CM API polyfill checklist

The document is public

(vasilii@chromium.org)

The doc answers the standard questions for polyfilling a feature in Bling. The CM API was shipped in Chrome on all the platforms but iOS.

Is implementing the feature before WebKit of large strategic importance?

1. Our stated goal is to get rid of passwords. The credential management API is an important stepping stone in that direction, but it can only be successful if it is broadly available.
2. With the recent changes to the CM API, it is now possible to implement a high fidelity polyfill. Before it wasn't possible.

Can your team own the polyfill implementation indefinitely?

The team is committed to owning the API on all the platforms. Essentially we own the spec for the API.
In addition, we assume that at some point we'll just switch to WebKit implementation.

Can the feature be fully implemented via Bling's API layer with reasonable performance?

The polyfill layer in Bling will do the same job the credentialmanager module currently does in Blink. It just passes all the calls to the browser making some simple security checks on the way. The calls are extremely rare

- Request a credential
- Store a credential
- Notify browser on sign out

Will we be able to implement future changes to the spec?

We own the spec. Currently there is one big extension planned. WebAuthN API is gonna be built on top of the CM API. That component is different from the traditional password/federated credentials and requires hardware support. We are not planning to polyfill it for now but see the WebAuthN API also of strategic importance. We may implement a polyfill if needed. It's definitely possible[1].
The second idea is to support iFrame. Currently there is no plan for that. It's technically doable in Bling but just requires a lot of work.
Anyway, JS is a thin wrapper that just talks to the browser.

Can the feature be feature detected?

```
if (navigator.credentials && navigator.credentials.preventSilentAccess) {
  // The new Credential Management API is available
}
```

Are there security or privacy concerns from implementing the feature in JS?

There are no secret values which are hidden behind the JS objects. Native implementation should always check the URL of the requester. The same is applicable to Blink actually.
All the methods are asynchronous. Currently only the main frame is served.

Can the feature be tested against the same test suite used by Blink?

It should be possible. Blink has some Layout tests that run without actual password store.

[1] See https://github.com/apowers313/webauthn-soft-authn and https://github.com/apowers313/webauthn-polyfill

Do we think the feature won't create enough user pressure for Safari to eventually ship the API?

The feature is user visible and improves the sign-in experience for the user. We also believe that WebAuthN part may seem of interest to Apple. Then they'd need to implement the API in WebKit.

Could the feature be unblocked by fixing something feasible in WebKit?

We have resources. It takes time to convince Apple to accept it. So far, our arguments were solid. In the past they committed to implement the API but the recent changes cooled them down.

Is it possible there may never be a usable version in WKWebView?

There is no technical problem to have the API in WKWebView. It's just the matter of time to reach an agreement.