Improving User-Interface Dependability through Mitigation of Human Error

Roy A. Maxion * Robert W. Reeder

Department of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213 USA

Abstract

Security may be compromised when humans make mistakes at the user interface. Cleartext is mistakenly sent to correspondents, sensitive files are left unprotected, and erroneously configured systems are left vulnerable to attackers. Such mistakes may be blamed on human error, but the regularity of human error suggests that mistakes may be preventable through better interface design. Certain user interface constructs drive users toward error, while others facilitate success.

Two security-sensitive user interfaces were evaluated in a laboratory user study: the Windows XP file-permissions interface and an alternative interface, called Salmon, designed in accordance with an error-avoiding principle to counteract the misleading constructs in the XP interface. The alternative interface was found to be more dependable; it increased successful task completion by up to 300%, reduced commission of a class of errors by up to 94%, and provided a nearly 3x speed-up in task completion time. Moreover, users spent less time searching for information with the alternative interface, and a greater proportion of time on essential task steps. An explanatory theory in its early stages of development is presented.

Key words: computer security, dependability, external representation, external subgoal support, file permissions, goal error, human error, user interfaces

1 Introduction

One locus of vulnerability in a computer system is an undependable user interface – one that does not meet its specification in terms of the speed or

* Corresponding author.

Email addresses: maxion@cs.cmu.edu (Roy A. Maxion), reeder@cs.cmu.edu (Robert W. Reeder).

accuracy with which users should complete tasks. One reason why some user interfaces fail to meet their speed and accuracy specifications is human error. Researchers have long recognized that human error has causes and manifestations that are similar across all domains of human endeavor, from aviation, to power plant operation, to making a cup of tea (Wiegmann and Shappell (2003); Reason (1990); Senders and Moray (1991); Norman (1988)). In the domain of software user interfaces, human error leads people off the path of correctly completing a task and on to lengthy task delays or total task failure. There is a need for user-interface designers to understand the common types and causes of human error, and the ways in which they may be prevented. When interfaces are designed to eliminate the conditions that lead people to make mistakes, interfaces will be more dependable, and the applications they serve will be more secure.

One domain in which user interface accuracy is critically important is computer security. Inaccurate security settings can have a high cost - they can make sensitive data vulnerable, or they can leave an entire system open to attack. Adding to this cost, security problems have what Whitten and Tygar (1999) have called the "barn door property" - once a system has had a vulnerability for any length of time, there may be no way to know if the vulnerability has been exploited, so the system will have to be considered compromised, whether it has been or not.

The present work investigates user interface dependability and human error in the security context of setting file permissions under Microsoft's Windows XP operating system, which uses Microsoft's NT file system (NTFS). NTFS file permissions were chosen as the domain of study because a significant amount of anecdotal evidence suggests that setting NTFS file permissions is a particularly error-prone task for which the consequences of failure can be severe. For example, there is the so-called "Memogate" scandal, in which staffers from one political party on the United States Senate Judiciary Committee stole confidential memos from the opposing party (U.S. Senate Sergeant at Arms, 2004). The memos were stored on a shared NTFS server. The theft was possible in part because an inexperienced system administrator had failed to set permissions correctly on the shared server. As another example, a Windows network administrator at Carnegie Mellon University reports that many users want to share their files so they can access them both at work and at home; they accidentally make their private files accessible to all (several hundred) users on the network, because it is too confusing to set permissions as actually desired (Smith, 2004). Finally, Microsoft publishes a list of "best practices" for NTFS security that advises users not to use several of the provided features of the NTFS permissions model, such as negative (i.e., deny) permissions and the ability to set permissions on individual files as opposed to folders (Microsoft Corporation, 2005a). The best-practices document states that use of these features "... could cause unexpected access problems or reduce security."

Providing access to features which are apparently problematic is bound to lead to errors.

As these anecdotes indicate, setting and checking permissions cannot always be left to expert system administrators – users in many environments need or want to take responsibility for protecting their own files. However, setting file permissions is not an everyday task; it may need to be done only every few weeks or months. Thus, those setting file permissions will often not be expert system administrators; they will be novice or occasional users who, from time to time, want to restrict access to data or to grant access for only a limited number of associates. They will not readily remember arcane details about how to operate a file-permissions-setting interface. The present work adopts the underlying assumption that file-permissions-setting interfaces should accommodate novice and occasional users.

This paper reports results of an investigation into and a solution for one type of human error encountered in file-permissions setting interfaces. First, an existing interface for setting NTFS permissions, the Windows XP File Permissions interface (hereafter abbreviated XPFP), was evaluated in a laboratory user study and shown to have accuracy rates as low as 25% on file-permissions-setting tasks. Errors made by users in the XPFP interface were identified and categorized into types according to an established human-error framework. Goal errors, the failures of users to understand what to do, were identified as the dominant type of error. A primary cause of goal errors, namely poor external representation of task-relevant information, was identified. A design principle, external subgoal support, was proposed to reduce goal errors and was implemented in a new interface, called Salmon, for setting NTFS file permissions. The design principle was evaluated in a laboratory user study identical in design to the XPFP study but employing the new interface. Salmon achieved a success rate of 100% on a task for which XPFP had achieved a 25% accuracy rate, achieved a 94% reduction in the number of goal errors users made on the same task, and achieved a nearly 3x task-performance speed-up in another task, compared to XPFP.

2 Objective

The objective of the present work is to understand the causes of user error in user interfaces generally, and file-permissions interfaces in particular. It is a further objective to find a design method that can be applied to new generations of user interfaces so that the same user errors are not encountered again and again and again in future user interfaces.

A specific problem, poor external representation, was observed in the XPFP

interface. It was expected that poor external representation would lead to user goal errors, i.e., the incorrect establishment or omission of cognitive goals. A solution was proposed to reduce the occurrence of goal errors. This solution, called external subgoal support (see section 5), led to this paper's hypothesis:

> Use of external subgoal support in user interface design reduces the likelihood that users will commit goal errors, thus improving task accuracy rates, and reduces time spent looking for information, thus reducing time to task completion. The combined improvements to speed and accuracy make the interface more dependable for quick and accurate accomplishment of tasks.

Task success rates, goal-error occurrences, and times to task completion were compared between the XPFP and Salmon studies to determine whether external subgoal support as implemented in Salmon was an effective means of improving success, reducing goal errors, and improving task-completion speed.

3 Background and related work

Related work falls into three areas. The first is prior work on usable file permissions and usable access control. The second, a superset of the first, is work in the emerging field of human-computer interaction and security, also known as HCISEC. The third is the traditional literature on human-computer interaction and cognitive science.

File permissions are an instance of the broader area of access control, including evaluation of interfaces for setting file access. Zurko et al. (1999) conducted a user study on the Visual Policy Builder, a graphical user interface for specifying access control policies for their Adage system. Good and Krekelberg (2003) showed that the Kazaa peer-to-peer file-sharing service's interface misled many users into unintentionally sharing confidential files. Long et al. (2003) evaluated a preliminary, paper-based interface for limiting applications' access to system resources. While these three interface evaluations were interesting in their specific task domains, none appear to lead to any conclusion about design principles for security interfaces in a larger context. Other work in usable access control in various domains includes Balfanz (2003), Sampemane et al. (2002), and Dewan and Shen (1998). With the exception of the Adage project and Long et al., work in this area involves outlining access control models, not evaluating access control interfaces, as the present work sets out to do.

In the broader human-computer interaction and security literature, those who have acknowledged the challenges of designing dependable user interfaces in security-related domains and have proposed principles for better security interface design include Whitten and Tygar (1999), Adams and Sasse (1999),

Besnard and Arief (2004), Zurko and Simon (1996), and Yee (2002). Such papers propose ideas for making security tasks easier to perform accurately, but do not evaluate their ideas empirically. Whitten and Tygar (2003), in another paper, present user-study results to validate a design principle for security interfaces, but their principle, "safe staging," is not related to external subgoal support, nor does it appear to be grounded in a theory of human error.

The traditional human-computer interaction and cognitive science literature contains a substantial amount of material on the importance of external representations in supporting problem-solving tasks (Woods (1985); Woods and Roth (1988); Zhang and Norman (1994); Zhang (1996, 1997); Ballard et al. (1995); Nielsen and Mack (1994) and Norman (1988)). External representations are information displays (e.g., notes on paper, graphics on a monitor, etc.) that reside outside the mind; they complement internal mental representations by providing additional memory capacity, cues to internal processes, and information structures that allow patterns to be easily perceived. Good external representations can facilitate fast and accurate problem solving, while poor external representations can impede it. Mitchell and Miller (1986), for example, suggest a methodology for information display design, although their method is intended mainly for industrial domains with expert users, rather than for novice and occasional users.

4 Example problem

The XPFP interface serves to illustrate the problem with user interfaces that offer poor external representations of task-relevant information. Some background on the NTFS file-permissions model is necessary to fully understand the XPFP interface and the errors it causes.

A computer system using NTFS will be populated with *entities* and *objects*. The entities are individual users and groups of users on the system. The objects are the files and folders on the system. NTFS defines 13 *atomic permissions*[1] that correspond to actions that users can perform on files and folders. The precise meanings of the 13 NTFS atomic permissions are not relevant to this paper, but are described in a Microsoft Technet article (Microsoft Corporation, 2005b). For purposes of this paper, it is sufficient to note that NTFS permissions can be grouped into five disjoint sets: Read, Write, Execute, Delete, and Administrate.[2] NTFS uses an *Access Control List (ACL)* model of file per-

[1] Note that NTFS documentation uses the term *special permission* where *atomic permission* is used here. The latter term makes it clearer that these are indivisible permissions, the lowest-level permissions in the system.

[2] Note that the XPFP interface and NTFS documentation use a different, non-

missions. Under the ACL model, each file and folder in the file system has an associated list of users and groups who have permissions on that file or folder.

For each file and for each permission for that file, each user and group on the file's ACL has a permission value explicitly granted to them for that file and that permission. Permissions are tri-valued, and can take on any of the following values: ALLOW, DENY, or NOTSET; the NOTSET value indicates that neither ALLOW nor DENY has been set. Under the rule of *group inheritance*, each user also inherits permissions from the groups of which they are members. A user's actual access to a file is computed according to a formula that sums the user's explicit permissions and their inherited permissions according to *precedence rules*. The precedence rules state that any DENY permission value takes precedence over all other permission values, and any ALLOW permission value takes precedence over any NOTSET permission value; NOTSET acts to deny access by default. Group inheritance leads to the distinction between *stated permissions*, the explicit permissions granted to each user, and *effective permissions*, the actual access a user will be allowed according to the combination of stated and inherited permissions.

It is the distinction between stated permissions and effective permissions that makes setting NTFS file permissions a difficult task. The person setting permissions operates directly on the low-level stated-permissions bits, which do not necessarily translate directly into the effective access that will be allowed to system files. Actual access in NTFS is determined by the nuanced formula alluded to above. However, users setting file permissions are ultimately concerned with who can access what, not with low-level bits and nuanced formulas. Thus users need to view the effective permissions in order to evaluate when they have fully completed their primary goal, and when they still have additional work to do to complete their goal.

Casual observation of the XPFP interface (see Figure 1) reveals that XPFP does not provide ready access to needed information. First of all, not all of the NTFS permissions are visible in the main XPFP window. ADMINISTRATE and DELETE permissions are notably absent, and are hidden two screens away. Without ADMINISTRATE and DELETE permissions visible on the main window, some users may not even realize they exist. Secondly, XPFP does not provide the group membership data to indicate what groups individual users belong to. For example, in Figure 1, Wesley is in the group ProjectF, but there is no indication of this in the XPFP window. In fact, users need to use an entirely different application to view group membership in Windows XP. Without group membership information, even those few users who understand

disjoint grouping of the 13 atomic permissions into six composite sets. The disjoint grouping discussed here is the authors' own, and is used for clarity of presentation to those readers not already familiar with the NTFS permissions model.

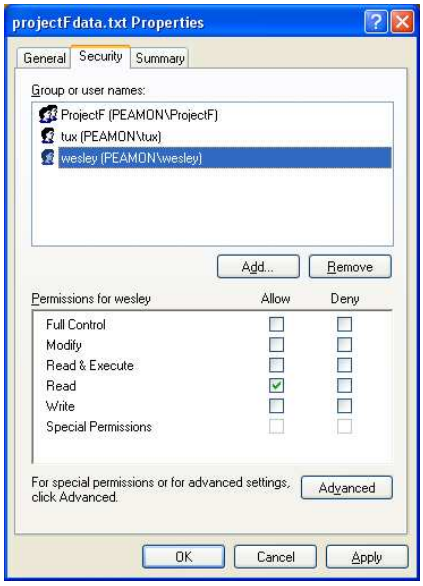

Fig. 1. A screenshot of the XPFP interface, showing information and functionality for setting permissions bits. Effective permissions are not visible, nor is group-membership information provided, making it virtually impossible for users to verify the completion status of their goals.

the group inheritance rules will not be able to figure out whether Wesley is inheriting permissions from ProjectF or not. Finally, the XPFP main window contains the checkboxes necessary for setting the permissions bits that will be used to determine effective permissions, but effective permissions themselves are nowhere to be seen. In fact, XPFP has an effective-permissions display, but it is two screens away, where hardly any non-expert user will find it. The difficulty of accessing essential task-relevant information in XPFP is likely to lead users into incorrectly determining when they have completed their goals, or to leave them not knowing how to proceed toward task completion. Without a salient external representation of task-relevant information to cue users toward proper behavior, users will be able to complete tasks only if they already have the information stored internally, or if they are fortunate enough to happen upon the information elsewhere. Those who do not have the needed information are likely to commit errors.

5 Solution - external subgoal support (ESS)

External subgoal support (ESS) is an interface design method rooted in cognitive principles regarding the importance of external representations: see Woods and Roth (1988); Zhang and Norman (1994); Zhang (1996, 1997); Ballard et al.

(1995). ESS was developed to ensure that all task-relevant information is represented saliently by the user interface, allowing users to check goal completion status and set appropriate subgoals. When users do not set appropriate subgoals, they commit *goal errors*, one of several specific types of human error. ESS mitigates one cause of goal errors by providing external cues as to what to do. Goal errors are discussed in more detail in Section 7.4.

5.1 Description

For an understanding of the cognitive causes of goal errors, the present work relies on Pocock et al.'s THEA – Technique for Human Error Assessment (Pocock et al., 2001). THEA was developed to analyze a user interface design for areas of potential user difficulty without conducting costly user studies. It includes an error framework which is based on Norman's well-known seven-stage execution-evaluation model of human information processing (Norman, 1988). THEA condenses Norman's seven stages down to four stages of information processing during which human error can occur. These four stages are the combination of perception, interpretation, and evaluation; goal formulation; plan formulation; and action execution. According to the Norman/THEA models, human information processing starts with a problem, the primary goal, and proceeds in the following loop:

(1) Perceive and interpret information from the environment, and evaluate whether the problem is solved;
(2) If the problem remains unsolved, formulate a subgoal, according to perceived information, for solving all or part of the problem; if the problem is solved, exit the loop;
(3) Formulate a plan to achieve the subgoal;
(4) Execute the actions in the plan.

Goal errors occur when the second step goes wrong. If the perceived information consulted in the second step is incorrect or is misinterpreted, the wrong subgoal may be set. If the wrong information is used to check whether the problem has been solved in the second step, either an unnecessary subgoal may be added (if the problem is assumed unsolved when it is already solved) or a necessary subgoal may be omitted (if the problem is assumed solved when it is not). Thus the availability of information to check progress toward the primary goal is critical to correct selection of subgoals.

If incorrect, misleading, or missing information causes goal errors, the logical solution is to make the necessary information available in a correct, easily interpretable form. Indeed, researchers in cognitive science have shown the significant effects that external (i.e., external to the human problem-solver)

problem representations can have on human performance: see Woods and Roth (1988); Zhang and Norman (1994); Zhang (1996, 1997); Ballard et al. (1995). External representations not only serve as memory aids, but also play a central role in shaping cognitive behavior (Zhang and Norman, 1994). Some representations of a problem lead to incorrect or slow problem-solving behavior, while other representations facilitate correct and efficient problem-solving behavior. Section 9 discusses how each of the user interfaces tested in the present study influences users to engage either more or less in activities like information-gathering, based on the information representations made available to the users.

5.2 Design method

A prerequisite for implementing ESS is a careful task analysis. Kirwan (1994), an excellent reference on how to perform task analyses, describes the Hierarchical Task Analysis (HTA) method, which includes a convenient representation of the results of a formal task analysis. An HTA represents the task as a hierarchy of goals and the operations that are needed to achieve them. At the root of an HTA hierarchy is a primary goal to be accomplished. Beneath the root are nodes that represent the subgoals necessary to achieve the primary goal, and each subgoal may have a tree of subgoals beneath it. At the leaf nodes of the hierarchy are the actionable operations necessary to achieve each of the lowest-level subgoals.

Once the hierarchical task analysis is complete, and its corresponding hierarchy of subgoals has been created, the method for designing according to external subgoal support can begin. It proceeds in two phases as shown below. The Salmon interface, shown in the next section, was designed according to this method:

- Phase 1 - Identify information required.

(1) For each goal, starting with the primary goal and proceeding through all subgoals in the HTA, identify the information a user will need:
   (a) to determine when the goal has been completed;
   (b) to set the subgoals beneath the goal.

(2) For each operation at the leaf nodes of the HTA, determine what information will be needed to execute the operation. This is usually:
   (a) procedural knowledge – information about *how* to execute the operation;
   (b) declarative knowledge – any parameters that must be supplied to the operation (e.g., the name of a user being added to an ACL).

- Phase 2 - Provide Phase-1 information in the interface.

(1) Incorporate, in the designed interface, an accurate, clear, and salient repre-

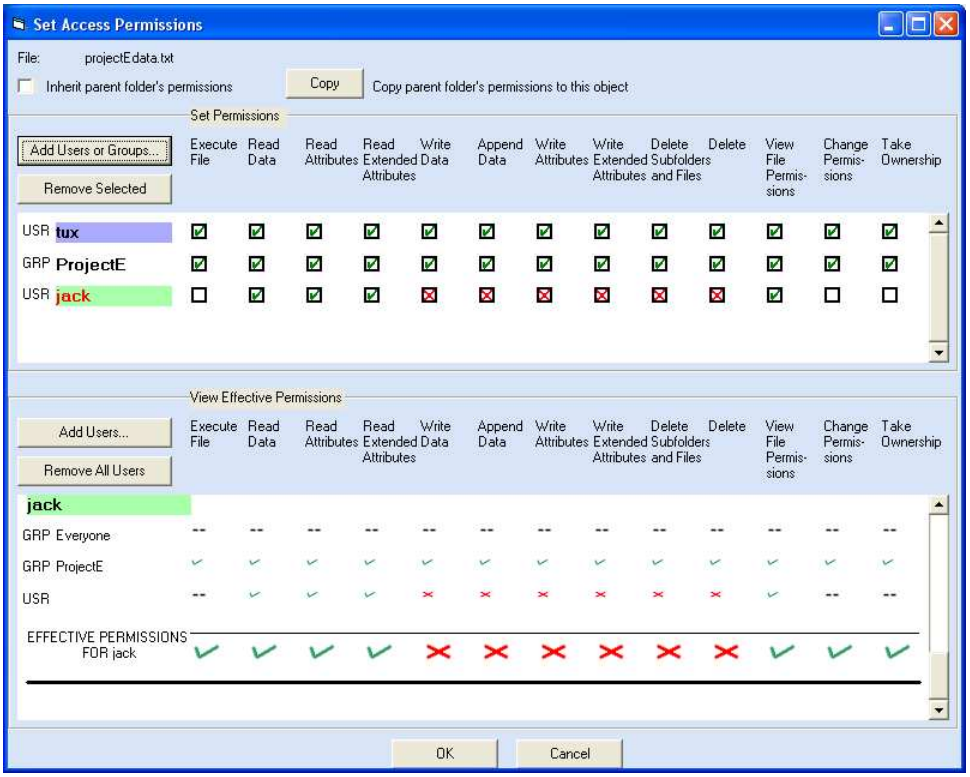

Fig. 2. The Salmon interface (screenshot) was designed to provide external subgoal support by having all 13 atomic permissions be settable in the upper pane, and by showing effective permissions in the lower pane.

sentation of the necessary information, as determined by the above steps. [3]

5.3 Salmon interface with ESS

The Salmon interface (see Figure 2) was designed according to the ESS method. Phase 1 of the ESS method identified the following information as necessary for establishing the right subgoals and executing the right operations in a file-permissions interface:

(1) The full list of 13 atomic permissions, with no permission values hidden;
(2) Stated permissions for all users and groups on the ACL;
(3) Group membership data, and how the data combine to form a user's effective permissions;
(4) Effective permissions for all users on the ACL.

[3] External representation design is a large topic and is not covered in detail here; see Card (2003) or Woods and Roth (1988) for more on this topic.

The Salmon main window contains all of this information. The window is split into two panes, upper and lower. In the upper pane, the 13 atomic permissions are listed across the top; immediately below them are the checkboxes necessary for setting the stated permissions for each user or group on the ACL. In the lower pane is an effective-permissions display. For any individual user selected, the effective-permissions display shows the groups of which that user is a member, the permissions inherited from those groups, the user's individual stated permissions, and the combination of all these permissions, i.e., the user's effective permissions.

6 User study and experimental method

A laboratory user study was conducted to observe and document errors in file-permissions-setting tasks. Two user interfaces were compared: XPFP and Salmon, a new interface for setting file permissions, designed in accordance with the principles of external subgoal support.

6.1 Participants

Twenty-four students and research staff at Carnegie Mellon University voluntarily participated in the study. Participants were recruited randomly. All participants' academic backgrounds were in science and engineering disciplines, and all were daily computer users. While a few usually used UNIX-based computer systems in their daily work, all had at least some experience using Windows, with 21 out of 24 claiming they used Windows at least a few times a week. Nineteen reported having some experience setting file permissions, on Windows or another operating system, while 5 reported having no experience setting file permissions whatsoever. All but four reported setting file permissions a few times a month or less.

6.2 Apparatus

All participants solved a series of permission-setting tasks on a computer running Windows XP, Version 2002, Service Pack 1, using either the XPFP or Salmon permission-setting user interface. Timestamped screen video and mouse and keyboard actions were recorded with a software tool developed for user study data collection. Participants' initial and final permissions settings were recorded for each task instance.

To simulate real permission-setting conditions, a hypothetical scenario was designed in which the participant worked in a generic "organization," shared her computer with other workers in the organization, and had to restrict access to the files and folders on her computer. On the laboratory Windows XP machine, the hypothetical organization's computer environment was created and populated with individual users, groups (containing users), files, and folders. The environment included 27 individual users, named for each letter of the alphabet (Ari, Bill, Catherine, Dave, Evelyn, etc.), plus one user named Tux, which represented the participant her/himself. The environment also included 6 groups named ProjectA through ProjectF, each of which contained 6 members drawn from the 27 users. No group contained another group as a member. There were also files and folders on which participants were to set permissions.

Participants were given three permission-setting tasks, called Jack, Wesley and Tux, and one training task called the Hakim task. The Hakim (training) task simply required the participant to add a user to the access control list (ACL), and to give that user READ permission. This training task gave participants experience with the mechanics of adding users and setting permissions – actions which were common to the experimental tasks.

The Jack and Wesley permission-setting tasks were chosen because they involved group inheritance, a feature of the NTFS permissions model that was expected to lead to a great deal of user error. These two tasks required participants to set permissions on a text file so that users Jack or Wesley could read the file, but not change it. The Tux task was chosen to determine how easily participants could find the DELETE permission checkbox if it were hidden, as it is in the XPFP interface, as opposed to being immediately visible, as it is in the Salmon interface. The Tux task required participants to set permissions on a text file so that Tux would not be able to delete the file by accident.

The task statements for the four tasks are shown below:

Hakim (training) task. You (username: tux) have just created the folder Stuff for Hakim, so that you can share private data with your friend Hakim (username: hakim). Set permissions on the folder so that Hakim will be able to read anything you put in the folder. Make sure no one else can read anything in the folder.

Jack task. The group ProjectE is working on projectEdata.txt, so everyone in ProjectE can read, write, or delete it. Jack (username: jack) has just been reassigned to another project and must not be allowed to change the file's contents, but should be allowed to read it. Make sure that effec-

tive now, Jack can read the file projectEdata.txt, but in no way change its contents.

Wesley task. [4] The group ProjectF is working on projectFdata.txt, so everyone in ProjectF can read, write, or delete it. Wesley (username: wesley) has just been reassigned to another project and must not be allowed to change the file's contents, but should be allowed to read it. Make sure that effective now, Wesley can read the file projectFdata.txt, but in no way change its contents.

Tux task. You (username: tux) have a checkbook-balancing program that writes to a file called myCheckbook.dat. You don't want to accidentally delete this file. Deny yourself the permission to delete it. Of course, you want all other permissions to remain unchanged.

In the Wesley and Jack tasks, there was one group that was already on the access control list (ACL) for the file, and the operative individual user was a member of that group. The difference between the Wesley and Jack tasks was that in the Wesley task, Wesley was inheriting Read and Write permissions from ProjectF, but not Administrate permission, while in the Jack task, Jack was inheriting Read and Write as well as Administrate permissions from ProjectE.

The simple solution to the Wesley task was to add Wesley to the ACL and explicitly deny him Write permission; he was already allowed Read permission from ProjectF. However, this simple solution would not work for Jack, since Jack was inheriting Administrate permission as well as Read and Write permission. If Jack was denied Write permission, but not explicitly denied Administrate permission, he would have been able to restore his Write permission. The task statement presented to users did not mention this nuance; it was left to the interfaces to provide the cues needed to understand that Jack's Administrate permission had to be removed.

6.4 Rules for completing tasks

Participants were asked to abide by certain rules to ensure as realistic an environment as possible without compromising the experimental comparison between XPFP and Salmon. First, they were allowed to check group memberships in the XP Computer Management application, which is a separate application from the file-permissions interfaces. However, in order to prevent users from completing tasks without using the file-permissions interfaces, they were asked not to use the Computer Management application to change group

[4] The task statement for Wesley was identical to that for Jack except for the names of specific files, users, and groups. The tasks differed, however, in the way they were initialized; see text following the task statements.

memberships. Thus, users were not allowed to complete the Wesley and Jack tasks by removing Wesley or Jack from their respective groups. Second, to compensate for the first restriction, participants were told that if a task statement did not explicitly mention a given user, any permission setting was permissible for that user. Thus, participants could change the group ProjectE's or the group ProjectF's permissions and not be concerned about the effects on members of those groups besides Wesley or Jack. Finally, users were allowed to access a selected set of Windows Help files that pertained specifically to file permissions, but they could not browse the full set of Help files at random.

6.5 Procedure

Participants were assigned randomly to use either the XPFP or Salmon interface. Each participant used only the single assigned interface for all tasks; no one used both interfaces. Twelve participants were assigned to each interface. Participants were asked to "think aloud" throughout the course of the experiment, and were instructed in thinking aloud according to directions adapted from Ericsson and Simon (1993). Participants were shown how to view system users, groups, and group memberships using the Computer Management interface and how to access the limited set of Windows Help files. Participants were given no instruction in using the XPFP or Salmon interfaces. The experimenter brought up the interface the participant was to use. Task statements were presented in text in a Web browser, and remained available to the participant throughout the task. All participants were given the same training task first, after which the presentation order of the remaining tasks was counterbalanced among participants using a full factorial design. Participants were given 8 minutes to complete each task. After each task was completed, participants were asked to rate their confidence on a 1-7 scale (7: very confident) that the task had been completed correctly.

7 Analytical procedures

This section presents the procedures for data analysis, the results of which are presented in Section 8. Data were logged from user sessions automatically by custom software. Separate, but consistently timestamped, files were kept for keyboard, mouse, video and verbal protocols (digital video and audio). These data sets needed to be analyzed to recover task-relevant information regarding speed (timing), accuracy (task success or failure), and errors (discrete actions, classification of action as error or non-error, and classification of errors into one of four types). Measuring elapsed times of events is straightforward, and will not be discussed in detail. Other analytical aspects of the experiment are

discussed below.

7.1 Accuracy - determining task success or failure

Accuracy was measured in terms of whether or not the participant succeeded in correctly executing the task. To determine task success or failure, participants' final permission settings were examined. For the Wesley and Jack tasks, a task instance was deemed successful if the individual (Wesley or Jack) had effective permissions allowing him READ permission and denying him WRITE and ADMINISTRATE permissions. A Wesley or Jack task instance was deemed a failure if the individual had effective permissions denying READ permission, or allowing WRITE and/or ADMINISTRATE permission. EXECUTE and DELETE permissions and all permissions for other entities were ignored. For the Tux task, a task instance was deemed successful if Tux had effective permissions denying DELETE and allowing both READ and WRITE permissions. A Tux task instance was deemed a failure if Tux had effective permissions allowing DELETE, or denying READ or WRITE permission.

7.2 Actions

Actions need to be defined so that user data can be divided into discrete units for error analysis. An *action* is defined as any change to the ACL, namely adding an entity to or removing an entity from the ACL, or altering the permissions of an entity already on the ACL.

7.3 Classifying actions as errors

An action is classified as an error if it is discrepant with a model of correct actions. A model of correct actions for each task was constructed by hierarchical task analysis (HTA) (Kirwan, 1994). An example HTA for the Jack task is shown in Figure 3; HTAs for other tasks were similarly constructed. The HTA for each task consists of the goals, plans and actions required to complete the task.

Each discrepancy between user actions and HTA model actions was classified as an *error of commission*, an *error of omission*, or a non-error. A user action was an error of commission if it was unnecessary according to the HTA. It was an error of omission if it was a necessary action according to the HTA, but the user failed to complete it. Non-errors included user actions that matched actions in the HTA and unnecessary but innocuous actions, such as changing

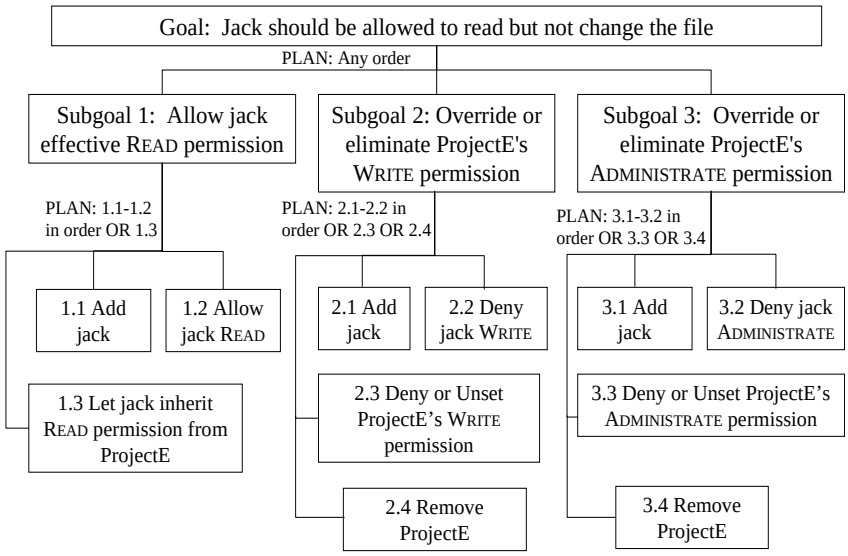

Fig. 3. Hierarchical task analysis (HTA) for the Jack task.

permissions in an interface to "see what happens" and then changing them back.

7.4 Classifying errors by type

Actions that were classified as errors were further classified according to error type: goal, plan, action, and perception errors. These four types were drawn from Pocock et al.'s THEA which describes four stages of human information processing (Pocock et al., 2001). There are many other frameworks that could have been used to classify human error. THEA was chosen because it was specifically designed for evaluating user interfaces, and because of its grounding in the familiar work of Norman (1988).

Goal errors, the focus of the present work, are described forthwith. Similar criteria were used to classify action, plan, and perception errors. An error was classified as a goal error if it was either: (1) an error of commission that was due to the user's establishing a wrong subgoal; or (2) an error of omission that was due to the user's failure to establish a necessary subgoal. An example of a common goal error from the Wesley task was a user failing to explicitly deny Wesley WRITE permission. In the Jack task, both failing to explicitly deny Jack WRITE permission (omitting Subgoal 2 in Figure 3) and failing to explicitly deny Jack ADMINISTRATE permission (omitting Subgoal 3 in Figure 3) were goal errors. In the Tux task, denying Tux WRITE permission was a goal error.

8 Results

This section presents the results of the study, including details of speed, accuracy and propensity for mitigating errors for each of the two interfaces under scrutiny. The results show that Salmon performed better than XPFP did.

8.1 Speed

Figure 4 shows the average task completion times for each of the two interfaces and three tasks. The solid bars show times for all participants, whether they succeeded or failed in the task; the striped bars show times only for participants who completed the tasks accurately. For Wesley and Jack, these results are of interest, because they show that the success of Salmon users was not due to having spent more time on task. In fact, those who completed the tasks successfully took less time using the Salmon interface. The difference between times for the two interfaces is not statistically significant (one-sided t-test for Wesley: t=0.3942, df=14.39, p=0.3496; for Jack: t=0.8973, df=9.367, p=0.1961). For the Tux task, successful Salmon users spent, on average, significantly less time (M=60.8s, sd=23.0) than the successful XPFP users did (M=178.0s, sd=108.6). A one-sided t-test showed this difference to be statistically significant at the .05 level (t=3.183, df=8.538, p=0.006).

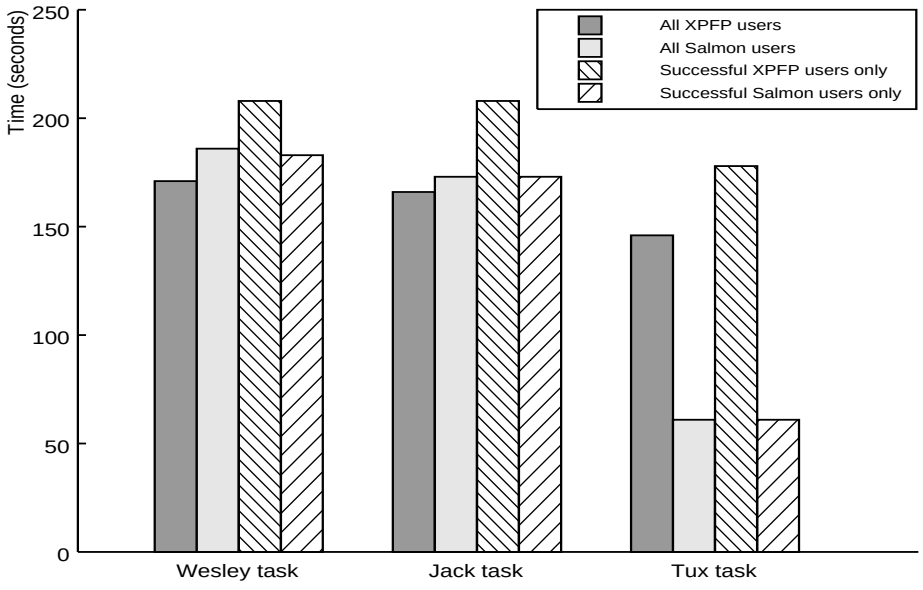

Fig. 4. Average time to complete Wesley, Jack and Tux tasks. Successful Salmon users took less time on average than successful XPFP users did, suggesting that Salmon's accuracy gains were not due simply to a speed-accuracy tradeoff.

8.2 Accuracy

Table 1 shows the percentage of participants who successfully (accurately) completed the tasks on the XPFP and Salmon interfaces. In testing whether or not Salmon gives superior performance in terms of accuracy (measured as task completions), a null hypothesis was posited: the proportion of task completions under Salmon was less than or equal to the proportion under XPFP; the alternative hypothesis is that the proportion for Salmon was greater. To test whether two sample proportions are equal, let $\hat{p}_S$ be the proportion of completions for Salmon and let $\hat{p}_X$ be the proportion for XPFP. The test statistic is $ts = \frac{\hat{p}_S - \hat{p}_X}{\sqrt{\bar{p}(1-\bar{p})(\frac{1}{n_S}+\frac{1}{n_X})}}$ where $\bar{p}$ is the total number of completions divided by the total number of subjects (e.g., for Jack, there were 15 (3+12) completions by 24 (12+12) subjects; thus $\bar{p} = 15/24$).

Table 1
Percent of accurate task completions for the Wesley, Jack and Tux tasks on the XPFP and Salmon interfaces. Salmon outperformed XPFP on all tasks.

	XPFP	Salmon
Wesley	58%	83%
Jack	25%	100%
Tux	75%	100%

A one-sided z-test was done for the equality of proportions. Since the null hypothesis specifies equality, one uses the pooled estimate of the standard error of the difference between the two proportions to calculate the likely size of the chance error in estimating the true difference between the proportions. The test statistic, *ts*, for the results of the Jack task was 3.795, which on referral to a z-table has significance probability less than .0001. Thus the null hypothesis can be strongly rejected; for the Jack task the Salmon interface is superior. Salmon is also superior for the Tux task, with a test statistic of 1.852 and significance probability .032; the Wesley task test statistic of 1.347 is weakly significant, with a significance probability of .089.

Recall that participants were asked to state their confidence in their work. Curiously, there was no significant difference between interfaces in participants' confidence ratings, on any task, as to whether tasks were completed correctly or not. XPFP users gave slightly higher confidence ratings than Salmon users for the Wesley (XPFP: M=5.58, sd=1.56; Salmon: M=4.83, sd=1.53) and Jack (XPFP: M=5.58, sd=1.39; Salmon: M=5.17, sd=1.64) tasks, and gave slightly lower confidence ratings for the Tux task (XPFP: M=5.50, sd=1.45; Salmon: M=6.17, sd=1.11). One-sided, unpaired t-tests

show Wesley (t=1.19, df=21.99, p=0.88); Jack (t=0.67, df=21.36, p=0.75); and Tux (t=-1.26, df=20.66, p=0.11) with no significant differences between XPFP and Salmon with respect to user confidence. This is interesting in view of the very different actual outcomes; that is, Salmon facilitated much better success than XPFP did, and yet the perception of participants regarding their success was about the same. This bears investigation, but is not further addressed here.

8.3 Errors

Table 2 shows results of the error analysis for the three tasks; XPFP dominated in terms of total errors and goal errors. For the Wesley task, 5 of 9 XPFP-user errors were goal errors; 1 of 4 Salmon-user errors was a goal error. For the Jack task, 15 of 16 XPFP-user errors were goal errors; 1 of 6 Salmon-user errors was a goal error. For the Tux task, 2 of 3 XPFP user errors were goal errors; Salmon users made no errors.

Table 2
Counts of errors by type for the Wesley, Jack and Tux tasks on XPFP and Salmon interfaces. Significantly fewer goal errors were made with Salmon than with XPFP on the same tasks.

	Goal	Plan	Action	Perception
Wesley				
XPFP	5	1	0	3
Salmon	1	2	1	0
Jack				
XPFP	15	0	0	1
Salmon	1	3	2	0
Tux				
XPFP	2	1	0	0
Salmon	0	0	0	0

In testing whether or not Salmon gives superior performance in terms of reducing the likelihood of goal errors, a null hypothesis was posited: the proportion of goal errors under Salmon was greater than or equal to the proportion under XPFP; the alternative hypothesis is that the proportion for Salmon was

smaller. As described in Section 8.2, a one-sided z-test was done for the equality of proportions. Since the null hypothesis specifies equality, one uses the pooled estimate of the standard error of the difference between the two proportions to calculate the likely size of the chance error in estimating the true difference between the proportions. The test statistic, *ts*, for the results of the Wesley task was -1.885, which on referral to a z-table has a significance probability of .0297. Thus one can reject the null hypothesis at the .05 level; the Salmon interface is superior for Wesley. For the Jack task, *ts* =-3.312, and the z-table significance probability is .0005, which is highly significant. For the Tux task, *ts* =-1.477, and the z-table significance probability is .0698, which is mildly significant. It can be concluded that in terms of mitigating goal errors, the Salmon interface performs better than the XPFP interface does.

9 Discussion

A straightforward comparison of accuracy results shows that Salmon outperformed XPFP in all three tasks. The causes for XPFP's poor performance and Salmon's improved performance can be inferred from the error classification data, which shows a substantial reduction in goal errors amongst Salmon users. For some insight into whether Salmon's improved external representation was responsible for the improved performance, users' protocols were analyzed to determine the amount of time spent on each of ten high-level behaviors. The analysis implies that Salmon's external representation of task-relevant data leads users more readily toward finding the information they need.

9.1 Goal errors in XPFP

Goal errors were common amongst XPFP users and led to high failure rates. The preponderance of goal errors observed in XPFP is not surprising; as noted in Section 4, the XPFP interface lacks information vital to the file-permissions problem solver (see Figure 1). In the main XPFP window, there is no display of effective permissions, no way to access group membership information, and no indication of the existence of Administrate permissions. Without effective permissions, progress toward the primary goal cannot be checked accurately. Many users, unaware of the distinction between stated and effective permissions, used the stated permissions to determine whether the primary goal had been completed. For example, during the Wesley task, many users saw the XPFP window in the state shown in Figure 1. From this display, it appeared to many users that Wesley was allowed Read permission because his Allow-Read checkbox is checked, but not allowed Write permission, because his Allow-Write checkbox is not checked. They did not realize that he has ef-

fective WRITE permission from ProjectF. Users hence were misled into thinking they had completed the task, even though they had not.

9.2 Fewer goal errors in Salmon

The improvement in task-completion successes and the dramatic reduction in goal errors achieved in the Salmon study can be accounted for primarily by the use of external subgoal support in the design of the Salmon interface. Users of Salmon were able to track their progress and identify the subgoals necessary to complete tasks using the effective permissions display, which provided an external representation supporting subgoaling. Although Salmon's design contains numerous superficial changes from the XPFP design (such as different fonts, labels, icons, colors, and layout), observation of participants' protocols strongly suggests that it was the effective permissions display that led users to formulate the correct goals. For example, one Salmon participant, about to commit an incorrect solution to the Jack task, said, "I see Jack over here now [pointing to Jack's stated permissions], and he doesn't have any access rights... Oh, wait! Jack has access rights over here [pointing to Salmon's effective permissions display]." After noticing the effective permissions display, the participant was able to correctly complete the task. In contrast, one typical XPFP participant, looking at Wesley's stated permissions in the XPFP window as shown in Figure 1, said, "And apparently his permissions are just READ. That's what we want." He had not explicitly denied WRITE permission to Wesley, and implemented an incorrect solution. In the absence of correct information to confirm that the task was complete, the participant used incorrect information, the stated permissions, to "confirm" that he had correctly completed the task. As opposed to the XPFP interface, Salmon's external representation enables the user to avoid having to build and maintain an internal mental representation of a complex permission-setting structure. The Salmon display provided the necessary goal cues by holding the representation externally, making it more accessible to the user.

9.3 Less time searching for information in Salmon

To determine in more detail how users spent their time, and to address the hypothesis that Salmon users' improvement was due to improved external representations, users' protocols were analyzed for time spent on specific behaviors. This kind of analysis reveals how an interface shapes users' behavior, by drawing attention to activities where users spent the bulk of their time. Ten behaviors were identified, and each user protocol was segmented into portions of time spent on each of the ten behaviors. Objective criteria determined

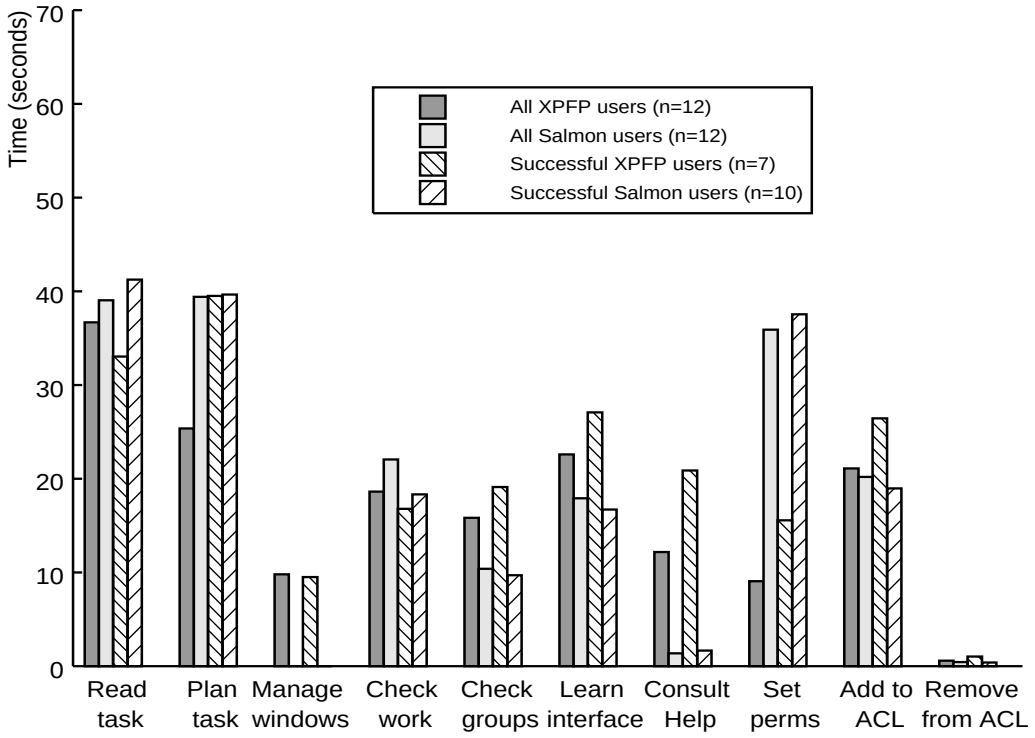

Fig. 5. Average time spent on each of ten behaviors identified in the Wesley task by all XPFP and Salmon users combined, whether successful or not, and only successful XPFP and Salmon users. Salmon users spent less time on information-gathering behaviors like check-groups, learn-interface, and consult-help.

whether a given segment of protocol matched a particular behavior. For example, a "Read task" behavior was identified any time a user had the Web browser with the task statement in the foreground. As another example, a "Learn interface" behavior was identified as the time between first reading the task statement and starting to enact a plan, or any action that followed a user statement like "Let's see what this does." Figure 5 shows the average amount of time users spent on each of these ten behaviors for the Wesley task; results for the Jack task are similar, and are not shown. Figure 6 shows the average amount of time that users spent on each of these ten behaviors for the Tux task. Each set of four bars shows, from left to right, the average times for all XPFP users, all Salmon users, successful XPFP users only, and successful Salmon users only. Descriptions and discussion of each of the ten behaviors is included below.

- **Read task.** This behavior consists of reading the task statement in the Web browser. Since the task statements were presented in the same manner to all users, it is to be expected that users of both interfaces would take about the same amount of time, on average, to read them and refer to them while working.

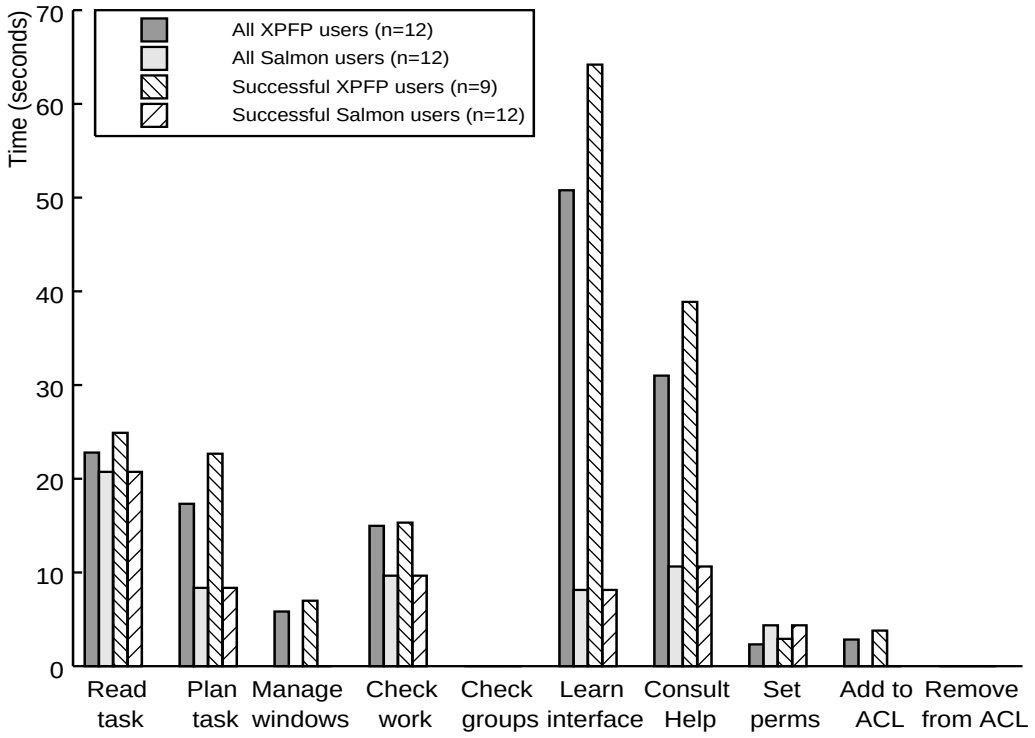

Fig. 6. Average time spent on each of ten behaviors identified in the Tux task by all XPFP and Salmon users, whether successful or not, and all successful XPFP and Salmon users. Salmon users spent less time on information-gathering behaviors like plan-task, learn-interface, and consult-help.

- **Plan.** This behavior is assumed to occur when the user is not engaged in any of the other nine behaviors on this list; it consists of thinking and trying to formulate a plan. Notice that, for the Wesley task, successful XPFP users and Salmon users spent more time planning than did all XPFP users. This can be explained by noting that those XPFP users who failed to complete the Wesley task typically failed by omitting a subgoal, and hence finished more quickly than those who successfully completed all subgoals. Notice, too, that XPFP users in the Tux task spent more than twice as much time planning, since without a visible DELETE permission-setting checkbox, it was harder to decide what to do.
- **Manage windows.** The manage-windows behavior includes the activities of moving, opening, closing, minimizing, or otherwise manipulating interface windows. Time spent on the dialog windows for Check groups, Read task, Consult Help, and Add to ACL behaviors is not included within manage-windows. Salmon users did not exhibit managing-windows behavior, since Salmon only has one interface window, while XPFP has as many as four. Consequently, it takes XPFP users longer to navigate through the multiple windows to find information. Salmon does take up more space on the screen,

however, in order to display all the relevant information in one window.

- **Check work.** This behavior consists of reviewing the interface to confirm that permissions look correct; it occurs after setting permissions and before ending the task or revising work. It is interesting to note that although users spent roughly the same amount of time checking work on both interfaces for all three tasks, there were still more failures amongst those using XPFP. This is likely because the information needed to check work, the effective permissions, is not visible in XPFP, so users employ the wrong information, the stated permissions, to check their work.
- **Check groups.** This behavior consists of checking group membership with the XP Computer Management interface. XPFP does not display group membership information, so it requires users to view the separate Computer Management application to check group membership. Salmon's lower pane displays group information, so it is not necessary to go to another application to check group membership. (Some users did anyway, because they had not yet noticed that the information was available in Salmon). Unsurprisingly, Salmon users spent less time checking group memberships than did XPFP users in the Wesley task. Since groups were not relevant to the Tux task, no users spent time checking groups.
- **Learn interface.** This behavior consists of either looking over the interface at the beginning of the task to see what can be done with it, or experimenting with the interface by clicking on interface elements whose function is unknown or confusing. Novice users spend time surveying a new interface and testing out its functionality to see how it works. Often, when at an impasse, users may test an unfamiliar interface function to see if it helps them accomplish their goal. Since Salmon was a completely new interface, while some users had already used the XPFP interface before participating in the study, it was expected that users would spend more time learning the Salmon interface. This does not appear to have been the case; apparently confusion led many XPFP users to experiment more with that interface. This phenomenon was especially apparent in the Tux task, where XPFP users spent, on average, six times longer than Salmon users exploring and experimenting with the interface.
- **Consult Help.** This behavior consists of browsing or reading the Help files. Users consult Help when confused about what to do next or about how an interface works. XPFP users spent far more time consulting Help than did Salmon users, presumably because they were having more difficulty using the XPFP interface.
- **Set permissions.** This behavior consists of reading permissions labels and clicking on their corresponding checkboxes. In the Wesley task, after users had added Wesley to the ACL, they would scan through the list of permissions labels in the interface. Salmon lists all 13 atomic permissions labels, while XPFP lists only 6 composite permissions on its main window. Since there are fewer labels to scan in XPFP, those users set permissions faster, on average. However, the added efficiency in XPFP comes at a price; all of

the users who failed at the Jack task failed in part because they did not notice that Jack had ADMINISTRATE permission. They failed because the ADMINISTRATE permission is hidden two screens away. Furthermore, in the Tux task, XPFP users spent far more time searching for the DELETE permission, since it is hidden in XPFP, but readily visible in Salmon.

- **Add to ACL.** This behavior consists of selecting a user or group, and adding it to the ACL. Salmon uses essentially the same dialog as XPFP does for adding users to the ACL, so little difference in the behavior times between the two interfaces is expected here.
- **Remove from ACL.** This behavior consists of selecting a user or group, and removing it from the ACL. The Remove operation was essentially the same in both interfaces, and was rarely used.

As this detailed behavior analysis shows, Salmon users spent less time than XPFP users on the behaviors related to information gathering (Check groups, Learn interface, and Consult Help) and spent a greater proportion of their time on essential task behaviors (Add to ACL, Set permissions, and Check work). This observation supports the hypothesis that an improved external representation of task-relevant information – due to the use of ESS in the design of Salmon – leads users to spend less time looking for information and leads users to find more readily the information they need to complete tasks.

10 Conclusion

In the course of completing tasks with a user interface, users look for information to formulate goals and to check progress. When the necessary information is misleading or absent, users fail to establish the correct goals and hence make goal errors. Goal errors may lead to total task failure, and thus impinge on interface dependability. Many goal errors can be prevented by providing an accurate, clear and salient external representation of the information needed to achieve the user's primary goal. A design method for producing such a representation has been named external subgoal support.

The Windows XP file permissions interface, which does not use external subgoal support, was shown to have unacceptably low success rates – 58%, 25% and 75% – on three representative file permission-setting tasks. Salmon, an alternative interface designed in accordance with the external subgoal support principle, was shown to increase percentage of successes to 83%, 100% and 100% on the same tasks. Furthermore, user tests with Salmon showed a dramatic reduction in the occurrence of goal errors compared to XPFP, with 80% fewer goal errors on one task, 94% fewer goal errors on another, and 100% on a third. In addition, Salmon maintained or substantially reduced time to task completion for all three tasks. These improvements in successful task comple-

tion, reductions in goal error occurrence, and time to task completion were due primarily to external subgoal support. These speed and success rates far more closely approach what is needed for dependable user interfaces in mission-critical systems like those required for setting security-related configurations of servers, firewalls, personal workstations, etc.

11 Future

External subgoal support has been demonstrated in a first step to be a successful design technique for reducing goal errors in the domain of file-permission setting, but the technique will need to be tested in other task domains before it is fully proven. Testing in additional task domains will also help to define its limits, and potentially reveal areas in which it cannot by itself reduce goal errors. The present work attempted to reduce only one type of user interface error – goal errors. Future work will look at means to reduce plan, action, and perception errors as well. In addition, interface retention could be explored by asking both Salmon and XPFP participants to return after two months for retesting, with the expectation that one interface or the other would be better retained and would therefore offer sustained (if not improved) performance over the earlier session. Such an exploration may yield hints about the form of interface best suited to occasional use, as would be the case for nearly any permission-setting application.

12 Acknowledgements

The authors are grateful for the generous help graciously given by our colleagues Fahd Arshad, David Banks (Duke University), Patricia Loring and Rachel Roberts. This work was partially supported by the Army Research Office through grant number DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to Carnegie Mellon University's CyLab, and partially supported by the Engineering and Physical Sciences Research Council, United Kingdom, grant number GR/S29911/01 (Senior Visiting Fellowship for first author).

References

Adams, A., Sasse, M. A., 1999. Users are not the enemy. Communications of the ACM 42 (12), 41–46.

Balfanz, D., 2003. Usable access control for the World Wide Web. In: Proceedings of the 19th Annual Computer Security Applications Conference. IEEE Computer Society, Los Alamitos, CA, pp. 406–415, 08-12 December 2003, Las Vegas, NV.

Ballard, D. H., Hayhoe, M. H., Pelz, J. B., 1995. Memory representations in natural tasks. Journal of Cognitive Neuroscience 7 (1), 66–80.

Besnard, D., Arief, B., 2004. Computer security impaired by legitimate users. Computers & Security 23 (3), 253–264.

Card, S., 2003. Information visualization. In: Jacko, J. A., Sears, A. (Eds.), The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications. Lawrence Erlbaum Associates, Mahwah, NJ, Ch. 28, pp. 544–582.

Dewan, P., Shen, H., 1998. Controlling access in multiuser interfaces. ACM Transactions on Computer-Human Interaction 5 (1), 34–62.

Ericsson, K. A., Simon, H. A., 1993. Protocol Analysis: Verbal Reports as Data, Revised Edition. MIT Press, Cambridge, MA.

Good, N. S., Krekelberg, A., 2003. Usability and privacy: a study of Kazaa P2P file-sharing. In: Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003). ACM Press, New York, NY, pp. 137–144, 05-10 April 2003, Fort Lauderdale, Florida.

Kirwan, B., 1994. A Guide to Practical Human Reliability Assessment. Taylor & Francis, London, United Kingdom.

Long, A. C., Moskowitz, C., Ganger, G., November 2003. A prototype user interface for coarse-grained desktop access control. Tech. Rep. CMU-CS-03-200, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Microsoft Corporation, 2005a. Best practices for permissions and user rights. Available at `http://www.microsoft.com/resources/documentation/windowsserv/2003/stand%ard/proddocs/en-us/sag_SEconceptsImpACBP.asp`.

Microsoft Corporation, 2005b. Microsoft Technet: Windows XP file permissions documentation. `http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prod%technol/winxppro/proddocs/acl_special_permissions.asp`.

Mitchell, C. M., Miller, R. A., 1986. A discrete control model of operator function: A methodology for information display design. IEEE Transactions on Systems, Man, and Cybernetics SMC-16 (3), 343–357.

Nielsen, J., Mack, R. L., 1994. Usability Inspection Methods. John Wiley & Sons, Inc., New York, NY.

Norman, D. A., 1988. The Design of Everyday Things. Doubleday, New York, NY.

Pocock, S., Harrison, M., Wright, P., Johnson, P., 2001. Thea: A technique for human error assessment early in design. In: Proceeding of 8th IFIP TC.13 Conference on Human-Computer Interaction (INTERACT'01). IOS Press, Amsterdam, pp. 247–254, 09-13 July 2001, Tokyo, Japan.

Reason, J., 1990. Human Error. Cambridge University Press, Cambridge, UK.

Sampemane, G., Naldurg, P., Campbell, R. H., 2002. Access control for active spaces. In: Proceedings of the 18th Annual Computer Security Applications Conference. IEEE Computer Society, Los Alamitos, CA, pp. 343–352, 09-13 December 2002, Las Vegas, NV.

Senders, J. W., Moray, N. P., 1991. Human Error: Cause, Prediction, and Reduction. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Smith, R., March 2004. Personal communication.

U.S. Senate Sergeant at Arms, March 2004. Report on the investigation into improper access to the Senate Judiciary Committees computer system. Available at `http://judiciary.senate.gov/testimony.cfm?id=1085&wit_id=2514`.

Whitten, A., Tygar, J., 1999. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In: Proceedings of the 8th USENIX Security Symposium. USENIX Association, Berkeley, California, pp. 169–184, 23-26 August 1999, Washington, DC.

Whitten, A., Tygar, J., 05-10 April 2003. Safe staging for computer security. Presented at the Workshop on Human-Computer Interaction and Security Systems, part of ACM Conference on Human Factors in Computing Systems (CHI 2003), Fort Lauderdale, Florida.

Wiegmann, D. A., Shappell, S. A., 2003. A Human Error Approach to Aviation Accident Analysis. Ashgate Publishing Co., Aldershot, Hants, United Kingdom.

Woods, D. D., 1985. Paradigms for intelligent decision support. In: Hollnagel, E., Mancini, G., Woods, D. D. (Eds.), Intelligent Decision Support in Process Environments. Springer-Verlag, Berlin, pp. 153–173.

Woods, D. D., Roth, E. M., 1988. Cognitive systems engineering. In: Helander, M. (Ed.), Handbook of Human-Computer Interaction, 1st Edition. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, Ch. 1, pp. 3–43.

Yee, K., 2002. User interaction design for secure systems. In: Information and Communications Security, 4th International Conference, ICICS 2002, Singapore. Lecture Notes in Computer Science, Vol. 2513. Springer, New York, NY, pp. 278–290, 09-12 December 2002, Singapore.

Zhang, J., 1996. A representational analysis of relational information. International Journal of Human-Computer Studies 45 (1), 59–74.

Zhang, J., 1997. The nature of external representations in problem solving. Cognitive Science 21 (2), 179–217.

Zhang, J., Norman, D. A., 1994. Representations in distributed cognitive tasks. Cognitive Science 18 (1), 87–122.

Zurko, M. E., Simon, R., Sanfilippo, T., 1999. A user-centered, modular authorization service built on an RBAC foundation. In: Proceedings 1999 IEEE Symposium on Security and Privacy. IEEE Computer Security Press, Los Alamitos, CA, pp. 57–71, 09-12 May 1999, Berkeley, California.

Zurko, M. E., Simon, R. T., 1996. User-centered security. In: Proceedings of Workshop on New Security Paradigms. ACM Press, New York, NY, pp.

27–33, 17-20 September 1996, Lake Arrowhead, CA.