User Interface Dependability through Goal-Error Prevention

Robert W. Reeder and Roy A. Maxion
reeder@cs.cmu.edu and maxion@cs.cmu.edu

Dependable Systems Laboratory
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 / USA

Abstract

User interfaces form a critical coupling between humans and computers. When the interface fails, the user fails, and the mission is lost. For example, in computer security applications, human-made configuration errors can expose entire systems to various forms of attack.

To avoid interaction failures, a dependable user interface must facilitate the speedy and accurate completion of user tasks. Defects in the interface cause user errors (e.g., goal, plan, action and perception errors), which impinge on speed and accuracy goals, and can lead to mission failure.

One source of user error is poor information representation in the interface. This can cause users to commit a specific class of errors – goal errors. A design principle (anchor-based subgoaling) for mitigating this cause was formulated. The principle was evaluated in the domain of setting Windows file permissions. The native Windows XP file permissions interface, which did not support anchor-based subgoaling, was compared to an alternative, called Salmon, which did. In an experiment with 24 users, Salmon achieved as much as a four-fold increase in accuracy for a representative task and a 94% reduction in the number of goal errors committed, compared to the XP interface.

1 Introduction

One locus of vulnerability in a computer system is the user interface. Undependable interfaces are those that do not meet their specifications in terms of the speed and 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 similar across all domains of human endeavor, from aviation, to power plant operation, to making a cup of tea [13, 15, 17, 21]. In the domain of software user interfaces, human error leads people off the path of correctly completing a task and on to lengthy delays or partial or total task failure. Thus, it is imperative for 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 humans to make mistakes, interfaces will be more dependable.

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 [20] 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). A significant amount of anecdotal evidence suggests that setting NTFS file permissions is a particularly error-prone task. For example, there is the so-called "Memogate" scandal, in which staffers from one party on the United States Senate Judiciary Committee stole confidential memos from the opposing party [19]. 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 [18]. Finally, Microsoft publishes a list of "best practices" for NTFS security that

advises users not to use several of the features of the NTFS permissions model, such as negative (i.e., DENY) permissions [10]. 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 data. Nevertheless, 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 files or grant access to a limited number of associates. They will not readily remember arcane details about how to operate a permission-setting interface. The present work adopts the position that permission-setting interfaces should accommodate novice and occasional users.

This paper reports an investigation into, and a solution for, one type of human error encountered in permission-setting interfaces. An existing interface for setting NTFS permissions, the Windows XP File Permissions interface (hereafter abbreviated XPFP), was shown to have accuracy rates as low as 25% on permission-setting tasks. Errors made by users of 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 a poor external representation of the information needed to complete the user's root goal – a representation that is sometimes called an anchor (for concepts held in human memory) – was identified. A design principle, anchor-based subgoaling, 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 comparing Salmon to XPFP. Salmon achieved a success rate of 100% on the task on which XPFP had achieved a 25% accuracy rate, and showed a 94% reduction in the number of goal errors users made on the same task.

2 Problem and approach

The objective of the present work is to understand the causes of user error in user interfaces generally, and in XPFP in particular, and to determine what can be done to mitigate or eliminate them. It is a further objective to find a design principle that can be applied to new generations of user interfaces so that the same user errors are not encountered again and again in future user interfaces.

A visual inspection and informal use of the XPFP interface revealed several instances of an interface problem. Specifically, information that was necessary to complete tasks accurately was hidden or was entirely missing from the interface. Cognitive theory suggests that without ready access to necessary information, users are likely to commit goal errors (see Section 5.1). A solution was proposed to reduce the occurrence of goal errors. This solution, called anchor-based subgoaling, led to the following hypothesis:

> Use of anchor-based subgoaling in user interface design reduces the likelihood that users will commit goal errors, and task accuracy rates should improve when goal errors are reduced.

This hypothesis was tested by implementing an interface, Salmon, designed in accordance with the anchor-based subgoaling procedure, and by conducting a laboratory user study comparing the XPFP interface to Salmon. Task success rates and goal-error occurrences were compared between XPFP and Salmon to determine whether anchor-based subgoaling as implemented in Salmon was an effective means of improving successful task completion and reducing goal errors.

3 Related work

File permissions are an instance of the broader area of access control in which several authors have published related work. Those who have evaluated interfaces for setting file access include Good and Krekelberg [7], Long et al. [9] and Zurko et al. [24]. Zurko et al. 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 showed that the Kazaa peer-to-peer file-sharing service's interface misled many users into unintentionally sharing confidential files. Long et al. 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 [2], Sampemane et al. [16], and Dewan and Shen [5]. 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 (an emerging field known as HCISEC), those who have proposed principles for better security interface design include Whitten and Tygar [20], Adams and Sasse [1], Besnard and Arief [3], Zurko and Simon [25], and Yee [23]. These authors propose ideas for making security tasks easier to perform accurately, but the ideas are not evaluated empirically, and neither do they appear to be grounded in any theory of cognition.

The most closely related concept in the traditional human-computer interaction literature to anchor-based subgoaling is the concept of feedback. Norman defines feedback as "sending back to the user information about what action has actually been done, what result has been accomplished [13]." Norman and Nielsen both include feedback as an important user interface design principle [12, 13], but neither explain *how* to ensure that adequate feedback is provided. Anchor-based subgoaling includes an actionable procedure (see Section 5.2) for ensuring that not only feedback, but also other forms of needed information are provided in the interface.

4 Problem in context

The problem of poor external representation can be observed in the Windows XP file permissions interface (XPFP). Users maintain mental models (internal representations) of tasks that may be incorrect or incomplete. In the course of performing a task, they will update their model with the information they perceive externally. Thus, for an interface to be robust to inaccurate mental models, it must provide the information users need to properly update their mental models, and it must provide this information in a way that users will perceive it. XPFP hides or does not show much of the information necessary to foster understanding of task requirements and state, while Salmon, an alternate file permissions interface, prominently displays this information and hence requires that only a minimal internal representation be maintained. In order to understand what information is lacking in the XPFP interface and present in Salmon, some background on the NTFS permissions model is necessary, and is introduced forthwith.

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 [11]. 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]. Of these, only READ, WRITE, and ADMINISTRATE will figure into the tasks for the present study. ADMINISTRATE permission deserves one special note – an entity that is allowed ADMINISTRATE permission can change its own permissions, so even if it were denied READ or WRITE permission on an object, it would still be able to read or write the object by allowing itself that permission first.

NTFS uses an *Access Control List (ACL)* model of file permissions. 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. An entry in this list is called an *access control entry*. The access control entry for each user or group on the list has a setting for each of the atomic permissions. This setting's value may be ALLOW, DENY, or NOTSET. The meaning of ALLOW and DENY are self-evident. NOTSET implicitly means that neither ALLOW nor DENY has been checked, or turned on; it acts as a DENY by default, but can be overridden by a competing ALLOW setting, as described below.

Under the rule of *group inheritance*, if a group entity has permissions on an object, all members of the group inherit the group's permissions for the object. Conflicts between a user's explicit permissions and their inherited permissions are resolved through precedence rules. Specifically, DENY settings take precedence over ALLOW settings, but ALLOW settings take precedence over NOTSET settings. So if a user has an ALLOW READ setting for a file, but inherits a DENY READ setting for that file, the user will be denied access to read the file. Group inheritance and precedence rules lead to the distinction between *stated permissions*, the permissions contained in a user's access control entry, and *effective permissions*, the actual access a user will be allowed.

The distinction between stated permissions and effective permissions can make setting file permissions a difficult task, because the low-level permission values on which a user operates do not necessarily translate directly into what access will be allowed to system data. Actual access in NTFS is determined by a subtle formula accompanying the precedence rules. However, users setting file permissions are ultimately concerned with who can access what, not with low-level values and nuanced formulas. Thus the necessary information for users to evaluate goal status is the *effective permissions*, which are the output of the nuanced formula, and which reflect the actual access that will be granted to files.

Casual observation of the XPFP interface (see Figure 1) reveals that XPFP provides a poor external representation of the file-permissions task. For example, the XPFP main window contains the checkboxes necessary for setting the permission values that will be used to determine effective permissions, but effective permissions themselves are nowhere to be seen. In fact, XPFP *can* display effective permissions, but they are two screens away. To see them, users must click the Advanced button, then select an "Effective Permissions" tab. Even then, users must go through an extra step to choose whose effective permissions they want to see; viewing multiple entities' effective permissions si-

[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 the lowest-level, indivisible permissions in the system.

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

multaneously is impossible. When users return to changing permission values, the effective-permissions display disappears. Users need to know the effective permissions to check their work. Without an effective-permissions display readily available, or even a cue to indicate their importance, users are forced not only to maintain in their minds the inheritance and precedence rules, but also to compute the effective permissions mentally.

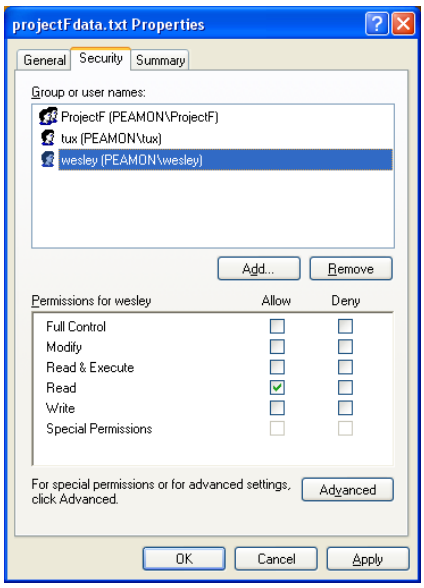

Figure 1: The XPFP interface. The interface contains information and functionality for setting permission values, but effective permissions, group-inherited permissions, and ADMINISTRATE permissions are not visible.

Besides the lack of an accessible effective-permissions display, XPFP also hides the ADMINISTRATE permission setting two screens away from the main window, behind the Advanced button, and does not display information about users' group membership or their group-inherited permissions anywhere. This makes it difficult to track down the source of an effective permission that was inherited from a group. Furthermore, users must maintain group membership and permission settings information in their heads while completing tasks – a situation that begs for error.

5 Anchor-based subgoaling

Anchor-based subgoaling (ABS) is a principle for ensuring that a user interface provides all the information a user will need to complete the tasks for which the interface is intended, and provides such information in a clear and accurate display that the user will notice. The XPFP interface illustrates the problem of omitting or obscuring necessary task information, the problem addressed by ABS.

5.1 Cognitive theory

Anchor-based subgoaling is rooted in the cognitive theory of Pocock et al.'s Technique for Human Error Assessment (THEA) [14], which is in turn based on Norman's well-known seven-stage execution-evaluation model of human information processing [13]. 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 root 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 problem is solved: exit 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 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 root goal is critical during correct subgoal selection. If the salient information suggests an inappropriate subgoal selection, then goal errors are likely to result.

The potential for goal errors can be seen in the XPFP interface. Since effective permissions are hidden, most users will not find them, and will determine problem completion based on either wrong, but visible, information in the interface, or on their own error-prone mental computations. Hidden group information may result in users not knowing when a group permission is being inherited. Since the ADMINISTRATE permission is hidden, users may never realize it exists. Lacking this information, users are prone to making goal errors.

5.2 Anchor-based subgoaling design procedure

Anchor-based subgoaling bridges the gap between task analysis and the creation of an external representation during the interface design process. The ABS procedure ensures that the necessary information is represented. A careful task analysis is a prerequisite for implementing ABS; Kirwan [8] is an excellent reference on how to perform task analyses. Kirwan describes the hierarchical task analysis (HTA) method, which includes a convenient representation

for the results of a formal task analysis. An HTA represents a 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; each subgoal may have a tree of subgoals beneath it. At the leaves of the hierarchy are the actionable operations necessary to achieve each of the lowest-level subgoals. A detailed example appears in Section 7.1.

After the task analysis is completed, the ABS design procedure can begin. It proceeds as follows:

- Phase 1: Identify the information that is 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 to:
     (a) determine when the goal has been completed;
     (b) set the subgoals beneath the goal.
  2. For each operation at the leaves 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 will have to be supplied to the operation.
- Phase 2: Provide the information in the interface. Incorporate into the interface design an accurate, clear, and salient representation of the necessary information, as determined by the above steps.[3]

5.3 Salmon interface

The Salmon interface (see Figure 2) was designed in accordance with anchor-based subgoaling. Anchor-based subgoaling identified the following information as necessary for establishing the correct subgoals and executing the correct operations in a file-permissions interface:

1. The full list of 13 atomic permissions;
2. Stated permissions for all users & groups on the ACL;
3. Group membership data, and how it combines to a user's effective permissions;
4. Effective permissions for all users on the ACL.

The Salmon interface was designed to provide this information. Its main window comprises two panes. In the upper pane are the checkboxes necessary for setting permission values. Each column of checkboxes has a label corresponding to one of the 13 atomic permissions. These checkboxes show the *stated permissions*. In the lower pane is an effective-permissions display that shows both the effective permissions and the group and individual permissions that combine to the *effective permissions*. The effective-permissions display is available at all times. When changes are made in the upper pane, they are reflected in the lower pane, and attention is drawn to the lower pane by highlighting the areas that have changed. While effective permissions for all users cannot be viewed on screen at the same time, Salmon's lower pane can be scrolled to access information that does not fit on the screen. The necessary information is readily available in the Salmon interface, and can be viewed as it is needed.

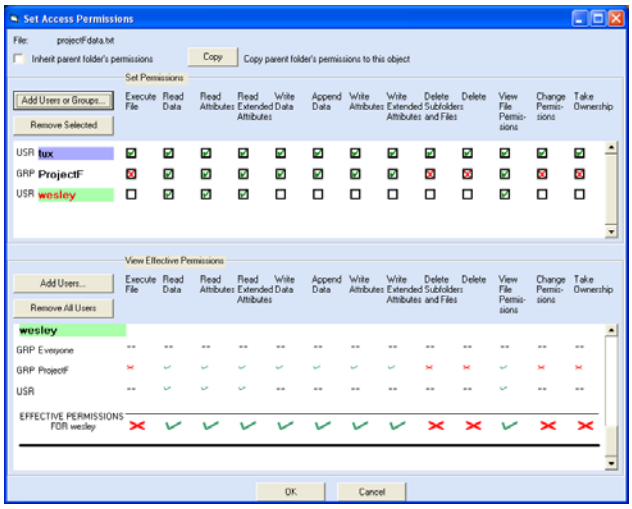

Figure 2: The Salmon interface. The upper pane of the Salmon interface contains the same information and functionality that is contained in the XPFP main window. The lower pane contains an effective-permissions display.

6 Methodology

A laboratory user study was conducted to observe and document errors in permission-setting tasks. Two interfaces were compared: XPFP, which lacks salient representations of task-relevant information, and Salmon, which was designed using the anchor-based subgoaling design procedure.

6.1 Participants

Twenty-four students and research staff at Carnegie Mellon University voluntarily participated in the study. 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

[3] External representation design is a large topic and is not covered in detail here; see Card et al. [4] or Woods and Roth [22] for more information on this subject.

operating system, while 5 reported having no experience whatsoever in setting file permissions. All but 4 reported setting file permissions a few times a month or less. Thus, the participant pool was consistent with the assumption of occasional users, who are likely to have to relearn the task, at least partially, due to infrequent use of the interface.

6.2 Apparatus

All participants worked on the same computer, a system running Windows XP, Version 2002, Service Pack 1. Using a standard think-aloud experimental paradigm [6], participants were asked to think aloud as they worked, while their voice was recorded. Screen video and mouse and keyboard actions were recorded with a software tool developed for user study data collection. Participants' final permissions settings were saved after each task instance.

6.3 Task design

To simulate real permission-setting conditions, a hypothetical scenario was designed in which the participant worked in a generic "organization," shared a computer with other workers in the organization, and had to restrict access to the files and folders on her computer. The hypothetical organization's computer environment – populated with individual users, groups containing users, files, and folders – was created on the laboratory Windows XP machine. 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 was to represent the participant. 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 randomly assigned to one of the two interfaces, Salmon or XPFP, and were each given seven tasks to perform with the same interface. The first of the tasks was a simple training task to give participants a quick introduction to the interface they were using. All participants performed this task first, and it was excluded from analysis. The remaining 6 tasks consisted of a variety of existing-permissions contexts, only two of which, called the Wesley and Jack tasks, are discussed here. These two tasks involved group inheritance, a feature of the NTFS permissions model that is especially error-prone. Four tasks were excluded from analysis: one had flawed instructions; two were too easy, providing no discrimination between interfaces; and one was not analyzed due to time constraints.

The Wesley and Jack tasks required participants to set permissions on a text file so that the entities "Wesley" or "Jack" could read the file, but could not change it. The task statement presented to participants for each task was identical except for the names of specific files, users, and groups. The task statement for the Wesley task read as follows:

> The group ProjectF is working on projectF-data.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 projectF-data.txt, but in no way change its contents.

The difference between the Wesley and Jack tasks was the way in which permissions were initially set up. In each task, there was one group (ProjectF or ProjectE, for the Wesley and Jack tasks, respectively) that was already on the access control list (ACL) for the file, and the operative individual user (Wesley or Jack) was a member of that group. The difference between the two 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 did 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

To ensure as realistic an environment as possible without compromising the experimental comparison between the two interfaces, it was necessary to establish certain rules for participants' interaction. First, participants in both interface conditions were allowed to look up group membership information using the XP Computer Management interface, which is a separate application from the file-permissions interfaces. However, participants were instructed not to use this interface to change group permissions. Had they been allowed to do so, they could have solved the Wesley and Jack tasks by removing Wesley or Jack from their respective groups without using the file-permissions interfaces, which would have defeated the purpose of comparing XPFP to Salmon. Second, to compensate for the restriction on changing group memberships, 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 permission settings for the groups ProjectE or ProjectF and not be concerned about the effects of these permission changes on members of the

groups other than Wesley and Jack. Finally, participants were allowed to access a set of online Windows Help files that applied to setting NTFS permissions, but they were not permitted to browse the entire set of Windows Help files.

6.5 Procedure

Participants were asked to think aloud during their sessions, and were instructed in doing so according to directions adapted from Ericsson and Simon [6]. Participants were shown how to view system users, groups, and group memberships using the XP Computer Management interface, and were shown how to access Help files. Participants were not given any instruction in using the XPFP or Salmon interfaces. Following instruction on the XP Computer Management interface and the Help files, participants were given the tasks. Before each task, the experimenter brought up the interface the participant was to use for the experiment. Then task statements were presented in text in a Web browser; these remained available to the participant throughout the task. All participants were given the training task first, but after that, presentation order of the remaining tasks was counterbalanced among participants using a Latin square design. Participants were given 8 minutes to complete each task (an expert could complete the task in under one minute).

7 Data analysis

Data from the user studies were analyzed for speed, accuracy, and error counts. Speed was straightforward to measure using time to task completion. Data analysis for accuracy and error results consisted of the following five steps:

1. For each of the two tasks, Wesley and Jack, apply a Hierarchical Task Analysis (HTA: see Section 7.1) to determine the steps necessary to complete the task;
2. For each task instance, determine whether the user succeeded or failed at completing the task;
3. For each task instance, list all actions taken by the user;
4. For each action taken, classify it as an error or a non-error by comparing it to the steps listed in the HTAs;
5. For each error, classify it as one of four types of error: goal, plan, action, or perception.

7.1 Step 1: Hierarchical Task Analysis

To aid in the identification of errors, a Hierarchical Task Analysis (HTA) was applied to the Wesley and Jack tasks. HTA, as described by Kirwan [8], is a tool for breaking a task into its constituents - the goals, plans, and actions required to complete the task. An HTA diagram for the Jack task is shown in Figure 3. As the figure shows, each task has a root goal that is decomposed into subgoals, which are in turn decomposed into actions. Plans express constraints on the choice or ordering of actions.

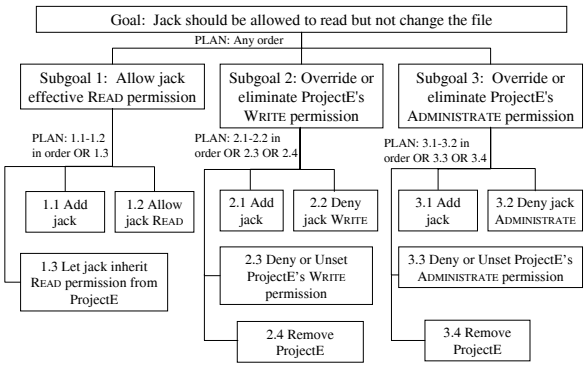

Figure 3: Hierarchical Task Analysis of the Jack task.

7.2 Step 2: Determining task success or failure

To determine task successes and failures, participants' final permission settings were examined. A task was judged successful if the operative individual (Wesley or Jack) had effective permissions allowing him READ permission and denying him WRITE and ADMINISTRATE permissions. A task was judged a failure if the operative 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.

7.3 Step 3: Listing actions

Actions were defined for the purpose of dividing user protocol data into discrete units for error analysis. An action was defined as any change to the access control list (ACL), i.e., adding an entity to or removing an entity from the ACL, or altering the permissions of an entity already on the ACL.

7.4 Step 4: Classifying actions as errors

Once actions for each task instance were listed, they were compared to the actions listed in the HTA for the corresponding task (Wesley or Jack). Each discrepancy between user actions and HTA 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 and could lead to failure if not recovered from. A user action 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 permissions in an interface to "see what happens" and then changing them back.

7.5 Step 5: Classifying errors by THEA type

Pocock et al.'s THEA [14] proposes four stages of human information processing that map directly to the four er-

ror types used to categorize errors in this work: goal, plan, action, and perception errors. Because error classification was not their main objective, Pocock et al. are not perfectly clear about the criteria for classifying a specific error as one of the four THEA types. However, all attempts were made to ensure that the error classification criteria used for this work remained faithful to Pocock et al.'s descriptions of the four types. Since goal errors are the focus of this paper, the criteria used to classify errors as goal errors are described below. Similar criteria were used for classifying the remaining errors as action, plan, or perception errors.

The data used to classify errors into types included verbal protocol, screen video, and mouse and keyboard logs. An error was classified as a *goal error* if it was either:

- An error of commission that was due to the user establishing a wrong subgoal; or
- An error of omission that was due to the user failing to establish a necessary subgoal.

Establishment of subgoals was determined mainly from the intentions users stated in their think-aloud protocols. 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 common goal errors.

There are numerous other frameworks that could be 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 [13].

8 Results

The Salmon and XPFP interfaces were evaluated with respect to speed, accuracy, and number of goal errors committed. Results for each of these are given in this section.

8.1 Speed

Salmon and XPFP were roughly comparable in speed, as measured by average time to task completion. Figure 4 shows the average time to task completion for all XPFP and Salmon users, and successful XPFP and Salmon users. Since many users who failed using XPFP failed by omitting essential task steps, they tended to reduce the average time to task completion, so comparing only successful users across the interfaces gives a more meaningful comparison. Although Salmon moderately outperformed XPFP in speed amongst successful users in both the Wesley (XPFP: M=208 seconds, sd=116; Salmon: M=183 seconds, sd=138) and Jack (XPFP: M=208 seconds, sd=42; Salmon: M=173 seconds, sd=109) tasks, the difference between the two interfaces was 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). Still, the results are of interest because they show that Salmon's gains in accuracy (next section) over XPFP are not simply due to a speed-accuracy tradeoff.

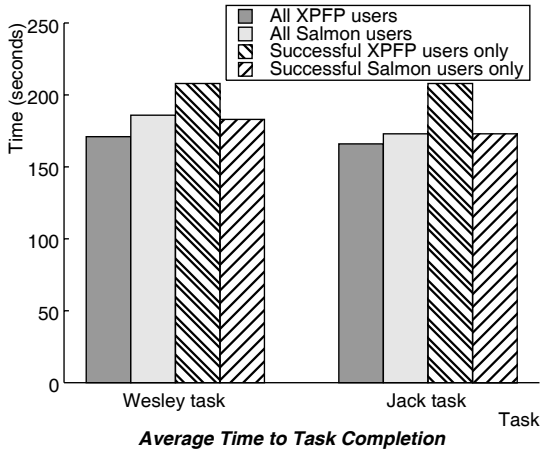

Figure 4: Average time to task completion for the Wesley and Jack tasks. Amongst successful users, Salmon moderately outperformed XPFP in time to task completion, but the differences were not statistically significant.

8.2 Accuracy

Table 1 shows the percentage of participants who successfully completed the Wesley and Jack tasks on the XPFP and Salmon interfaces. For the Wesley task, 7 of 12 XPFP users (58%) and 10 of 12 Salmon users (83%) successfully completed the task. For the Jack task, 3 of 12 XPFP users (25%) and 12 of 12 Salmon users (100%) successfully completed the task. These numbers represent a 43% improvement in accuracy for Salmon over XPFP on the Wesley task and a 300% improvement on the Jack task. A one-sided z-test for equality of proportions showed Salmon's superiority over XPFP in successful task completions to be weakly significant for the Wesley task (z=1.347, p=0.089) and strongly significant for the Jack task (z=3.795, $p < 0.0001$).

	Wesley task	Jack task
XPFP	58%	25%
Salmon	83%	100%

Table 1: Accuracy rates for the Wesley and Jack tasks on the XPFP and Salmon interfaces. Salmon showed 43% and 100% improvements in accuracy over XPFP on the Wesley and Jack tasks, respectively.

	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

Table 2: Count of errors by type for Wesley and Jack tasks on XPFP and Salmon interfaces. Salmon users committed significantly fewer goal errors than did XPFP users for both the Wesley and Jack tasks.

8.3 Goal Errors

The error analysis revealed a substantial reduction in the number of goal errors committed by Salmon users, compared to XPFP users. For the Wesley task, XPFP users made 9 total errors, of which 5 were goal errors, while Salmon users made 4 total errors, of which one was a goal error (see Table 2). This represents an 80% reduction in average goal errors per participant for Salmon (M=0.083, sd=0.29) over XPFP (M=0.42, sd=0.51). A one-sided z-test for equality of proportions showed this difference to be statistically significant (z=-1.885, p=0.0297) at the 0.05 level. For the Jack task, XPFP users made 16 total errors, of which 15 were goal errors, while Salmon users made 6 total errors, of which one was a goal error (see Table 2). This represents a 94% reduction in goal errors per participant for Salmon (M=0.083, sd=0.29) over XPFP (M=1.33, sd=0.87). A one-sided z-test showed this difference to be statistically significant (z=-3.312, p=0.0005) at the 0.05 level. It can be concluded that Salmon performs better than XPFP in mitigating goal errors.

Error analysis results show that the Salmon interface led users to more plan and action errors than did XPFP. However, the impact of these errors was not as significant as was the impact of goal errors, because while most goal errors led directly to task failure, most plan and action errors were recovered from. Nevertheless, plan and action errors in Salmon bear further investigation.

9 Discussion

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 anchor-based subgoaling in the design of the Salmon interface. 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 suggested 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 on 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, several XPFP users, looking at Wesley's stated permissions in the XPFP window as shown in Figure 1, thought that Wesley was allowed READ permission because his "Allow Read" checkbox is checked, but was not allowed WRITE permission, because his "Allow Write" checkbox is not checked. They did not realize that he has effective WRITE permission from ProjectF. One such XPFP participant, looking at the XPFP window in the state 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 committed his 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.

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 partial or total task failure, and to the extent that interfaces lead to goal errors, they are undependable. Many goal errors can be prevented by providing a comprehensive and correct external representation of the information relevant to completing the user's root goal. The design principle which calls for such a representation has been named anchor-based subgoaling.

The Windows XP file permissions interface, which does not use anchor-based subgoaling, was shown to have unacceptably low success rates, 58% and 25%, on two representative permission-setting tasks. Salmon, an alternative interface designed in accordance with the anchor-based subgoaling principle, was shown to increase the percentage of successes to 83% and 100%, respectively, 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 and 94% fewer goal errors on the other. These substantial improvements in successful task completion and reductions in goal-error occurrence were due to anchor-based subgoaling. These success rates more closely approach what is needed for dependable user interfaces in mission-critical systems like those required for setting security-related configurations.

11 Future work

Anchor-based subgoaling has been demonstrated to be a successful design technique for reducing goal errors in the domain of setting file permissions, 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 define its limits, and potentially reveal areas in which it cannot by itself reduce goal errors.

The present work attempted only to reduce one type of user interface error, goal errors. Future work will look at means to reduce plan, action, and perception errors.

12 Acknowledgements

The authors are grateful for help from our colleagues Fahd Arshad, David Banks, 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.

References

[1] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):41–46, 1999.

[2] D. Balfanz. Usable access control for the World Wide Web. In *Proceedings of 19th Annual Computer Security Applications Conference*, pages 406–415, Los Alamitos, CA, 2003. IEEE Comp. Society. 08-12 Dec 2003, Las Vegas, NV.

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

[4] S. Card. Information visualization. In J. A. Jacko and A. Sears, editors, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, chapter 28, pages 544–582. Lawrence Erlbaum Associates, Mahwah, NJ, 2003.

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

[6] K. A. Ericsson and H. A. Simon. *Protocol Analysis: Verbal Reports as Data*. MIT Press, Cambridge, MA, Revised edition, 1993.

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

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

[9] A. C. Long, C. Moskowitz, and G. Ganger. A prototype user interface for coarse-grained desktop access control. Technical Report CMU-CS-03-200, Comp. Sci. Dept, Carnegie Mellon University, Pittsburgh, PA, Nov. 2003.

[10] Microsoft Corporation. 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`, 2005.

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

[12] J. Nielsen and R. L. Mack. *Usability Inspection Methods*. John Wiley & Sons, Inc., New York, NY, 1994.

[13] D. A. Norman. *The Design of Everyday Things*. Doubleday, New York, NY, 1988.

[14] S. Pocock, M. Harrison, P. Wright, and P. Johnson. Thea: A technique for human error assessment early in design. In *Proceeding of 8th IFIP TC.13 Conference on Human-Computer Interaction*, pages 247–254, Amsterdam, 2001. IOS Press. 09-13 July 2001, Tokyo, Japan.

[15] J. Reason. *Human Error*. Cambridge University Press, Cambridge, UK, 1990.

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

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

[18] R. Smith. Personal communication, March 2004.

[19] U.S. Senate Sergeant at Arms. 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`, March 2004.

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

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

[22] D. D. Woods and E. M. Roth. Cognitive systems engineering. In M. Helander, editor, *Handbook of Human-Computer Interaction*, chapter 1, pages 3–43. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1st edition, 1988.

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

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

[25] M. E. Zurko and R. T. Simon. User-centered security. In *Proceedings of Workshop on New Security Paradigms*, pages 27–33, New York, NY, 1996. ACM Press. 17-20 September 1996, Lake Arrowhead, CA.