Security Assurance for Web Device APIs

Maritza Johnson and Steven M. Bellovin
{maritzaj,smb}@cs.columbia.edu
Columbia University

Abstract

There are currently proposals for web access to devices. The security threats are obvious. We propose design principles intended to ensure that the user actually controls access, despite potential errors in judgment, tricky web pages, or flaws in browsers.

1 Introduction

There are many obvious reasons why web developers would like enhanced access to the facilities of the client platform. It is equally obvious that such access represents an enormous potential threat: if the solution is designed or implemented incorrectly, assorted security and privacy problems could result, ranging from loss of personal privacy to the complete takeover of the host platform. Accordingly, strong protections must be designed in from the beginning.

We do not claim to have a complete solution. Rather, we wish to propose certain design principles. We focus on two areas: usability of the human interface, and guarantees that the browser is suitably isolated from all but authorized devices.

Problems in this space are legion. Permission problems, for example, go back to at least 1993. In CERT Advisory CA-1993-15, problems with `/dev/audio` were described:

> Any user with access to the system can eavesdrop on conversations held in the vicinity of the microphone.

It is important to note that the problem is not merely one of incorrect permissions. Rather, SunOS had (and has) a mechanism to assign certain devices, such as the microphone and the mouse, to the user logged in to the console. `/dev/audio` was omitted from this list. In other words, the failure was of configuration of a dynamic mechanism. From our perspective, though, the problem is more fundamental: there was no *high-assurance* way to ensure that proper permissions were set for all devices, including perhaps new ones. Similarly, there was no *high-assurance* way for users to understand who can access such devices.

The web problem is considerably harder. Users of today's brwosers are considerably less sophisticated about access controls than those who used Sun

workstations in the early 1990s; furthermore, their machines frequently don't have professional system administrators to make any necessary corrections.

2 Usability

We suggest that the following principles are essential for security and privacy:

1. The user must explicitly authorize any and all accesses to devices
2. The user must understand the consequences of any change
3. The state of the system must be visible at all times

While much of this is primarily of interest to browser designers, there are implications for the API as well.

Consider principle 2. Users request web pages, not their embedded IFRAMEs. If a user is to understand to whom permission to use, say, the microphone is granted, any request from that page must grant permission to the owner of the page. This in turn has two consequences: the URL of the stream receiver *must* be on the same host as the page is loaded from; furthermore, if the URL is in in IFRAME, it, too, just come from that host. We realize that this will inhibit a possible market for third-party sound service providers. We submit that it is necessary, and suggest that any necessary hand-off be done by relay from the web site, rather than from the customer's machine.

Another reason for this is that empirically, it is very hard for users to tell which web site they are at [1]. This, of course, is why phishing attacks succeed. Increasing the severity of the threat — such as by letting users bug themselves — while relying on the same flawed assumptions seems unwise.

Principle 3 applies to both static configuration — what sites have already been authorized — and whether or not the device is currently being accessed. The former appears straight-forward but raises the question of whether users can be relied on to actively manage the list, how does a user revisit a security decision? The latter poses a more difficult issue: how should the user be told? Web page content should not be used to communicate security context information because an attacker can easily spoof the indicators[4]; indicators in the browser chrome are likely to be ineffective, and are also vulnerable since users cannot reliably distinguish between chrome and content [5, 1]. One is tempted to suggest that access can only be done when, say, the mouse is moved to some window; this would place constraints (and probably unacceptable constraints) on what such web pages could look like.

Finally, following principle 1 suggests that having the permission request to the user appear as a result of an implicit request in a web page is probably unacceptable — inactive warnings are ineffective and users have been habituated to just click whatever is needed to make annoying pop-ups go away [2, 3]. This in turn suggests that a two-page sequence is necessary before monitoring can take place: a first page to check the permissions; it will either go on to the next page automatically, or it will display an error message telling the user to

correct permissions. The act of making the change should perhaps trigger an auto-refresh of the first page.

3 Isolation

Our second major area of concern is how to be sure that the implementation is correct. That is, what should be done in the protocol or API definitions such that one can have confidence that that end-users systems are secure, across many years of upgrades, and with capabilities not dreamt of when the specification document is written.

One possible approach is to put each possible monitored device into a category; each category would have a sensitivity label associated with it. For example, cameras and microphones might be in the catagory `physworld`, which is marked extremely sensitive: these devices are, in effect, self-deployed bugs. Access that might disclose persistent identifiers — battery serial numbers, MAC addresses, etc. — might be in category `privacy`; these would be less restricted. Devices that could be used to secure transactions might be grouped as `security-token` and left unprotected, but only if manual action, such as swiping a magstripe card, is used to activate it.

Requests from the net for access would specify both the class name and the device within the class. Access would be granted or denied based on the permissions given to that site for the particular class.

This scheme has certain inherent advantages. It reduces the management complexity for the user; it also guarantees that new devices cannot be access at all unless they are explicitly assigned to a class.

A second layer of protection is needed to protect the user against the sort of attack described in the aforementioned CERT advisory. We do this by using operating system protection mechanisms to isolate devices from users who don't want them acessed. We give an example using Unix-style permissions; similar schemes can be devised for other operating systems.

Create groups for each category: `phys-world`, `privacy`, etc. Set the group for each protected device so that it is in the group for its category; set the permissions so that only that group has access to it. Thus, `/dev/camera` would be in group `phys-world`, with mode `060`, i.e., group read/write but no other access. If a users wishes to permit some sites to have access to the camera, create a setgid program owned by that user with no execute permissions for anyone but the user, i.e., mode `2100`, and no other permissions. Only one user can run the program; when it is running, it will be able to access the device. Enabling a category, then, entails creating this program, presumably via a setuid helper program that demands proper assent by the user; disabling a category is as simple as deleting the program.

The goal of this excercise in Unix file system arcana is to ensure that no possible web browser flaw can result in a user being bugged. This does happen today; see Figure 1 for an example. The basic principle of our design is simple: something that does not exist cannot be abused.

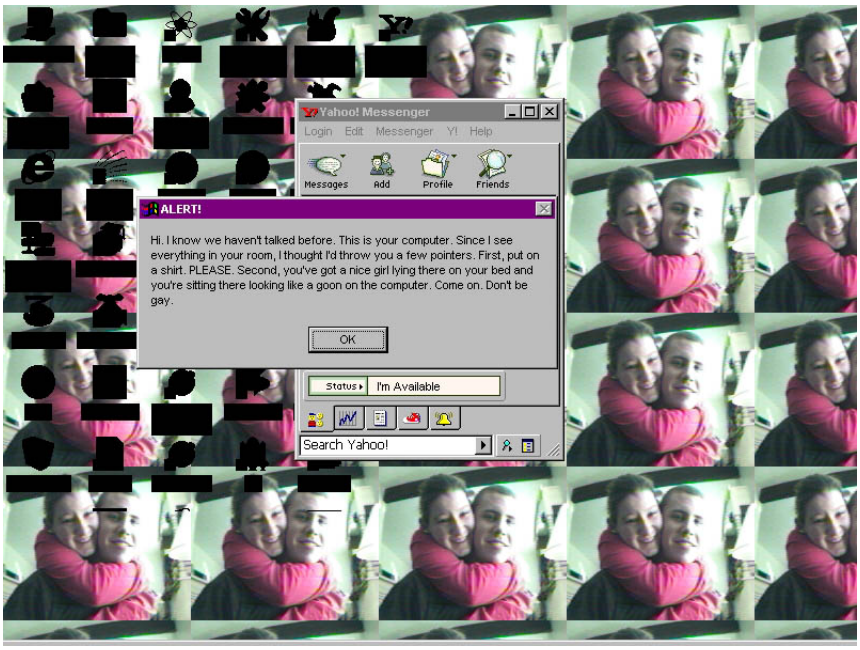

Figure 1: A screen capture after a hacker took control of a user's camera.

4 Conclusion

There are many obvious scenarios where web access to devices is desirable. That said, the risks are at least as obvious. We suggest several security principles: ensuring that the security model is indeed as the user wishes; and minimizing the consequences of flaws in code.

References

[1] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, New York, NY, USA, 2006. ACM Press.

[2] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1065–1074, New York, NY, USA, 2008. ACM.

[3] Nathaniel Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Joseph Konstan. Stopping spyware at

the gate: A user study of privacy, notice and spyware. In Lorrie Faith Cranor, editor, *Symposium On Usable Privacy and Security (SOUPS) 2005*. Symposium On Usable Privacy and Security (SOUPS), July 2005.

[4] W3C Web Security Context Working Group. Working draft - web security context: User interface guidelines. `http://www.w3.org/TR/wsc-ui/`.

[5] Collin Jackson, Danial R. Simon, Desney S. Tan, and Adam Barth. An evaluation of extended validation and picture-in-picture att acks. In *Proceedings of the 2007 Usable Security (USEC '07) Workshop*, 2007.