Usable Set-up of Runtime Security Policies

Almut Herzog, Nahid Shahmehri
Dept. of Computer and Information Science
Linköpings universitet, Sweden
{almhe,nahsh}@ida.liu.se

Abstract

Setting up runtime security policies as required for firewalls or as envisioned by policy languages for the Semantic Web is a difficult task, especially for lay users who have little knowledge in the security domain. While technical solutions for runtime protection and advanced security policy languages abound, little effort has so far been spent on enabling users to actually use these systems to set up a security policy, and certainly not at runtime.

To start filling this gap, we give concrete and verified guidelines for designers that are faced with the task of delegating security decisions to lay users. We advocate, for example, that security policies be set up at runtime, not off-line, that the principle of least privilege be enforced and that alert windows be compact but still contain information about the consequences of a chosen action.

These guidelines have emerged from our own and others' research on usability and security. They are further strengthened through the implementation of the prototype JPerm, which follows our guidelines. JPerm is used for the runtime set-up of security policies for Java applications. Its specific design and evaluation are described in this work and serve as an illustration of the presented guidelines.

Keywords: security policy management, access control, usability, Java, application surveillance

1 Introduction

Setting up a security policy or security rules for a personal firewall, for application surveillance on one's computer, or for how one's web browser should interact with privacy policies of visited web sites, is a difficult task. It is technically difficult in the sense that lay users must have some grasp of technical terms, the limitations of the policy system, and policy syntax or available options. It is also difficult for users to accept the whole concept in the first place, because users can easily perceive security measures as an extra strain whose gain is not readily apparent.

Still, at least one security tool for setting up security policies at runtime has succeeded: Personal firewalls are on many people's personal computers and quite a number of non-expert users have come to appreciate and master them. But firewalls are not very complex in their runtime rule syntax. They will either allow or disallow a network connection based on some criteria of the connection type—typically port and host—and the name of the local application.

Research has envisioned many advanced security policy systems and languages for end users, ranging from runtime application rules, as seen in the Java runtime environment and rules for intrusion detection systems, to policy languages for trust negotiation (Seamons et al., 2002) and advanced access control (Herrmann and Krumm, 2001). So far, no usable end-user

interface has been presented for any of these advanced security controls. Thus, we are interested in studying whether users can handle more advanced security policy set-up than firewall rules and what is required from a graphical user interface for such security policy set-up.

The contribution of this work is consequently to present and discuss concrete guidelines for enhancing the usability and security of software that delegates security decisions to lay users and captures these user decisions as a security policy. The guidelines have emerged from pre-studies on how users want to and are capable of setting up runtime security rules (see section 2), from previous work on the usability of personal firewalls (Herzog and Shahmehri, 2007), from a usability study of a tool for off-line setting of a Java security policy (Herzog and Shahmehri, 2006) and from literature studies on usability and security. The validity of our guidelines is strengthened by a prototype implementation of a tool for setting up an access control policy for Java applications that follows these guidelines and that was received positively from users.

We chose Java and the set-up of Java security policies because Java is a language that supports runtime monitoring of security properties. But due to usability lapses (see our study (Herzog and Shahmehri, 2006)) and because of alleged slowness of the Java security mechanism, which is only partially true as shown in Herzog and Shahmehri (2005), this Java feature is seldom used. Consequently, our implementation fills a need in the Java community, but by choosing Java we also arrive at an extensible test bed for user interfaces for policy languages, because many policy languages—for example object-oriented Ponder (Damianou et al., 2001) or OWL-based policy languages (Kagal et al., 2003)—are implemented in or easily integrated into Java.

The paper is structured as follows: in the next section we report on two pre-studies, which together with our and others' work, have lead to a number of guidelines for applications that must ask their users for a security decision, and to a prototype implementation of an application monitor. The Java application monitor JPerm is briefly presented in section 3. Section 4 presents our guidelines in the light of JPerm and in contrast to other guidelines, which are presented in general in section 6. Section 5 reports on the evaluation of JPerm. Section 7 concludes and names future work.

2 Pre-studies

Two prestudies provided useful input about how users want to and can handle runtime security policies in the form of rules for access control to sensitive resources.

2.1 Study 1: Do users want application access control?

The first pre-study explored whether users were interested in application access control, what they would like to see controlled and how they want to interact with such an application.

22 students from social, technical and business programs completed a questionnaire-guided interview. The interviews dealt with the respondents' Internet activities, their security concerns, how they address them and which, if any, security-critical actions they would like to see monitored.

Respondents were engaged in Internet activities like searching, browsing, downloading files, e-mailing and to a lesser degree Internet banking, purchasing items over the Internet and using chat. 10 of the participants employed IP telephony, and 8 more respondents expressed that they are considering IP telephony as an Internet activity for themselves in the near future. All of the respondents had downloaded some software, most often updates and free software such as Acrobat Reader.

The perceived personal risk level was considered to be low to medium, with the typical explanation that nothing much had ever happened and/or that there was no crucial data on the Internet-connected computer. 9 respondents had never had any problems with malware, 9 respondents had problems with viruses, sometimes to the extent of having to reinstall their system; 8 respondents had experienced problems, usually a slow computer, with adware or spyware. Anti-virus software was the most popular defence mechanism against malware (18 of 22); 14 respondents knew that they had a personal firewall, 8 had anti-spyware software. A common strategy of defending against malware in unknown software was also to only download from known sources or to only download known products.

However, despite low risk levels and defence strategies in place, 20 respondents expressed that they would like to have application surveillance on their PC. Half or more of the respondents wanted alerts for network connections, file operations such as reading, modifying, creating, deleting, executing, and also alerts before an application starts controlling mouse, keyboard or screen and for the setting of environment or system variables.

When asked to suggest an alert or elements of an alert for a security-critical action of their choice, respondents described a dialog window with the typical components of what is happening, what that means, what one should do and where one can find more information and advice. The text in the initial window should be untechnical, but details should be readily available. A number of users also described the case of allowing an action too quickly and then having difficulties in revoking their grant and and said they would like to have this addressed.

Thus, results show that people are interested in application monitoring. The positive answers may partly be a study artefact since a number of users admitted that they would not actively search for such an application but they would not mind using it if it happened to be on their computer. The less computer-literate respondents found participation in the study educational and expressed astonishment about how much could go wrong. Their new security awareness might have made them overly interested in a monitoring application.

2.2 Study 2: Can users handle application access control?

In a second pilot study, described in detail in (Herzog, 2006), we prepared a paper prototype of an application monitor. 6 students from cognitive science, who are used to the idea of paper prototypes, were confronted with alerts from an application monitor built into a browser and observing browser plugins. There, it became clear that users had great difficulty in recognising alerts as genuine security warnings and subsequently abandoning their task because of security concerns. The security warnings interfered with the user task of (1) working with the prototype for the study and (2) following the scenario of the study which said that users should download some music files for a friend. These two-fold social settings may have made it especially difficult to accept the security warnings. In 4 of the 6 trials, users invoked a malicious application on their computer despite security warnings. However, two users did not; one of the two did not even download the malicious application.

As planned, the paper prototype evolved during the study, therefore it may be a success story for the prototype that the two users that did not install the malware were scheduled towards the end of the study (occasions 4 and 6 of 6). After user feedback, we (1) changed the wording of the alert message from very technical formulations, like “The application tries to open a socket connection IP address 68.142.226.56:80.” to “The application tries to communicate with another host on the Internet: 68.142.226.56:80.”, (2) added colour coding of alerts to show the severity, and (3) refrained from showing low-severity alerts at all.

But regardless of the design of the alert message, the final decision to continue or to discontinue using an application is put on the user; and the user may not be inclined to distrust the

application or to spend time pondering security decisions. One user made this explicit:

"'attacking the other computer' [reading aloud the alert text for an outgoing connection to the Yahoo web site] that sounds quite scary. Well, I assume that this is okay. I am only downloading a file from a page that a friend recommended. I cannot imagine that my computer is attacking another computer." [The same user when confronted with a high-risk alert and about to run the malicious application:] "This is a high-risk [action]. But I disregard this, I assume that this is perfectly okay."

Therefore, for JPerm, we made the alert texts more explicit by informing the user of the security consequences for allowing the action and by allowing explanations of technical details through links.

3 Background and implementation of JPerm

We implemented JPerm, an application monitor for Java applications, to fill a need in the Java security community but also to have a test bed for testing graphical user interfaces for security alerts. In this section, we briefly present the technical background of JPerm.

The Java language (Arnold et al., 2005) contains a runtime monitor called *security manager* that intercepts potentially dangerous calls such as file operations, socket operations, setting of Java properties (Java environment variables) and access to Java-internal objects such as the security policy, Java threads, class loader creation and many more, fully described in e.g. (Gong et al., 2003; Oaks, 2001). Upon interception, the security manager checks the Java security policy to determine whether the action should be allowed or not. If the action is allowed, the sensitive resource is accessed. If the action is not allowed, an exception is thrown. Usually, this results in the application terminating because it cannot proceed. For regular Java applications, the security manager is by default switched off, but for applets it is always switched on. The Java policy resides in a text file and can be edited either through a text editor or with the crude policytool, which is part of every Java distribution.

JPerm introduces a new security manager that, if it notices that a permission is not granted by the policy, invokes an alert window. JPerm is technically inspired by JSEF (Java Secure Execution Framework (Hauswirth et al., 2000)), which shows that it is technically possible to ask the user for every Java permission that the code needs. However, prompting with very technical text for every permission is not user-friendly, therefore JPerm focuses on the usability aspect of setting the policy. Thus, the focus of this article is JPerm's usability design and user interface (see fig. 1), which emerged from the guidelines presented in the following section and which were continuously evaluated through user feedback.

4 Guidelines for usable set-up of security policies

Our guidelines are specific for *applications that must ask lay users for security decisions and capture these decisions in a security policy*. The guidelines are influenced by our previously published work, the studies described in sections 2.1 and 2.2, as well as more general guidelines from literature on usability and security as presented in section 6.

In the following text, we discuss the guidelines and describe how they are addressed in JPerm. A discussion on the influence of other work on our guidelines is shown in detail in table 1 and in general in section 6.

Security must be visible without being intrusive. If there is a useful security tool on the computer, users must be made aware of its existence. Study 1 revealed that a number of respondents did not know whether they had a firewall on their home computer. Also, our firewall evaluation showed that two firewall products do not display the firewall name in their security alert, thus

Table 1: Design guidelines for applications that must set a security policy, their origin and motivation

Guideline	Origin and motivation
1. *Security must be visible without being intrusive.*	Johnston et al. (2003); Nielsen (1994); Yee (2002) propose visibility of system status as one criterion for successful HCI in security applications. Visibility contributes to the building of trust in the security application. However, users do not want to be ambushed with security alerts at all times (Sasse et al., 2003).
2. *Security applications must encourage learning.*	As a first step towards learning, Nielsen (1994) demands that applications use the language of the users to enhance their understanding and consequently to support the learning process. Whitten and Tygar (1999); Whitten (2004) have shown that security is not easy to understand and that concepts from educational software could and should be borrowed. Johnston et al. (2003) propose *learnability*, which we take one step further: not only should the software be learnable but also encourage the user to learn about security issues.
3. *Give the user a chance to revise a hasty decision later.*	Our studies in section 2 have shown that users are aware of making hasty decisions, driven by the need of getting a primary task done. While security in principle has the barn-door property (Whitten and Tygar, 1999) that the late closing of a security door may be exactly too late, because the damage is already done, this is not always or absolutely the case. But if there is no convenient way for the user to "close the door", it will remain open, and this must be avoided. This issue is also recognised as *revocability* by Yee (2002) or *easy reversal of actions* by Shneiderman and Plaisant (2004), even though true reversal may not be possible because of the barn-door property.
4. *Decisions cannot be handled off-line; runtime set-up is to be preferred.*	This guideline is in conflict with the guideline *support internal locus of control by making the user initiate actions, not respond to system output* by Shneiderman and Plaisant (2004) and shows clearly that not all usability guidelines can be uncritically transferred to security applications, which are typically supportive and not primary-task applications, and the user is not likely to take any actions if not prompted to do so.
5. *Enforce least privilege wherever possible.*	The principle of least privilege comes from Saltzer and Schroeder (1975) and is one important principle of computer security and specifically access control, which is what security policies are about. Garfinkel (2005) warns in this context of *hyperconfigurability*. Users have difficulties in managing too many options and cannot take in the consequences of their modifications. Garfinkel rather suggests "a range of well-vetted, understood and teachable policies" instead of exposing the user to fine-grained policy set-up.
6. *In a security alert, the user should be informed of the severity of the event and what to do.*	Nielsen (1994) proposes that error messages should contain instructions on what to do, not only what has happened. Still, the texts must be short and focused so that they are actually read. Details and additional explanations should be accessible but not blur the main message. Yee (2002) demands *clarity* so that the effect of any actions the user may take are clear to him/her before performing the action. Also Hardee et al. (2006) state that any decision support should contain the consequence of any action taken.
7. *Spend time on icons.*	Johnston et al. (2003) state that well-chosen icons can increase learnability. This is supported by Whitten (2004), who suggests icons for public-key encryption and motivates icon choices, and Pettersson (2005), who comments on the difficulty of choosing icons for privacy settings.
8. *Know and follow general usability guidelines and test, test, and test again.*	General usability guidelines are e.g. described by Shneiderman and Plaisant (2004) or Nielsen (1993). However, these guidelines are often so general that they can be difficult to implement for a specific case. Therefore, actual usability testing with users from the intended user segment is essential.

leaving the user at a loss on what software is giving her/him a warning. On the other hand, study 2 showed clearly that user will quickly automate their response to alerts if there are many of them. So the frequency of warnings must be well-balanced. In JPerm, we choose the default setting to only warn for medium and high-risk actions and silently allow all highly frequent low-risk actions (but capture them in the policy).

Security applications must encourage learning. If the security software ambushes the user with technical details like IP address or port numbers without explanations of what these mean, a lay user will be discouraged, resign and set up security guided by ad-hoc strategies. Rarely is learning encouraged. Therefore we designed JPerm so that it would be easy for users to get help on the meaning of concepts. Explanatory tooltips for concepts like 'IP address' or 'HTTP protocol' and links to web pages with more information are provided.

Give the user a chance to revise a hasty decision later. Users that are busy with a primary task take security chances to get their primary task done. They may need a reminder of their suboptimal security settings and a chance to revise their settings. In the firewall evaluation, we realised that no firewall gives access to its full configuration interface from an alert window. At best, settings for the current action can be fine-tuned. JPerm contains two hooks for revising previous actions: Firstly, there is a button in the alert window that can invoke the full-fledged policy editor. Secondly, the history shows previously granted actions for this application and is meant to also allow editing in a future version.

Decisions cannot be handled off-line; runtime set-up is to be preferred. In principle, security policies can be edited off-line, for example by editing the settings of the security application after installation, and for corporate firewalls this is the opus moderandi. But lay users of security applications do not make security a primary task (unless forced), therefore it is more straightforward and usable to allow the set-up of security policies at runtime. Off-line editing must still be supported for revising decisions or for fine-tuning them. But the coarse work should be done by a runtime set-up. In fact, offline editing is a rather impossible task in application monitoring because one cannot anticipate which permissions an application will need. JPerm successfully implements runtime set-up of Java security policies and has hooks for plugging in a more usable tool for off-line policy editing than the existing Java policytool.

Enforce least privilege wherever possible. The principle of least privilege says that a subject should only receive the privileges needed to perform its requested task but not more. Least-privilege and usability may be a trade-off: It is easier to let a user set up a coarse-grained rule (e.g. "Completely trust this software?") than prompting for every needed permission. Severity classification can help to find the right balance, but the more complicated the policy, the less likely it becomes that this trade-off can be automated. While a personal firewall can assume that the lookup of a host name with the DNS server can be granted without too much risk for security, policies that contain conditions—"Application X is allowed to execute only if application Y is not currently running."—cannot be anticipated by a monitor but must be user-provided.

JPerm addresses least privilege by never automatically granting permissions that contain wildcards. Such permissions must be supplied by the user by explicitly answering questions like "Allow connections to any host on port 22?". By default, JPerm remembers the permission exactly as it was required by the resource access and ignores probing attempts that check for wildcard permissions.

In a security alert, the user should be informed of the severity of the event and what to do. Study 2 made clear that users are not interested in dealing with low-risk events. If the events are not classified by severity, users do not have the energy to understand each and every alert and they resort to allowing everything in order to get their primary task done. It is advisable to set up the monitor so that warnings only appear for certain classes of events. Users also indicated that they need specific, non-technical guidance on what this event means, what it can lead to

and what they should be doing. JPerm implements this by means of a clear structure in the alert: what has happened, why this is dangerous, what should be done now. The risk level is prominently illustrated by a traffic-light icon.

Spend time on icons. As Whitten (2004) and Pettersson (2005) have shown, icons are important in enhancing—and also in destroying—the understanding of security concepts. If icons are used they must be carefully tested for their understanding by users. The one prominent icon in JPerm is the traffic light for signalling the severity of an alert. It came about after realising that alert classifications in personal firewalls are not visible enough when using sliders (one personal firewall), general colour coding or texts (some firewalls) or even no classifications at all (roughly half of the tested firewalls).

Know and follow general usability guidelines and test, test, and test again. This final guideline acknowledges the importance of considering other general and specific guidelines. Our guidelines provide further focus but general guidelines will also contribute to usability. We wish to stress that the best design guideline for a specific application is to do tests. Tests can be done with paper prototypes (Snyder, 2003), which are cheap to do in terms of both time and resources and can be adapted as late as at test-time by creating new windows and widgets as the need arises. But there must also be prototype tests with the actual application at the earliest possible stage because retrofitting usability as well as security is not possible. We show in the next section how we went about in designing and evaluating JPerm.

5 Evaluation of JPerm

JPerm was developed in a continuous feedback loop with users. First, we explored usability issues with a paper prototype as described in section 2.2. When we later had a running program, the first round of users (5 participants) consisted of students interested and educated in usability aspects. Their feedback was decisive for implementation changes for the second round of users (12 participants), where only minor changes were made.

The JPerm evaluation for both rounds of users consisted of three tasks which would expose users to the alert window of JPerm in different situations. Users were asked to set up (1) the same policy as in (Herzog and Shahmehri, 2006) in order to compare JPerm to the previously evaluated Java policytool by Sun, (2) a policy for a benign application, namely an SSH (secure shell) application, to test usability in a real application, and (3) a policy for a malicious application, in order to test whether users would recognise the maliciousness from the warnings given by JPerm. The setting for the study was that the user was at home at his computer with all three applications newly installed and JPerm running in the background. A screen recorder recorded each user session, including what happened on screen and what was said in the room.

In the following, we take up the most prominent findings from this evaluation.

Out of 16 trials *only two users fell for the malicious application*. One said she was assuming that the malicious application came from a reasonably trusted source such as `www.download.com`. The other user was busy in making the primary application work and did not take in the JPerm warnings.

Everybody understood and reacted positively to the *traffic light icon*, on the left-hand side of fig. 1, which clearly signalled the severity of the action to the users. Many users said for the medium (yellow) severity something like “This action sounds quite serious but it is only medium, so I think I will allow it.”, while the high (red) severity alert caused users to sharpen their attention one more step and they would read the texts with more concentration than if it had been yet another yellow medium alert.

Many users gave *positive feedback to the clear structure of the alert* (fig. 1) which presents

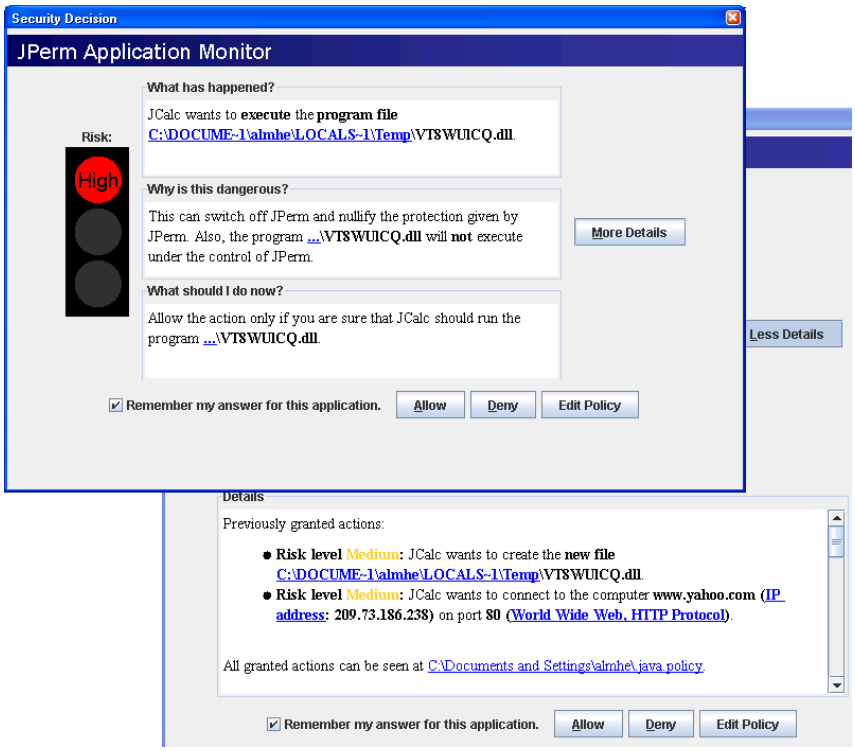

Figure 1: JPerm by the end of the study when alerting for the execution of a file. In the background, one sees the lower part of the JPerm window when the More Details-button is pressed.

on top what has happened, in the middle why this is dangerous and what the consequences of allowing this action could be and on the bottom what the user should do now. No one remarked negatively on this structure.

Shortening long file names and making directories accessible through links was understood by all users. Long file names were shortened with an ellipsis instead of the path name after a first round of user comments that long file names were tiring and should appear only once. Directory exploration was introduced after having several users wonder what *other* files would reside in the directory in which a new file was about to be created.

Customised advice with e.g. the concerned file names (`VT8WUICQ.dll` in fig. 1) and bold lettering in every text box leads to users reading or at least parsing the text more thoroughly than when there is generic advice. The first round of users that saw generic advice complained about it being too generic, whereas it was noticeable that later users pondered more about the subsequently customised text.

It was noticeable that *only about half of the users would invoke the button for more details, follow the links or invoke policy editing*. Most users would only interact with the alert window. This is consistent with security being a secondary task that users are not willing to spend time on unless absolutely necessary. This behaviour must be anticipated by alert designers and must result in compact windows with compact information.

JPerm clearly proved to be *superior to the Java policytool* for offline policy editing. No user had problems setting up the policies for the application that we had designed for our previous study (Herzog and Shahmehri, 2006) for evaluating the offline editing of the security policy using the Sun-provided policytool. While in that study, it took 15 to 30 minutes to complete the running of that applications, JPerm users would this time run the program, set up the policy while the program runs, finish the task within 3 minutes—and wonder what this was really

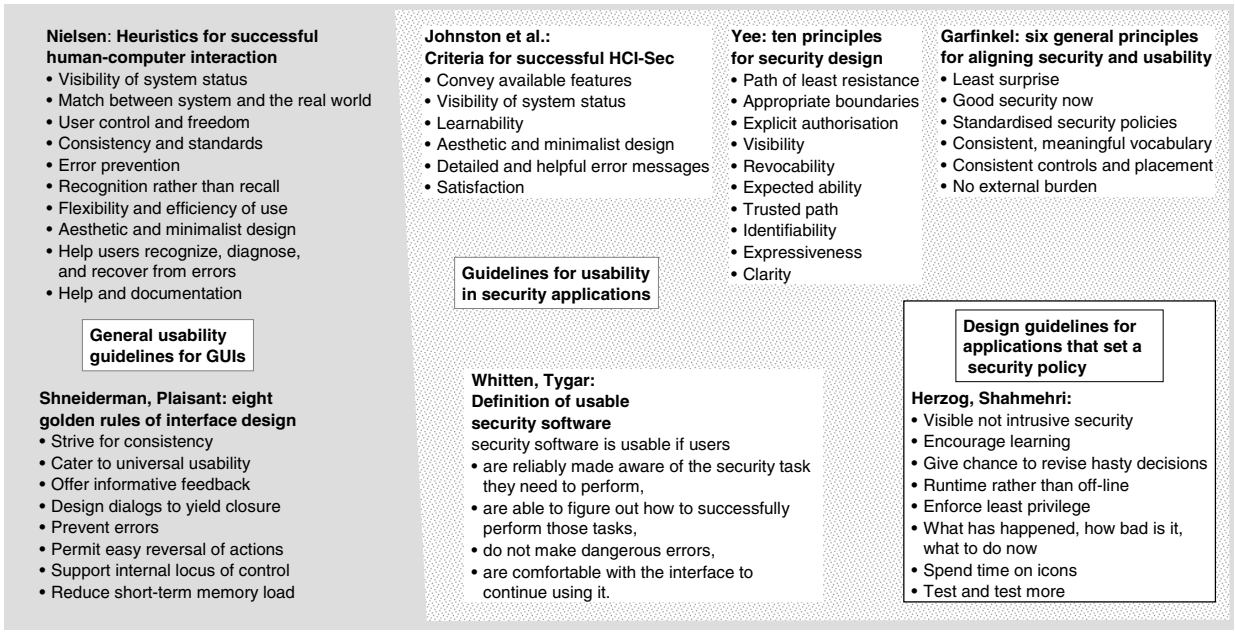

Figure 2: Structured overview of guidelines for usability in security applications

about. The previous study users never got to the point of wondering what this truly nonsensical application was doing, as they were so busy setting up the security policy.

Some suggestions for configuration were made and will be taken up for the main interface of JPerm, e.g. the setting of the warning level. Some users also wanted to be able to easily see low-risk warnings when running especially untrusted applications for the first time. Also, two of the 16 users did not like that JPerm remembers the answers by default and would like this to be configurable.

In general, users had a positive experience with JPerm, even though, as a security application, JPerm faces a difficult task. It wants to warn users *before* an application performs a potentially dangerous task and thus gives the user a chance to abandon the application. However, what the user *really* would need to know for his/her decision is *why* the application wants to perform this dangerous action. And this cannot be answered by an application monitor; at best, the user can take a guess by looking at the history of previously granted actions for this and other applications. We implemented this and users found it useful.

6 Related Work

Our work is positioned in the area of *usability* of *security applications*. The ISO standard 9241 defines *usability* as "the *effectiveness, efficiency*, and *satisfaction* with which specified users achieve specified goals in particular environments". That *security applications* or security features in applications differ from regular features or applications is recognised by Whitten and Tygar (1999), who note that security is typically a secondary goal, contains difficult concepts that may be unintuitive to lay users and suffers from the "barn door property", i.e. that true reversal of actions is not possible.

Figure 2 shows a number of guidelines from general usability of user interfaces (on the left-hand side of fig. 2) and guidelines for designing security features or applications (on the right-hand side of fig. 2. We position our own work as a subcategory of design guidelines for security applications, because our guidelines are specific for applications that set up a security

policy through runtime user decisions.

A comparison of the guideline keywords of figure 2 shows that the guidelines overlap. Usability of security applications is strongly influenced by general usability guidelines such as those given by Nielsen (1994) and Shneiderman and Plaisant (2004). The guidelines by Johnston et al. (2003) build, for example, explicitly upon the general guidelines of Nielsen (1994) and also take up each of the points identified by the early work of Whitten and Tygar (1999). However, for the other sources, there are often a number of specific and new issues that do not appear elsewhere.

Yee (2002) describes ten principles for security design that, while presented in a general fashion, were derived from the analysis of requirements for a secure desktop shell and thus are at times quite close to principles for secure operating systems. This is evident in the guideline of "appropriate boundaries" for automatic granting of user privileges to newly spawned processes or in the guideline of "trusted path" and "identifiability" for unspoofable communication channels or system windows.

Garfinkel (2005) puts up six general principles that summarise his long-term work on security and usability. New among these are the principles of 'good security now' (existing and available security solutions should be deployed even though they might have imperfections), standardized security policies that should keep the number of configuration options down, and the principle not to place external burden on the work-flow or routines of non-users of a security technology.

How our guidelines specifically relate to and are influenced by these existing guidelines is shown in detail in table 1. To summarise we can say that we put up specific guidelines for a specific subcategory of security applications. Our guidelines are therefore refinements of existing guidelines (our guidelines 1, 2, 3), focus specifically on the set up of runtime security policies (guidelines 4, 5) and therefore on the design of alert windows (guidelines 6, 7). Our final guideline that advocates repeated tests is surprisingly absent in other design guidelines.

Additional inspiration for the design guidelines for security applications can be found in the area of safety-critical computing. Leveson (1995) presents 60 guidelines for safe human machine interaction design. The guidelines are targeted at e.g. nuclear power plant user interfaces, but some of the recommendations, such as "Make potentially dangerous actions difficult or impossible", "Minimize activities requiring passive or repetitive action" and "Avoid displaying absolute values: Show changes...", also fit well in a security software context.

Two previous systems have attempted usable access control. Zurko et al. (1999) describe Adage for letting administrators set up security policies for distributed applications. But for administrators, security is a primary task. Also, policies were edited offline. We focus on lay users that must make a security decision at runtime.

Brostoff et al. (2005) describe their attempts to let lay users create role-based access control policies for their PERMIS system and show clearly how difficult this is. Their work consists of a description of their implementation and usability study but it is difficult to apply their experiences to other access control applications due to the lack of specific or general advice in the form of clearly stated guidelines or principles.

7 Conclusion and future work

While efforts for runtime control and policy languages abound, little effort has been spent on how security policies can be set up by those lay users for whom they are intended. This work has taken exploratory steps in the direction of application monitoring to explore what users want and need for a successful runtime set-up.

We have presented eight specific guidelines for designers of security applications that rely on a user-provided security policy. The guidelines have been presented in comparison to previous research in the area of usability and security. The guidelines are also presented in via the implementation of the application monitor JPerm, which enhances the security architecture of Java by setting up a Java security policy at runtime.

Specific future work for JPerm will deal with signed code and provide a general interface for setting JPerm start-up options. A more general continuation is to see to what extent the guidelines can be applied to more advanced policy settings as described in typical scenarios for trust negotiation by e.g. Winsborough et al. (2000).

References

Arnold, K., Gosling, J., and Holmes, D. (2005). *The Java Programming Language*. Addison Wesley, 4th edition.

Brostoff, S., Sasse, M. A., Chadwick, D., Cunningham, J., Mbanaso, U., and Otenko, S. (2005). 'R-What?' Development of a role-based access control policy-writing tool for e-scientists. *Software—Practice and Experience*, 35:835–856.

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (Policy'01)*, volume LNCS 1995, pages 18–38. Springer-Verlag.

Garfinkel, S. L. (2005). *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, Massachusetts Institute of Technology.

Gong, L., Ellison, G., and Dageforde, M. (2003). *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*. Addison Wesley, 2nd edition.

Hardee, J. B., West, R., and Mayhorn, C. B. (2006). To download or not to download: an examination of computer security decision making. *interactions*, 13(3):32–37.

Hauswirth, M., Kerer, C., and Kurmanowytsch, R. (2000). A secure execution framework for Java. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS'00)*, pages 43–52. ACM Press.

Herrmann, P. and Krumm, H. (2001). Trust-adapted enforcement of security policies in distributed component-structured applications. In *Proceedings of the 6th IEEE Symposium on Computers and Communications*, pages 2–8. IEEE.

Herzog, A. (2006). A pilot study on setting an applet access control policy. Technical report, Linköpings universitet, Dept. of Computer and Information Science.

Herzog, A. and Shahmehri, N. (2005). Performance of the Java security manager. *Computers & Security*, 24(3):192–207.

Herzog, A. and Shahmehri, N. (2006). A usability study of security policy managment. In Fischer-Hübner, S., Rannenberg, K., and Louise Yngström, S. L., editors, *Security and Privacy in Dynamic Environments, Proceedings of the 21st International Information Security Conference (IFIP TC-11) (SEC'06)*, pages 296–306. Springer-Verlag.

Herzog, A. and Shahmehri, N. (2007). Usability and security of personal firewalls. In *Proceedings of the International Information Security Conference (IFIP TC-11) (SEC'07)*. Springer-Verlag.

Johnston, J., Eloff, J. H. P., and Labuschagne, L. (2003). Security and human computer interfaces. *Computers & Security*, 22(8):675–684.

Kagal, L., Finin, T., and Joshi, A. (2003). A policy based approach to security for the semantic web. In *Proceedings of the International Semantic Web Conference (ISWC'03)*, volume LNCS2870, pages 402–418. Springer-Verlag.

Leveson, N. (1995). *Safeware: System Safety and Computers*. Addison Wesley.

Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann Publishers.

Nielsen, J. (1994). Heuristic evaluation. In Nielsen and Mack (1994), pages 25–62.

Nielsen, J. and Mack, R. L., editors (1994). *Usability Inspection Methods*. Wiley & Sons.

Oaks, S. (2001). *Java Security*. O'Reilly, 2nd edition.

Pettersson, J. S. (2005). HCI guidance and proposals. Deliverable D06.1.c, PRIME Project.

Saltzer, J. H. and Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Sasse, M. A., Brostoff, S., and Weirich, D. (2003). Transforming the weakest link—a human/computer interaction approach to usable and effective security. *BT Technology Journal*, 19(3):122–131.

Seamons, K. E., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., and Yu, L. (2002). Requirements for policy languages for trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (Policy'02)*, pages 68–79. IEEE.

Shneiderman, B. and Plaisant, C. (2004). *Designing the User Interface*. Addison Wesley, 4th edition.

Snyder, C. (2003). *Paper Prototyping*. Morgan Kaufmann Publishers.

Whitten, A. (2004). *Making Security Usable*. PhD thesis, School of Computer Science, Carnegie Mellon University. CMU-CS-04-135.

Whitten, A. and Tygar, J. D. (1999). Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium (Security'99)*. Usenix.

Winsborough, W. H., Seamons, K. E., and Jones, V. E. (2000). Automated trust negotiation. In *Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX'00)*, volume 1, pages 88–102. IEEE.

Yee, K.-P. (2002). User interaction design for secure systems. In *Proceedings of the International Conference on Information and Communications Security (ICICS'02)*, pages 278–290. Springer-Verlag.

Zurko, M. E., Simon, R., and Sanfilippo, T. (1999). A user-centered, modular authorization service built on an RBAC foundation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'99)*, pages 57–71. IEEE.