What You See is What They Get
Protecting users from unwanted use of microphones, cameras, and other sensors

Jon Howell and Stuart Schechter
Microsoft Research
{*howell,stus*}*@microsoft.com*

Abstract

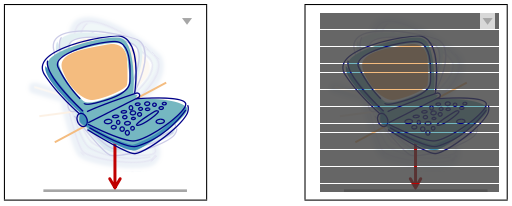
Sensors such as cameras and microphones collect privacy-sensitive data streams without the user's explicit action. Conventional sensor access policies either hassle users to grant applications access to sensors or grant with no approval at all. Once access is granted, an application may collect sensor data even after the application's interface suggests that the sensor is no longer being accessed.

We introduce the *sensor-access widget*, a graphical user interface element that resides within an application's display. The widget provides an animated representation of the personal data being collected by its corresponding sensor, calling attention to the application's attempt to collect the data. The widget indicates whether the sensor data is currently allowed to flow to the application. The widget also acts as a control point through which the user can configure the sensor and grant or deny the application access. By building perpetual disclosure of sensor data collection into the platform, sensor-access widgets enable new access-control policies that relax the tension between the user's privacy needs and applications' ease of access.

1 Introduction

Operating systems and web browsers are increasingly supporting application access to privacy-sensitive device sensors: cameras, microphones, accelerometers, and geolocation positioning systems. Mobile phones offer a wide variety of sensors and provide applications with APIs to access them. Flash [1] and Silverlight [12] provide web applications with access to cameras and microphones today, and an addendum to the HTML 5 standard promises to allow device access direct from pure JavaScript web applications [11, 15]. Lagging behind these rapid developments have been the tools required to help users manage the disclosure of sensor data.

We propose a new GUI primitive, the *sensor-access widget*, to enable users to manage application access to sensors. Sensor-access widgets borrow from an existing concept: hardware sensor-use indicators, such as camera-use LED indicators, that light up when the sensor is collecting data. Whenever an application requests access to a sensor, a sensor-access widget will appear and remain until the application releases access. A sensor-access widget displays a real-time human-comprehensible visualization of the data streaming from the sensor, much in the same way as the self-image widget used by video-chat systems displays the input from a user's own camera. For example, a sensor-access widget for an accelerometer might animate the movement of the user's laptop with respect to the ground, as illustrated in Figure 1. The sensor-access widget also indicates whether or not the application currently has access to the data stream, and acts as a control point through which the user can grant or deny access. If a sensor-access widget is obscured, the application's access to the corresponding stream is interrupted. This prevents sneaky applications from burying the widget behind a popup, and it also lets the user momentarily "cover the camera lens".



(a) Access allowed: sensor (accelerometer) readings are being collected by the application

(b) Access denied: virtual blinds indicate that sensor (accelerometer) data can not be seen by the application

Figure 1: A possible design for a sensor-access widget to appear within applications that request access to the accelerometer. The widget represents the data collected by the accelerometer by animating the position and movement of the user's laptop in relation to a gravity source. The widget indicates whether the application can currently access the sensor, using the metaphor of window blinds in this example.

The goal of sensor-access widgets is to make passively providing sensor data as conscious and preventable as actively entering data via a keyboard, mouse, or touchscreen. Keyboards, mice, and touchscreens are active-feedback input devices; data is only input as a result of user action and this action results in immediate visual feedback that indicates the application to which it is currently directed. Like active-feedback input de-

Figure 2: The countdown timer and the animated opening of window blinds inform the user that the application will soon be able to access a video stream—unless he disables it first.

vices, sensor-access widgets provide feedback to illustrate the data flowing into the system and, because they appear within an application's display region, they indicate which application is receiving this flow.

To make sensors true active-feedback input devices, we pair sensor-access widgets with a new access control policy we call *Show Widget and Allow After Input and Delay* (SWAAID). When an application requests access to a sensor controlled by this policy, the widget appears, indicates that access is not currently granted, and shows a countdown until the application will receive access, as illustrated in Figure 2. The countdown period should be long enough to allow the user to deny the application access. To prevent access from being granted when the user's attention is elsewhere, the countdown begins immediately after active user input is directed to the application, such as input from a keyboard, mouse, or touchscreen.

2 Related Work

In conventional desktop operating systems, including Windows and UNIX derivatives, applications running on behalf of the device owner have traditionally received access to all devices.

Systems to restrict device access and other privileges have addressed the problem both in terms of application plug-ins and the OS applications themselves. For example, in the mid 1990s Goldberg *et al.* described a system to sandbox native-code application plug-ins and limit their access to system APIs [9]. In roughly the same period, the Java language was introduced with a browser-based applet model that enabled the execution of "subversive code" restricted from accessing system resources [10]. Further efforts to confine applications and impose policies that restrict access to system APIs included Cowan *et al.*'s AppArmor (then called SubDomain) [4] and Provos's SysTrace [17]. SysTrace would request user's consent before granting access to previously-restricted resources.

Another approach to restricting device access in conventional operating systems has been to make the device accessible only to the superuser (admin/root), run applications as a user other than superuser, and request an elevation of privileges in order to access the device. In UNIX-derivatives this elevation is performed using the `sudo` command. Elevation was introduced to Windows as Vista's User Account Control (UAC) [13]. Alas, these mechanisms provide all-or-nothing elevation to superuser mode and their user interfaces (including the GUI dialogs on Windows and Mac OS) are much maligned. Dissatisfaction with the frequency of UAC's elevation requests led the Windows 7 team to create a default mode with fewer prompts [14].

Adobe's Flash player and Microsoft's Silverlight, browser plug-ins that enable web applications to access sensors today, require user approval before devices such as cameras can be accessed [1, 12]. Users of the Flash player can grant or deny access to a resource for a single session or may check the *remember* box to apply their decision indefinitely, as illustrated in Figure 3.

Figure 3: *Adobe's Flash player asks users to grant camera and microphone access to a website.*

Explicit authorization before use is also the policy implemented by many mobile phones. Google's Android OS grants access to sensors such as cameras and audio inputs only if their use is disclosed at installation time, as illustrated in Figure 4 [2]. At installation time, a user may not understand an application well enough to determine why it would need sensor data or gauge its trustworthiness.

The early Apple iPhone limits application access to the camera by routing all camera use through a standardized OS interface, as illustrated in Figure 5a [3]. When an iPhone application is loaded and requests access to location (GPS) sensor data, the OS prompts the user to approve access, as shown in Figure 5b. A recently-announced update takes a first step in the direction of sensor-access widgets, displaying an icon when an application is using location data [7].

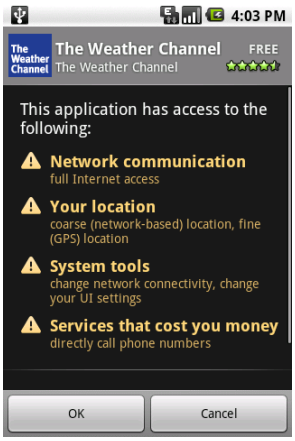
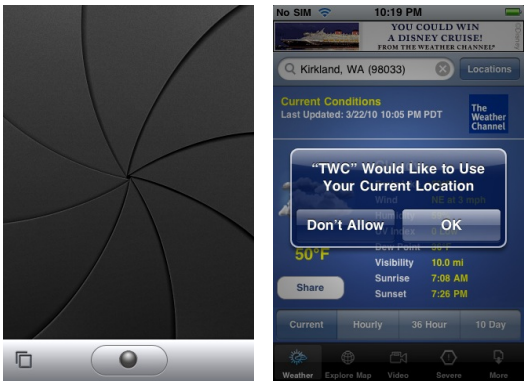


Figure 4: *Google's Android OS discloses applications' use of sensor data (GPS location) before the application is installed.*

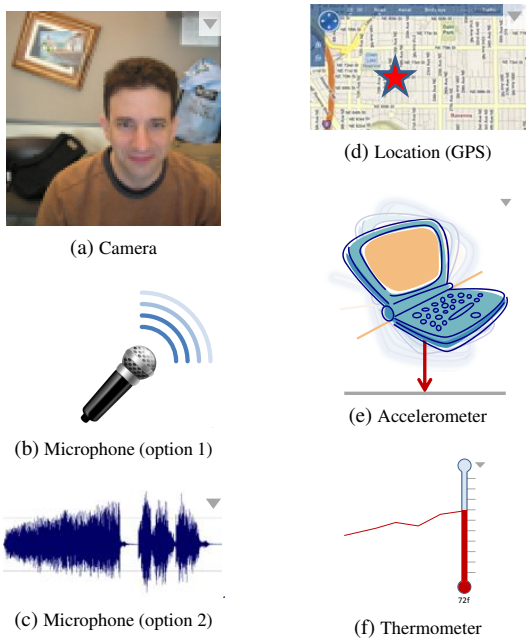


(a) (b)

Figure 5: *Apple's iPhone OS (5a) limits applications to using the device's camera only through this OS-provided user interface, and (5b) asks the user to approve access when an application requests access to a location sensor.*

Alas, per-session grants may not be sufficiently granular. In order to authorize access to sensors for a five-minute phone call, the user must grant access for an entire session, which could last days. Yet unless the user grants indefinite access she will still have to interrupt her workflow to re-authorize every time she restarts her browser, application, or computer.

Acquiring the consent of a device owner alone may be insufficient for collecting privacy-sensitive data when the device owner's incentives do not align with those of a legitimate device user. This became abundantly clear when a Pennsylvania high school, which had provided laptops to students, was found to be using the cameras on these devices to spy on the students to whom they were given [8, 21].



(a) Camera

(d) Location (GPS)

(b) Microphone (option 1)

(e) Accelerometer

(c) Microphone (option 2)

(f) Thermometer

Figure 6: Sensor inputs are converted into animated representations that match the flow of real-time sensor data. This figure shows possible animations for five sensor types.

3 Sensor-access widgets

We propose that when an application requests access to a sensor, the runtime environment (such as the desktop OS or the browser) should overlay a GUI widget onto the portion of the display controlled by the application. This sensor-access widget displays the live data streaming from the sensor, indicates whether the application can see the sensor data (the effective access policy), and acts as a control point from which the user can configure devices and control application access to them.

Displaying data collected by the sensor. The sensor-access widget animates a representation of the data being streamed by the sensor in a form that is meant to be recognizable to the user; users should notice the correlation between the animation in the widget and their environment (what they see, hear, or feel). For example, consider a video widget of your own movements or a graphic-equalizer display moving in time with your voice or the sounds of your immediate environment. To accomplish this goal, different types of sensor data will need to be displayed in different ways, as illustrated in Figure 6. Some sensors, such as location sensors, may not be as easily correlatable to a user's experience as microphone or video data.

Displaying effective access policy. Sensor-access widgets indicate whether applications have access to sensor data by changing their appearance appropriately. When an application is allowed to access sensor data, the user

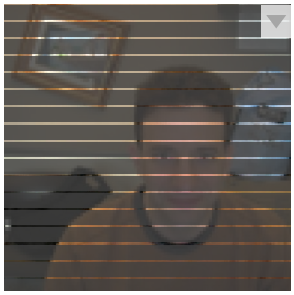


Figure 7: Window blinds illustrate that applications cannot see a data stream.

sees a clear stream of data as the application would—hence "what you see is what they get". When policy prohibits the application from accessing sensor data, the widget modifies its appearance to indicate that the application cannot get access to the sensor data; it could show virtual window blinds, as illustrated in Figures 1b and 7, overlay a red filter over the image, or lay an indicator such as an X or 'disabled' on top of the image.

Prior work on file system access control interfaces shows that users appreciate seeing *effective policy*–the outcome of policy rules rather than the policy itself [18]. Sensor-access widgets readily indicate whether an application is receiving data, and can indicate the granularity of the data stream graphically, such as by drawing an uncertainty disk on a GPS map.

Managing an application's access to sensor data. Sensor-access widgets act as control points from which users may specify when applications may access the sensor. For example, a right-click on a widget might allow the user to set the policy to *allow* or *deny* as they can today using preference dialogs. The use of a widget as a control point is illustrated in Figure 8.

Choosing the right sensor and configuring it

In addition to helping users protect their privacy, sensor-access widgets can also solve an existing usability problem: specifying which sensor of a given type should be used. For example, many users find it challenging to change the microphone to be used by audio- and video-conferencing software. A user may have a microphone in her laptop, built into her USB webcam, and on her bluetooth headset. If the wrong one is chosen by default, users are forced to search through application-specific interfaces or to change the default device for the full system. Sensor-access widgets can solve this problem by including the option to switch to another sensor of the same type from within the widget. Applications would no longer need to provide a custom selection interface in an unfamiliar menu. Users would benefit from a uniform experience for changing devices. The widget could even provide a unified interface for managing the amplitude of the input signal.

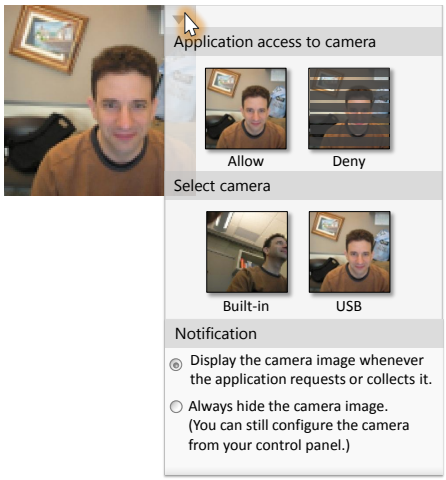


Figure 8: This sensor-access widget is also a control point: a context menu lets the user grant or deny access, select alternate sensors of the same type, or hide the widget.

Using the same widget to both handle sensor privacy and configuration is complementary: Configuration tasks help users understand and familiarize themselves with the sensor-access widget, and policy-setting activities remind the user of where to go to configure their sensors.

Further complementary uses include replacing the self-image widget in videoconferencing applications with a user-resizable sensor-access widget for the camera. Applications could grow the widget within their space, such as to create a viewfinder, and then return it to its prior user-selected size. Similar widgets could also be created for outputs (e.g. speakers), though these widgets might have less strict rules about when they appear and when they can be hidden.

4 Widget policies

Sensor-access widgets support the same per-session or indefinite *allow* or *deny* choices that today's access management interfaces provide. However, pairing these policies with widgets' perpetual disclosure makes the *effective policy* and the mechanisms for changing policy more visible. This visibility is by analogy to the mute button on a conference room telephone: it is a control point that lets room users quickly shut off the audio stream for a private conversation; its colored light makes it quick to find and perpetually discloses mute status.

Show Widget Allow (SWA)
When sensor data is requested, the widget must appear within the screen real estate associated with the requesting application. Sensor data flows to the application so long as display of the widget is entirely unobstructed.

Show Widget Deny (SWD)
When sensor data is requested, the widget must appear within the screen real estate associated with the requesting application. Sensor data may not flow to the application without a user-initiated policy change.

Alas, adding disclosure to existing allow and deny policies does not remove the tension between usability and privacy when choosing a default. When an application requests access to a sensor, a default-allow policy (such as SWA) will allow data to be collected without first checking if this use instance is appropriate. Thus, a default allow policy would allow a video-chat application to turn on the user's camera in the middle of the night. On the other hand, default-deny policies hassle the user for authority every time an application requires access to one or more sensors.

4.1 SWAAID

The sensor-access widget goes beyond improving existing policies: It enables a new default policy that provides privacy without sacrificing usability. We call this policy *Show Widget and Allow After Input and Delay* (SWAAID). SWAAID is designed to protect against unwanted sensor access without requiring user interaction to grant user-desired access. SWAAID aims to make passive input devices (sensors) work similarly to active input devices (keyboards, mice, touchscreens), wherein an application cannot receive sensitive data without the user deliberately directing input at the application.

Show Widget, Allow After Input and Delay (SWAAID)

Visibility The widget appears unobstructed within the screen real estate associated with the requesting application. If the widget becomes obstructed the flow of data immediately stops and all requirements for turning the flow back on are reset.

Active input The runtime environment has received an active user input event (key press, mouse movement, or screen touch) directed toward the application since the visibility requirement was met.

Waiting period A waiting period (e.g. a five-second delay) has passed since both the visibility and active input requirements were met. The user may choose to terminate the waiting period to start sending sensor data earlier (e.g. by left-clicking on the widget).

During the waiting period the widget may overlay its animation with a countdown of the time until the application will have access, as illustrated in Figure 2. So long as the user doesn't take action to deny access, the application gains real-time access to sensor data when the waiting period expires.

The visibility requirement, shared with *Show Widget and Allow*, ensures that grant of access to the sensor data is disclosed to the user. The waiting period requirement is intended to give the user sufficient time to notice that pending sensor data access to be granted and, if she deems this undesirable, sufficient time to turn it off. The combination of the active input and waiting period requirements ensures that an application cannot obtain access to a sensor unless there is some indication that the user is paying attention to the application (the active input) and has had sufficient time to decline the access request. These requirements close the input/output loop to make passive input devices (sensors) work more like active input devices. They protect against attempts to access sensors when the user is absent or her attention is outside of the application. For example, these requirements prevent a mobile phone application from requesting new sensor access while the phone is pressed against the user's ear or in the user's pocket.

A well-behaved application might willingly release access to a sensor, such as releasing the microphone after a VoIP call ends. Under the SWAAID policy, such an application would need to re-establish permission for the next call, but the application builds user trust by using the sensor only when required.

A malicious application might simultaneously request access to more sensors than a user can decline during the waiting period. One way to address this threat is to provide a disable-all-sensors option when the user sets an individual policy. Alternatively, when the countdown periods for two or more widgets overlap, their individual countdown periods can be set to the sum of their original periods.

A malicious application might try to occlude the widget to prevent the user from noticing its appearance or continued presence. For example, it might cause an apparently-unrelated pop-up advertisement to appear conveniently over the widget part of its primary display. This is the reason the visibility requirement specifies that the widget must "remain unobstructed": as soon as such a dialog even partially obscures the widget, the application ceases receiving the data stream. Since you can't see the data stream, 'they' shouldn't be able to get it.

4.2 Additional policy options

We recommend systems use a default policy of *Show Widget and Allow After Input and Delay* (SWAAID) offering users easily-accessible knobs to reduce access down to *Show Widget and Deny* (SWD) or up to *Show Widget and Allow* (SWA) for a given application.

Those users who prefer a more conservative policy can opt for a default of *Show Widget and Deny* (SWD). SWD policy is similar to that of Internet Explorer's yellow "information bar", introduced in March 2004 [20], which appears when a pop-up, software installation, or other likely-undesirable action is prevented. Internet explorer applies the safe policy (denying access) by default and the information bar provides a non-modal notification to the user and a mechanism through which the user can override the default policy to allow the action. Unlike the information bar, combining a default-off policy with widgets gives the user a more concrete view of the consequences of changing policy; she can see a representation of the stream of data that will be revealed if the policy is changed.

Less common policies on the outer edges of the policy spectrum (see Figure 9) can be applied by users on an as-needed basis to cover such cases as benign background applications or useful applications that overzealously request sensor access. For background applications that the user trusts with sensor data at all times, the *Hide Widget and Allow* (HWA) policy is suitable. For example, a user may prefer to run a life-blogging application in the background, with persistent, invisible access to the camera and microphone. Broadly, we argue that SWAAID is the best initial policy; permanent permission (HWA) should be rare, applied for those few applications that require it. Devices that support use by non-owners who may not have set policy, and who may have personal data at risk (e.g. students using school-provided laptops) should make it easy to discover that sensor-access widgets have been hidden.

At the other extreme, a user may find a text chat application useful despite its annoying requests for access to the user's camera access on every invocation. Here, a policy of *Hide Widget and Deny* (HWD) allows the user to permanently deny camera access while still taking advantage of the text chatting feature.

One way to make users more comfortable with looser access policies is to combine them with proxies that reduce data sensitivity before it reaches the application. For example, a proxy reads accelerometer data and degrades it to just the level required so the application can decide whether to present a portrait or landscape UI.

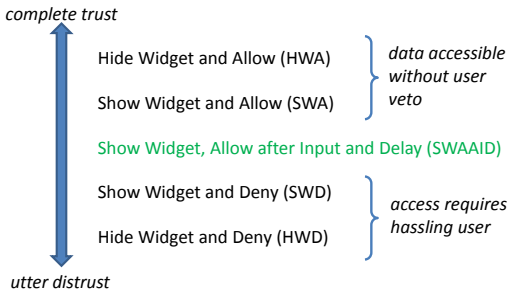


Figure 9: The SWAAID policy allows users of partially-trusted applications to allow sensor access by default, yet veto inappropriate accesses before sensor data is exposed to the application.

5 Evaluation proposal

Access control solutions should be evaluated on both their impact both to security and to usability.

Potential usability impacts of sensor-access widgets include loss of screen real-estate, user distraction, user annoyance, time and effort required to interact with the UI, and the delayed or denied access to sensor data when access was desired. The loss of screen real-estate can be measured concretely in terms of pixels. Conventional user studies can help quantify such usability factors as distraction, annoyance, time, effort, and access delays or denials.

We posit that users will not find sensor-access widgets distracting in the cases where sensor access is desired, because the data will simply mirror the environment around them. Telephone users tune out sidetone, the redundant sound of their own voice [16], and in fact modulate their voices based on the sidetone cue. Videoconferencing users tune out the small images of themselves that allow them to ensure they remain within the view of their conversation partners.

For application access to sensors, we would want to measure the likelihood that an application could obtain access to a sensor in an instance that the user deemed undesirable either ex-ante (before the event) or ex-post (after the event). Sensor-access widgets attempt to bring users' ex-ante beliefs of whether access is desirable closer to their ex-post beliefs by illustrating to users what it is they will be exposing should access be granted. However, sensor-access widgets cannot tell users how applications may use (or misuse) data. Furthermore, failings in an access control mechanism may allow an application to gain access to sensor data even if a user's ex-ante preference is to deny access. Access control failures may occur when an application issues a new request for access, or when an application continues receiving access after the user no longer deemed access desirable.

When users ex-ante preferences are to deny an application access to a sensor, the security of sensor-access

widgets with the *Show Widget and Allow After Input and Delay* (SWAAID) policy rests on the hypothesis that users will notice sensor-access widgets when the application's use is inappropriate, and do so with sufficient time to take action. Prior work by Egelman et al. warns of the risks of habituation on user's attention to warnings they should find alarming [6]. It is thus important to test whether habituation to sensor-access widgets under legitimate circumstances will lead users to ignore them in illegitimate ones. Increased exposure to sensor-access widgets may have security benefits as well as costs. Increased familiarity may help users understand how and when sensor data is flowing to applications, potentially increasing the likelihood that users will experience alarm when an undesired access is imminent. The fact that sensor-access widgets indicate the presence of risk makes them more likely to receive users' attention than those that indicate the absence of risk, such as the poorly-understood and heeded browser padlock icon [5, 19].

Testing our security hypothesis poses two challenges. First, we must ensure participants are fully habituated to widgets from sufficient use in cases where they deem sensor access desirable. Second, we must construct a test environment in which participants believe they have something to lose. Prior research has illustrated the loss of ecological validity that results when participants in security studies do not have as much to lose as their real-world counterparts, such as studies with designs that use role-playing [19]. Habituation concerns would seem to rule out a single-session study. Ethics prevent researchers from requiring laboratory participants to engage in the types of behavior that would yield the greatest concerns for sharing sensor data outside a laboratory environment. One possible study design would be to deploy the feature to a subset of users along with a test infrastructure. The test infrastructure would, after a sufficient habituation period, insert fake requests for sensor-access and gauge the user response. It might then reveal that the request was purely for research purposes and ask users to explain the choices they made.

Such a study might include:

1. a control group (using an existing allow/deny all interface),
2. a sensor-access widget interface that forces the user to make a decision, then hides the widget (differing from the control group only in the interface mechanism),
3. a sensor-access widget interface with a default *Show Widget and Deny* (SWD) policy, and
4. a sensor-access widget implementing *Show Widget, Allow After Input and Delay* (SWAAID).

For an in-the-wild deployment enabling a large number of participants, the treatment groups could be further subdivided by sensor type: camera, microphone, accelerometer, geolocation, and thermometer. They could also be subdivided by type of attacks: a request for access from an application that should not need access to a sensor (such as camera access for a news site) versus access by an application when semantics no longer demand it (microphone access after a phone call).

6 Limitations and future work

The sensor access widgets we have described are intended to help users understand what information is passing from sensors to applications, but do not help users understand why this information may be dangerous to disclose. For example, two individuals who associate with each other in secret may not understand that revealing accelerometer data will allow a third party to determine that they are in the same vehicle. Finding ways to communicate the risks of disclosing sensor data, especially when these risks are not intuitive to users, is an important area for future work.

Sensor access widgets may also fail to represent sensor data well enough to expose its sensitivity. For example, a camera access widget may show a near-black image of a dark room, while an attacker may be able to process the image to reveal objects not seen in the widget. Image processing techniques may mitigate but not eliminate these limitations. For example, a low-resolution widget may not convey just how many details of the environment are exposed by the camera. Similarly, a microphone-access widget that shows little activity due to the absence of high-amplitude sounds may lead a user to believe no data is being collected, when in fact low-amplitude sounds such as keystrokes may be collected. Careful design is required to ensure that the data representation in the widget automatically scales with average amplitude.

It may be difficult to pack many sensor access widgets into a small display, for applications that use several at once. Enumeration of real applications may indicate that this is not a common case, or further work may suggest how to display naturally group or prioritize sensor access widget displays.

If devices are prone to accidental user input, malicious applications may be able to subvert SWAAID, and other policies conditioned on active input, by requesting sensor access at times when accidental input is likely. For example, phones left in pockets or that treat ear contact as input may allow applications to gain access to sensor data when the user is not paying attention. As accidental input causes myriad other problems when it reaches applications, this is an important UI problem to solve independent of its security implications.

Finally, applications may be able to spoof sensor access widgets. An malicious application could create fake sensor access widgets, though without access to the right sensors it might be not create a believable match with the user's environment. However, the utility of fake sensor access widgets is limited as they would not actually be able to grant access to sensors. The application could present a flood of sensor access widgets to desensitize the user to them when real ones appear. It could create fake sensor access widgets at the same time a real one appears in order to make it hard for the user to find the right one to turn off during the waiting period. However, both these attacks send strong signals that the application is malicious and might inspire the user to quit the application altogether rather than try to turn of individual sensor access requests.

More realistically, a malicious application might create a fake sensor access widget in order to convince the user that the portion of the screen containing the fake widget is controlled by the OS. The application might then created a spoofed interface in which the sensor access widget asks the user to install a new application or change a system configuration. For example, it might justify the need for the user to perform these actions by claiming the sensor driver needed. One might attempt to address such attacks, which are endemic to all OS control mechanisms, by ensuring that controls only appear in platform-only screen real estate (e.g. the 'chrome' in browser platforms). We have chosen not to make this a requirement for sensor access widgets because studies have shown that few users actually understand the concept of chrome [5].

7 Conclusion

We introduce the *sensor-access widget* which provides perpetual disclosure of sensor data, communicates the effective policy regulating applications' access to that data, and provides an easily-discoverable control point for changing this policy.

The sensor-access widget also enables a sensor access control new policy, *Show Widget and Allow After Input and Delay*. SWAAID provides interaction-free access to well-behaved applications, while still preventing disclosure to misbehaving applications. SWAAID makes passive input sensors act more like active input devices, such as keyboards and mice, requiring the user to direct input at an application before the application can receive sensitive sensor data.

8 Acknowledgments

The authors would like to thank John Douceur, Jeremy Elson, David Molnar, Alex Moshchuck, and Helen Wang for comments on drafts. We thank Jaeyeon Jung (Intel Labs) for comments and screen images.

References

[1] Adobe Systems Incorporated. Flash player help: Privacy settings. *Archived on March 15, 2010.* http://macromedia.com/support/documentation/en/flashplayer/help/help09.html.

[2] Android Development Team. Manifest.permission, July 2009. *Archived on March 16, 2010.* http://developer.android.com/reference/android/Manifest.permission.html.

[3] Apple Inc. iPhone OS reference library: UIImagePickerController class reference. *Archived on March 22, 2010.* http://developer.apple.com/iphone/library/documentation/UIKit/Reference/UIImagePickerController_Class/UIImagePickerController/UIImagePickerController.html#//apple_ref/occ/cl/UIImagePickerController.

[4] Crispin Cowan, Steve Beattie, Calton Pu, Perry Wagle, and Virgil Gligor. SubDomain: Parsimonious security for server appliances. In *Proceedings of the 14th USENIX System Administration Conference (LISA 2000)*, 2000.

[5] Rachna Dhamija, J. Doug Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the 24th SIGCHI Conference on Human Factors in Computing Systems (CHI 2006)*, pages 581–590, New York, NY, USA, 2006. ACM.

[6] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems (CHI 2008)*, pages 1065–1074, New York, NY, USA, 2008. ACM.

[7] Engadget. Live from Apple's iPhone OS 4 event! http://www.engadget.com/2010/04/08/live-from-apples-iphone-os-4-event/.

[8] Elizabeth Fiedler. Student sues over alleged webcam spying. *All Things Considered*, National Public Radio, February 24 2010. http://www.npr.org/templates/story/story.php?storyId=124043452.

[9] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *Proceedings of the 6th Usenix Security Symposium*, 1996.

[10] James Gosling and Henry McGilton. The Java language environment: A white paper. Sun Microsystems, May 1995, http://java.sun.com/docs/white/langenv/.

[11] Ian Hickson. HTML Device: An addition to HTML. W3C Editor's Draft dated March 5 2010, http://dev.w3.org/html5/html-device/.

[12] Microsoft Corporation. MSDN: Silverlight developer center: Silverlight 4 rc: Security. *Archived on March 16, 2010.* http://msdn.microsoft.com/en-us/library/cc972657(VS.96).aspx.

[13] Microsoft Corporation. Understanding and configuring user account control in Windows Vista. *Archived on March 16, 2010.* http://technet.microsoft.com/en-us/library/cc709628(WS.10).aspx.

[14] Microsoft Corporation (UAC, Kernel, and Security program managers). Engineering Windows 7: User Account Control (UAC) – quick update, January 15 2009. *Archived on March 16, 2010.* http://blogs.msdn.com/e7/archive/2009/01/15/user-account-control-uac-quick-update.aspx.

[15] OpenAjax Alliance. Mobile device APIs style guide. http://www.openajax.org/member/wiki/Mobile_Device_APIs_Style_Guide.

[16] Naotoshi Osaka, Kazuhiko Kakehi, Satori Iai, and Nobuhiko Kitawaki. A model for evaluating talker echo and sidetone in a telephone transmission network. *IEEE Transactions on Communications*, 40(11), November 1992.

[17] Niels Provos. Improving host security with system call policies. In *Proceedings of the 12th Usenix Security Symposium*, pages 257–272, August 4–8 2003.

[18] Robert W. Reeder, Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, Kelli Bacon, Keisha How, and Heather Strong. Expandable grids for visualizing and authoring computer security policies. In *Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems (CHI 2008)*, pages 1473–1482, New York, NY, USA, 2008. ACM.

[19] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 51–65, Washington, DC, USA, May 2007. IEEE Computer Society.

[20] Tony Schreiner. IE in XP SP2 (part 2): Information Bar - Stopping the modal dialog madness, March 21 2004. *Archived March 17 2010.* http://blogs.msdn.com/tonyschr/archive/2004/03/21/93551.aspx.

[21] Joseph Tanfani. Two Lower Merion school district IT workers placed on leave. The Philadelphia Inquirer, March 4 2010. http://www.philly.com/philly/news/breaking/86444922.html.