Relating Declarative Semantics and Usability in Access Control

Vivek Krishnan
ECE, University of Waterloo
Waterloo, ON, Canada
vkv@uwaterloo.ca

Mahesh V. Tripunitara
ECE, University of Waterloo
Waterloo, ON, Canada
tripunit@uwaterloo.ca

Kinson Chik[*]
IBM
Toronto, ON, Canada
chik.kinson@gmail.com

Tony Bergstrom
Desire2Learn Ltd.
Waterloo, ON, Canada
tony@mortsgreb.com

ABSTRACT

Usability is widely recognized as a problem in the context of the administration of access control systems. We seek to relate the notion of declarative semantics, a recurring theme in research in access control, with usability. We adopt the concrete context of POSIX ACLs and the traditional interface for it that comprises two utilities `getfacl` and `setfacl` whose natural semantics is operational. We have designed and implemented an alternate interface that we call `askfacl` whose natural semantics is declarative. We discuss our design of `askfacl`. We then discuss a human-subject usability study that we have designed and conducted that compares the two interfaces. Our results measurably demonstrate the goodness of declarative semantics in access control.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection-Access Controls; H.1.2 [**User/Machine Systems**]: Human Factors; H.5.2 [**UserInterfaces**]: Evaluation/Methodology

General Terms

Security, Human Factors

Keywords

Access Control Lists, POSIX, semantics, declarative

1. INTRODUCTION

We relate two recurring, but heretofore disparate, themes in research in access control: declarative semantics and usability. Access control is used to regulate accesses to resources by principals, and is acknowledged to be an important aspect of security. Examples of an access are read and write, of resources are files and devices, and of principals are users and computer processes. Whether a principal has a particular kind of access to a resource is governed by a policy. The policy may dictate, for example, that Alice is allowed to read, but not write a file.

The policy is expressed using some syntax. There are several such syntaxes (or models) that have been proposed in past work in access control. Our work regards a family of syntaxes called Access Control Lists (ACLs). An ACL is a collection of entries. Each entry typically identifies a principal, a resource and a right. It may also additionally specify that the right over the resource is to be awarded to the principal (an "allow entry") or that it should be disallowed (a "deny entry"). There is no single syntax for ACLs that is used universally.

As in other contexts of computing in which a syntax is adopted (e.g., programming languages), it is necessary to specify a semantics or meaning for the syntax. It is only with a semantics that we understand what an access control policy expresses. In our example above of an ACL with an entry of the form $\langle \text{Alice}, \text{file } f, \text{read}, \text{allow} \rangle$, a semantics would need to unambiguously specify what the consequence of the existence of such an entry as part of the ACL is to authorization. That is, a semantics would provide a 'yes' or 'no' answer to a question such as "can Alice read the file f?"

It may appear that the answer to such a question, given the existence of such a entry, is obvious. For ACLs that have been deployed in practice, however, this is not necessarily the case. It is possible, for example, that the rules in the ACL follow a particular precedence relationship, and an entry of higher precedence overrides this "allow" entry for Alice to the file f.

There are different approaches to specify a semantics for a syntax. Two examples that are relevant to this paper are operational and declarative. With operational semantics [21], the semantics is specified procedurally. That is, when one executes a sequence of steps, one is able to derive the precise meaning of something expressed in the syntax. Declarative semantics [13], on the other hand, is "what you see is what it is." That is, there is a more direct specification of the meaning associated with something expressed in the syntax.

Given a syntax, there is often a natural semantics for it. For example, operational semantics is natural for imperative programming languages such as C. Declarative semantics is natural for logic programming.

A question that arises is whether a syntax whose natural semantics is declarative is "better" than a syntax whose nat-

[*]This author's contribution was made when he was a student at the University of Waterloo.

Symposium on Usable Privacy and Security (SOUPS) 2012, July 11-13, 2012, Washington, DC, USA. ACM 978-1-4503-1532-6/12/07.

ural semantics is operational in the context of access control systems. A 'yes' answer to this question has been assumed in a number of pieces of prior work. Part of the motivation for a logic-based approach to access control (see, for example, [12, 17, 26]) is that there is a natural declarative semantics for it. Declarative semantics certainly seems to have intuitive appeal in the context of access control. To our knowledge, this goodness has not been established in practice in any measurable way. We do so in this paper.

In characterizing what is good from the standpoint of a semantics, one must ask what the consequence of a particular choice is. One of the most challenging aspects of an access control system is its administration. Administration encompasses the inspection of a policy to understand what it expresses, and changes to it. In access control, it is typically human users that need to administer the policy. Indeed, as ACLs are typically used for Discretionary Access Control [9], every user is an administrator for the ACLs of resources that she owns.

There have been a number of pieces of prior work that discuss the challenges with the administration of access control systems from this standpoint (see Section 6). What we need to consider is the workload that is imposed on a humans cognitive processes in answering questions related to inspection ("is Alice allowed to read this file?") and change ("how do I ensure that Alice can read this file?")

Ease of administration by a human user, then, is our notion of goodness. We infer that a particular choice for the semantics is better than another if the access control policy is easier to administer with that choice than the other. In the context of access control policies, ease of administration has been called usability [18]. In the context of usability, we care only about the semantics of the interface to an ACL system, rather than the semantics of the underlying system itself. Our thesis is:

> If the natural semantics of the interface for ACLs is declarative, then a user is able to more quickly, accurately and confidently, inspect and edit ACLs than if the semantics is operational.

We instantiate this thesis for a particular ACL system and interfaces to it, and assess its truth value. The system that we adopt is POSIX ACLs (Section 2). By POSIX ACLs, we mean the de facto standard for ACLs to which POSIX-conformant systems such as OpenBSD [3] and Linux [2] adhere. (What we call POSIX ACLs are sometimes called extended or file ACLs.) POSIX ACLs subsume the traditional file permission bits in Unix and Linux systems. They are used to control accesses to objects in a filesystem.

Two utilities customarily comprise the interface to POSIX ACLs: `getfacl` and `setfacl`. The former is used to inspect an ACL, and the latter is used to specify and modify it. We observe that the natural semantics for these utilities is operational. This is because the utilities are low-level; they are a thin layer over the underlying ACL implementation.

There are two possible ways to modify POSIX ACLs to have a natural declarative semantics. One is to modify the underlying design of POSIX ACLs so it has a declarative semantics, and provide a corresponding interface for it. The other is to leave the underlying design as is, and only provide a new interface which provides a declarative way of inspecting and changing ACLs.

We have chosen the latter approach. Our reason is that then, we can readily compare our new interface with the existing `get/setfacl` interface. We call our interface `askfacl`. If we are able to prove our thesis by comparing `get/setfacl` and `askfacl`, then a next step may be to think about the manner in which POSIX ACLs (and other ACL systems) may be redesigned, i.e., whether one should change the interface only, or entirely redesign the underlying system.

A by-product of our choice of changing the interface only is that we are then able to explore some issues beyond our thesis with `get/setfacl`. We observe that the manipulation of ACLs with `get/setfacl` can result in side-effects and redundancy in the ACL. A side-effect is an authorization or unauthorization that is not intended. A redundancy is an extra entry in the ACL that serves no purpose. Side-effects are undesirable as they change the authorization state in an unintended way. Redundancy is undesirable because it may make the ACL more difficult to administer in future.

Our interface, `askfacl`, has been designed so that the user is not burdened with managing side-effects and redundancies. Indeed, the semantics of `askfacl` require this.

We have conducted a human-subject usability study to compare the usability of `get/setfacl` and `askfacl` (see Sections 4 and 5). The main objective of the study is to establish our thesis from above. We also present results related to the confidence users have in accomplishing tasks with each interface, and the issue of side-effects and redundancy. In the context of the latter, our results show that if we tighten the notion of accurately accomplishing a goal to not only include ensuring that a particular authorization or unauthorization is effected by a change to the ACL, but also that this should happen with no side-effects or redundancy, then the accuracy rate for `get/setfacl` drops significantly.

Layout The remainder of this paper is organized as follows. In the next section, we overview POSIX ACLs and the `get/setfacl` interface. In Section 3 we discuss the interface provided by `askfacl` and a brief discussion of its design. In Section 4, we discuss the design of our human-subject usability study to compare `get/setfacl` and `askfacl`. In Section 5, we present our results from the study. We discuss related work in Section 6 and conclude with Section 7.

2. POSIX ACLS

In this section, we discuss POSIX ACLs, and the commandline utilities that are used to administer them. As we mention in the previous section, POSIX ACLs are the concrete context in which conduct our evaluations.

In the context of POSIX ACLs, a principal may be either a user or a group. In the remainder of this paper, we use the term subject when we want to be agnostic to the particular kind of principal. When we refer to a file, we mean any resource that is accessible as a file via the filesystem. Derivatives of Unix expose devices such as printers as entries in the filesystem. Similarly, files may be grouped into directories, which also are accessible via the filesystem.

POSIX is a family of standards for the Unix operating system and its derivatives. Operating systems that are in use today such as Linux are POSIX-conformant in that they implement several of the POSIX standards.

Part of the effort on POSIX was an attempt to standardize ACLs. This effort did not evolve into a standard. However, the work was considered to be of sufficiently high quality that a draft was made publicly available [27]. This draft is a de facto standard for ACLs in systems such as OpenBSD and Linux that are POSIX-conformant. Such systems implement all the functionality that is prescribed in the POSIX draft. A discussion of the conformance to the POSIX draft is included in the manual pages for ACLs in OpenBSD and Linux. When we refer to "POSIX ACLs," we mean what is specified by this draft. We refer to the draft document as "the POSIX standard (for ACLs)."

The POSIX standard [27] discusses the design criteria that were adopted by the committee. Ease of inspection and modification is not one of them.

In the context of POSIX ACLs, an access request comprises a user, a mode of access and a file. A user is the smallest unit of a principal and is atomic. Every user is associated with a unique integer identifier called the uid, and a mnemonic called the username. There are three possible modes of access to a file: read, write and execute. The semantics of each depends on the nature of the file. Each mode of access is atomic; the right to one does not imply the right to another.

Every file is associated with two owners: an owning user, and an owning group. A group is a set of users, and has an identifier called a gid, and a name called the groupname associated with it. Every user is associated with a group whose only member is that user and whose groupname is the same as his username.

An ACL comprises one or more entries. Each entry identifies a principal and the subset of the three modes of access (read, write, execute) to which the principal is authorized. There are five types of principals that may be identified in an ACL entry: the owning user, a user identified by his uid, the owning group, a group identified by its gid, and other.

The mnemonic "other" refers to all users that do not fall in any of the other four categories. An ACL may be associated with a mask, which is used to identify the maximum permissions a user identified by his uid or any group may have to the file.

By default, every file has an ACL with which it is associated. This ACL is called its base ACL. The entries in the base ACL correspond to the permission bits of the file. Permission bits predate POSIX ACLs, and are the traditional approach to file access control in Unix and its variants.

Every file has nine bits associated with it as meta-data. (We do not discuss other bits that are maintained along with the nine permission bits, such as the "sticky" bit.) For each of the three access modes, read, write and execute, a bit identifies whether the owning user, a member of the owning group and anyone else, has that access to the file. POSIX ACLs subsume the permission bits. The entries that correspond to the owning user, owning group and "other" principals in a file's ACL reflect the permission bits exactly, and vice versa.

Example 1. *An example of the output produced by the utility* `getfacl` *that is used to examine an ACL follows. It is for a file that contains ASCII text, called* `data.txt` *in a user* `alice`*'s home directory,* `/home/alice`*, on a host called* `Host`.

```
carol@Host:~$ getfacl data.txt
# file: data.txt
# owner: alice
# group: alice
user::rw-
user:bob:r--
group::---
group:profs:rwx          #effective:r--
mask::r--
other::---

carol@Host:~$
```

The command `getfacl` *and its output above are as executed by a user* `carol`*. The output identifies* `alice` *as the owning user and* `alice` *as the owning group of* `data.txt`*. It expresses that the owning user has read and write privileges, but no execute privilege. It expresses that a user* `bob` *has read, but no write or execute privilege. The owning group,* `alice`*, has no privileges, and a group called* `profs` *is configured to have all privileges. However,* `profs` *is limited to only the read privilege by the* `mask`*, as indicated by the "*`#effective:r---`*" annotation against its entry. All others are configured to have no privileges.*

Access check Given an access request to a file, the POSIX standard [27] specifies an algorithm that the reference monitor is to use for checking the request against the file's ACL to determine whether the request should be allowed or denied. As we mention above, an access request is a triple, $\langle \text{user}, \text{mode}, \text{file} \rangle$. We show the algorithm in Figure 1 .

The algorithm maps the user in the request to exactly one of the following four principals in order: (a) the owning user, (b) a user for whom there is an ACL entry, (c) either the owning group, or a group for which there is an ACL entry, or, (d) "other." The algorithm then checks whether the user in the request has the privilege that corresponds to the mode in the request.

Example 2. *Consider the file* `data.txt` *from Example 1. If Alice requests read or write access, she is allowed. She is disallowed if she requests execute access. The reason is that her request is decided based on the entry for the owing user, "*`user::rw-`*", only. Bob is allowed read access only. His request is decided based on the "*`user:bob:r--`*" entry that is specific to him only. It is immaterial whether he is a member of* `profs`.

Semantics.

The natural semantics of POSIX ACLs, as accessed via `get/setfacl`, is operational. To answer a question such as "can Carl read this file?," an administrator needs to effectively run the algorithm from Figure 1 cognitively after getting the output from `getfacl`.

Changing an ACL using `setfacl` involves a similar cognitive process. An administrator needs to consider a prospective change, and then run the algorithm from Figure 1 cognitively and determine whether the new ACL meets his objective. In this context, he must also consider whether he introduces side-effects (authorizations or denial of authorizations that are not intended) and redundancy (extraneous entries).

There are a number of subtleties of which a `get/setfacl` user must be aware when inspecting or changing an ACL. For example, if a user has a right to a file via groups only, then the ACL has the monotonicity property that if any of the groups of which the user is a member has the right, then the user has the right. If, on the other hand, there is

```
check($v, p, f$)
    01: if $v$ is the owning user then
    02:   if $p \in$ rights of "user::" entry then
    03:     allow and return
    04:   else deny and return
    05: $M \leftarrow$ rights of the "mask::" entry
    06: if $\exists$ a "user:$v$:" entry then
    07:   let $R \leftarrow$ the entry's set of rights
    08:   if $p \in R \cap M$ then
    09:     allow and return
    10:   else deny and return
    11: if $v$ is a member of the owning group $g$ then
    12:   let $R \leftarrow$ the rights of the "group::" entry
    13:   if $p \in R \cap M$ then
    14:     allow and return
    15: for each "group:$g$:" entry
    16:   if $v$ is a member of $g$ then
    17:     let $R \leftarrow$ the rights of the "group:$g$:" entry
    18:     if $p \in R \cap M$ then
    19:       allow and return
    20: if $v$ is not a member of any group in ACL of $f$ then
    21:   let $R \leftarrow$ rights of the "other::" entry
    22:   if $p \in R$ then allow and return
    23: deny and return
```

Figure 1: The algorithm that is used for checking a user v's access p to the file f in POSIX ACLs. This is also the algorithm that a user must run cognitively when using `get/setfacl`.

an entry of the type "`user:`" that matches the user, then such a monotonicity property does not hold. Exactly one of those entries determines whether the user has the right, notwithstanding whether other entries that have the right match the user.

3. `askfacl`

In this section, we discuss our design of `askfacl`. As we mention in Section 1, the main design consideration is that its natural semantics is declarative. We have also adopted POSIX ACLs as is, and made no changes to that design. Consequently, `askfacl` can be seen as an interface to POSIX ACLs with declarative semantics. We first discuss how it is used. Then, we discuss its semantics and design.

Usage. The utility `askfacl` has two main options: get, and set. The get option is used to inspect an ACL. The set option is used to modify it.

For get, there is an optional sub-option: exact. If the exact sub-option is not specified, then the set of rights that are specified as part of the command are interpreted as a lower-bound. For example, if the administrator issues `askfacl --get u:bob:r-- data.txt`, we say "yes," if Bob has read access to `data.txt`; we don't care whether he has any privileges other than read or not. If the user issues `askfacl --get --exact u:bob:r-- data.txt` instead, then we say "yes" only if Bob has read privileges only. If he has the write or execute privilege, we say "no." (We show the exact output in Example 3 below)

As is customary for command-line utilities in Unix and Linux, every option has a long form, and a short form. The short form has only one `-` and uses the first letter of the long form. For example, "`--get`" specifies the get option in long form. The same option is short form is specified as "`-g`."

Example 3. *For the ACL in Example 1, the output of* `askfacl -g -e g:profs:r-- data.txt` *is the following. (The option "*`-e`*" is short for "*`--exact`*.") We assume that Alice is a member of* `profs`*.*

```
No, profs do not have read access only.
file: data.txt, group: profs
read: yes
write: specific to member
execute: no
```

The output indicates that every member of `profs` *has read access, some members have write access, and no member has execute access to* `data.txt`*.*

For the set option, there are three sub-options: exact, add and subtract. In the exact sub-option, the user needs to specify privileges for each of read, write and execute. For example, if the user issues `askfacl -s -e u:bob:rw- data.txt`, it means that Bob should have read and write access, and must not have execute access to `data.txt`.

The add sub-option is used to ensure that the subject has a right, and the subtract sub-option is used to ensure that the subject does not have a right. For example, if the user issues `askfacl -s -a u:bob:w data.txt`, then we ensure that Bob has write access. No change may be made to the ACL; this occurs when Bob already has write access. We present a fuller example of the set option below, once we discuss side-effects and redundancy in the context of `askfacl`.

Semantics.

`askfacl` is designed to have a declarative semantics. When an administrator wants to inspect an ACL, he queries using `askfacl -g`, and gets an answer for the subject (user or group) in the query. When he wants to edit the ACL using `askfacl -s`, he specifies the subject and the new set of rights. If `askfacl` does not return an error, then that subject is guaranteed to have that access.

There are some design choices we have made that are somewhat subjective. For example, if the subject that is the argument to `askfacl -g` or `-s` is a group, then we interpret this as "for every member of the group." A consequence is shown in Example 3 above. When we ask whether the group `profs` has write access to a file, and only some members of `profs` have the access, then the output is "no," with another line that says that write is "specific to member." In particular, this inference may be different from the list of rights against the `groups:profs:` entry in the ACL.

We mention another design choice in the algorithm in Figure 2 in our discussions on the design of `askfacl` below.

Design.

The algorithm that underlies `askfacl` is similar to the one that we expect a human user to have to run cognitively to inspect or change an ACL. In Figure 2 we show the portion

```
add($v, p, f$)
    01: if askfacl -g u:$v$:p f is true, then return
    02: if $u$ is the owning user of $f$ then
    03:     add $p$ to the "user::" entry and return
    04: addToMask($p, f$)
    05: if $\exists$ a "user:$v$:" entry then
    06:     add $p$ to the "user:$v$:" entry and return
    07: $G \leftarrow$ all groups that appear in $f$'s ACL
    08: $G \leftarrow G \cup$ owning group of $f$
    09: for each $g \in G$
    10:     if $v$ is the only member of $g$ then
    11:         add $p$ to the "group:g:" entry and return
    12: $R \leftarrow$ all effective rights to $f$ of $v$ via groups
    13: if $v$ is not a member of any group in ACL of $f$ then
    14:     $R \leftarrow$ all rights of "other::"
    15:     for each $r \in R$ addToMask($r, f$)
    16: create a user:$v$ entry with rights $R \cup \{p\}$

addToMask($p, f$)
    17: let $R \leftarrow$ rights of "mask::" entry
    18: if $p \in R$ return
    19: for each "user:$\mu$:", "group:$g$:" and the "group::" entry
    20:     let $S \leftarrow$ rights of the entry
    21:     assign rights $R \cap S$ to the entry
    22: add $p$ to the "mask::" entry
```

Figure 2: The algorithm within askfacl for adding the right p in the ACL for the file f for a user v.

of the algorithm for `askfacl -s -a u:`v`:p f` that is used to add the right p for some user v to the ACL for the file f.

An express design consideration of `askfacl` is for it to result in no side-effects and no redundancy when we use the set option. A side-effect is an authorization that changes without intent. For example, in the ACL of Example 1, if we want to give `profs` write access, we need to change the mask. However, this may affect the authorizations of some users that are not members of `profs`. For example, if Bob is not a member of `profs`, but the ACL has an entry of the form `user:bob:rw-`, then we should ensure that we remove `w` from the entry for Bob when we change the mask.

In the algorithm in Figure 2, we invoke the auxiliary routine addToMask($\cdot$) in Lines 04 and 15 precisely for this reason. The routine first explicitly assigns the effective rights to every entry of the form "`user:u:`", "`group::`" and "`group:g:`". Then, it adds p to the set of rights of the "`mask::`" entry. This ensures that giving the user v the right p does not result in p being given also to other users.

Redundancy deals with whether we add extraneous entries to the ACL that are unnecessary. This is particularly relevant to entries of the form `user:`, as these can be viewed as exceptions. It is known that exceptions in access control policies make them difficult to administer [7]. One may argue that if `askfacl` is always use to manage an ACL, then the redundancy should not matter. However, it may be desirable to maintain an ACL without such redundancy in case an administrator chooses to view or edit the underlying ACL directly using `get/setfacl`.

Our design of `askfacl` ensures that if the input ACL has no redundancy, then the output ACL, after changes by `askfacl`, has no redundancy. Line 01 of the algorithm in Figure 2 is a simple example of a check to preclude the addition of a redundant entry. The algorithm also contains a design choice in this context. In Lines 09–11, if we determine that there is already an entry for a group g of which v is the only member, then we add the right p to that group. If this is considered undesirable (e.g., because other users may be made members of g at a later time), then those lines can be removed from the algorithm without affecting its correctness. (Of course, then, the notion of redundancy is different from the one we adopt.)

We discuss some limitations of `askfacl` in Section 4.6.

Example 4. *Assume that Bob is a member of* `profs` *for the ACL in Example 1. Then, if we issue* `askfacl -s -a u:bob:w data.txt`*, we get the following ACL (as output by* `getfacl`*).*

```
carol@Host:~$ getfacl data.txt
# file: data.txt
# owner: alice
# group: alice
user::rw-
user:bob:rw-
group::---
group:profs:r--
mask::rw-
other::---

carol@Host:~$
```

The group `profs` *has had its explicit list of rights set to* `r-` *from* `rwx`*, as that was its effective set of rights before the change. The algorithm makes that change to the entry for* `group:profs` *because it needs to add* `w` *to the mask so that the write privilege from the* `user:bob` *entry can have effect.*

Assume that the members of `profs` *are Alice, Bob and Carol. Issuance of* `askfacl -s -a u:carol:r data.txt` *has no effect on the ACL as she already has that right from being a member of* `profs`*. Issuance of* `askfacl -s -m u:carol:r data.txt` *(the "-m" stands for "minus" or remove) results in the entry* `group:profs` *being removed from the ACL. This is because, now, all members of* `profs` *are either the owning user (Alice), in a "*`user:`*" category (Bob) or in the "*`other::`*" category (Carol).*

Example 5. *For the ACL in Example 1, assume that the members of the group* `committee-members` *are Alice and Bob, and the members of* `profs` *are Alice, Bob and Carol. Then, Issuance of* `askfacl -s -a g:committee-members:x` *results in the following.*

```
carol@Host:~$ getfacl data.txt
# file: data.txt
# owner: alice
# group: alice
user::rwx
group::---
group:profs:r--
group:committee-members:--x
```

```
mask::r-x
other::---

carol@Host:~$
```

Alice has `x` *added to her list of rights. The entry* `user:bob` *is removed as Bob acquires* `r` *from* `profs` *and* `x` *from* `committee-members`*. And Carol acquires* `r` *from* `profs`*. As* `x` *needed to be added to the mask, the list of rights for* `profs` *is changed to* `r--`*.*

4. HUMAN-SUBJECT STUDY

We conducted a human-subject study with 42 participants. We obtained approval from the Office of Research Ethics prior to the study.

We used a between-participants design, so 21 participants used `askfacl` and the other 21 used `get/setfacl`. Each participant was provided training as we discuss below in Section 4.3, and completed 6 tasks related to inspecting and modifying ACLs. We measured accuracy, time for accurately completing each task and a self-rated confidence of a participant in having accurately completed the task. The first two tests from each set (which were our first four participants) were used as pilots. Consequently, we ended up with 19 participants for each of `get/setfacl` and `askfacl`.

4.1 Participants

Our participants were undergraduate and graduate students from science and engineering disciplines at the University of Waterloo. We recruited them via email to class-lists, bulletin boards on campus and word-of-mouth. To be qualified to participate, a participant had to be a regular user of Linux, be familiar with a UNIX command-line shell such as bash or csh, and have had no familiarity with `get/setfacl`. We wanted a participant to be technical savvy enough to be able to quickly learn to use `get/setfacl` or `askfacl`, but not be biased as a consequence of prior familiarity with `get/setfacl`. We conjecture that the technical profile of each of our participants matches that of a junior-level systems administrator. The participants were each given $20 as an honorarium.

4.2 Experimental Setup

The study was conducted in an office room at the university dedicated to the study over its duration. Only the participant and the person that conducted the study were present in the room during each session. The same person (the first author) conducted all the sessions.

The participants worked on a desktop that ran the Ubuntu 10.04 LTS operating system. Each participant was given a hard-copy of the training materials and the tasks. We recorded the user's entire shell session using the `script` utility [5]. We also recorded the final policies that users created, and the sheets of paper on which the user made notes and gave a confidence rating as that indicated how confident she was in having completed a task accurately. We also collected audio recordings of the participants as they thought aloud while performing tasks.

4.3 Training

Prior to commencement of the tasks, the participants were trained on the respective interface that they used. Both sets of users were initially shown the following.

- What the broad use of the interface is — visualizing and manipulating ACLs.
- A discussion on the syntax of the interface.
- Instructions on how to think aloud.

Users of `get/setfacl` were briefed also about the structure of the access control list.

After the above, the participants were given a set of training tasks to complete. As the nature of the interfaces and their semantics are different, we gave them different training tasks but with same objective.

> After the training, a participant should know how to obtain information about, and create and edit entries within an ACL.

For `get/setfacl`, the participants were trained using the following types of tasks, in order.

- Intuiting the state of and modifying the ACL is straightforward — the user needs to deal with only entries related to users, groups or "other."
- Intuiting the state of and modifying the ACL involves consideration of the mask entry.
- Intuiting the state of and modifying the ACL involves information about groups to which a user belongs. This involves the use of auxiliary tools to determine group memberships.

For the participants that were assigned `askfacl`, the training tasks involved inspecting and modifying ACL entries for users and groups. The modifications involved setting and adding rights. We provided no training to either group of participants on removing rights from ACL entries.

4.4 Tasks

The participants were given the following 6 tasks to complete. The tasks are ordered in what we believe to be increasing difficulty. For Tasks 1–5, each task has an underlying rationale that is independent of the interface. The rationale pertains to objectives that we assume are customary for inspecting and modifying ACLs. Consequently, each task except Task 6 has no intentional bias towards or against either interface. We have not accounted for subconscious bias.

Task 6 was designed as a "negative test" of our implementation of `askfacl`. It is biased against `askfacl` by intent. Our design and development of `askfacl` was done before and independently of the design of our human-subject study. While we were designing our human-subject study, we realized a weakness in our design of `askfacl`. It is easy to correct this weakness — we can simply provide a mnemonic "all" in `askfacl` which is a special group that refers to all users in the system. However, we chose to leave the weakness as is, and incorporate a task to test the manner in which it impacts usability. We discuss the task in this section, and the results for it in Section 5.4.

In the description of tasks that follows, we provide for each task an objective, a rationale, the ACL state prior to the task, and possible solutions for `get/setfacl` and `askfacl`. We recognize that the solutions we provide below are not the only possible ones, and they are not the only ones that we accepted when considering whether a participant completed a task successfully. We accepted any solution that met the objective. A deeper consideration, for example using Hierarchical Task Analysis [16], is beyond the scope of this paper.

4.4.1 *Task 1: user-get/set*

Objective: Give user `alice` write access to the file `task1.txt`.
Rationale: A basic task to give a user a right. As we point out below, she already has the right, so an option is to make no changes.
ACL state prior to task:

```
# file: task1.txt
# owner: harry
# group: harry
user::rw-
user:harry:r--
group::r--
mask::r--
other::-w-
```

Solution for `get/setfacl` : A participant has one of two choices. He may inspect the ACL and observe that `alice` already has write access via the "other" entry.

```
getfacl task1.txt
```

Or, he may add an entry for `alice`.

```
setfacl -m u:alice:-w- task1.txt
```

Solution for `askfacl` : Two choices. Inspect the ACL and observe that `alice` already has write access.

```
askfacl -g u:alice:-w- task1.txt
```

Or, ask for the right to be added, which has no effect.

```
askfacl -s -a u:alice:w task1.txt
```

4.4.2 *Task 2: user-set*

Objective: Give user `harry` write access to the file `task2.txt`.
Rationale: Harry does not have the access. Therefore it needs to be given.
ACL state prior to task:

```
# file: task2.txt
# owner: harry
# group: harry
user::r--
user:harry:r--
group::r--
mask::rw-
other::-w-
```

Solution for `get/setfacl` : A participant should inspect the ACL and observe that `harry` is the owner and does not have write access. He should then provide the write access to the entry that corresponds to the owner.

```
getfacl task2.txt
setfacl -n -m u::rw- task2.txt
```

Solution for `askfacl` : Give `harry` write access. Optionally check beforehand that he does not have it.

```
askfacl -g u:harry:-w- task2.txt
askfacl -s -a u:harry:w task2.txt
```

4.4.3 *Task 3: user-get*

Objective: Check if user `david` has write access to the file `task3.txt`.
Rationale: A task that involves inspection only.
ACL state prior to task:

```
# file: task3.txt
# owner: harry
# group: harry
user::rw-
user:alice:r--
user:bob:rw-
user:carol:rwx              #effective:rw-
user:edward:-w-
user:fred:---
group::r--
group:engineers:--x         #effective:---
group:employees:r--
group:managers:r-x          #effective:r--
group:supportstaff:rw-
group:students:-w-
mask::rw-
other::rw-
```

Solution for `get/setfacl` : In addition to inspecting the ACL, a participant needs to check of what groups `david` is a member.

```
getfacl task3
groups david
```

Solution for `askfacl` :

```
askfacl -g u:david:r-- task3.txt
```

4.4.4 *Task 4: user-remove*

Objective: Remove the read permission that user `david` has to the file `task4.txt`.
Rationale: A task that involves unauthorization.
ACL state prior to task:

```
# file: task4.txt
# owner: harry
# group: harry
user::rw-
user:alice:r--
user:bob:rw-
user:carol:rwx             #effective:rw-
user:edward:-w-
user:fred:---
group::r--
group:supportstaff:--      #effective:---
group:lecturers:r--
group:employees:r--
group:managers:rw-
mask::rw-
other::r--
```

Solution for `get/setfacl` : Create a "`user:david`" entry, and assign it no rights.

```
setfacl -n -m u:david:--- task4.txt
```

If the "`-n`" option indicates that the mask should not be changed. If it is omitted, then the mask may be affected.

And therefore, in addition, the following should be specified to restore the mask.

```
setfacl -m m::rw- task4.txt
```

Solution for `askfacl` :

```
askfacl -s -m u:david:r task4.txt
```

4.4.5 Task 5: group-set

Objective: Ensure that group `engineers` has write access to the file `task5.txt`.
Rationale: Authorization for a group.
ACL state prior to task:

```
# file: task5.txt
# owner: harry
# group: harry
user::rw-
user:alice:r--
user:bob:rw-               #effective:r--
user:carol:rwx             #effective:r--
user:edward:-w-            #effective:---
user:fred:---
group::r--
group:employees:rw-        #effective:r--
group:engineers:r--
group:managers:rw-         #effective:r--
group:supportstaff:rw-     #effective:r--
mask::r--
other::r--
```

Solution for `get/setfacl` :

```
setfacl -n -m g:engineers:rw- task5.txt
```

In addition to the above if one wants to eliminate side-effects, one has to also issue the following.

```
setfacl -n -m u:bob:r-- task5.txt
setfacl -n -m u:carol:r-- task5.txt
setfacl -n -m u:edward:--- task5.txt
setfacl -n -m g:employees:r-- task5.txt
setfacl -n -m g:managers:r-- task5.txt
setfacl -n -m g:supportstaff:r-- task5.txt
setfacl -n -m m::rw- task5.txt
```

Solution for `askfacl` :

```
askfacl -s -a g:engineers:w task5.txt
```

4.4.6 Task 6: all-set

Objective: Ensure that all the users and groups in the system have only read access to the file `task6.txt`.
Rationale: A task that is biased against `askfacl` by intent. It targets a deficiency in the current implementation of `askfacl`. We do not support a mnemonic "all," that allows for a task such as this to be achieved easily using `askfacl` for the given ACL. For `get/setfacl`, however, it involves the manipulation of two entries only.
ACL state prior to task:

```
# file: task6.txt
# owner: harry
# group: harry
user::rw-
group::r--
other::-w-
```

Solution for `get/setfacl` :

```
getfacl task6.txt
setfacl -n -m o::r-- task6.txt
setfacl -n -m u::r-- task6.txt
```

Solution for `askfacl` : Get a list of all groups and users in the system and issue `askfacl` commands to change the permissions of each.

4.5 Procedure

The participants were alternatingly assigned `get/setfacl` and `askfacl`. Participants thought aloud throughout the session. The person that conducted the session brought up the terminal that the participant was to use. Task statements were presented on paper. The order of tasks was the same for all the participants, as the tasks were designed in increasing levels of difficulty, and the participants performed them in order.

Participants were not given any time limit to complete a task. After each task was completed, participants were asked to record the answers wherever appropriate and also to rate their confidence on a 1-10 scale (10: most confident) that the task had been completed accurately. After completion of all the tasks the participants were asked to talk about their experience while working with the interface.

4.6 Limitations

There are several limitations and issues with our study and our design of `askfacl` that we must point out. Our study is not intended to be a holistic study about policy management and expression using ACLs. As our tasks indicate, ours is a targeted study to prove our thesis from Section 1. This is also the reason that our tasks may not seem to be as comprehensive as those of Reeder et al. [22].

Another important issue is that we assume that if a user Alice has a permission p to a file, then she has the ability to exercise p on that file. This is not always true in POSIX systems. For a user to be able to read a file, she must not only have the read permission to that file, but also execute permission to the directory in which the file resides. We made our choice so we could focus on the problem of interpreting and editing an ACL than a more holistic study of the management of ACLs and the semantics of permissions.

We have chosen a fixed ordering of tasks. Our ordering is based on what we believe to be increasing difficulty for participants. This could lead to effects due to ordering for which we have not accounted. We do not consider seasoned systems administrators, but only students that we believe to be equivalent in technical expertise to junior-level systems administrators. We argue, however, that our user-population is appropriate given the thesis that we seek to prove.

The design of our tasks may also have some nuances that may have affected our results. For example, in Task 2 we use the verb "give," and in Task 5 we use "ensure." The latter may have provided a user a hint that the status of the ACL should be checked before any changes are made. Also, we chose to implement both the get and set functionalities using switches in the same application, `askfacl`, rather than as two different commands similar to `getfacl` and `setfacl`. This may have introduced a bias in favour of `askfacl`. This was unintended, and in retrospect, we should have implemented `askfacl` as two commands.

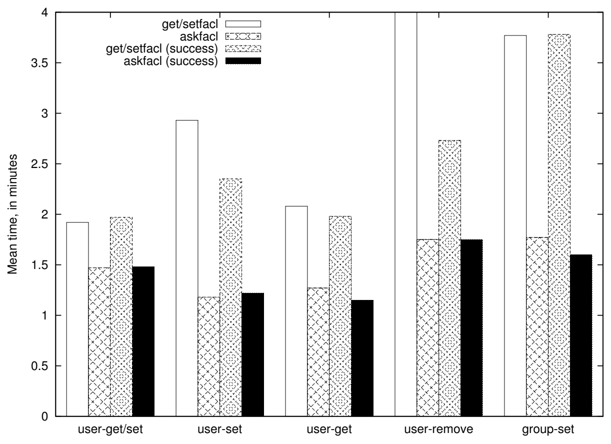

Figure 3: Mean completion time.

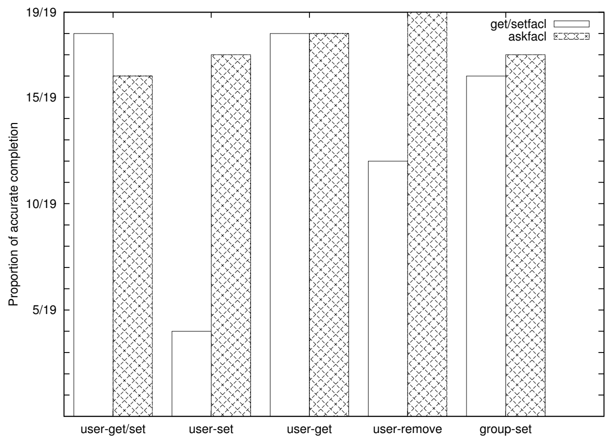

Figure 4: Accuracy rate.

We point out also that `askfacl`'s semantics when it comes to groups is somewhat limiting. For example, if a user Alice already has a permission, and we use `askfacl` to grant that permission to a group of which she is a member, our approach to controlling redundancy may remove her individual access. Thus, there is no easy way to undo this change.

Yet another issue regards how the mask changes in `setfacl` as opposed to `askfacl`. In the former, unless one specifically asks for the mask to not change with the `-n` option, it changes with requested changes to the entries that are affected by it. We have not incorporated a task to test the impact of this difference between `setfacl` and `askfacl`.

5. RESULTS

We present four sets of results. In Section 5.1, we present results for accuracy and time for task completion. We present both per-task results for Tasks 1–5 and experiment-wide results. We consider also the correlation between accuracy and confidence. In Section 5.2, we present results related to the confidence of users in using the two interfaces.

In Section 5.3, we consider a stricter notion of accuracy that requires that a participant that uses `get/setfacl` achieve an objective with neither side-effects nor redundancy. We study the drop in accuracy as a consequence. Our intent is to assess the usefulness of `askfacl` in handling side-effects and redundancy by itself, and not impose that burden on the user as `get/setfacl` does. Finally, in Section 5.4, we consider the results for Task 6, which is biased against `askfacl` by intent, in that it targets a weakness in its implementation. We conclude with a discussion of our results in Section 5.5.

Presentation and Statistical Significance.

We present graphs for means and proportions. For results on statistical significance, we use the APA format [1], which is summarized in [4]. For statistical significance of interval variables, we have used the Mann Whitney U test [20]. This is a non-parametric test. The reason we chose it over, for example, the t-test, is that we are unsure about whether some of our data is normal. Therefore, as Schechter [25] suggests, we adopt a non-parametric test. For statistical significance of non-interval variables, particularly accuracy rates within a task, we use Fisher's exact test [11]. We adopt a threshold of 0.05 for p values to deem that a difference is statistically significant. We report p values that are smaller than 0.001 simply as "< 0.001." We have accounted for multiple comparisons by adjusting the p values using the Benjamini Hochberg method [8].

5.1 Accuracy and Time

In this section we present results on the two main goodness criteria that we consider: accuracy and time for accurate completion of tasks. We present two types of results: experiment-wide and task-specific. We also present our results on the correlation between confidence and accuracy.

For both time and accuracy, our null hypothesis is that `askfacl` performs no better than `get/setfacl`. That is, participants take at least as much time, and are at most as accurate. Our alternate hypothesis is that `askfacl` is better than `get/setfacl`. For time, we consider only those that accurately complete a task.

5.1.1 Time

Experiment-Wide Results As we mention in Section 4, we had 19 users for each interface and 5 tasks for each of them to complete. Our experiment-wide results were calculated on the time taken for those tasks that a user completed accurately across all the $19 \times 5 = 95$ values. Users of `askfacl` took a mean time of 1:26 (minutes:seconds) to accurately complete a task while `get/setfacl` users took 2:33. To check for statistical significance of the difference, we first computed the mean time of accurate tasks for each participant, and then performed a Mann Whitney U test on those composite scores: $N = 38$, $U = 318$, $p_{exact} < 0.001$. Due to the low p value, we reject the null hypothesis and conclude that `askfacl` performs significantly better than `get/setfacl` from the standpoint of time to accurately complete a task.

Our choice to consider only those tasks that were completed accurately by each participant is the same as prior work [22]. Our reason is an assumption we have made: a participant is accurate only if she was focused on achieving the goal associated with a task. Under the assumption, we are guaranteed that we consider time values for only those tasks for which the participant was focused on achieving the goal. Thereby, we exclude cases such as a participant taking an exceedingly long time for a task as a consequence of engaging in activities unrelated to achieving the goal.

Task by Task Results We show the mean per-task accurate completion times in Figure 3. We then computed the difference for users that completed each task accurately and checked the differences for statistical significance using the Mann Whitney U Test. As we mention above, we have adjusted the p values to account for multiple comparisons. Based on the following, we strongly reject the null hypothesis for all the tasks and establish that `askfacl` is significantly faster than `get/setfacl` for those tasks.

- For Task 1, `get/setfacl` users spent significantly more time than the `askfacl` users: $N = 34$, $U = 197$, $p_{exact} = 0.034$.
- For Task 2, `get/setfacl` users spent significantly more time than the `askfacl` users: $N = 21$, $U = 59$, $p_{exact} = 0.024$.
- For Task 3, `get/setfacl` users spent significantly more time than the `askfacl` users: $N = 36$, $U = 257.5$, $p_{exact} = 0.004$.
- For Task 4, `get/setfacl` users spent significantly more time than the `askfacl` users: $N = 31$, $U = 184.5$, $p_{exact} = 0.005$.
- For Task 5, `get/setfacl` users spent significantly more time than the `askfacl` users: $N = 33$, $U = 219.5$, $p_{exact} = 0.004$.

5.1.2 Accuracy

Experiment-Wide Results We determined the proportion of the $19 \times 5 = 95$ tasks that were completed accurately for each interface. The proportion was 68/95 for `get/setfacl`, and 87/95 for `askfacl`. To check for statistical significance of the difference, we first determined the proportion of accurate completion across all 5 tasks for each user, and then performed a Mann Whitney U test on the composite (ordinal) scores: $N = 38, W = 70.5$, $p_{exact} < 0.001$. Hence, we strongly reject the null hypothesis and conclude that `askfacl` is better than `get/setfacl` from the standpoint of accuracy.

Task by Task Results We show the per-task accuracy rates in Figure 4. To check the statistical significance of the difference in accuracy rates, we performed a one sided Fisher's exact test for each of the tasks. Based on the following, we strongly reject the null hypothesis for Tasks 2 and 4.

- For Task 1, The result of the Fisher's exact test was inconclusive: `get/setfacl` = 18/19, `askfacl` = 16/19, $p = 0.757$.
- For Task 2, users of `askfacl` were significantly more accurate than users of `get/setfacl`: `get/setfacl` = 4/19, `askfacl` = 17/19, $p = 0.012$.
- For Task 3, The result of the Fisher's exact test was inconclusive: `get/setfacl` = 18/19, `askfacl` = 18/19, $p = 0.757$.
- For Task 4, users of `askfacl` were significantly more accurate than users of `get/setfacl`: `get/setfacl` = 12/19, `askfacl` = 19/19, $p = 0.036$.
- For Task 5, The result of the Fisher's exact test was inconclusive: `get/setfacl` = 16/19, `askfacl` = 19/19, $p = 0.757$.

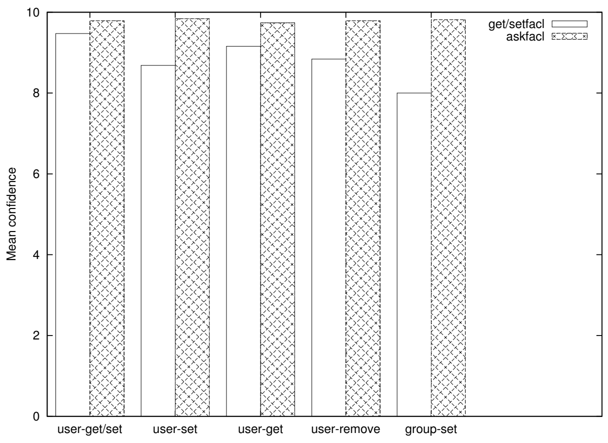

Figure 5: Mean confidence.

5.1.3 Correlation between Accuracy and Confidence

We tested for a possible correlation between accuracy and confidence. As we mention in Section 4.5, all participants provided us a self-rated confidence between 1 and 10 for each task. We computed the point biserial correlation coefficient between accuracy and confidence. We were unable to find a significant correlation for either `get/setfacl` (r_{pb}=0.128, $p_{2-tailed} = 0.217$) or `askfacl` (r_{pb}=0.156, $p_{2-tailed} = 0.132$). We have not addressed the issue of independence of the data points in computing the correlation as the result is not statistically significant regardless.

5.2 Confidence

Apart from accuracy and time for accurate completion, one may consider the confidence of users of an interface as indicative of its goodness. In this section, we discuss our results on the difference in confidence of users of the two interfaces. Our null hypothesis is that the users of `askfacl` are no more confident than the users of `get/setfacl`, and our alternate hypothesis that they are.

Experiment-Wide Results To check the statistical significance of the difference between the mean confidence of those that used `get/setfacl` as opposed to `askfacl`, we first computed the mean confidence across all 5 tasks for each user, and then performed a Mann Whitney U test on those composite scores: $U = 66, N = 38, p_{exact} < 0.001$. Hence we strongly reject the null hypothesis.

Task by Task Results In Figure 5, we show the mean per-task confidence. We performed a Mann Whitney U test to check the statistical significance of the differences. We strongly reject the null hypothesis for Tasks 4 and 5.

- For Task 1, the Mann Whitney U test was inconclusive: $N = 38$, $U = 166.5$, $p_{exact} = 0.757$.
- For Task 2, the Mann Whitney U test was inconclusive: $N = 38$, $U = 113$, $p_{exact} = 0.062$.
- For Task 3, the Mann Whitney U test was inconclusive: $N = 38$, $U = 130$, $p_{exact} = 0.175$.
- For Task 4, the askfacl users were significantly more confident: $N = 38$, $U = 86.5$, $p_{exact} = 0.009$.
- For Task 5, the askfacl users were significantly more confident: $N = 38$, $U = 101.5$, $p_{exact} = 0.027$.

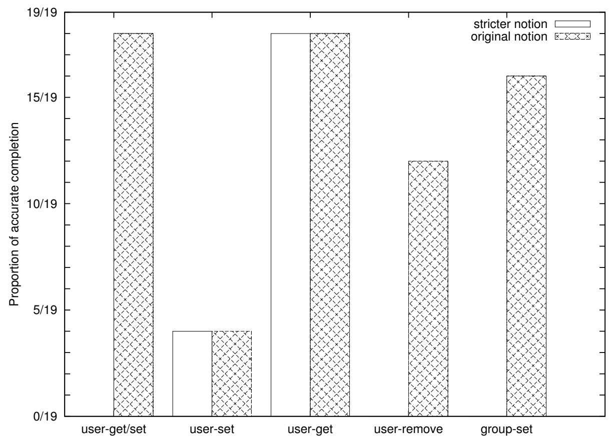

Figure 6: Accuracy rates for two different notions.

5.3 A Stricter Notion of Accuracy

As we mention in Section 2, the use of `get/setfacl` can result in side-effects and redundancy. The `askfacl` interface is designed to handle this automatically and does not burden the user. We revisited the data for `get/setfacl` from our human-subject study with the stricter notion of accuracy that not only must the objective of a task be met, but also that it must be met with neither side-effects nor redundancy. We considered the drop in accuracy rates as a consequence. Our intent is to assess the usefulness of this feature of `askfacl`.

In the following discussions, our null hypothesis is that the accuracy rates for `get/setfacl` under the stricter notion of accuracy are at least as high as that for the original notion of accuracy, and our alternate hypothesis is that it is not.

Experiment-Wide Results We determined the proportion of the $19 \times 5 = 95$ tasks that were completed accurately with `get/setfacl` under each notion of accuracy. The proportion was 68/95 under the original notion, and 22/95 under the stricter notion. To check for statistical significance of the difference, we first determined the proportion across all 5 tasks for each user, and then performed a Mann Whitney U test on the composite (ordinal) scores: $N = 38, U = 357, p_{exact} < 0.001$. The p value indicates that a Type 1 error is almost absent and hence we strongly reject the null hypothesis and conclude that there is a significant difference in accuracy rates between the stricter and original notions of accuracy for `get/setfacl`.

Task by Task Results We show the per-task accuracy rates for the two notions in Figure 6. To check the statistical significance of the difference in accuracy rates, we performed a one sided Fisher's exact test for each of the tasks.

- For Task 1, the accuracy rate is significantly higher under the original notion: stricter notion = 0/19, original notion = 18/19, $p < 0.001$.
- For Task 2, The result of the Fisher's exact test was inconclusive: stricter notion = 4/19, original notion = 4/19, $p = 0.654$.
- For Task 3, The result of the Fisher's exact test was inconclusive: stricter notion = 18/19, original notion = 18/19, $p = 0.757$.
- For Task 4, the accuracy rate is significantly higher under the original notion: stricter notion = 0/19, original notion = 12/19, $p < 0.001$.
- For Task 5, the accuracy rate is significantly higher under the original notion: stricter notion = 0/19, original notion = 16/19, $p < 0.001$.

Consequently, we strongly reject the null hypothesis for Tasks 1, 4 and 5. Our results suggest that the feature of `askfacl` that it deals automatically with side-effects and redundancy, and does not burden the user with them, is useful.

5.4 A Negative Result

As we mention in Section 4.4, we devised Task 6: all-set to be biased against `askfacl` by intent. It is intended to target a weakness in our current implementation of `askfacl`. Specifically, the task requires the setting of a kind of default policy for all users and groups of the system on a file. As the mnemonic "all" is not meaningful in `askfacl`, the only option for a user of `askfacl` is to iterate through every user and group — certainly a burdensome endeavour.

We included this task even though adding a mnemonic "all" is easy. We wanted to establish that this is indeed a problem for users (which our results that we discuss below clearly demonstrate). Our results indicate that it is not enough to simply design an interface to have a declarative semantics. It must be designed carefully taking into consideration various operational aspects. This negative result also validates the use of human-subject and field studies of access control systems, notwithstanding the thought that may have gone into the design of their interfaces.

Our null hypothesis is that users of `get/setfacl` complete the task at best as fast, accurately and confidently as those of `askfacl`. Our alternate hypothesis is that they are strictly faster, and more accurate and confident. As with our other results, we analyze time for accurate completion, accuracy rate and confidence.

The task by task results shows a significant difference in mean completion times: 4:10 (minutes:seconds) and 3:25 respectively for all participants and accurate participants of `get/setfacl`, and 12:09 and 8:07 respectively for all participants and accurate participants of `askfacl`. As only 1 person that used `askfacl` completed the task accurately, we did not perform any tests of statistical significance of differences between the two interfaces for time for accurate completion.

The accuracy rates of the participants were 12/19 and 1/19 respectively for `get/setfacl` and `askfacl`. For statistical significance of the difference, we performed a one-sided Fisher's exact test, which yielded $p < 0.001$. Therefore we strongly reject the null hypothesis and conclude that for Task 6, users of `get/setfacl` are significantly more accurate than those of `askfacl`.

The mean confidence was 8.74 for `get/setfacl` and 3.79 for `askfacl`. To check the statistical significance of the difference in confidence, we performed a Mann Whitney U Test: $N = 38, U = 322.5, p_{exact} < 0.001$. Therefore, we strongly reject the null hypothesis and conclude that for Task 6, users of `get/setfacl` were significantly more confident.

5.5 Discussion

Our main goodness criteria were time for accurate completion and accuracy. The experiment-wide results for both are strongly in favour of askfacl. Of the 10 task-specific tests, 7 yielded significant results, all in favour of askfacl.

A closer look at the tasks that did not yield a significant difference in favour of either interface is instructive. Tasks 1, 3 and 5 yielded no significant difference from the standpoint of accuracy. For Task 1, we conjecture that its relative ease is the cause. For Task 3, the ACL has no explicit entry for david, the subject in question. Our users have sufficient technical nous to dig deeper and solve the task with this observation. For Task 5, we conjecture that our wording may have helped the users of get/setfacl. As we mention in Section 4.6, we use the verb "ensure" rather than "give" as we do, for example, for Task 2, for which there is a significant difference in accuracy.

6. RELATED WORK

Our work is related to usability in the context of access control, and specifically the relationship between semantics and usability. The piece of prior work that is closest to ours is the work of Reeder et al. [23]. That work points out that even subtle changes in semantics in an access control system can impact usability. Our work is similar in that we also consider semantics. It is different in that we show that a declarative semantics can lend an ACL system to usability.

Other prior work extensively documents usability issues in the context of access control. For example, Cao and Iverson [10], Maxion and Reeder [18] and Reeder et al. [22] have pointed out usability issues with Windows ACLs, and proposed alternate interfaces for it. Our work has similarities to such work, and important differences. We are similar in that we find issues with an existing interface to an ACL system. We are different in that we consider command-line as opposed to graphical-user interfaces, we consider a de facto standard as opposed to a proprietary system, and our goal is to relate an existing thread of research (declarative semantics) with usability.

There are similarities between what we call declarative semantics and what Reeder et al. [22] call effective permissions. Our study is targeted at the relationship between declarative semantics and usability. Reeder et al. [22], on the other hand, have several features of Expandable Grids that they tout in addition to effective permissions. We argue that their conflation of a number of new features does not provide a clear verdict regarding effective permissions. In particular, that study does not include the work on Salmon [18] for comparison. Salmon improves on usability over the traditional Windows interface while still retaining a list-of-rules design.

There has also been other work in the context of access control policy management that proposes the use of graphical user interfaces. These include the following. Zurko et al. [29] have included a graphical user interface for access control policy authoring with their Adage system. The HP Select Access Policy Builder [19] includes a grid-like user interface. The work of Karat et al. [15] is mostly about allowing policy editors to make specifications in a natural language. Their approach to visualization is a graphical user interface. Rode et al. [24] also present an approach to visualization that is a graphical user interface. In their case, the authorization policy is shown as a pie chart. The differences from our work are the same as what we say above for the three pieces of work we discuss in the paragraph prior.

There has been work also on the use of access control systems. Such work provides valuable insights to work such as ours. For example, our assertions about the badness of exceptions in ACLs are based on the observations of Bauer et al. [7]. Similarly, the work of Smetters and Good [28] discusses the value of groups and also makes some insightful observations. For example, we have called out in the previous section their observation that while changes to ACLs are infrequent in practice, when changes are made, they are of a complex nature. Their work also proposes a particular simplifying approach to perceiving access control policies. Bauer et al. [6] discuss a study that compares what they call ideal access control policies for physical access based on a smartphone-based system, as opposed to using physical keys. Finally, there has been work such as that of Johnson et al. [14] and Rode et al. [24] that presents new kinds of policies for sharing resources. It is unclear whether such policies can be supported by POSIX ACLs. In any case, such considerations are beyond the scope of our work.

7. CONCLUSIONS AND FUTURE WORK

We have related declarative semantics and usability in the context of POSIX ACLs. The two are recurring, but heretofore disparate, themes in research in access control. We have designed and implemented a new interface to POSIX ACLs that we call askfacl, which has a natural declarative semantics. We have presented our design and results of a human-subject study that establishes that askfacl enables users to inspect and edit ACLs faster and with better accuracy than the existing interface, get/setfacl.

There are a number of topics for future work. One is a deeper analysis of our data, for example, a Hierarchical Task Analysis [16] to determine the kinds of errors users have made. A second is a larger field-study with experienced systems administrators as to whether there is a difference to them between two such interfaces. The third is a tighter binding of the interface with the underlying system — that is, to design, implement and usability-test an ACL system that is built with declarative semantics from the ground up. We plan to tackle some of these topics as future work.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Stuart Schechter, for their valuable comments.

8. REFERENCES

[1] *Publication manual of the American Psychological Association.* American Psychological Association, 6 edition, 2010.

[2] The linux kernel archives, Mar. 2012. http://www.kernel.org/.

[3] OpenBSD, Mar. 2012. http://www.openbsd.org/.

[4] Reporting statistics in APA format, Mar. 2012. Available from http://www.writingcenter.uconn.edu/pdf/Reporting_Statistics.pdf.

[5] script - make typescript of terminal session, Mar. 2012. http://unixhelp.ed.ac.uk/CGI/man-cgi?script.

[6] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea. A user study of policy creation in a flexible access-control system. In *CHI 2008 Proceedings - Policy, Telemedicine, and Enterprise*, pages 543–552, Apr. 2008.

[7] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea. Real life challenges in access-control management. In *CHI 2009 Proceedings - Security*, pages 899–908, Apr. 2009.

[8] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. Roy. Statist. Soc. Ser. B*, 57(1):289–300, 1995.

[9] M. Bishop. *Introduction to Computer Security.* Addison-Wesley, 2004.

[10] X. Cao and L. Iverson. Intentional access management: Making access control usable for end-users. In *Proceedings of the second Symposium on Usable Privacy and Security (SOUPS)*, pages 20–31, July 2006.

[11] R. A. Fisher. *Statistical Methods for Research Workers.* Oliver and Boyd, 1925.

[12] D. Garg, L. Bauer, K. Bowers, F. Pfenning, and M. Reiter. A linear logic of authorization and knowledge. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *Computer Security – ESORICS 2006*, volume 4189 of *Lecture Notes in Computer Science*, pages 297–312. Springer Berlin / Heidelberg, 2006. 10.1007/11863908_19.

[13] M. Huth and M. Ryan. *Logic in Computer Science.* Cambridge University Press, Cambridge, UK, 2nd edition, 2004.

[14] M. L. Johnson, S. M. Bellovin, R. W. Reeder, and S. E. Schechter. Laizzez-faire file sharing. In *Proceedings of the New Security Paradigms Workshop (NSPW'09)*, Sept. 2009.

[15] J. Karat, C.-M. Karat, C. Brodie, and J. Feng. Privacy in information technology: Designing to enable privacy policy management in organizations. *International Journal of Human-Computer Studies*, 63(1-2):153–174, July 2005.

[16] B. Kirwan. *A Guide To Practical Human Reliability Assessment.* CRC Press, Nov. 1994.

[17] N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 114–130, 2002.

[18] R. A. Maxion and R. W. Reeder. Improving user-interface dependability through mitigation of human error. *International Journal of Human-Computer Studies*, 63(1-2):25–50, July 2005.

[19] M. C. Mont, R. Thyne, and P. Bramhall. Privacy enforcement with hp select access for regulatory compliance. Technical report, HP Labs, Bristol, UK, Jan. 2008.

[20] N. Nachar. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in Quantitative Methods for Psychology*, 4(1):13–20, 2008.

[21] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:3–15, 2004.

[22] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *CHI 2008 Proceedings - Visualizations*, pages 1473–1482. ACM, Apr. 2008.

[23] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea. More than skin deep: measuring effects of the underlying model on access-control system usability. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2065–2074, New York, NY, USA, 2011. ACM.

[24] J. Rode, C. Johansson, P. DiGioia, R. S Filho, K. Nies, D. H. Nguyen, J. Ren, P. Dourish, and D. Redmiles. Seeing further: Extending visualization as a basis for usable security. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS)*, pages 145–155, 2006.

[25] S. Schechter. Common pitfalls in writing about security and privacy human subjects experiments, and how to avoid them, 2012. Available from `https://cups.cs.cmu.edu/soups/2010/howtosoups.pdf`.

[26] F. B. Schneider, K. Walsh, and E. G. Sirer. Nexus authorization logic (NAL): Design rationale and applications. *ACM Trans. Inf. Syst. Secur.*, 14(1):8:1–8:28, June 2011.

[27] Security Working Group, IEEE Computer Society. IEEE 1003.1e and 1003.2c: Draft Standard for Information Technology–Portable Operating System Interface (POSIX)–Part 1: System Application Program Interface (API) and Part 2: Shell and Utilities, draft 17 (withdrawn). Available from `http://ece.uwaterloo.ca/~tripunit/Posix1003.1e990310.pdf`, Oct. 1997.

[28] D. K. Smetters and N. Good. How users use access control. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, July 2009.

[29] M. E. Zurko, R. Simon, and T. Sanfilippo. A user-centered, modular authorization service built on an RBAC foundation. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 57–71, May 1999.