Feature Policy Integration for Clipboard API Explainer

Authors:

- huangdarwin@chromium.org

Participate

- https://crbug.com/1074489

Introduction

The Clipboard API provides an API surface that supersedes the legacy document.execCommand(‘copy’) API, by being more performant, explicit, and safe. Part of securing this API is the implementation of feature policy integration, which through an oversight is currently not working correctly. We therefore aim to fix use of this API in iframes, by fully implementing feature policy integration.

Goals

Implement feature policy (aka permission policy) flags for the Clipboard API. The flag names will be `‘clipboard-read’` and `‘clipboard-write’`, to be consistent with the Clipboard API permissions, as feature policy is now merged with permissions and renamed to permissions policy. This will improve security for the Clipboard API, by disallowing access on unapproved iframes. The Clipboard API is an existing API and will be changed only by the addition of a feature policy restriction.

Our ultimate goal here is to re-allow use of the Clipboard API in iframes, while preserving the protections of feature policy restrictions in iframes.

Non-goals

Discuss other APIs or permissions at length, though we may cover a bit of this regardless for context in this explainer.

Feature Policy “clipboard-read”

The feature policy “clipboard-read” will be implemented, and will control use of APIs controlled by the corresponding “clipboard-read” permission. This includes the Clipboard API’s `navigator.clipboard.read()` and `navigator.clipboard.readText()` functions.

“clipboard-read” may be used per the feature policy specification. Namely, enabling this feature may be done using:

- an HTTP Header Field, like below:

```
Feature-Policy: clipboard-read *
```

- or an iframe allow attribute, like below.

```
<iframe allow="clipboard-read">
```

If the Feature Policy is not enabled in an iframe, then use of these controlled features will be disallowed, and lead to the returned promise being rejected, as shown here.

Feature Policy “clipboard-write”

The feature policy “clipboard-write” will be implemented, and will control use of APIs controlled by the corresponding “clipboard-write” permission. This includes the Clipboard API’s `navigator.clipboard.write()` and `navigator.clipboard.writeText()` functions.

“clipboard-write” may be used per the feature policy specification. Namely, enabling this feature may be done using:

- an HTTP Header Field, like below:

```
Feature-Policy: clipboard-write *
```

- or an iframe allow attribute, like below.

```
<iframe allow="clipboard-write">
```

If the Feature Policy is not enabled in an iframe, then use of these controlled features will be disallowed, and lead to the returned promise being rejected, as shown here.

Detailed history/design discussion

Specification History

Feature Policies names should match their corresponding Permission names. While the original “clipboard-write” and “clipboard-read” permissions were added to the specification and implemented in Chrome long ago (in 2017), they seem to be under debate again now. Here’s some history...

1. 2017: TAG Review + Clipboard Spec Issue + intent to ship discussing permission. Edge, Firefox, and Chrome representatives were involved. Spec change made, with corresponding implementation, of clipboard-read/clipboard-write. The TR spec still has this same language now.
2. Feb 2020: Safari notes they’re not interested in these permissions and Firefox notes they’re unsure, well after the original review and implementation. Unshipping clipboard permissions is not feasible for Chrome, as it will break sites. (bikeshedding after shipping?)
3. Feature Policy and Permissions names merged, with Feature Policy being renamed to Permissions Policy. Therefore, any new feature policy names, including this feature policy, should match the relevant permissions names.
4. Updated Permissions spec permission name to clipboard-read/clipboard-write, to match the Clipboard Spec, but then some pushback post-merge, as we realized there may not be a second interested implementor. (We were unfortunately not aware of the lack of interest from other implementers until after this merged, so the ordering between 2 and this does look odd here)
5. Adding Feature Policy to Clipboard Spec. This is the specification change we’d like here, but is blocked on disagreements regarding permission name, which the feature policy should match.

Broken Feature

Feature policy for the Clipboard API was previously implemented to improve security, after a plan cross-cutting security, privacy, and feature teams, for various clipboard APIs. This integration was unfortunately incomplete, so that the Clipboard API is currently always disallowed and broken on iframes. The issue was only discovered after the change reached M81 stable, and the change was a security fix, so we didn’t revert this change. Completing this feature policy integration will therefore also fix use of the Clipboard API in iframes.

Other Clipboard APIs

While document.execCommand can also copy and paste, this API is not currently covered by the “clipboard-read” and “clipboard-write” permissions, so it will not be restricted by the relevant feature policy. We do plan to do this in a future change though, per the Clipboard API Restrictions proposed between Security, Privacy, and Feature teams.

Considered alternatives

The ultimate goal is to fix sites who wish to allow use of the Clipboard API in iframes. As there’s currently some pushback on the name of these feature policies, there are two other potential courses of action.

Remove feature policy restrictions on the Clipboard API

This would fix sites, and allow their iframes to use the Clipboard API. However, this will likely not be acceptable for protecting our users’ security, as the feature should not be accessible to all iframes without delegation by top-level frames.

Do nothing

Not really an alternative, but currently all sites wishing to use this feature in iframes are currently broken. Doing nothing is almost certainly the worst option, but could circumvent both security concerns and spec disagreements.

Stakeholder Feedback / Opposition

- Gecko: Defer, as Firefox would prefer to use one unified permission
- Webkit: Negative, as blocked on disagreements with the matching permission name.
- Web Developers: Strongly positive, as they’d like to be able to use this feature again. Publicly interested parties include bwater.com, hearthsim, and

code-server, though there are many other interested parties who have privately notified us about this issue.

References & acknowledgements

Many thanks for valuable feedback and advice from:

- mek@chromium.org
- slightlyoff@chromium.org