Modeling and Reasoning About Effective User Permissions in Social-Sharing Systems

This project lays a foundation upon which to build formal analysis for usable security on web. Building off the work of Ka-Ping Yee, we developed a set of design principles for on-line sharing applications, particularly social sharing networks. Social sharing is distinct from professional or business sharing, and consequently requires different models of privacy and permissions. This necessitates investigation into the shape and design of social sharing systems.

Security and privacy policies in social networks depend on a number of factors. The policy embedded in the program takes users' personal settings as input. Because users must manipulate these settings through the user interface, the design of the interface also plays a significant role. Therefore, in order to study the privacy of a user interface, we must carefully consider of two things. First, we have to consider the design of the user interface, in particular how it presents security policy choices to the user. Second, we have to evaluate the interaction between the user's actions in the interface and the privacy policy that exists on the server.

Building systems for social sharing presents a number of unique design challenges. The system must manage such issues as aggregation data at a large scale, interactions between multiple users' policies, and the intrinsically open nature of data on these social networks. These issues make much of the prior work on security through UI design difficult to apply to social applications. Two undergraduates and I redesigned Yee's principles to address these concerns. Our principles address the effect of these system quality on the design of the interface of the systems.

To address the effect of user actions on the privacy policy of the program, it would be useful to have a model of the systems that can capture this interplay. The undergraduates and I created a general model of sharing systems capable of describing how user decisions define the actual *effective* policy of a system. We were able to use this model to instantiate specific applications, giving us confidence that the model cleaves to the actual systems we are interested in.

In analyzing user interfaces, we suspect that human design input is indispensable and cannot be completely eliminated by formal analysis. Thus, we envision this model as a basis for tools intended to assist the system designer. For each of our principles, we have identified aspects of the principles that are amenable to static analysis, and aspects the require user experience design expertise. By identifying tasks suited to static analyses, we reduce the need for expensive and error-prone human analyses. Furthermore, by clarifying the human-factors components of each principle, we provide specific guidance to user interface engineers on how to evaluate those parts of the interface that need manual examination.

This work was done along with two undergraduate students. Much of the work was done jointly, including determining defining qualities of social systems that effect user interface design, such as the scale of the number of users and connections between them in the system. However, because each undergraduate had very different strengths, I partitioned the project to better utilize their expertise. The student with a stronger sense of systems and software worked more on instantiating systems in the model, while the student with a strong mathematical logic background focused more on the design of logical properties to correspond to the design principles. I worked with both students, which ensured that the disparate aspects of the project are consistent with one another (for example, the logical properties made sense with respect to the system model.)

Modeling and Reasoning About Effective User Permissions in Social-Sharing Systems

Hannah Quay-de la Vallee
Brown University Dept of Computer Science
Providence, RI 02906
hannahqd@cs.brown.edu

James M. Walsh
Brown University Dept of Computer Science
Providence, RI 02906

William Zimrin
Brown University Dept of Computer Science
Providence, RI 02906

Kathi Fisler
WPI Dept of Computer Science
Worcester, MA 01609
kfisler@cs.wpi.edu

Shriram Krishnamurthi
Brown University Dept of Computer Science
Providence, RI 02906
sk@cs.brown.edu

ABSTRACT

Most modern Web sites—ranging from Facebook to GitHub and beyond—have a strong sharing, and hence access control, component. Multiple factors determine whether sharing occurs: the access-control policy, the program, the user-interface, and the choices made by users. Indeed, the actual permitted accesses depend on decisions made not only by the owner of the data but also by third parties. Due to these complex interactions, it is difficult for developers to reason about the extent of control that users do and do not have over their data.

This paper presents a model of sites with social sharing and discusses how to analyze end-user data-access properties against that model. Our model encompasses all the factors identified earlier, enabling comprehensive description of a site's *effective* access policy. We show that the model is rich enough to capture several useful properties about end-users' control over their data. Several nuances arise when formalizing these properties under realistic assumptions about scale and available user operations. We illustrate these nuances and their consequences in the context of several mainstream websites.

1. INTRODUCTION

Many modern web applications support data-sharing among users. Some sites, such as social networks and photo repositories, are designed specifically to help users share data; others, such as version-control systems and conference managers, share data as part of the overall workflow that they support. Regardless of whether data sharing is a primary or secondary goal, applications must provide appropriate protections on data. In general, data creators should retain a certain degree of control over when sharing happens.

In a web application, a user typically accesses data through links or text on pages presented to the user. Many factors determine which information is presented to a user: an access-control policy affects which pages or links a user is able to access; the user-interface determines which of the available information is actually displayed; the access-control policy may depend on conditions or attributes that get set as various users take actions through a site. Thus, the *effective* data-sharing policy that a site follows results from combining information about policies, programs, user-interfaces, and user actions. The complex interactions among these sources in a real system suggests that errors are likely. Developers would thus benefit from tools that explore systems' actual data-sharing consequences. This in turn demands the models and analyses that underlie such tools.

Consider the widely-publicized incident in which Facebook revealed the sexual orientation of two students who had taken strong measures to conceal this information [**?**]. The students friended the owner of a Facebook group for a homosexual chorus to which they belonged. The owner added the students to the group (an action which did not require the students' permission because of the existing friends relationship). The owner had configured the group to make membership public, so Facebook announced the students' group membership to their friends (including their parents). The question here is not whether Facebook's policy is reasonable; rather, it is whether analysis tools could have helped developers determine whether the system had a way to leak this information that the student users sought to keep private, or, alternatively, whether the system provided the students sufficient controls to prevent the leak from occurring.

This paper presents a model, formal properties, and proposals for tools to help developers reason about end-users' control of data. Our work embraces the complexity of modern web applications, using several mainstream sites to guide our results. We show why reasoning about end-user data access demands modeling far more than access-control policies, or even just the interactions of access-policies and the programs that use them. We argue that developers need tools that help reason about issues traditionally left to HCI and

usability analyses. Leveraging existing work in usable security, we show how formal analyses of access-control policies and their programmatic contexts can provide considerable support to usability analysis. Overall, our goal is to set the foundations for tools that enable developers to assess end-user access-controls over data in real-world systems.

2. REPRESENTATIVE WEBSITES

In developing our model and data-access properties, we calibrated our work against four specific web-based sharing applications. Each of the four is an example of a modern data-sharing system, but collectively they represent systems with different broad sharing goals and different authority-granting features. We will refer to these examples throughout the paper to illustrate aspects of our models and property formalizations.

GitHub, a version-control system: GitHub manages repositories of files. Each repository has an owner who can grant other users authority to take from ("pull") or add to ("push") the repository. The owner can also transfer her ownership to another user. GitHub represents systems with simple sharing models: an owner of a datum grants or revokes access to others through a single action.

Facebook, a social network: Facebook users share personal details, photos, and comments with other users. Users share individual data either publicly or with select groups of other users. Groups include Facebook's built-in options for "Friends" and "Friends-of-friends", as well as user-defined groups (such as "Co-workers" or "My Football Team").

Facebook's information-sharing model is interesting because it involves roles (through groups), delegation (through friends-of-friends), tags, and many different types of content on which a user might want different sharing policies. The friends relation also induces a two-step authorization process, since certain authority is only granted after one user issues a friend request and another user accepts it.

Google Drive, a service for collaborative editing and file sharing: Drive users create, edit, and share files (such as spreadsheets, presentations, and documents) with self-defined groups of other users. Drive represents systems designed for fine-grained access-controls (separate policies are often defined per document), as well as systems in which many users may configure access to the same content. File-sharing systems are a common case study in for capability-based access-control [**?**], a finer-grained mechanism than typical of access-control policies.

Resumé, a job-application manager: Resumé (a private system developed by one of the authors) manages submission and review of applications for faculty positions. Applicants and reference-letter writers upload materials to Resumé. An applicant can create an account and add materials to it before formally submitting her application. Following submission, members of the department can see the materials but neither the applicant nor the reference letter writers can edit them further. Department members may submit comments on applications. The applicant never has access to the comments or the reference letters. Applicants and letter writers may email materials to a staff member who has authority to upload materials on behalf of others.

Resumé is typical of systems with rich access-control policies whose rules take effect at different points in a broader workflow. The richness in the policy arises from rules that consult several attributes beyond the typical subject, action and resource. The delegation of tasks to support staff also raises questions about least authority and the trust boundaries within which software systems get deployed.

2.1 Relevant System Traits

These four systems raise several traits of modern social-sharing systems that models and analyses must support:

Massive Scale in Users and Data.

Github, Drive, and Facebook manage massive numbers of users and amounts of data; in Facebook, users and data are richly connected. At these scales, users cannot manage authorities at the level of individuals. All three systems support user-defined groups and multiple classes of data; users configure access at the level of groups, as in an RBAC policy. Simply seeing these systems as instances of RBAC, however, misses the more critical point that RBAC can be as much a problem as a solution in these systems. In an effort to control sharing, users might create more groups than they can manage at a cognitive (or time) level. To mitigate that, users often choose to violate least-privilege; they share with a superset of their desired audience to save the hassle of creating and managing yet another group. Attempts to reason about end-user sharing must account for the over-approximation of groups and other forms of tagging.

Delegation and Privacy.

All four systems support some form of delegation (friends-of-friends in Facebook, granting administrative privilege in Github, etc). By design, users who have delegated authority will no longer have sole control of others' data access. Lacking control though, questions arise as to whether owners should have knowledge of how their data is accessed and managed. This rapidly gets into privacy considerations: in Facebook, for example, knowledge of how data propagates might effectively expose a user's list of friends. This suggests that analyses will need tunable parameters about which situations constitute or violate privacy.

Privacy Across Organizational Boundaries.

Resumé raises privacy and confidentiality concerns. Job applicants should not know which specific people are on the hiring committee; typically, they do not know which groups of users (e.g., faculty versus graduate students) are represented on the committee. In general, some details of how an organization performs a task should be withheld from users who have provided information, which in turn limits users' access to information about who sees their data. Health-care policies, however, are subject to transparency regulations that require releasing some of this information to users. Thus, models and properties must account for varying privacy and confidentiality concerns.

Workflow-Based Systems.

In systems (such as Resumé) that support complex workflows, authority can vary with the phase of a workflow (e.g., conference papers may not be submitted once reviewing has begun), or change as the result of user actions (e.g., Resumé applicants lose the ability to revoke the department's access upon submitting their completed application). This raises questions about what access, control, or information users should retain once they lose effective control of data such as job applications. These questions affect how we model own-

ership of resources: does an owner retain rights to a resource even when the system no longer provides control over access, or should ownership somehow imply a degree of access?

Ascribing Ownership.

Interpretations of "a user's resources" must account for *tagging*, a social network feature in which one user (or an algorithm within Facebook itself!) associates a resource with another user. In the name of privacy, tagged users should have revocation access and be notified of tagging. This suggests that ownership is not a sufficient concept for modeling or reasoning about social-sharing systems.

3. ANALYZING DATA-SHARING, INFORMALLY

Before we design a model for reasoning about end-user control of data, we should explore the information that such a model demands. Consider the following simple data-sharing property that one might expect of a system. It checks whether data owners control other users' access to their data:

> *Access to a user-owned datum is granted or revoked only with permission of the owner.*

The challenge in formalizing this statement lies in defining "permission of the owner". In many systems, users do not grant permission through a single action with that express purpose. Assume that a Facebook user configures a photo album to be shared with her friends. If she issues a new friend request that is later accepted, the new friend will see the album. The album owner did not directly edit a configuration such as an access-control policy; in fact, the access-control policy didn't even change in order to grant the new friend access to the album. This indicates that our model must encompass more than just the policy.

Three aspects of this example are particularly interesting when we consider how to reason about users granting authority to one another:

- Multiple actions from multiple users were required to grant the new friend access to the album: issuing the friend invitation (by the album owner), and accepting the invitation (by the new friend). Thus, **enabling new permissions is a multi-step process**. In the model, this will manifest in access-control rules with conditions beyond simple role/action/resource triples.

- The action that corresponds to the owner granting permission (here, issuing the invitation) is not necessarily the last user action that occurs before the new permission is active on the website. Thus, **permission-granting actions may be temporally separated from activation of permissions**. Thus, our model needs to capture the sequences of actions taken in the program, as well as information on how users' actions update the conditions referenced in access-control rules.

- User actions may have implicit data-sharing consequences. The link or button to invite a friend says nothing about sharing the album: sharing is an implicit consequence of adding a friend. Users are responsible for understanding what information new friends will be able to access. Thus, **some aspect of the model and analyses must connect user actions with data-sharing consequences**. The model will need to capture the consequences of actions and perhaps information about how the interface presents these to users; analyses will need to consider whether users understand actions' consequences.

This last point suggests that reasoning about end-user data access ultimately involves questions about usability and human-factors concerns. Ka-ping Yee connected data access with usable security back in 2004 [**?**]. He proposed ten principles of usable security, many of which focused on whether an application gives end-users sufficient information to make informed decisions about data access and sharing. This paper adapts several of Yee's principles to reasoning about modern social-sharing sites. When Yee developed his principles, massive-scale, cloud-based web-based applications such as our benchmark systems were nascent or non-existent; his principles thus implicitly codify the expectations of closed, corporate systems, and many of the assumptions no longer directly apply to social sharing-based applications. We discuss these issues and their impact as we discuss property formalization in Section 5.

4. A FORMAL SYSTEM MODEL

Our focus on web-based systems demands a model of client/server systems in which users access shared data stored on the server, and initiate operations through elements on (web) pages. Most of our model is straightforward: a website manages data and relations on that data, providing users with actions to create, modify, and view data and relations. We capture conditions for authority over resources through an explicit access-control policy. For simplicity, we assume that the "resources" under access control are application data (documents, passwords, etc), rather than references to system state, processes, or other artifacts that some security work also treats as resources.

Handling client pages raises two interesting issues: modeling elements of user interfaces, and handling elements that become stale relative to the viewer's authority. On the interface side, we model which data and information about authority a user can access; this helps us write properties about whether users have enough information to make informed data-sharing decisions (Section 4.3). On the authority side, we explicitly account for TOCTOU violations by checking that users are still eligible to execute actions through webpage links (Section 4.4). We describe our handling of both issues in more detail within the cited sections.

4.1 Data Schemas and Access Policies

Every website manages certain relations over certain data, governed in part by an access-control policy. We use the term "application" for this core structure of a website.

DEFINITION 1. *An* Application *contains:*

- *A set D of* Domains*, representing system-specific information of relevance to the application.*

- *A distinguished element* Users $\in D$*, representing potential users of the system.*

- *A set $R_1, \ldots, R_n$ of* Relations *maintained by the application. Each relation has a type $D_1, \ldots, D_k$ where each $D_i \in D$.*

- *A set* Data $= D_1 \cup \ldots \cup D_j$ *where each* $D_i \in D$. *This distinguishes domains corresponding to securable data.*
- *A set Act of* Actions. *Each action act has a type* $D_0, \ldots, D_n$ *where each* $D_i \in D$ *and* $n \geq 0$.
- *A list of* Access-Control Rules *of the form*

  $$Permit(u, d_0, act(d_1, \ldots, d_n)) \textbf{ if } \exists \overline{v} : \phi$$

  where act is an action of type $D_0, \ldots, D_n$ *and* ϕ *is a conjunction of terms over* u, $d_0, \ldots, d_n$, $\overline{v}$, *and the domains and relations in A.*

Our definition of actions and the form of access rules are closely linked. Permissions typically capture actions of a single user on a single resource, but actions may require multiple inputs. Our action model assumes that the primary resource is the first argument to the action; the access rules separate this argument from the others. Rules may need to reference data stored on the server that are not parameters to the action; the existentially-quantified variables capture these. We illustrate actions and rules using two examples from conference managers. A rule that allows a user to read a review during the reviewing phase if the user is assigned the paper and has already submitted a review would be:

$$\text{Permit}(u, r, readRev()) \textbf{ if}$$
$$\exists r_2, forp : r \neq r_2 \wedge Review(r) \wedge Paper(forp) \wedge Assigned(u, forp) \wedge SubmittedReview(u, r_2, forp)$$

In the following rule that allows a program chair to upload a review on behalf of a reviewer, the main resource (the review) is related to other other data: the reviewer, and the paper being reviewed. Our rules capture this by making the review r the primary resource for the permission, while the auziliary data about the reviewer and paper being reviewed are arguments to the *submitRevFor* action.

$$\text{Permit}(u, r, submitRevFor(pc, forp)) \textbf{ if}$$
$$Chair(u) \wedge reviewer(pc) \wedge Paper(forp) \wedge Assigned(pc, forp)$$

The rest of the paper abbreviates permissions as $\langle u, d, act \rangle$, suppressing the arguments to the action unless they are relevant in context.

4.2 Server

Servers instantiate the Data, Users, and Relations sets in an application with concrete data values, users, and relational tuples. Whereas the Application for GitHub would have domains such as Files and Folders and a relation such as Owns (on Files × Users), a Server for GitHub would populate Files with specific documents, Users with specific users, and the Owns relation with tuples indicating which user owns which files.

Definition 2. *A* Server for Application *A consists of:*

- *A set* KnownUsers ⊆ Users
- *A distinguished element* UnknownUser ∈ Users *that is not in* KnownUsers
- *A set* KnownData ⊆ *Data*
- *For each domain* D_i *in A, a set* DE_i *of elements of* D_i
- *For each relation* $R \in D_0 \times \ldots \times D_n$ *in A, a set of tuples over* $DE_0 \times \ldots \times DE_n$

The *KnownUser* and *KnownData* sets help distinguish elements of *Users* and *Data* that are currently active on the server. As a website runs, new users become known as people create accounts; new data arises as people upload files; similarly, users may leave a site and files can be deleted. Since we may need to reason about new or former users, we distinguish the currently known Users and Data from all those that may exist in the application. The *UnknownUser* provides a way to talk about someone who views pages without having an account or being recognized on a site.

Servers contain sufficient information to evaluate access-control rules. Intuitively, a user has the authority to perform an action if the body of some rule evaluates to true under the domain and relation contents in the server.

Definition 3. *Given a server S, user u, action act, and concrete data* $d_0, \ldots, d_n$ *from S that respect the type signature of act, S* permits $\langle u, d_0, act(d_1, \ldots, d_n) \rangle$ *if there exists an access-control rule such that* $Permit(u, d, act(d_0, \ldots, d_n))$ *returns true when the rule's formula* ϕ *is interpreted under the domain and relation contents of S.*

4.3 Clients and Pages

Clients correspond to website users (which could be either a *KnownUser* or the *UnknownUser*); each user has a set of active *pages*, which intuitively correspond to different tabs or windows in a web-browser. In a real website, pages contain descriptive text, information stored in the server, and ways to execute actions. Our model of pages abstracts away descriptive text, leaving only bits of server data and current permission information that would otherwise be embedded within free-form text. Our model also uses a generic concept of links to cover links, buttons, and other forms of executing actions against the server.

Many users can have pages open on the same site at the same time. Users' actions can affect each others' permissions, as this paper has already discussed. Thus, when modeling client web pages, one must assume that links and data on a page may have become stale relative to the server state. In particular, a user may have a link to execute an action for which she is no longer authorized (typical TOCTOU—"time of check to time of use"—problems). Even modern web systems that automatically refresh displays may have brief periods of stale page content. Our model thus assumes that links and data may be stale, and checks users' authority at both time of display and time of execution.

Definition 4. *Let S be a server for an Application A. A* page for *S contains*

- *A set of* Links, *each corresponding to an action*

  $$act(d_0, \ldots, d_k)$$

  and supplying a concrete value for each d_i *from the corresponding domain* D_i *as given in the type for act.*
- *A set* Contents ⊆ *S*.Data
- *A set of* Permissions *of the form* $\langle u, d, act \rangle$ *for* $u \in$ Users, d *in some* DE_i *in S, and act in Act*

Returning to the conference manager, a link to submit a review might look like *submitRevFor(review2, Bob, paper1)*, where *review2* is a specific review and *paper1* is a specific paper. Note that in this form, the primary resource (*review2*) is included as an argument to the *submitRevFor* action; in contrast, when we used *submitRevFor* in the access-control rule, we moved *review2* out to the level of the permission.

The distinctions among Links, Contents, and Permissions are key to our model. Links represent actions that the user viewing the page is permitted to take (or was, at the time the page was generated). Permissions, in contrast, purely convey information about the state of authority in the system. In the context of Facebook, the ability to invite a friend would be a Link; that another user, Susie, is allowed to view the page-viewer's vacation album would be a permission. If a page chose to embed Alice's photo in the user's page, that photo would lie in Contents. In the context of GitHub, a repository might be available to a user as Contents; indications of what kinds of access the user has to that repository, in contrast, would appear as Permissions.

4.4 Transition Function and Page Generation

Intuitively, applications generate pages dynamically in response to users' requests to execute actions. Executing an action yields both a new page to display to the client and an updated server. The generated page, however, must reflect the authority of the user who took the action (and hence receives that page). Authority is determined by the rules of the access-control policy.

DEFINITION 5. *Page P is* valid *for user u and server S iff $P.Contents \subseteq S.$*KnownData*, and for every $act(d_0, \ldots, d_k) \in P.Links$, S permits $\langle u, d_0, act(d_1, \ldots, d_k)\rangle$.*

The model captures the effects of actions in two functions: Act_{pg} maps each user u, action invocation, and server S to a valid page for u and S. Act_{op} maps each user, action invocation, and server to a new server, reflecting how actions change the server state (e.g., adding friends or documents).

Given Act_{pg} and Act_{op}, we can define the transition system for a website. The website state reflects the contents or pages and the server state. Intuitively, a transition occurs when a user takes a permitted action by clicking a link on a page. The resulting next state replaces the old page with a new one and updates the server state according to Act_{op}. The check in Definition 7 that linked actions be authorized in the server state accounts for potential TOCTOU violations.

DEFINITION 6. *Let S be a server, P be a page on client C for user u, and $act(d_0, \ldots, d_k)$ be in $P.Links$. act is* authorized *for C in S iff S permits $\langle u, d_0, act(d_1, \ldots, d_k)\rangle$.*

DEFINITION 7. *An* Application State *consists of a server and a set of clients. Let $\langle S, \{C_1, \ldots, C_j\}\rangle$ be an Application State and act be an action linked to some page P in some C_i. If act is authorized for C_i in S, then the* next Application State *is $\langle Act_{op}(act, S), \{C_1, \ldots, C_j, C_i'\} - C_i\rangle$ where C_i' is identical to C_i except P has been replaced with $Act_{pg}(act, S)$. If act is not authorized for C_i in S, then the next Application State is the same as the given Application State.*

Preserving the Application State on unauthorized links makes sense for the server; how to handle stale links on the page is more subtle. Some websites do have links that are stale briefly (e.g., waiting for a transaction to finish); some sites regenerate pages. This choice is irrelevant to our work. Our model thus takes the safe default of retaining stale links; modelers of sites may refine this decision as appropriate.

5. PROPERTIES AND ANALYSIS FOR END-USER DATA SHARING

With a model in hand, we now propose and formalize several end-user data-sharing properties over the model. Our proposed properties are variants of ones proposed by Yee [**?**]. We find these properties interesting because they emphasize different ways that a website might help end-users make data-access decisions. As we will demonstrate between this section and Section 6, these properties also illustrate the interplay between reasoning about information content and reasoning about human-factors concerns. Both kinds of reasoning are important for assessing whether a website provides users with good data-sharing facilities.

Several of our properties reference user actions that contribute to the granting or revocation of data-access authority. Thus, we begin by defining how actions in the model can affect authority. First, we define what it means for authority to change on a single transition, and on a path:

DEFINITION 8. *A transition from state s to s'* grants permission *p if s does not permit p but s' permits p. Similarly, a transition* revokes permission *p if s permits p but s' does not permit p. Action act is* authority editing *from state s if a transition from s on act grants or revokes some permission. A path of transitions is* authority-preserving *if no permissions are granted or revoked along the path.*

Next, we define what it means for a transition to make progress towards granting permission. The computation underlying this definition is local to the access-control policy within the model; had our model captured access controls without explicit policy rules, this definition would have been more complicated. Once we identify actions that make progress towards permissions, we can examine paths to states that eventually grant permissions to determine whether an action effectively constitutes a user's consent. Some of the properties in this section will address whether the effective consent is intentional or sufficiently informed.

DEFINITION 9. *Let S be a server, p be a permission that does not hold in S, act be a valid action for user u in S, and S' be the server that would result if u took act from S.*

- *act* advances *p* in *S if for some access rule r for p, more conjuncts of r are satisfied in S' than in S.*
- *act is* consent-granting for *u* and *p* in *S if act advances p in S and there exists a path from s' on which p is granted without further actions from u.*

We now discuss candidate properties and their formalizations against our model. Each of the following subsections starts with an informal description of a property, then formalizes its concepts. Within each formalization, underlined terms represent parameters that must be instantiated relative to data and relations in a particular website, while boldface terms represent human-factors components to the property. Section 6 will discuss how we could analyze handles the human-factors components.

5.1 Change Authority Only with Consent

Grant or revoke authority to others in accordance with user actions indicating consent.

The key concepts to formalize here are "who can consent" and "what constitutes consent". Our definition of consent-granting actions (Definition 9) captures the latter. For the former, we assume that each site has a notion of which users administer or manage each datum: Github tracks repository owners, Google Drive tracks document owners, Facebook users own data that they uploaded. To use this property, a developer would instantiate the *administers* term in the formalization with references to whichever relations in the data schema reflect this concept. For simplicity, we formalize the property in terms of granting authority; the version handling revocation is similar.

Formalization: *For every state s, every permission $p = \langle u, d, act \rangle$ that does not hold in s, and every path Π from s to a state s_2 in which p first holds, then Π contains a transition t by a user u_a such that u_a administers d and either (1) the action on t is consent-granting for p by u_a, or (2) the action on t gave administrative privilege for d to a user u_d, who subsequently took a consent-granting action for p within Π. In either case, u_a* **understood** *that the action on t would grant consent or delegate authority, respectively.*

This first formalization illustrates our notions of property parameters and human-factors components. Parameters should be instantiated in terms of data and relations in the model. Whenever a formalization raises a human-factors component, that component could be ignored (by omitting a clause of the property) during a traditional static analysis, then handled as discussed in Section 6.

5.2 Allow Reduction of Authority

Offer the user ways to revoke others' authority to access the user's resources unless the user previously took an action he understood would (eventually) relinquish that authority.

The previous property checked that revocation of access only happens with the consent of an appropriate user; it did not mandate that users have access to actions that revoke permissions over their data. This property requires the latter. Exceptions to this property can arise in workflow-based systems, as users take actions that are expected to relinquish authority. In Resumé, for example, once a candidate submits a job application, the candidate no longer controls its propagation (though the candidate may retain rights to withdraw the application). Our formalization therefore requires users to maintain access to actions that revoke authority unless they took actions specifically intended to relinquish their administrative control.

One other subtlety arises in this property: users should have final say as to whether access to their data is revoked. With multi-step permissions, users may consent to an permission, but actions of other users are required to realize the permission. With revocation, an action by an administering user should suffice to revoke another's permission (otherwise, a user might be in a position to consent to having his authority revoked). When access-policy rules all specify permits (rather than permit/deny/not-applicable) and have only conjunctions in their rule-bodies, this requirement simply needs a revoking action to falsify a conjunct in the policy rules. Our formalization, however, uses a more general statement that applies to a wider range of policies.

Formalization: *In every state s, if user u_a administers datum d and s has permission $p = \langle u, d, act \rangle$, then there exists an authority-preserving path from s on which u_a can revoke p unless u_a previously took an action that he* **understood** *would eventually revoke p and that revoking action has occurred. Furthermore, on all paths from s, u_a can reach an action to revoke p, u_a stops administering d, or the action that revokes p occurs.*

5.3 Summarize Others' Authority

Maintain accurate awareness of others' authority at a granularity relevant to user decisions, without violating others' privacy.

Imagine that a Google Drive user has several folders, each of which is shared with a different group of other users. She needs to share a somewhat sensitive document with some other users. In order to decide whether to upload the new document to an existing folder or to a new one, she needs to review who can access each of her shared folders. This property checks that the user can view this information. Formally, we capture ability to view as having access to a web page that contains the information (through the Permissions component of a page from Definition 4); the path to such a page should not be guarded by the actions of other users.

In systems with massive numbers of users and data, direct presentation of others' permissions could be overwhelming. Furthermore, relevant information might infringe on another user's privacy (such as if a Facebook user wants to know who her friend's friends are before using the "friends-of-friends" setting). Our formalization must account for these nuances.

The key challenge in formalizing this property lies in defining when an existing permission is "relevant to a user's decision". We restrict "user decisions" as "choosing to take a consent-granting action". A narrow interpretation of "relevance" to a consent-granting action for a specific permission would look at the conditions on that permission in the access-control policy: information that affects those conditions would be deemed relevant. Such a definition would not, however, capture concerns about higher-level connections between system data. For example, a user might be willing to share anonymized medical data in a corporate file-sharing system (a private version of Google Drive), as long as the viewers didn't also have access to the file with the mapping from anonymous tags to names. Given the challenge of a suitable general definition, we leave "relevance" as a parameter for developers to tailor relative to the data and relations in a particular website.

Formalization: *In every state s, if user u_c can take a consent-granting action for permission $p = \langle u, d, act \rangle$ in s, then for every related permission p, there exists an authority-preserving path to a page for u_c that* **summarizes** *p, unless doing so would* **violate u's privacy**. *Furthermore, on all paths from s, either u_c can reach a page that summarizes p or some action revokes p.*

Yee's original principles also stated that users should have access to information about their own current permissions.

This reduces to a formal property about paths to pages displaying a user's own permissions. As such a property does not add insight above the formalization for others' permissions, we do not discuss it further in this paper.

5.4 Explain Consequences

> *Indicate clearly the consequences of decisions that the user is expected to make, without violating others' privacy.*

As with the principle on others' authority, we focus on users' decisions to execute actions within the website. User actions can have one of three kinds of consequences relative to data access: an action can change a permission, an action can consent to changing a permission, or an action can advance a permission. In each case, the affected permission could be either to the user executing the action or to some other user. Under even moderate numbers of users and data, the set of all potentially-affected permissions for an action could be overwhelming. As a result, our formalization focuses on changed permissions and consent. The formalization is easily adapted, however, to include advances of particular permissions.

As when summarizing others' authority, naïvely showing permissions has the potential to violate other's privacy. Our formalization leaves privacy violation as a parameter, as in the case for others' authority.

Formalization: *For every state s and action act available to user u, u has an authority-preserving path to a page displaying all of the permissions that will change between s and the next state s' of s on act and that do not* **violate another user's privacy**. *If act is consent-granting for u and permission p, then u has an authority-preserving path to a page displaying p.*

5.5 Encourage Least-Privilege

> *Match the most comfortable way to do security-oriented tasks with the least granting of authority. Match comfortable ways to do sharing-oriented tasks with acceptable granting of authority.*

A "task" corresponds to a user's goal, such as creating a new document or sharing specific photos with certain friends. While some tasks correspond to atomic actions in a system (such as "create a document"), others correspond to sequences of actions ("upload photos and share them"). We therefore view tasks as paths to goal states within a system. Accordingly, we interpret this principle as asking whether paths that grant the least authority are also sufficiently easy that users will take them (a site that made it difficult to share with individuals instead of everyone would, for example, violate this property).

This principle is interesting for two reasons: first, the human-factors issues are harder to separate out from a core information-based principle; second, massive scale of users and data often makes least-privilege an impractical standard. One could certainly approximate this principle by weighing the number of permissions granted along a path against the number of actions required of a user along that path. In practice, we expect developers will need more nuanced interpretations of "comfort".

Finer-grained formalizations of comfort would also require finer-grained models of pages. Our page model (Definition 4) could be expanded to include link styling information (bold fonts, colors, general position on the page, etc). Given that modern web systems codify many of their design choices in CSS stylesheets, models that at least partly draw on UI decisions seem feasible, and worthy of additional investigation.

DEFINITION 10. *A path's* authority-weight *is the total number of permissions granted along its transitions.*

Formalization: *The* **low-difficulty-weight paths** *to a goal state for a task should be a subset of the least-authority-weight paths to a goal state for that task. Furthermore, every* **comfortable path** *to a goal state for a task yields* **acceptable authority** *for that task.*

6. BUILDING TOOLS ON THIS WORK

This work strives to provide foundations for useful analysis tools for developers. We want to see our models and properties used to create tools that help developers identify subtle bugs in the complex interactions that underlie modern web-based data-sharing systems. The combination of our model (Section 4) and preliminary properties (Section 5) suggest three broad categories of tools.

First, one might build a verification tool that could verify properties (ours or others') against instances of our model. We have begun building such a tool ourselves within the Alloy analyzer [**?**]. To date, we have a model of GitHub with repositories as data, basic push and pull operations as allowed accesses, and various actions that affect users' authority to act on repositories. The model differs a bit from that in Section 4, most notably in embedding the access-control policy in specifications of actions (rather than model a standalone policy); this simplifies the state space, though we cannot yet quantify the impact of this modification. Against the embedded access-control model, we have been able to check the "authority with consent" property against this model on traces up to five states in length, with each check requiring a few minutes in real time. Work on this tool is ongoing.

Second, one could build tools that compute data-sharing consequences of model instances without performing verification. For example, one could compute all sequences of user-level actions under which a (kind of) datum gets shared with a (kind of) user. Such an computation could start from the access-control rules, compute the various requirements for permitting an access under those rules, then compute the program and user actions required to satisfy those requirements. Using such an approach against a model of Facebook, for example, one might determine that "A user u will be able to see a photo owned by user o if o sends a friend request to u, u accepts the request, o uploads the photo, and o sets the sharing permissions on the photo to her friends". (Some of these steps could also have been permuted, and the analysis would report these alternatives as well.) Presenting these data-sharing scenarios to a developer might help her identify cases in which data-sharing is not guarded by anticipated user actions, without burdening the developer with stating formal properties. This is in the same spirit of property-free analysis provided in policy-analysis tools such as Margrave [**?**].

Third, one might use our model and formalizations to build tools to help developers assess human-factors aspects

of web-based systems. Many of the human-factors concerns around data-sharing are about the accessibility of information, or whether end-users understand the consequences of system actions. One could use our model to compute consequences or information content, leaving developers with concrete guidance about what to assess on the human-factors end. Example tools in this style might include:

- Compute the consent-granting actions for each permission, leaving the developer to assess whether users understand these actions will grant consent.
- Report paths that a user can take to reduce authority, as well as permissions for which no such paths exist.
- Show instances of permissions related to consent-granting actions, to help a developer decide which should be conveyed to users.
- Compute the permissions that change by virtue of taking each consent-granting action, to help the developer identify unintended consequences between actions and permissions.
- Given a task, present the developer with alternate paths to the task, summarizing their authorities and highlighting those with the least authority.

In general, assessing a system's support for end-user data sharing cannot overlook human-factor considerations. We believe our work has something to offer in this regard: concretely, the properties presented in this paper can profitably be decomposed into parts that can be discharged entirely through computational means, and *residual* portions that require user intervention (such as through a user-interface designer or analyst). We are therefore especially intrigued by the prospect of this kind of decomposition in future work.

7. RELATED WORK

Several projects have formalized access-control within social networks. Fong et al. present a formal model of Facebook that is easily generalized to other social networks [**?**]. Carminati et al. [**?**] and Mika [**?**] use ontologies in the semantic web to design such a formalization. Those works represents user actions as well as user data and relationships between users. However, while they model which actions are available to each user, they do not investigate the effects of those actions. Besmer et al. formalize access-control policies of social networks with a specific eye towards the permissions of third party applications on those networks [**?**]. They focus on limiting the permissions of applications so that those permissions align more closely with users' privacy settings.

On the end-user side, several social-network researchers have developed tools to help users manage access to their resources. Cheek and Shehab's tool leverages users' existing social-network contacts to inform policies for other contacts [**?**]. Wang et al. also incorporate user relationships in order to suggest appropriate privacy settings, as well as offering users fine-grained controls [**?**]; Fang and LeFevre use limited user input to craft potential privacy settings [**?**]. These tools are designed to help users navigate the existing privacy controls of social networks. Our work focuses on helping developers and designers ensure that the access-control policy and privacy controls interact as expected.

Wang and Jin propose a system for minimizing the leakage caused by user error in cloud collaboration [**?**]. Their work relies heavily on the ability to instate company-wide mandatory access controls and to require company employees to adhere to file tagging standards. These restrictions would be infeasible in the context of a social network, where the data is owned and administrated by arbitrary users.

Targeting end-users' control of data-access seems reminiscent of ARBAC [**?**], a formalism for administrative access control. Including an ARBAC policy in our model would not address the core problem in this paper, which is the connection between user-facing actions and actual permission changes. ARBAC also says nothing about the human-factors issues discussed in this paper.

Much research has been done on the interplay between usability and security in computer systems. Several researchers have shown that user actions can unwittingly circumvent seemingly sensible security policies and mechanisms. Whitten and Tygar's classic case study [**?**], as well as other usable-security projects [**?**, **?**, **?**], illustrate the challenge to designing security features that people use properly.

Yee's work was one of the early, but not the only, efforts to propose design guidelines for usable security or privacy. Lederer, et al. [**?**] present five design pitfalls when designing interactive systems with privacy implications. Many of their principles resonate with Yee's, and could be similarly supported with static-analysis tools that target residuals. One of their pitfalls explicitly raises the need for security solutions to integrate into, rather than compete with, established workflows. At Microsoft, Reeder, Kowalczyk and Shostack [**?**] have developed guidelines for developers on how to present decisions to users in a way that encourages them to make secure choices. These guidelines contain more human-factors advice than Lederer's or Yee's, but also hint at avenues for formal tool support.

Yee's original principles [**?**] lacked the nuances of granularity, delegation, groups of users, privacy, and workflows that appear in our informal property statements. An interested reader should have no problem identifying which of Yee's principles corresponds to each of our informal properties. Our updating of Yee's statements to modern systems is a minor contribution of this work.

Usable security was one design goal in the Polaris [**?**], ScoopFS [**?**], and CapDesk [**?**] projects that inspired Yee's principles. All of these systems employed capability-based security, in which a user's access is embodied in objects or references, rather than access-control policies. Our system model, in contrast, assumes an explicit access-control policy. Based on existing research into reasoning about policies [**?**] and their interactions with programs [**?**], we believe an explicit access policy is an essential artifact for formal analysis: many interesting analysis questions can be answered (at least in part) on the policy alone, leaving lighter-weight analyses for the full system model. One can reconcile these perspectives by viewing capabilities as a way to implement a separately-declared policy (at the modeling level).

Cognitive walkthroughs enable usability evaluation without expensive and time-consuming user studies. Walkthrough frameworks propose questions that developers should ask about their systems' expectations of users' mental models and goals. Rieman et al. [**?**] and Blackmon et al. [**?**] propose automated tools to support walkthroughs, but their tools automate only the process of conducting a walkthrough. They

do not automate any of the artifact analysis within the walkthrough, which is the promise and purview of static analysis.

Automated formal analysis tools have been applied to specific usability concerns. Both Leveson et al. [**?**] and Butler et al. [**?**] proposed design metrics and tool support for detecting mode confusion in interfaces. Curzon and Blandford [**?**] analyzed usability design criteria themselves against a formal model of human cognitive processes. Other tools formalize user task models for systems [**?**] or log all user interactions with the system and check whether these correspond to expected usability patterns. Ivory and Hearst [**?**] provide a detailed summary of projects on the latter.

These earlier techniques focus mainly on the users' interactions with the interface, not the users' interactions *with the underlying system model* via the interface. Our work tackles the latter. A system model is essential for reasoning about usable security: the system controls how permissions are used, while the user ideally controls how they are granted and revoked. Our formalization of Yee's principles can be used to check (a) that users have the expected control over how permissions are granted and revoked, and (b) that the system applies those permissions in ways that the user expects. Formal modeling and analysis of interface components alone is likely a useful complement, but not a substitute, for our work.

Garg, et al. [**?**] analyze audit logs for compliance with security and privacy policies. Their work uses formal analysis to discharge objective components of the compliance checks, leaving a subjective residual for manual analysis. Our residual tools are in the same spirit, but with a focus on residuals about human-factors decisions.

8. DISCUSSION AND FUTURE WORK

As this paper has shown, a proper evaluation of social sharing systems demands an analysis of both systems and interfaces. Thus, it ultimately puts the problem in the realm of usable security. This is a fascinating and difficult problem because the two goals can sometimes seem at odds, though as authors like Yee point out [**?**], sometimes this is an artifact of our interface designs and methods for designating authority. When these two are brought closer together, seemingly conflicting requirements can actually become harmonious ones.

Our paper has presented a formal model that is readily amenable to tool support. We also identify how properties decompose into formal and non-formal parts, where the former can in principle be discharged by automated tools, leaving the latter as residual tasks to be evaluated by humans. This ensures that the critical element of human judgement is not eliminated, but the parts that can be handled automatically by a compiler are nevertheless treated in that way. In principle, for instance, the tools can be run frequently but the expensive human evaluation might happen more rarely.

Reasoning effectively about usable security demands system models that readily reveal the connections between user actions and authority. This paper argues for a separate access-control policy as part of the system model: the policy rules provide a clear starting point for computing information about authority within a system. Policies for mainstream web applications, however, consult information stored in the system state as well as in the request for access. Accordingly, we propose a fairly rich system model, in which individual states are effectively first-order relational models. We believe such models are essential in order to capture multi-step permissions and other complexities of authority in web applications.

Our discussions hint at several areas for ongoing work:

- *Security versus privacy:* Security and privacy are widely accepted as different concerns with different norms, but the line between them can be blurry. Principles about "accurate awareness" clearly raise privacy issues. Our current formalizations of the principles leave developers to flag privacy violations. Extending the system model with a privacy policy could better support this task. Such a policy would affect our definition of valid pages (Definition 5) to not leak private information. This requires care, however, as it implicitly expects that users are capable of both crafting and maintaining such policies.

- *Administrative Permissions:* The notion of administrative control is our model is coarse: it does not distinguish between different operations on a datum. In practice, administrative controls are finer-grained: someone may have control over who edits, but not who deletes a file. Administrative access-control policies [**?**] capture these nuances in separate policies from the access controls. We consciously chose to interpret administrative control coarsely in this paper, to allow us to focus on a coherent set of principles. A more realistic model, however, might include an administrative access policy (which would in turn affect the formalizations of the presented principles).

- *Usability Over Time:* Time introduces subtle complications in the management of information. It is tempting to look primarily at the availability and impact of actions that change authority in individual system states. Optimizing for temporally local usable security, however, can make the system less usable over time. In a system for small-scale collaboration within a large set of users, creating many groups (to maintain local security) increases the effort needed later (as users have to search for the right group for their task). Furthermore, there exist studies showing how too many security choices lead users to use systems insecurely [**?**]. We are, however, not aware of proposed principles that balance immediate and long-term usable security.

- *Accounting for Human Processes and Behavior:* Software tools are often used within rich systems of interactions between people and software. In social sharing systems, many aspects of security lie in the human processes rather than in the software: operating policies, incentives, and trust relationships are good examples. The principles we've discussed focus on usable security within the software. How would the residuals or our proposed tools change if we extended our model with process models of surrounding workflows, or task models of users with different security expertise? If these additional models proved effective, where would we get them to use in analysis of real systems?

- *Formalizing Presentation:* Web-based applications contain layout and formatting specifications through CSS stylesheets. If our model of pages included CSS-like

styling tags on individual data, automated analyses might be able to flag certain formatting concerns, such as a more secure action being presented in smaller, lower-contrast, text than a less secure one. Given the plethora of usability design rules in the literature, it is worth exploring which affect security and which, if any, could be analyzed against the tools and languages used to actually build production web applications.