Making Policy Decisions Disappear into the User's Workflow

Alan H. Karp, Marc Stiegler

HP Laboratories
HPL-2009-341

Keyword(s):
Usable security

Abstract:

Complaints of security interfering with getting work done are commonplace. They often arise when users are distracted from their tasks to make policy decisions. We have identified what is missing from earlier security interaction designs that leads to these interruptions. Explicitly representing policy decisions in the user interface and pro-viding controls for changing those policies has allowed us to reliably infers users' desired policy decisions from actions they take to get their work done. This paper describes the underlying principles and how they resulted in an interaction design that never interferes with the user's work.

External Posting Date: October 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: October 6, 2009 [Fulltext]

© Copyright 2009 Hewlett-Packard Development Company, L.P.

Making Policy Decisions Disappear into the User's Workflow

Alan H. Karp
Hewlett-Packard Laboratories
Palo Alto, CA
alan.karp@hp.com

Marc Stiegler
Hewlett-Packard Laboratories
Palo Alto, CA
marc.d.stiegler@hp.com

ABSTRACT
Complaints of security interfering with getting work done are commonplace. They often arise when users are distracted from their tasks to make policy decisions. We have identified what is missing from earlier security interaction designs that leads to these interruptions. Explicitly representing policy decisions in the user interface and providing controls for changing those policies has allowed us to reliably infers users' desired policy decisions from actions they take to get their work done. This paper describes the underlying principles and how they resulted in an interaction design that never interferes with the user's work.

Author Keywords
Usable security

ACM Classification Keywords
D.4.6 **[Security and Protection]** Access Controls; H.5.2 [**User Interfaces**]: User-centered design

INTRODUCTION

In spite of many years spent including security early in the design process, most people agree with the statement, "There is an inevitable tension between security and usability." Clearly, that alone isn't enough to avoid the tension. Either the tension is inevitable, or something is missing in the way security is included in user interactions.

The user interfaces we encounter daily often interrupt our work to ask us to make policy decisions. Open this dangerous file type? Connect to this server? Grant access to that file? The interaction designer is asking those questions because only the user is in a position to know what policy to apply. Fortunately, while that observation is true, we don't need to ask the user. The key contribution of this paper is to present principles for interaction designs that make it possible to infer the user's desired policy without asking.

Submitted to *CHI 2010,* April 10–15, 2010, Atlanta,Georgia USA.

MAKING SECURITY DISAPPEAR

There are three dimensions to avoiding the need to trade usability for security – Information, Expressiveness, Control. First, we need to provide users with sufficient information to make intelligent policy decisions. If we don't, they are likely to be unhappy with the result of having made policy decisions without being able to understand their implications. Second, we need to support the modes of sharing that users need to get their work done. If we don't, they will find the workarounds required to do their jobs an impediment. Finally, we must provide an interaction design that lets users make these decisions without interrupting their work. Otherwise, they won't like having to focus their attention on something extraneous to the task at hand.

The systems in common use today get a low score on each axis of this three dimensional space. Windows will warn you that a spreadsheet contains a macro that could be dangerous, but it doesn't tell you what harm that macro might do, nor does it tell you what benefit it provides. The access control mechanisms used by Linux limit the policies we can express. For example, it is impossible to delegate a read only permission if you don't also have the authority to update the access control list, in which case you effectively have write authority. In the third dimension, the controls to express a policy decision are almost always independent of the application. Granting another user permission to read one of your files requires leaving your application to change the access control list. We can do better.

Dimension 1: Information

There are 10 guidelines for building systems that enable users to make wise policy decisions [13], most of which are related to giving users the information they need. They are just common sense, such as, "Maintain accurate awareness of others' authority relevant to user decisions." and "Present objects and actions using distinguishable, truthful appearances." Surprisingly, most of the systems in common use implement none of them. For example, a file dialog box on a conventional desktop environment doesn't tell the user what application it is attached to, leaving users to guess their exposure.

The challenge for the interaction designer is finding a way to represent this information in a manner that is meaningful in the application context. Consider the file dialog box. We can let the user know which application is asking for the file access by putting its name in the title bar. But what if there are two instances running, one on the local machine and one from a remote desktop? Listing the process ID provides the information, but not in the application context. A better approach is to use graphic elements to connect the dialog box with the requesting window.

Dimension 2: Expressiveness

People need to cooperate to get their jobs done. Often this cooperation is with others, but quite frequently users share with themselves across both applications and machines. The security community focuses on building walls. Not surprisingly, the mechanisms they choose often make cooperation hard. When these mechanisms block cooperation, security makes it harder for people to get their work done. Users become frustrated when the system doesn't let them express what to them are simple patterns of cooperation.

There are six aspects of sharing we rely on in the physical world that we have learned to live without online.[1]

- Dynamic: No third party needed to approve a change.
- Cross-domain: No one party is in charge.
- Attenuated: You can have a dollar, but not my wallet.
- Chained: A delegated right can be further delegated.
- Composable: Combine rights from different sources.
- Accountable: Who gave a right as well as who used it.

Each time the interaction design blocks one of these modes of sharing, the user must find a workaround. Despite the abundance of collaboration software products, it is likely that email is so widely used for collaborating on documents because it supports all six aspects of sharing.

Examples of systems that don't support rich sharing abound. One of us has a spouse who manages the family accounts, but that spouse can't get to the employee's electronic pay stub because it's behind the company firewall (cross-domain). HP managers share their Windows domain credentials with their admins, so the admins can take care of minor budgeting and personnel matters (attenuated). The implementation of simple service chaining done for the US Navy can achieve either the desired functionality (chained) or the required security (composable), but not both at the same time [5]. Microsoft Live Mesh [8] supports two of the six aspects (dynamic, cross-domain), explicitly does not support two (attenuated, accountable), and allows two only if users are willing to share essentially all their rights (chained, composable).

[1] We have been surprised by our inability to find references for such a list in either the computer science or the sociology literature.

Dimension 3: Control

Users must be able to express their policy decisions in order to get their work done. One approach, dubbed by some the "Vista UAC nightmare," is to open a dialog box for every decision, which leads to *Just-Say-Yes* conditioning [4]. Such security by admonition may absolve the software vendor from blame when something goes wrong, but it interferes with the user's work, often without enforcing the user's policy.

CapDesk [11] demonstrates that we can do better. It uses capabilities (See the Appendix.), because that is the only mechanism that has been shown to support all six aspects of sharing, while enforcing access control at extremely fine granularity. CapDesk adapts the fundamental property of capabilities, combining designation with authorization, to the interaction design by using acts of designation to denote the desired authorizations. For example, in CapDesk dragging the icon for a file onto the icon for an editor designates that the user wants to use that editor with the file. CapDesk infers that the user wants to grant the process running the application the authority to read and write the designated file. The result is that the user is not distracted from the task of editing the file to specify the desired policy.

One shortcoming of CapDesk, which is imposed by the desire to emulate a standard desktop user experience, is that there is no place within the application context to represent the policy decisions that have been made or to put controls to modify those decisions. Some of them are represented implicitly. For example, the user knows an application has permission to edit a file by looking at the application window to see what file it is editing. The user can revoke that access only by closing that instance of the editor.

POLICY DECISIONS AS CONTROLLABLE OBJECTS

Policy decisions are subject to change, and we don't want to interfere with the user's work when they do. CapDesk replicates an existing user experience, that of a conventional desktop. That constraint limits the policy decisions that can be expressed directly in the user interface. We had more freedom with the design of SCoopFS [6],[2] a tool for collaborating on a document. We used this freedom to design an interaction that lets us infer users' policy decisions as well as changes to those previously made.

SCoopFS was designed to get a high value on the information axis, and achieved 8+ out of 10. In addition, SCoopFS uses capabilities because we know that lets us support all six aspects of sharing, giving it a high score on the expressiveness axis. Like CapDesk, SCoopFS uses acts of designation in the user interface to infer acts of authorization. However, SCoopFS scores higher on the control axis by representing policy decisions as elements in the application's user interface and providing application specific mechanisms for manipulating these policies.

[2] Simple Cooperative File Sharing, the "F" is silent.

We have distilled what needs to be done to make user's policy decisions disappear into their workflow into the following four principles.

1. Every object separately controllable by the user should be represented in the application user interface by a capability that is uniquely distinguishable to the user.
2. Every possible policy decision on an object should appear as a unique affordance in the application user interface.
3. Every policy decision the user has made should be represented in the application user interface by a capability that is uniquely distinguishable to the user.
4. Every possible change to a previously made policy decision should appear in the application user interface as a unique affordance.

It is the support for the last two of these principles that distinguishes SCoopFS from earlier work, such as CapDesk.

These principles separate the concepts of resources, operations on resources, policy decision, and operations on policy decisions. So, a file is a "separately controllable object," granting read permission on that file is a "policy decision" accomplished with some affordance in the user interface, and revoking that grant is a "change to a previously made policy decision" also accomplished with some affordance in the interface. If the interaction design follows these principles, every action taken in the user interface that affects policy will be unique. Since there is no ambiguity in determining the user's intent, there is no need to interrupt the user's work with a question.

Figure 1 shows an application of the last two of these principles. The second line in the grid shows that "Me," the SCoopFS pseudonym for the user, granted read and write permission (the double headed arrow under "Mode") for the file "decideRightsSetup.zip" to his Pal "AlanXP" around Noon on November 25. The grayed out buttons show the actions that the user can take on this sharing relationship. In this case they are only "Unshare" and "Snapshot Share," which makes a private copy of the file in its current state.

Figure 1 also shows how we use filtering to simplify dealing with large numbers of policy decisions. Clicking the "Show Shares" button when an item is selected results in a view showing all sharing relationships involving that file. Another view shows all the sharing relationships with a given Pal.

We have shown how these principles apply to access control decisions, but there are different kinds of policy. For example, deciding whether or not to encrypt communications is often left to the user. Our approach is to make the communication channel a "separately controllable object" and include separate buttons for sending encrypted or not. It is up to the interaction designer to decide if that decision is best made once per channel or once per message. Making the message itself a separately controllable object lets the interaction designer supply controls for other policies, such as digital signing.

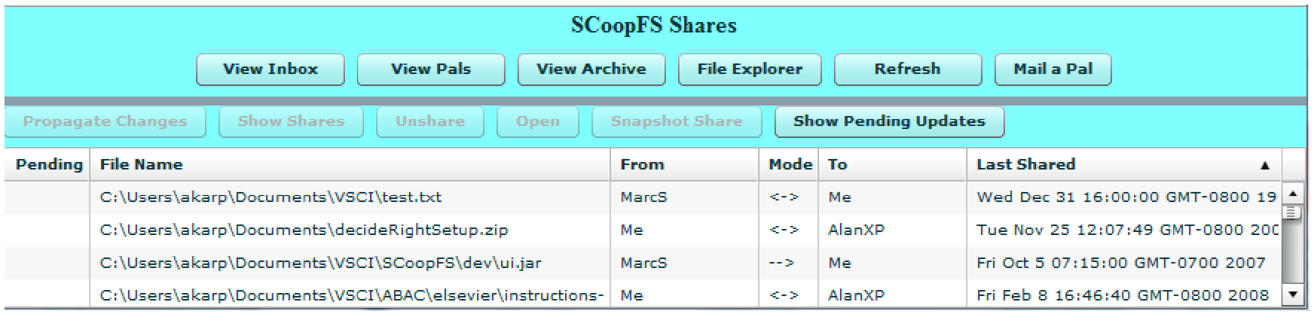
The danger with all these affordances is an overly complex user interface. How to avoid this problem depends on the application space. CapDesk makes the most common policy the default and requires a separate action, such as a right click instead of a left click, to denote an exception. SCoopFS attaches some properties to the communication channel and others to the sharing relationship as well as eliminating some choices by always encrypting messages and authenticating the servers. These design decisions limit the user's options, but in ways that make sense for the application domain.

An interface that lets the user specify dozens of actions on each of millions of objects will be complex. That complexity is unavoidable. However, following the guidelines presented here avoids increasing the perceived complexity by interrupting the user's work to make policy decisions.

RELATED WORK

There are numerous guidelines for designing for usability, *e.g.*, [3], recommending early focus on the user, measurement, and iterative design. What's missing is guidance specific to security and details of what elements must be presented to the user. Other work, *e.g.,* [1], makes interruptions of the user's work less onerous, while our goal is to avoid those interruptions entirely. In addition, that work uses a separate GUI to represent the policy decisions, while we present those decisions in the application context.

A key goal of Chameleon [7] is not to "interfere too much



Pending	File Name	From	Mode	To	Last Shared ▲
	C:\Users\akarp\Documents\VSCI\test.txt	MarcS	<->	Me	Wed Dec 31 16:00:00 GMT-0800 19
	C:\Users\akarp\Documents\decideRightSetup.zip	Me	<->	AlanXP	Tue Nov 25 12:07:49 GMT-0800 20C
	C:\Users\akarp\Documents\VSCI\SCoopFS\dev\ui.jar	MarcS	-->	Me	Fri Oct 5 07:15:00 GMT-0700 2007
	C:\Users\akarp\Documents\VSCI\ABAC\elsevier\instructions-	Me	<->	AlanXP	Fri Feb 8 16:46:40 GMT-0800 2008

Figure 1: Shares view.

with the primary task" nor "intrude on the ordinary activities that people want to perform." Our principles go beyond those goals by not interfering at all with the primary task and never intruding on the user's tasks.

Sometimes we can avoid the perception of interfering with the user's task by changing when certain actions are taken. Groove [9] asks users to determine the trustworthiness of a message sender's authentication when the message is received, which interferes with the user's task of reading the message. SCoopFS makes this authentication step part of a different user task, that of establishing a new relationship. It's the same work, but it's done at a different time so that it becomes part of the user's workflow.

CONCLUSION

We didn't start out by dreaming up a set of principles and building tools using them. Instead, we built tools and discovered that we weren't bothering users with questions unrelated to what they want to do. The articulation of the principles came from asking ourselves how we did it, a form of *post-hoc* synthesis [2].

It may be that we did too good a job. One of our users said, "This tool would be a lot better if it had some security. Is there any way I can turn some on?" While that question shows we achieved our goal, it also indicates that achieving our goal is not enough. While "security reality" is necessary, the "feeling of security" is important, too [10]. Apparently, people are so used to security interfering with their work that they don't feel secure when it doesn't. Overcoming this bias is a challenge.

We have demonstrated that giving the user sufficient information to make wise decisions, adequate support for sharing, inferring policy decisions from acts of designation, representing those decisions, and providing meaningful controls, all in the application context, allowed us to build an application that did not trade off usability for security. The primary contribution of this paper is to point out that making policy decisions explicitly controllable objects makes it possible to give the user the desired control without needing to leave the task at hand.

ACKNOWLEDGMENTS

We thank Jhilmil Jain for guiding the writing of this paper.

REFERENCES

1. Cao, X. and Iverson, L. Intentional access management: making access control usable for end-users. Proc. 2nd Symposium on Usable Privacy and Security, Pittsburgh, Pennsylvania July 12 - 14, (2006)
2. Cockton, G. Getting There: Six Meta-Principles and Interaction Design. CHI 2009, Boston, MA. (2009)
3. Gould, J. and Lewis, C. Designing for Usability: Key Principles and What Designers Think. CACM, 28(3), 300-311, (1985)
4. Gutman, P. The Psychology of Usability Security, http://www.cs.auckland.ac.nz/~pgut001/pubs/usability.pdf
5. Karp, A. H. and Li, J. Solving the Transitive Access Problem for the Services Oriented Architecture, HP Labs Technical Report, HPL-2008-204R1 (2008)
6. Karp, A. H., Stiegler, M., Close, T., Not One Click for Security, HP Labs Tech Report, HPL-2009-53 (2009)
7. Long, C. A. and Moskowitz, C. Simple Desktop Security with Chameleon. In *Security and Usability: Designing Secure Systems That People Can Use*, by Lorrie Faith Cranor and Simson Garfinkel, 247-273. Sebastopol, CA: O'Reilly Media, Inc., (2005)
8. Microsoft Corp. What's inside Live Mesh? https://www.mesh.com/Welcome/features/features.aspx.
9. Moromisato, G., Boyd, P., and Asthagiri, N. Achieving Usable Security in Groove. In *Security and Usability: Designing Secure Systems That People Can Use*, by Lorrie Faith Cranor and Simson Garfinkel, 247-273. Sebastopol, CA: O'Reilly Media, Inc., (2005)
10. Schneier, Bruce. Reconceptualizing Security. *LISA '08: 22nd Large Installation System Administration Conf.* San Diego: Usenix, 2008.
11. Stiegler, M. A Capability Based Client: The Darpa-Browser. http://www.combex.com/papers/darpa-report/darpaBrowserFinalReport.pdf (2002)
12. Stiegler, M. Rich Sharing for the Web. HP Labs Technical Report, HPL-2009-169 (2009)
13. Yee, Ka-Ping. "Guidelines and Strategies for Secure Interaction Design." In *Security and Usability: Designing Secure Systems That People Can Use*, by Lorrie Faith Cranor and Simson Garfinkel, 247-273. Sebastopol, CA: O'Reilly Media, Inc., (2005)

APPENDIX: A BRIEF HISTORY OF ACCESS CONTROL

When timesharing was first introduced, there was a debate about how to isolate users from each other while allowing them to specify a sharing policy. One approach assigns an identity to each user and labels resources to indicate which actions each user can take. That's the approach that won and is what we use today on all systems in common use, Windows, Mac, Linux, *etc*. A fundamental flaw in that model is the way it interferes with sharing, making it the primary reason security gets in the way of doing work.

The other approach is to provide each user with a list of references, a capability list, each entry of which represents a specific action on a particular resource. Possession of a capability represents the right to take that action on that resource. Users specify the actions they wish to take by denoting the relevant capability from the list. The key feature of a capability is that it combines designation with authorization. The way that capabilities can be passed around supports all six aspects of sharing.