Improving BFCache Hit-rate

TPAC 2022
fergal@chromium.org
kbx@chromium.org
Contribs from many others
Biggest blockers

- What % of history navigations are directly blocked?
- Chrome telemetry tells us
  - Android
    - Cache-Control: no-store 16.5%
  - Desktop
    - Cache-Control: no-store 7%
    - Unload handlers: 14%
- Numbers are much higher if we consider history navigations blocked by more than 1 thing.
- Most other things are
  - below 1%
  - blocked for Chromium implementation reasons
Not Restored Reasons API
Not Restored Reasons API

- Not the topic of this talk
  - of interest to this audience
  - fairly clear path
- **Explainer**
- Debug your cache-misses in the wild
- Similar to Chrome's devtools but with cross-site privacy constraints
- When a history navigation occurs and the page is not restored from BFCache, JS can access a tree of reasons (based the frame-tree of the page that was not restored) telling why it was not restored
Unload handlers
Unload handlers are problematic

- No right time to run the handler with BFCache
  - Can't run it as you enter because we hope to restore you
  - Can't run it as we evict due to privacy reasons (and others)
- Unreliable (especially on mobile)
- Chrome/Firefox/WebKit choose to prioritise BFCache on mobile
- WebKit also prioritises BFCache on desktop
Reducing unload usage

- We worked with big providers of 3rd party iframes
- Worked with some big partners
- Provide alternatives
  - Pending Beacon API
- Devtools
- Monkey patching addEventListener (not great)
- Provide opt-out
  - Permissions Policy: unload
Permissions Policy: unload

- Give outer frames control over inner frames.
- Allow gradual removal (PP doesn't allow this as well as we'd like)
- Prevent reintroduction (big teams)
- Enforce policy (e.g. ad networks)
- Override hard-to-change script
- In some ways, never works > sometimes works
Problems

- That unload handler might be important
- Cross-site control of code execution
  - Unload is already unreliable
  - sync-xhr and document-domain
Discussion
Cache-Control: no-store
Current state

- All browser implementations block BFCache with CCNS
- Not specced to have any interaction with BFCache(!)
- Intentionally used to prevent BFCache
Can we just unblock? No.

- Logout then hit back (*especially bad on shared devices*)
  - auth cookies have been cleared
  - page is restored from BFCache
  - information is visible that should not be
We can solve the logout case

- If we evict site's pages on logout
- Give sites a JS API to directly evict their pages?
  - Hard to deploy (needs to be called at all the right times)
  - Easy to over-use
- Tie eviction to secure cookies
  - If secure cookies change/delete/expire, evict site's CCNS pages
  - No action needed by authors
  - Is it safe?
Does that make it safe?

- **Problems**
  - cookies disabled
  - auth tokens held outside of cookies
  - site using 3rd party auth tokens but no auth tokens of its own
  - server-side logout
  - stale information
  - non-secure cookies for auth

- **Remediations**
  - don't cache if cookies or JS is disabled
  - update content in pageshow
  - cookie-eviction [API](#)

- **What are we missing?**
Cookie-eviction API

- **Explainer**
- **Very straw-person example**
  ```javascript
  backForwardCacheController.evictionCookies = ['SID'];
  ```
- **2 purposes**
  - Site may want to evict on more than just secure cookies
  - Site may want to improve hit-rate and evict on a smaller set of cookies
- **Using the API is a signal that we can BFCache the page despite CCNS**
- **Alternatives**
  - list cookies in a header
  - set metadata on cookies
Proposed end state

- Cookie eviction API is available
- HTML spec is explicit
  - that CCNS pages are cacheable
  - evicted on cookie changes - API-specified or all secure