W3C Workshop on Security for Access to Device APIs from the Web - Position Paper

Olli Immonen (olli.immonen@nokia.com)

Security model for browsing and widgets

It has been common to think that the security model for device API access, involving also related user experience aspects, is somewhat different for widgets and for browsing. In browsing, users are used to feel relatively safe without thinking too much about trustworthiness of the sites they browse. Something bad might happen if you download code. Of course one must think before entering private data. But just browsing and entering some innocent data should be safe. For widgets, the situation is somewhat different. User should think whether to take a widget into use. That is why it would be easier to grant more powerful device access to widgets.

On the other hand, there might be a widget that just implements an entry point to a service. All code is fetched from a web site. Or, a browsing bookmark may be presented so that the user experience is quite similar to that of a standalone widget. This is why it useful to consider both cases in order to create a maximally uniform solution.

Identification

How to identify the entity allowed to certain access? Options include

- Site (2nd level (base) domain, full domain, full address)
- Name of the responsible entity like the Distinguished Name (DN) from a certificate of a TLS server or software signing entity (widget or web page script).

The user obviously should understand the meaning of the identity in order to grant access. However, studies related to phishing attacks indicate that users usually do *not* understand URLs even if they try to. It is questionable how useful it is to identify the site (e.g. DNS name) to the user. "This site" or "This page" might be effectively as good, or everything that the user understands (or bothers to read).

Authentication and authorization

How strongly does the entity need to be identified/authenticated. Options include

- No authentication, i.e. just DNS
- TLS (https)
- Signed scripts (of web pages) or widgets

Anything besides just DNS would be somewhat of a deployment issue. SSL/TLS itself is widely deployed but without a controlled certificate issuance process its value is limited. One can argue that TLS does not really protect against phishing since average users can't reliably tell if TLS is used or not, and sending user input to non-TLS sites is allowed. For controlling device APIs, TLS could actually be helpful, if the platform would require TLS being used for certain API access, possibly with some requirements regarding site certification processes (CAs).

Signing works in closed systems like intranets but in the open internet it is problematic, at least if it

is based on users understanding the PKI and identities involved. An approval type of signing where the signing entity guarantees not just the identity of the provider (developer) but the trustworthiness of the code, might not be robust enough.

Whitelist and blacklists, possibly based on site's or widget developer's reputation within a community, are an interesting option here.

Following the *principle of least authority* (POLA), only those rights should be granted that are absolutely necessary. There are several ways to limit the access right

- less features (e.g. just READ instead of READ&WRITE)
- limited time (one-shot, session, time period)
- measurement accuracy (e.g. location accuracy)
- scope (e.g. access to just one person instead of entire address book)
- implicitly controlled access (e.g. UI to select a file to to upload, see more below)

For a user, in principle it would be easier to give less rights to entities that he/she is not ready to trust fully. On the other hand, controlling things in detail may become laborious and lead to unpleasant user experience. So, services would be tempted to ask for "full" rights and users to grant them even in cases they actually should not.

Policy

A policy language could be able to describe all the above options. It could be useful for an organization (corporate IT, telco operators). Anyway, at some point the policy should be presented to users. What should be visible as choices?

- Option 1
  - Defining which are the "highly trusted" entities/sites, and "moderately trusted"
  - What is allowed to "highly trusted" sites and to "moderately trusted" sites
- Option 2
  - Explicitly allowing each entity/site those rights it needs and should have

The advantage in option 1 is that authorization is less cumbersome. The question to answer is shorter ("trust or not"). Risky APIs and combinations can be better taken into account when setting policies.

The advantage in option 2 is that question to answer is more concrete and so easier to understand. Also, it is better aligned with POLA.

Declaring rights

Does the widget or the web page need to declare beforehand which kind of rights it will need? Or, is all this done on the fly, when actual API call are used.

Declaring has been seen useful for MIDP applications and same applies probably web widgets. The decision about granting necessary rights is typically made when the application is installed or when it is started, anyway before the user has seen the application doing anything.

The benefit of declaration approach is that an entity can declare minimal rights (POLA!). Even if something goes wrong (e.g. site gets XSS-attacked) the damage is limited to declared rights. Here we assume that changing the declaration is harder than changing the code (like using XSS and eval()). This can be achieved by signing the rights declaration for widgets.

But would declaration be feasible for web browsing? Besides technical difficulties there are application issues.

- How would the declaration be made harder to change than the code. Signing a "site capability certificate" could be possible but hard to deploy.
- From application logic point of view, it may be non-optimal to declare the rights in advance

Attacks and risks

Besides attacks relying on technical issues (buffer overflow etc.), application and user behavior level attacks are of special interest here:

- **User giving access unknowingly.** The user is lured to giving access without noticing or understanding what is happening, possibly as a result of tempting content.

- **Impersonation.** A malicious site masquerading a good one may request access to users device. The situation is similar to phishing attacks where users are lured to enter their sensitive information.

- **Vulnerable site.** If the site you trust to access your device is vulnerable (to XSS or similar) then your device will also be vulnerable. Apart from the usual anti-XSS methods, one way to deal with this is to grant only minimal access. Even trusted entities should only ask and be given rights they absolutely need.

It is impossible to foresee all risks. Risk assessment is still beneficial in order to

- evaluate how well the system is able to deal with the risks identified
- see how serious the risks are; what if the system fails

There are general risks and risks specific to each API. Risk analysis should be the basis for setting the policy (and enabling an API in the first place!).

Mashups

In mashups there are many entities involved. In a client mashup web page the content comes from multiple sources. Widgets (provided by a certain entity) can fetch content and code from multiple entities that are possibly different from the one providing widgets. This is challenging for controlling access to device APIs: Do all entities need to be identified, authenticated and given access rights separately or would it be enough to rely on one entity (page, widget)? The users might have the idea about accessing a certain site (and be willing to grant access to that) but in fact it might be another entity (providing a script) that wishes to utilize the device API. Enforcing access control technically might take us to quite complex models like those in Java 2 security where permissions of components from various sources are analyzed in the execution stack.

Implicitly controlled access

For many kinds of functionality it is possible to design a UI that implicitly controls access to device APIs. In this case, no separate authorization is necessary. Available techniques include

- HTML form input e.g.
    - Reading a file from the file system (form input type "file")
- Special URL schemes

    - Sending an email (mailto:), making a phone call (tel:)
- JavaScript APIs with a UI
    - crypto.signText, UI for signing text using private keys and certificates of the device

The same principle could perhaps be applied to other APIs as well e.g. (disclaimer: just examples)

- Camera (take a picture)
- Addressbook (select a person's email address)

The benefit of this approach is that no separate authorization is needed. Implicit authorization (selecting a particular file) is easier to understand than a technical question ("do you allow the site to access your filesystem?"). Also, users are tend to just click yes to questions like like allow/not that are just hindering them from getting their job done.

The drawback is that that this approach limits application UI design. And of course this approach is hardly feasible for many features like

- continuous monitoring of a device parameter
- reaction to a device event (like receiving a phone call)

Exercise: Thermometer API

It is good to start from thinking something less critical...

Use case: A web site wants personalize product offerings depending on is it hot or cold out there, is the device used inside or outdoors.

Continuous temperature monitoring would be an issue. While no actual risk can be thought about, still: not every site should be allowed to monitor the user's personal space!

Implicitly controlled method. Platform-generated thermometer UI: Clicking a button is necessary to measure the temperature. No separate access control is needed.

Continuous monitoring. An ad portal would really like to have this in order to avoid unnecessary user actions. This requires granting a permission. For everyday use, it should be granted "always". For sites accessed seldom "session" might do. Regarding authentication, just DNS would probably be OK for this feature. Regarding the policy, "trusted sites" should certainly have right to this information. A bank, as a trusted site, would not use this information. But if it would, what would be the risk...

Conclusions

- There should be maximally uniform solution for device API access from widgets and browsing
- Declaring rights (and only those absolutely needed) would be a good practice
- Enforcing access control for mashups requires taking multiple entities into account
- APIs providing implicitly controlled access are an interesting option