Practical Privacy Concerns in a Real World Browser

Ian Fette IFETTE@GOOGLE.COM
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043 USA

Jochen Eisinger EISINGER@GOOGLE.COM
Google Germany GmbH, Dienerstraße 12, D-80331 Munich, Germany

Abstract

Google Chrome has implemented a number of "HTML5" APIs, including the Geolocation API and various storage APIs. In this paper we discuss some of our experiences on the Google Chrome team in implementing these APIs, as well as our thoughts around privacy for new APIs we are considering implementing. Specifically, we discuss our ideas of how providing access to things such as speech, web cameras, and filesystems can be done in ways that are understandable and in the natural flow of users.

1. Introduction

Over the past three years, the rate at which browsers are innovating and introducing additional APIs has skyrocketed (Pichai, 2010). Over the same time, we have not seen any significant new user experience paradigms take hold in widely deployed user agents. By and large, there is still an over-reliance on the traditional fallback of being able to consult an authoritative source, be that the user or a policy defined by some third party. We believe that new APIs being proposed for use on the web must either be "safe" by default, or where some capability is being granted that may negatively impact the user's security or privacy, this capability should be granted through a user interaction that is natural and understandable to the user.

We will begin by giving some background information on the APIs we discuss in this paper and on browser security. We will continue by discussing our experience in implementing the W3C geolocation API (Popescu, 2009), after which we will discuss some upcoming APIs around speech, web cameras, and filesystems. We end our paper with concluding remarks on what will be required for a successful development platform that meets users' privacy and security concerns while still being attractive to developers.

Preliminary work. Under review by the W3C Workshop on Privacy for Advanced Web APIs. Do not distribute.

2. Background

Traditionally, applications running in the web browser have had little access to capabilities of the device they were running on. For the purposes of this paper, we define a "web application" as a page or collection of pages and resources accessible via a web browser that is designed to provide a service to a user. There are many examples of web applications, including mail applications such as Gmail and Hotmail, mapping applications like Google Maps and Bing Maps, as well as applications with no desktop analog, such as Facebook. Many of these applications were written using mostly HTML 4.01 (Raggett et al., 1999) and JavaScript (Fulman & Wilmer, 1999). These languages provide the capability for information to be processed and displayed within a user agent, but with a few notable exceptions (cookies), provide no further access to capabilities of the device on which the user agent runs, such as persistent storage, access to devices such as GPS units, video cameras and microphones, and the like.

Today, specifications such as HTML5 (HTML 5) and contemporary specifications being published by working groups such as the Web Applications Working Group and the Device APIs and Policy Working Group give capabilities to web applications that have previously only been accessible to native applications, or accessible via running native code (e.g. plugins) in the browser. The W3C Geolocation API (Popescu, 2009) lets a web application request, via JavaScript, the user's location as a (latitude, longitude) pair, as well as the capability to receive updates to this location as they become available. The specification in-

cludes a section on security considerations, which provides motivation for the security concerns expressed as well as some guidance on what information is relevant in making security decisions around this API. The specification does not mandate a particular flow or user experience that must be followed to determine whether to give the requesting website access to the user's location, nor do most other APIs prescribe a prcecise user experience. In the next section, we will discuss our experience in trying to come up with a suitable user experience for the Google Chrome browser.

3. Geolocation Experiences

The relevant requirements from the W3C Geolocation API specification can be distilled into four succint points.

- User agents must not send location information to web sites without the express permission of the user.
- User agents must acquire permission through a user interface, unless they have prearranged trust relationships with users.
- The user interface must include the URI of the document origin.
- Those permissions [...] that are preserved beyond the current browsing session [...] must be revocable and User Agents must respect revoked permissions.

The Android browser was the first Google product to implement the W3C Geolocation API. When you visit a site that requests your location using this API, you are prompted for your location as shown in Figure 1. When we were ready to implement this API in Google Chrome, it seemed as if it ought to be relatively straightforward, as we as a company already had experience creating a compliant UI once. However, when the Chrome user experience team designed the interface for Chrome, they asked a critical question - what about <iframe>s, and other resources embedded from other origins? For instance, if the user has granted maps.google.com permission to use the Geolocation API, and example.com embeds a Google Maps gadget, should the user's location be shown, or not?

We considered various scenarios to answer the question of permission models for embedded resources requesting a user's location, from embedded maps used to provide directions to advertisements and more. We considered using the origin of the top level document, the

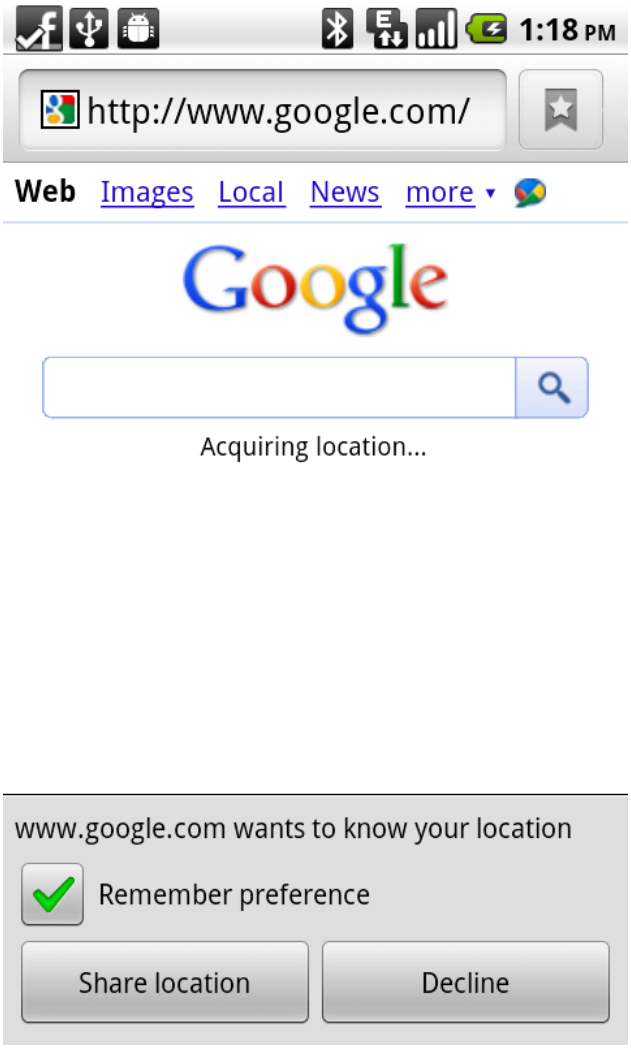

Figure 1. Geolocation access request prompt on the Android browser

origin of the resource or frame requesting the location, a permissive "OR"ing of the two, and a more restrictive cross-product of the two. From a purely security standpoint, the answer should not matter as code running in an <iframe> that had access to your location should not be able to be interrogated by code from an other site, thanks to the same origin policy (sam, 2009). However, a user's perception of the security and privacy implications may be very different than a browser implementer's perception.

As implementers, we understand the security model of the browser, and understand the notion of sites embedding resources from other origins, and the protections afforded to the data within such other origins by policies such as the same origin policy. Many users do not

share this same level of understanding, however. We believed that users might be surprised if they visited a site for the first time, and saw on that site a maps gadget showing their exact location. As developers we could rationalize that we had granted access to the origin of the maps gadget in the past, and it's still the same origin getting our location data. As a user however, it appears that a totally new site now has access to location data, and whether or not this is technically true, it can create a perception issue and unexpected behavior that we wish to avoid.

To meet the user's expectations, we came up with a scheme whereby a user's preference was stored for a tuple of the origin of the top-level document, as well as the origin of the resource requesting the user's location. In order for the user's location to be disclosed to a page, a user must have explicitly allowed the combination of the top-level origin and the origin of the resource requesting the location, and if this condiditon is not met, then the user is asked and has the option of allowing or denying the request. The result can be seen in Figure 2, which shows an example of how the list of allowed combinations appears in the settings dialog.

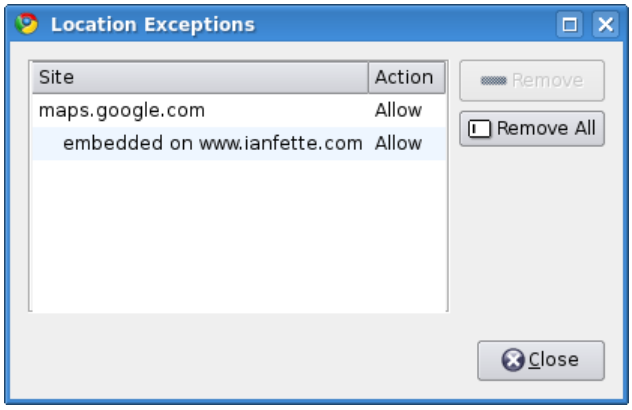

Figure 2. Google Chrome's Geolocation Settings Dialog

Implementing the Geolocation API in Chrome was a much more difficult task than expected. This difficulty is largely because there exist multiple mismatches between the actual security model of the Web, and what users perceive. In designing new APIs for the web, and new security models for these APIs, we must keep in mind how the capabilities will be perceived by users, especially in an era where pages are increasingly leveraging shared components and "widgets" provided by third parties. Simply saying "prompt the user" is not an acceptable answer, unless one can be sure that the question the user is prompted with is a question that the user can understand, and is not a question that assumes a mental model radically different or more complicated than the model the user posesses.

4. Upcoming APIs

4.1. Audio and Video

We are working on providing APIs to give webpages access to audio and video capabilities of the platform – specifically, a speech recognition and text to speech API, as well as access to the video camera (if available). Where possible, we prefer leveraging the <input> tag for as many of these capabilities as possible. As a concrete example, we are planning to expose a speech recognition API. Rather than letting a website call RecognizeSpeech() as a JavaScript API however, we are planning to offer an API of the form <input type="speech"> that will appear as a text input field, with a microphone icon next to it. When the user clicks the microphone, a visual element will appear that clearly indicates the microphone is being used, and the browser will begin analyzing microphone input, making a text transcription available via the input form element. By doing so, we can provide an API where we have an intuitive user interaction (the user clicks a clearly identifiable button), that avoids prompting the user with a question. We want to limit the number of permissions we have to request in the form of "This site wants to do X, is that OK?" and instead implicitly grant permissions based on user actions, whenever possible.

4.2. Filesystem

Another API under development is a filesystem API (Uhrhane, 2010), which exposes a sandboxed, per-origin filesystem to web applications. Many web applications wish to store arbitrary data (email, attachments, photos as an example) on a user's disk, either for access whilst offline, or as a cache for performance reasons. While there are great benefits to such an API, there is also a concern that a malicious attacker could use such an API to fill up a user's hard disk. As such, we are proposing that there be two types of filesystem stores – one that is persistent, and one that is temporary. By default, we will allow applications access to a temporary filesystem. This provides a safe default path, as the user agent can delete data from this temporary filesystem at any time (for instance, should disk space become scarce, the browser could delete the contents of temporary filesystems). At the same time, websites can use the API for many use cases where strong persistence guarantees are not required, and so in the default case, we provide a useful API without

burdening the user with a series of permission questions. In the case of persistent quota, user consent can still be requested, though this use case is likely rarer, and so for what we believe to be the default use case, we have managed to provide an API that is safe by default and does not require yet another user prompt.

5. Bundling Permissions

While we believe it is important to reduce the number of permissions that must be explicitly requested by an application by providing safe-by-default APIs, and modeling APIs as <input> elements where appropriate, there is a limit to how far this will take us. Eventually, when building complex applications, we will hit cases where the number of permissions that must be requested, or the number of user interactions that must be completed to grant permissions in in-line workflows, exceeds what is reasonable to expect from a user. Imagine, for instance, a video-conferencing application written solely with web standard APIs. One could imagine requiring access to video cameras, microphones, allowing a peer-to-peer connection, allowing geolocation access, and more, such that the number of user interactions to start a video conference would be prohibitive. There are also some permissions, such as the ability to access the clipboard, that while they could be requested of the user, are too fine-graned and nuianced to actually express as a meaningful question to the user. For these scenarios where a large number of permissions are to be requested, or where permissions are too fine-graned to make sense as a user prompt, we need a better answer.

One answer that the Chrome team is working on is the notion of an installable application. In the desktop environment, a user installs an application once, and grants virtually all privileges to that application. This has clear downsides as is evidenced by the number of computers with viruses and malware installed, but does have the advantage of bundling everything up into a single question – do you, the user, trust this application? We believe that with new efforts, we can help users answer this trust question by providing additional data about the application (including reputation data such as the number of other users using the application). More importantly, we can reverse this trust decision and actually revoke access in a single action if the user desires that, or we discover that the application is malicious. As such, we believe that the packaging of permissions into a single question makes sense, where the application specifies up front exactly which permissions it would like, and are pursuing various options along this vein. This will enable more natural user experiences, and avoid forcing a user to grant multiple permissions serially (and potentially spread out over a period of time), before a web application is useable.

6. Conclusion

In this paper, we have discussed our experience with implementing the W3C Geolocation API, and how it has motivated us to be particularly sensitive to the fact that implementers of user agents and end users have different mental models. As such, we believe that it is critical for other implementers and spec authors to keep in mind the mental model of the user, and make sure it is possible to phrase questions to the user that match their mental model of what is happening on a website. We have discussed new APIs we are in the process of implementing, and how we are attempting whenever possible to avoid creating more permissions for which we must prompt the user. Finally, we have discussed the need for a mechanism where multiple permissions can be granted, and revoked, in a single action. We hope that these proposals prove useful for others, and provide an interesting starting point for discussion in the workshop.

References

Same origin policy for JavaScript. Mozilla Specification, June 2009. URL https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript.

Fulman, Jason and Wilmer, Elizabeth L. Ecmascript language specification. http://www.ecma-international.org/publications/files/ecma-st/ecma-262.pdf. *Ann. Appl. Probab*, 9:1–13, 1999.

HTML 5. HTML 5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft, August 2009. URL http://www.w3.org/TR/2009/WD-html5-20090825/.

Pichai, Sundar. Google I/O Keynote. May 2010. URL http://code.google.com/events/io/2010/.

Popescu, Andrei. Geolocation API Specification. W3C Recommendation, July 2009.

Raggett, Dave, Hors, Arnaud Le, and Jacobs, Ian. Html 4.01 specification. W3C Recommendation, December 1999. URL http://www.w3.org/TR/html4.

Uhrhane, Eric. File api: Directories and system. W3C Editor's Draft, June 2010. URL http://dev.w3.org/2009/dap/file-system/file-dir-sys.html.