Safety against Clickjacking / UI Redressing Attacks

Brad Hill <bhill {at} paypal-inc.com>

Jeff Hodges <jeff.hodges {at} kingsmountain.com>

Clickjacking / UI Redressing

- A web resource or application can induce the web user agent to include, frame or embed another application from a different security domain.
- In so doing, it may be able to convince the user to interact with the nested application out-of-context, by obscuring or modifying the target application's presentation to the user.

Two types of cross-origin mixing:

- Transclusion
- Framing / Embedding

Transclusion

- Content included *inline*
- Same browsing context / DOM

- E.g. images, fonts, css, <script src=>

- Once transcluded, part of the total instantiated resource / application, single effective origin for access control

- NOT IN SCOPE FOR THIS WORK

Framing / Embedding

- Explicitly distinct browsing contexts, with different security principals (origins) and enforced security boundaries
  - frames / iframes
  - Some plugin content using object/embed tags

- Attacks arise due to incomplete isolation at the *User Interface* level

Difficult problem to solve

- User Interface context mixing is *by design* and a *desirable* property of the web user agent
  - Except when it isn't
- No unambiguous fixes possible at the protocol or browsing context security model
- Diversity of user agent / user interface features:
  - Modal vs. multi-window, mouse vs. touch, voice or assistive technologies

X-Frame-Options Header

- DENY, SAMEORIGIN, [ALLOW-FROM]
  - All-or-nothing means that use cases which require framing cannot use this policy
- Application authors need more granularity:
  - Allow, and apply protections if possible
  - Only allow if possible to apply protections
  - Report, don’t block, if things look suspicious

“UI Safety” spec @ W3C WebAppSec

http://dvcs.w3.org/hg/user-interface-safety/raw-file/tip/user-interface-safety.html

- Use Content Security Policy header to convey UI safety requirements and tuning hints to the user agent
- Non-normative recommendations on how to apply such recommendations at the user agent
  - Screenshot comparisons to detect overlays, repositioned content
  - Click timing measurements
  - Etc.

From the application owner’s perspective, features of XFO and UI Safety are part of a single risk management policy around how the web user agent manages the application’s user interface.

Going forward, it may make the most sense to define both policy pieces in the same spec.

Advantages to moving XFO features to CSP UI Safety specification:

Clearest policy combination mechanism for resource owners.

- If UI Safety directives are specified and understood by the user agent, they apply exclusively
- XFO policy applied by user agents that does not understand or find a CSP UI Safety directive for a resource

XFO features can take advantage of CSP features

- CSP specifies a reporting channel and is developing a DOM API
  - Application authors may wish to use these for risk management with the XFO features

- Re-use CSP definition of origin
  - Likely source of error to require authors to continue to use two syntaxes and two headers to express one intention

Single conveyance mechanism may give broader adoption

- XFO policies really are associated to the *content user interface*, not the protocol
- Chrome extensions have a way to set a Content Security Policy in the application manifest, do not have a way to set XFO
- Widgets, app cache, etc.
  - All could have a mechanism to attach or persist XFO *and* CSP, but easier to just do one