in future anyways.

Activation state replicated across browser and renderers

We will replicate the activation bits across all "copies" of a frame, across the browser process and all renderers. In other words, when the info changes in any particular frame, all "copies" of the frame (i.e. the `FrameTreeNode` in the browser and all `LocalFrame`/`RemoteFrame` objects in renderers) will be updated. All "write" operations on the bits would initiate at the browser, i.e., detection of activation and consumption of activation. After a user activation is detected, all ancestor frames of the frame

where the activation occurred (in all copies of frame tree) will have the bits set. When a frame consumes activation, all nodes (in all copies of the frame tree) will have the transient bits reset.

One implicit state update that is never propagated across frames is the expiration of the transient activation bit after a constant time (currently 10sec). Each copy of the bit will expire “locally”.

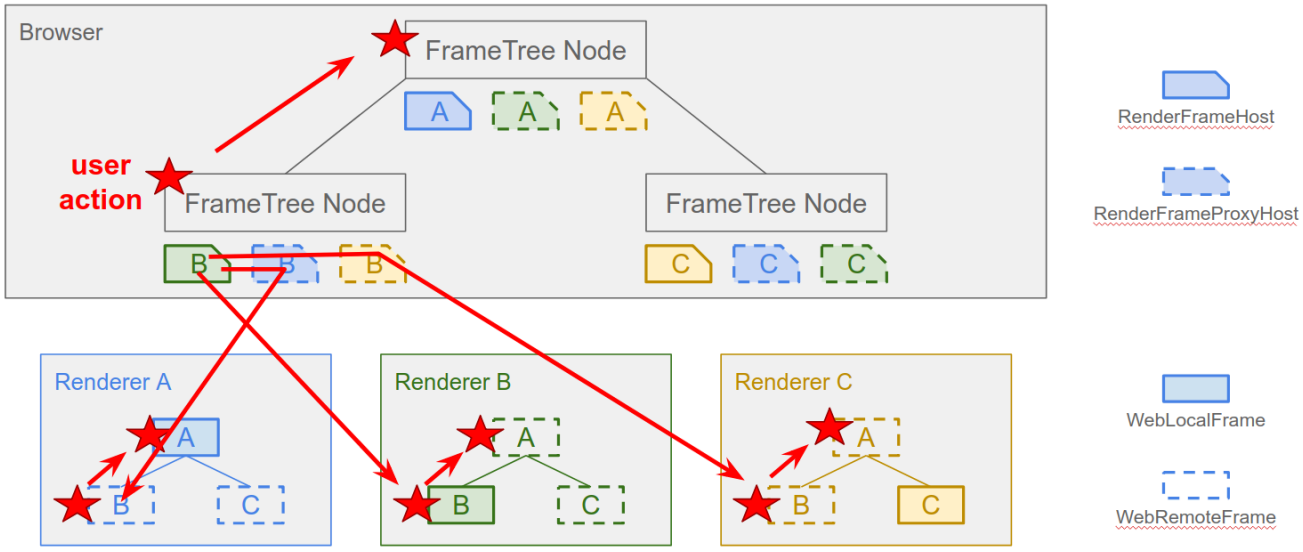

Figure 1: Replicating activation state across browser/renderers: no “write” would start at a renderer.

Proposed plan

The end result would have the browser-side states as ground truth and renderer-side states as cached values. We would reach this point in an incremental manner because currently UAv2 code is kind-of mixed up with v1 code, and we want to keep v1 behavior as is to avoid fighting unnecessary regressions. We will achieve this in two main steps:

A. Move all consumption calls to browser side, after adding UAv2 states the to browser side frame tree. This would affect both v1 & v2 popups but we would try to keep v1 behavior unchanged.
B. Rewrite user activation detection calls on browser side only for v2, after splitting v1/v2 notification calls. This would affect only v2.

We will leave all calls to check current user activation as is except for popups.

Security Considerations

The underlying model change through UAv2 (i.e. the core proposal here) could possibly allow/disallow popups in the rare case that a frame (the main frame or an iframe) could be trying to open a popup right after (within a sec) the user interacted with a parent/child/sibling iframe. To elaborate:

- In the old (current) model, Chrome allows popups:
    - (before site-isolation) if the user interacted with any frame, or

    - (after site-isolation) if the user interacted with any same-process renderer frame.

  Note that these checks constitute the first step in popup blocking. Chrome's popup blocker mechanism kicks in after any of these checks.
- In the new model, Chrome defines the scope of activation by (frame) containment, so Chrome will now allow popups from a frame if the user interacted with the frame itself or any contained frame (subframe). After this check, Chrome's popup blocker mechanism kicks in just like today (and this is independent from UAv2).

UAv2 is not a concern from security perspective because the browser-side user activation states proposed in this document guarantees that there can never be more than one popup from a single user activation notification on the renderer side. This is because the browser consumes the activation right before a popup window is created. In fact, this solution was suggested by the Site Isolation team in response to a timing analysis we attempted in an earlier draft.

(Note that neither Chrome's existing user activation model nor UAv2 can guard against false user activation notifications from an untrusted renderer. We plan to fix this through a separate project.)

Challenges/concerns

1. [v1 only] On mouse drag across a cross-process frame boundary, v1 tokens would be split effectively allowing two popups.
    - Not a major problem from input perspective. It's limited to two popups, and arguably `mousedown` and `mouseup` are two different user action.
    - We don't know any site-isolation bug around it.
2. [v2 only] Isolation of checking (in renderer) vs consumption (in the browser) of v2 states for popups.
    - Activation state is checked in renderer multiple times before consuming. [2018-05-03] `window.open()` in JS triggers 3 calls to `HasTransientUserActivation()` before consuming.
        - Call#1 at `create_window.cc::CreateWindow` is used in FrameLoadRequest.
        - Call#2 at `create_window.cc::CreateWindow` is used in "probing" for devtools?
        - Call#3 `RenverViewImpl::CreateWindow` sets a `param` for `FHM_ShowCreatedWindow` IPC. Ultimately used (through `WebContentsDelegate`) in two places: `MultiUserWindowManagerChromeOS::SetWindowOwner`, and `browser_navigator.cc::Navigate`.
    - Stale states in renderer may possibly screw up Blink bookkeeping and UMA.
    - One solution is to nuke the renderer side state caches and instead make the activation checks at renderers blocking-IPCs to the browser. However, such blocking calls are discouraged.

3. [v1+v2] Recently video picture-in-picture has started consuming from the renderer side.
    - Not sure if the browser knows about it. In any case, we need a new blocking call to consume on browser side.
4. [v2 only] With ~30 APIs relying on activation, any change in v2 code is expected to cause new test failures.
5. [v2 only] Current activation detection logic is designed around v1 token passing, many of the calls are not needed in v2. We have to check each of them carefully.
6. [v2 only] Current activation detection logic relies somewhat on “event states” that are not exposed to the browser process. It will be tricky to move these calls to the browser. Here are some cases like these:
    - `mousemove` is not activation, but drag is!
    - Special keyboard events: while autoscrolling, modifier-only input, browser hotkeys, cancelable hotkeys.
    - IME events.
    - Web_plugin_container_impl script execution.
    - WebLocalFrameImpl::ExecuteCommand.
    - Accessibility.
    - Thread debugger.
7. [v2 only] Event targeting at the browser doesn’t necessarily have frame-level granularity. It knows only about the receiving widget. When a widget contains multiple (same-origin) frames, the browser process need to find the frame to correctly update the frame-node states.