Secure Interaction Design and the Principle of Least Authority

Ka-Ping Yee
Group for User Interface Research
University of California, Berkeley
ping@zesty.ca

ABSTRACT

The security of any computer system that is configured or operated by human beings critically depends on the information conveyed by the user interface, the decisions of the users, and the interpretation of their actions. This position paper puts forth some starting points for reasoning about security from a user-centred point of view. I rebut the common assumption that security and usability are always in conflict, propose a user model based on the subjective actor-ability state, and identify ten key principles for secure interaction design. I argue that adherence to the Principle of Least Authority is essential to usability goals for secure systems, and call for increased attention to this well-known security principle in the security community.

Keywords

Secure interaction design principles, intentional stance, actor-ability model, Principle of Least Authority.

INTRODUCTION

Researchers and designers of human-computer interaction face a tremendous challenge in the computer security arena: the pervasive assumption that any secure system must be a compromise between the conflicting goals of security and usability. This belief is so strongly held in some circles that it is not uncommon for security system designers to openly adopt an attitude of contempt toward their users. I find this unfortunate and counterproductive, and feel that it is vital for the security and usability communities to work together on these problems. To this end, my colleagues and I have developed a user model and a set of interaction design principles for secure systems.

The bulk of computer security research concerns methods for establishing software correctness – that is, assuring that software implementations will perform according to a particular specification in theory. However, I claim that today's most widespread and damaging security problems are failures to adequately meet user expectations, and have nothing to do with software correctness.

Saltzer and Schroeder's seminal paper [4] introduced a basic design principle for secure systems known as the principle of least privilege or least authority. It states that each system component should be granted the *least authority necessary to perform its function.* It has long been accepted that this simple principle is fundamental to security at the system level; it is important to recognize that it is also fundamental to usability for secure systems.

The remainder of this paper will address four questions:

1. Why is software correctness insufficient for security?
2. How can security and usability work together rather than against each other?
3. What user models and design principles can we employ to improve secure interaction design?
4. What is the relationship between the Principle of Least Authority and usability concerns for secure systems?

INSUFFICIENCY OF SOFTWARE CORRECTNESS

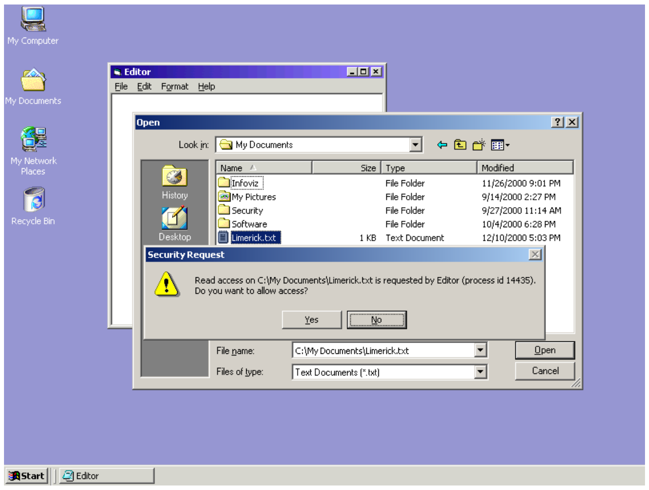
Plenty of work has been done on building and verifying software to correctly meet security policy specifications. However, the literature largely neglects the question of how these specifications come about in the first place. In order for a computer system to be secure in practice, the security restrictions must match human expectations of system behaviour. It is impossible to even define what "security" means without addressing user expectations.

Here is a high-profile example to support this argument. E-mail viruses have been among the most spectacular and damaging security problems in recent history. Yet many of these, such as the infamous "Melissa" and "Love Letter" viruses, involve no software errors whatsoever. In fact, these viruses depend on the correct operation of software in order to propagate. At no point in their propagation does any application or system software behaviour differ from exactly what its programmers would expect. Rather, the problem exists because the functionally correct behaviour is inconsistent with the user's desires and expectations.

ALIGNING USABILITY AND SECURITY

Because of the essential relationship between computer security and human expectations, it makes more sense for security and usability to be complementary rather than conflicting goals. A system that is more secure is more controllable, more reliable, and hence more usable. Conversely, a more usable system reduces confusion and is more likely to be secure. In general, security advocates and usability advocates both want the computer to correctly do what the user wants – no more and no less.

Here is a simple example of how this perspective can lead to better designs than the conventional wisdom. The "security confirmation" prompt is a common sight in user interfaces for secure systems. A typical conception of an interface for opening files in a text editor in a secure environment might look like this:



Look in: My Documents

Name	Size	Type	Modified
Infoviz		File Folder	11/26/2000 9:01 PM
My Pictures		File Folder	9/14/2000 2:27 PM
Security		File Folder	9/27/2000 11:14 AM
Software		File Folder	10/4/2000 6:28 PM
Limerick.txt	1 KB	Text Document	12/10/2000 5:03 PM

Read access on C:\My Documents\Limerick.txt is requested by Editor (process id 14435).
Do you want to allow access?

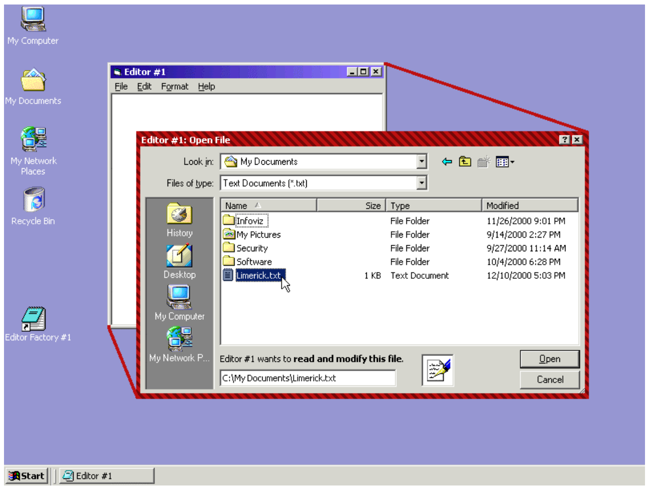
File name: C:\My Documents\Limerick.txt

Files of type: Text Documents (*.txt)

Figure 1. Conventional design for a secure text editor.

The reasoning behind such a design is that reading files is a privileged operation on a secure system, so it ought to be confirmed by the user before the editor is granted access. The consequential observation is that a security improvement has impeded usability by introducing an annoying extra step into the user's workflow.

However, if we step back and examine the situation from the perspective of user intentions, we can find a much better solution. Notice that the purpose of the file browser itself is to designate a file for reading. Therefore, in making a selection the file browser, the user has already precisely specified an expected privilege relationship: the editor should get read access to the selected file. All that is needed is for the type of access and the receiver of the access to be clearly identified, and for the file browser to be a system-controlled interface component that conveys just the selected access to the text editor:



Look in: My Documents

Files of type: Text Documents (*.txt)

Name	Size	Type	Modified
Infoviz		File Folder	11/26/2000 9:01 PM
My Pictures		File Folder	9/14/2000 2:27 PM
Security		File Folder	9/27/2000 11:14 AM
Software		File Folder	10/4/2000 6:28 PM
Limerick.txt	1 KB	Text Document	12/10/2000 5:03 PM

Editor #1 wants to **read and modify this file.**

C:\My Documents\Limerick.txt

Figure 2. Combining designation with authority. The file browser's special border marks it as a privileged entity.

The result is a design that is more secure and more clear, while remaining just as convenient for the user. A security specification has been combined with user's natural task into a single operation, as it should be.

The inconvenience of security confirmation prompts is often a result of separating a designation operation from an authority-manipulating operation. Many security problems can be avoided by *combining designation with authority*, as recommended by Norm Hardy's discussion of the Confused Deputy problem [3].

DEFINING THE PROBLEM

Here is one way to frame the problem of secure interaction design: a computer user interacts within a universe of programs and other users, often unpredictable and sometimes even adversarial. The user employs a *user agent* (such as a desktop shell or Web browser) to mediate interactions with this universe. How can we design the user agent to serve and protect the user's interests?

Notice that the scope of this definition avoids systems designed to balance the user's interests against conflicting interests. I believe that the ability to meet the user's interests is a prerequisite for the ability to balance multiple interests, that the simpler problem has not been solved, and therefore that we need to solve that simpler problem first.

For example, digital copy restriction mechanisms often make media content harder to use. This is properly seen as a conflict between different groups of people, not a conflict between security and usability. Even in a hypothetical system that would manage this conflict, the ability for the user to accurately and conveniently express expectations and desires remains crucial to secure operation.

REQUIREMENTS FOR SECURE SYSTEMS

We can begin to address the problem by considering a set of requirements for a user to be able to use a computer system securely. In the tradition of computer security discourse, we will suppose that our user is called Alice. The requirements proposed here fall into two broad classes.

Firstly, if "unsafe" means a state where the computer would permit access contrary to Alice's wishes, what is necessary for Alice to keep her system safe?

i. The system should not become unsafe all by itself.

ii. Alice should be able to determine whether her system is in a safe state.

iii. Alice should be able to make her system safer.

iv. Alice should not intentionally or inadvertently make choices that cause her system to become unsafe.

v. Alice should not make choices that depend on abilities she does not have.

Secondly, what is necessary in order for Alice to be able to communicate effectively with her computer?

vi. In order to protect things that matter to her, Alice should be able to identify and distinguish those things.

vii. Alice should be able to express what she wants.

viii. Alice should know what she is telling her system to do.

ix. The system should protect Alice from being fooled, whether inadvertently or purposefully (by maliciously written applications).

THE ACTOR-ABILITY MODEL

I have proposed a user model of interaction with secure systems in terms of *actors* and *abilities*.

The suggestion is that, when they interact with computers, users model both application programs and other users of multi-user systems as independent actors. In Dennett's terminology, they model these things by adopting a *design stance* or *intentional stance* [2]. To adopt a physical stance is to predict behaviour according to physical laws; to adopt a design stance is to predict behaviour according to an understanding of the purpose for which an object was designed; and to adopt an intentional stance is to predict behaviour according to an understanding of the beliefs and intentions of a sentient entity. Here are some rough examples of how people apply these stances to the real world and analogously to the computer world (though it is now becoming common for users to speak about computer programs as if they were intentional):

	real-world example	*computer example*
physical stance	ball	text file
design stance	toaster	application
intentional stance	person	another user

The actor-ability model suggests that user expectations can be described in terms of the user's set of extant actors and the abilities associated with each of those actors. Note that although a computer system in reality may involve the interaction of many hundreds of software components, the user model is constructed in terms of actors as defined from the user's perspective. For instance, a single actor might be an instance of a running application program, which aggregates several software components and data files. The user is also an actor – the primary actor – in the user's own model.

Maintaining security is then a question of ensuring that (a) every other actor's true abilities are a subset of that actor's abilities in the user's current actor-ability state; and (b) the user's own abilities are a superset of the abilities in the user's current actor-ability state. The first condition ensures that other actors will not be able to take unexpected actions; the second condition ensures that the user will not expect to have abilities and find them missing.

DESIGN PRINCIPLES

The requirements stated above translate into a set of ten design principles for interaction design in secure systems, which I presented in a recent paper [5]. I list them here, but there is not enough space to go into each one in detail. Noted after each principle is the requirement it addresses.

1. *Path of Least Resistance.* The most natural way to do any task should also be the most secure way. (iv)
2. *Appropriate Boundaries.* The interface should expose, and the system should enforce, distinctions between objects and between actions along boundaries that matter to the user. (vi)
3. *Explicit Authorization.* A user's authorities must only be provided to other actors as a result of an explicit user action that is understood to imply granting. (i)
4. *Visibility*. The interface should allow the user to easily review any active actors and authority relationships that would affect security-relevant decisions. (ii)
5. *Revocability*. The interface should allow the user to easily revoke authorities that the user has granted, wherever revocation is possible. (iii)
6. *Expected Ability*. The interface must not give the user the impression that it is possible to do something that cannot actually be done. (v)
7. *Trusted Path*. The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf. (ix)
8. *Identifiability*. The interface should enforce that distinct objects and distinct actions have unspoofably identifiable and distinguishable representations. (ix)
9. *Expressiveness*. The interface should provide enough expressive power (a) to describe a safe security policy without undue difficulty; and (b) to allow users to express security policies in terms that fit their goals. (vii)
10. *Clarity*. The effect of any security-relevant action must be clearly apparent to the user before the action is taken. (viii)

These principles were developed through extensive discussions with designers and users of software intended to be secure. The set you see here is the most recent of several iterations of refinement during which the number of principles has varied from seven to eleven.

As recommendations, these principles might all appear to be quite obvious. In my opinion, this is no great deficiency. To my knowledge, the appearance of [5] at an international security conference is the first time that a comprehensive set of interaction design recommendations has been gathered, formulated, and presented to a security audience, and I consider that somewhat of a breakthrough. I hope at least that this has helped to open discussion on these matters. I am sure the principles will need further revision as we learn more over time.

Despite their apparent obviousness, I know of no deployed system that comes close to satisfying these principles. Attempting to satisfy them all appears to be a significant design challenge. However, I believe this challenge is not completely unrealistic: there is important ongoing research on a prototype desktop interface called CapDesk [1] that aims to meet most of the principles.

THE PRINCIPLE OF LEAST AUTHORITY

The importance of the Principle of Least Authority to the construction of secure systems cannot be overemphasized. From a systems point of view, transferring too much authority is dangerous because we are then forced to rely on the software to exercise authority only in the desired ways, and rigourously predicting the behaviour of software is extremely hard. It is better to execute software in an environment where authority is minimized to begin with.

From a usability point of view, there are even more reasons why authority should be minimized. Transferring too much authority forces the user to rely on the software to perform as advertised. Since in most cases the user has no access to the source code or the expertise to verify it, the user is forced to trust not only the good intentions but also the competence of the software author. Forcing this kind of trust is unacceptable for at least four reasons: (a) users are unlikely to have the resources to legally pursue the authors of harmful software; (b) regardless of what legal recourse a user may have, it can only take place after the damage has already occurred; (c) users are unlikely to have any way to identify which software is at fault; and (d) damage can be caused just as easily by a bug as by a malicious program.

Moreover, when too much authority is transferred at once, it becomes difficult for the user to ascertain what can go wrong or whether a particular action will be safe. The safety of the system comes to depend on a large amount of invisible state, which rapidly exceeds the user's short-term memory.

Unfortunately, popular operating systems have a great tendency to transfer enormous bundles of authority *en masse*. In fact, both Windows and Unix blatantly ignore the Principle of Least Authority every time they start any program, by implicitly transferring *every authority the user possesses* to the newly running process. This absolutely has to stop if we are to make any progress in creating secure, usable systems.

Microsoft's ActiveX certification mechanism is a specific example of the wrong approach to security. Rather than confining downloaded software so that it has limited authority to damage the system, it assumes that users will be able to establish trust relationships with software sources. It also assumes that all software will always perform exactly as its author intended, which as we all know is utterly absurd. The user only gets an all-or-nothing choice: either the user trusts the software with universal authority, or the software will not run at all. Given this option, most users choose to run the software, which leaves them no better off than if they had no security to begin with. The only difference is that Microsoft can now disclaim responsibility for any problem.

Compared to this, Java's security model is a move in the right direction, though its design is not yet able to support true software confinement as defined in the security community. However, the E language [1] *is* primarily designed on the Principle of Least Authority, and provides a very promising avenue for research into effective, secure, usable systems.

CONCLUSION

I have presented a user model and a set of design principles that I believe will be an interesting starting point for discussion. I hope we can continue to seek ways that security and usability goals can be harmonized in the design of secure systems.

Developing truly secure systems for real-world use requires a fundamental change in the way computer security experts and practitioners think about security. Although the Principle of Least Authority is well known and widely cited, most systems do not even try to enforce this basic requirement in practice. Given the increasing level of control handed over to computers in every aspect of today's society, I believe this to be one of the most important things we have to change in order to achieve the goal of a safer world.

REFERENCES

1. Combex. E and CapDesk: POLA for the Distributed Desktop (see http://www.combex.com/tech/edesk.html).
2. D. Dennett. *The Intentional Stance*. MIT Press (1987).
3. N. Hardy. The Confused Deputy. In *Operating Systems Review*, 22(4)36–38.
4. J. H. Saltzer, M. D. Schroeder. The Protection of Information in Computer Systems. In *Proc. IEEE*, 63(9)1278–1308 (see http://web.mit.edu/Saltzer/www/publications/protection/).
5. K.-P. Yee. User Interaction Design for Secure Systems. In *Proceedings of the 4th International Conference on Information and Communications Security*, Singapore, 2002.