Permissions Workshop Position Paper

Alex Russell <slightlyoff@google.com>
Thomas Nattestad <nattestad>
August 20th, 2018

The web's existing permission model has evolved in an ad-hoc way. This evolution has been shaped by many forces, including:

- Abuse and annoyance of users
- Concerns about user understanding of, and control over, potentially dangerous capabilities
- Implications of the Same Origin Policy, the Web's fundamental actor model
- Evolving API design practice and the need for backwards compatibility

An overriding concern for browser vendors has been (and continues to be) the need to adjust policies in a relatively dynamic way, for instance by denying iframes the ability to request certain permissions by default, or the need to deny certain capabilities in various situations. These adjustments are not part of any specification, and browsers require the flexibility to independently intervene on the user's behalf. This requirement has motivated the Chrome Team's dual pursuits of a more uniform Permissions API and a modern stance that standards must not attempt to be prescriptive about when and how browsers request that users mediate permission choices.

In our view, today's Permissions model and API surface area should be heavily revised to better support these goals and to enable more flexible permission policies. This view comes from an acknowledgement that existing permission-requesting APIs frequently annoy users to such an extent that browser intervention to prevent low-quality prompts is inevitable (e.g., the case of the current scourge of prompt-on-landing requests for Push Notifications).

Therefore, our position is that the current reflection-only surface area of the Web Permissions API should be expanded in the following ways:

1. The ability to request user mediation of a capability (e.g., a prompt) is itself a system-gated capability, and our APIs should model this capability. These decisions may be made based on complex inputs, both local and remote. For example, we wish to gate the ability to request Notifications on a calculation based on user engagement with a particular site as well as a global reputation score.

   Therefore, we propose that Permission-requesting APIs need to introduce the ability for developers to register interest in a capability before ever being allowed to prompt users

and to be called back (by event, e.g.) when the situation changes in their favor.

2. Permissions requests should be centralized on a single method to enable the introduction of better user controls and developer consistency. For instance, we are interested in providing developers and users the ability to negotiate time-limited permission grants. Retrofitting the `request*()` methods for every API to create a consistent way to model this time limiting presents a years-long task.

   Therefore, we propose that we should, instead, fast-track something like the unified permission request proposal as a modern, Promise-compatible, extensible entrypoint for developers to supplant the myriad, inconsistent legacy permission request methods. We recognize that the previous proposals may only serve as a jumping-off point for a new design effort.

3. Developers should be able to ensure their applications operate with least-privilege. This is critical in allowing the construction of accurate in-app toggle UI for permissions as well as for security-conscious sites to ensure that they can attenuate incorrectly granted capabilities post-incident.

   Therefore, we propose that the Permissions API should be extended with a `revoke()` method (naming very much TBD).

4. Developers should be able to declaratively specify maximum-privilege extent.

   Therefore, we propose that the Web Application Manifest should be extended to include fields which allow sites to identify to the runtime a maximum set of permissions. Requests for permissions not included in this list should fail (when the list is provided).

We believe that these additions, combined with the web's historically cautious grant of capabilities, provide invaluable tools to developers, users, and browsers. Browsers will continue to expand the set of capabilities available to developers to enable them to compete against other platforms that compete for developer investment; a unified model for requesting and managing them is long overdue.