User-Centered Security

Mary Ellen Zurko
zurko@opengroup.org
The Open Group Research Institute
Eleven Cambridge Center
Cambridge, MA 02142

Richard T. Simon
rsimon@opengroup.org
The Open Group Research Institute
Eleven Cambridge Center
Cambridge, MA 02142

Abstract— **We introduce the term user-centered security to refer to security models, mechanisms, systems, and software that have usability as a primary motivation or goal. We discuss the history of usable secure systems, citing both past problems and present studies. We develop three categories for work in user-friendly security: applying usability testing and techniques to secure systems, developing security models and mechanisms for user-friendly systems, and considering user needs as a primary design goal at the start of secure system development. We discuss our work on user-centered authorization, which started with a rules-based authorization engine (MAP) and will continue with Adage. We outline the lessons we have learned to date and how they apply to our future work.**

Keywords— **user-centered, security, authorization**

I. Introduction

As computing power becomes more prevalent on the office desktop and in the home, usability becomes increasingly important in determining which software will be deployed and which software will not. For example, usability made the World Wide Web [1] a success. While the utility of the Web relies on a variety of technologies such as its transport protocol and linking and naming standard, for most people the World Wide Web is the browser. A graphical, easy-to-use browser (Mosaic [1]) made the Web accessible and desirable. End users will not purchase or use security products they cannot understand.

In this paper, we hope to revive the study of user-friendly secure systems within the security community. We introduce the term *user-centered security* to refer to security models, mechanisms, systems, and software that have usability as a primary motivation or goal. The timing seems right for a renewal of interest in synthesizing usability and security. There is increasing pressure on government funded researchers to produce results that can be used to solve real world problems. The renewed interest in roles, led by NIST [9], is motivated in part by the similarity between roles for security and roles used in organizations. Baldwin's work on roles [2] was rooted in his experience with real users and the usability problems of some long-standing database security models. We discuss the history of usability in the security community in the next section of this paper. We then discuss the range of user-centered security work to date. In the fourth section, we discuss our early exploration of an authorization engine designed to support our vision of user-centered authorization.

1996 ACM New Security Paradigm Workshop Lake Arrowhead CA

We evaluate the pros and cons of this effort, as a precursor to further work in this area, and include a brief description of our current work in user-centered authorization. As our conclusion points out, we hope to see more work in user-centered security in the future; work that enables users to choose and use the protection they want, that matches their intuitions about security and privacy, and that supports the policies that teams and organizations need and use to get their work done.

II. Usability in Secure Software

Usability has yet to greatly influence the security community. This lack of impact is not from lack of need, nor from lack of understanding usability's importance in general and to security in particular. In 1975, Saltzer and Schroeder identified *psychological acceptability* as one of eight design principles for computer protection mechanisms [25]. While other principles such as *least privilege* and *fail-safe defaults* have become standards in the security literature, there has been very little work done on user-friendly security. Much of the work that has been done has appeared in publications for the Computer Human Interface (CHI) community [6][10][16][15][21][26].

Usability can be measured in a variety of ways. Representative users are assigned tasks to perform, and their performance is measured against usability goals couched in terms of number or type of errors or time to complete the task. Usability goals can also take the form of user satisfaction ratings gathered via survey after testing or use of the software. In addition to setting usability goals, usable software may be specified and designed based on interviews or task analysis with target users. As with most aspects of computer software (such as performance, modularity, extensibility, and flexibility), it is rarely achieved unless it is designed in.

The history of merging security and usability does not make us sanguine about its future. Secure systems have a particularly rich tradition of indifference to the user, whether the user is a security administrator, a programmer, or an end-user. The most widely used authentication mechanism, the password, is unsuitable for providing both easy-to-use and effective security to most end-users. People tend to pick short or easy-to-guess passwords, making them less effective. Measures that make passwords more effective (such as computer-generated pronounceable passwords) make passwords difficult to use. Military Multi-Level Systems (MLS) are prone to "label float" - information gets overclassified as a by-product of the mismatch between how users produce and use information and how labelling is determined by the MLS model. Gasser [13] reports a similar problem with Unix systems, which support the setting of default access modes for new files on a per-session basis. As a user's work moves from public to private files, the user is unlikely to remember to restrict their default access mode. A better solution, provided by Multics and VMS, is to set the default access modes on a per-directory basis. This example shows that considerations for users' natural working patterns can strengthen the security of the system.

Secure systems and security applications produce their own special challenges to usability. Since secure systems have been traditionally difficult to use, some people are under the impression that usability and security are innately in conflict. Most research and development in secure systems has strong roots in the military. People in the military are selected and trained to follow rules and procedures precisely, no matter how onerous. This user training and selection decreased the pressure on early secure systems to be user friendly. In another example of military influence, the first security model to achieve widespread attention in the security literature (Bell and LaPadula [3]) encoded military classification levels and need-to-know categories. Much effort was then spent trying to apply this model to all uses of secure systems. Clark and Wilson [5] redirected these efforts with a commercial model which could not be emulated by the Bell and Lapadula military model. Mathematical rigor was emphasized over usability. While social systems can be mathematically modeled, starting from a mathematical model does not guarantee a system that emulates user intuitions or current practice and is easy to use (although formal methods can be of service to usable secure systems [11]).

Some aspects of secure systems are difficult to make easy to use. In one example, the principle of Least Privilege [25] dictates a user should only have the privileges needed to accomplish the particular task at hand. This principle guards against user mistakes and trojan horses. However, defining the contents and limits of a task is rarely attempted in systems supporting Least Privilege. Some groupware systems are just beginning to make in-roads in this area. In addition, users often multi-process, working on bits of several tasks because of interruptions or limits on computer resources. The correspondence between privileges and tasks has been undefined by the system and is unclear to the user. Finally, it is frustrating to the user to have to enable privileges explicitly that they know they are cleared for and are necessary for the request they have made.

III. Directions in User-Centered Security

Recall that user-centered security refers to security models, mechanisms, systems, or software that have usability as a primary motivation or goal. Most work on usability emphasizes design process and testing (a situation that is partially familiar to secure systems engineers). Many techniques are available to help ensure consideration user needs and work habits at design time, and to test the utility and obviousness of proposed interfaces and features with target users.

Since user-interface technology is constantly evolving and user needs are so diverse, no particular technology or architecture is always "user-friendly." While some usability researchers work on producing theoretical results that can be successfully used to guide initial UI design, very few principles are robust enough to be generally applicable and specific enough to directly influence engineering decisions. One of the earliest concrete UI design principles is based on cognitive research which indicates that humans can only keep "seven plus or minus two" chunks of information in short-term memory. This guideline has helped bound menu and display design, and task analysis. More recently, usability practitioners have found the design philosophy of "affordances" [20] useful. The affordances of an object help a potential user determine how that object can be used. For example, when picking up a hammer, most people pick it up by the handle. The weight in the head of the hammer feels like the heavy side of the head can be used for knocking on things.

In the rest of this section, we consider several potential approaches to achieving user-centered security. We discuss the results available so far within each approach.

A. Applying Usability to Secure Systems

One obvious approach to synthesizing usability engineering and secure systems is to apply established procedures for enhancing usability to developing or existing secure systems. Techniques for enhancing the usability of software cover a wide range of context and sophistication:

- Contextual Design [27] uses in-depth studies of potential users' work habits and needs to determine initial product goals.
- Discount Usability Testing [22] involves user testing with low-tech paper mock-ups to get rapid feedback on early design concepts.
- In Lab Testing [24] users are asked to perform particular tasks with the software and their reactions and problems are monitored.
- Contextual Inquiry [27] provides usability testing on a deployed product, where real users using the system in their daily work chores allow observers to record this use.

Given the rich set of tools available in the usability community and the long history of their use, it is surprising that we have only discovered two published studies on their application to secure systems since 1985. Mosteller and Ballas [21] collected information on the error behavior of users of a secure interactive system that was instrumented to log error messages per user session. They document the most common errors (indicating particular areas where improvements could have the most impact) and report that error patterns do not change with increased experience with the system. Karat [15] goes farther towards the goal of usable secure systems. She reports that they were able to set and meet the product usability objective of a security application for data entry and inquiry by using a combination of mock-ups and iterative testing and design. These two datapoints suggest that secure applications are no more resistant to usability enhancements than other products. In particular, Karat's group was able to meet their usability goals using the same methods for producing usability enhancements that have been successful for other types of products.

B. Applying Security to Usable Systems

Another approach to synthesizing usability and security is to integrate security services that have software with a strong usability component, such as mass-market applications or groupware. Groupware has been the focus of the majority of security work in the Computer Human Interface (CHI) community. This is due to the need to consider the interactions between multiple users in groupware systems. Shen and Dewan [26] discuss a framework for access control in collaborative environments. This framework supports positive and negative access rights and their inheritance (much like Zurko's [28] privileges and prohibitions). More recently, Foley and Jacob [11] developed an approach for the formal specification of functionality and confidentiality requirements of Computer-Supported Cooperative Work (CSCW) applications.

Privacy has been an issue in groupware since the field began. CSCW '92 had both a workshop [6] and a panel [16] on privacy considerations, motivated by monitoring technologies such as active badges that were beginning to appear in experiments. Active

badge technology allows the whereabouts of each member of a group to be queried or monitored by others within any area equipped with badge sensors. One research group in England even equipped the local pub with them. Tutorials and workshops in audiovisual conferencing [10] and user interaction history logging [14] include privacy as a major issue.

C. User-Centered Design of Security

A more recent approach to synthesizing usability and security is the development of user-centered security models and applications. This approach takes consideration of user needs as a primary motivator when defining the security model, interface, or features of a system. The target user may be an end-user, an application programmer, a system or security administrator, or a group of users or social unit. The first security application to articulate a user-centered design philosophy was Privacy-Enhanced Mail (PEM) [18]: "The set of supported measures offers added value to users, enhancing rather than restricting the set of capabilities available to users." This was a startling vision in the security community that largely perceived security requirements as watching and restricting users. While usability problems with the certificate authority infrastructure kept PEM from being widely deployed, its primary motivation to offer users desirable security services such as privacy for their daily electronic mail remains a laudable goal still largely unmet today.

Pretty Good Privacy (PGP) [12] attacked the most obvious problem in PEM, the trust infrastructure. PGP's trust model puts each user squarely in the center of trust relationships. For each public key on the user's keychain, the user indicates how trustworthy that key is when it vouches for other keys (unknown, untrusted, marginally trusted, or completely trusted). The user then tells PGP how many signatures of any type are required to certify a new public key. This model has been referred to as a "web of trust." But that has inaccurate implications since the user is never more than one hop away from a certified key. Moreover, the trust model enables each user to designate who is trusted for what, overcoming the problem with trusting a single rooted hierarchy of Certificate Authorities (CAs).

PGP has other usability problems, notably the complexity of managing keys and trust. It is unlikely that casual computer users would be able to protect and manage their keys safely and effectively. Nevertheless, PGP's user-centered trust model is a step in the right direction. More recent work in user-centered trust models was done by Blaze, Feigenbaum, and Lacy on decentralized trust management [4]. Their system allows any user or site to specify what a key is trusted to do (including who a key is trusted to vouch for). Users can specify their own policy or accept digitally signed policy statements from someone else. Applications call this system with key and request information to determine whether a request will be granted.

IV. MAP: A Prototype of a User-Centered Authorization Engine

We had the opportunity to explore the implications of user-centered design on an authorization engine. MAP (Management of Authorization for site Policy) [30] was a six-month project that built prototype tools to support policy-oriented operations on a site's underlying authorization mechanism. This prototype augmented an existing Access Control List (ACL) mechanism with sensitivity labels, object groups, and access rules. We hypothesized that a rules-based engine would more easily support natural language site policy than one based on ACLs or capabilities, because of the greater similarity of rules to user policy. We will continue testing that hypothesis in our Adage work (see next section). Because of the short time-frame for this prototype, we extended an existing application (DCE-Web [7][17]), and made use of the DCE infrastructure, most notably its support for users and groups. The implications of these constraints are considered below.

MAP extended the authorization engine of the DCE-Web Server (Wand) with the label and rule mechanisms outlined below. We also extended the Secure Local Proxy (SLP) client to provide a Web forms-based user interface to our management functions. The SLP is a small, local HTTP proxy server that speaks DCE to the Wand Server on behalf of the user's off-the-shelf Web browser. The Wand Server and Secure Local Proxy were enhanced in two general ways: the management of the rule and label databases, and the use of the new authorization information. Users created rules and labels using a simple forms interface. That interface was used through an off-the-shelf Web browser that communicated with DCE-Web's Secure Local Proxy (SLP). The SLP used a new API in the Wand Server to actually perform the management functions. The functions provided were basic: rules and labels each had create, delete, view, and modify functions. The following subsection discusses MAP design in detail. The subsequent subsection summarizes the advantages and disadvantages of the MAP work.

A. MAP: User-Centered Authorization for DCE-Web

ACLs allow users to create access policies for individual objects by specifying lists of principals who can access the object and what kinds of access are permitted. However, ACLs fail to provide the mechanisms needed to categorize and group information objects throughout the namespace or to apply ranked levels of trust to principals. These criteria are often the ones used to express security policies. In addition, ACLs do not have any generally accepted well-defined semantics. Every system is different, and this inconsistency makes using ACLs difficult when transferring between different systems or working in a heterogeneous environment.

The Distributed Computing Environment (DCE) [23] allows groups to be defined for users and stores the group memberships in a user's Privilege Attribute Certificate (PAC); these groups are used in ACLs to make authorization decisions and so are analogous to clearances or user labels. However, no similar group memberships (analogous to sensitivity labels) are defined for objects in DCE.

MAP added the following capabilities to the DCE-Web authorization engine:

- the ability to define labels that can be applied to objects. The object labels are lists of existing DCE groups and perform two functions:
  - they identify the group memberships of the objects.
  - they act as sensitivity labels for the data in the objects. Sensitivity labels say what kind of information is in an object and how or in what way it is sensitive. They are a user-defined measure of the damage incurred if the information is revealed or corrupted.
- the ability to define rules that grant or deny access based on the relationship between the principal's label (the list of groups from the PAC) and the object's label. Example rules are in the Results subsection below.

1) Object Labels

Object labels are lists of DCE groups. The object labels may be applied to single objects or to all objects in one or more branches of the object namespace. For example, all the objects in "/projects/DCE-Web" might be labelled "ATO-Confidential," while work in progress in "/projects/DCE-Web/snapshot" might have the additional label "DCE-Web-team." The branches covered by different labels may overlap so objects may have more than one label applied to them. The effective group membership for an object is the union of all the groups in all the labels applied to the object (with redundancies removed). We will refer to the effective group membership of an object as "the object's groups."

2) Rules

Rules contain a list of groups and a relationship. Rules are applied to objects in the same way labels are; rules may be applied to single objects or to all objects in one or more branches of the object namespace. These branches can overlap so objects may have more than one rule applied to them. The access control policy that applies to a given object is the conjunction of all rules that apply to the object; every rule must be satisfied for access to be granted.

The list of groups in a rule determines the groups to which the rule applies. Generally, if the intersection of the groups in a rule and the groups to which an object belongs is empty, then the rule is not used to determine access to that object. The single exception to this is if the list of groups in a rule is empty (in other words, there are no groups defined for the rule); in that case, the rule is interpreted as applying to all the groups to which the object belongs, rather than to no groups. This awkward syntax was motivated by our use of standard DCE group semantics. It allowed us to construct rules that refer to all of an object's groups, whatever they might be.

There are two kinds of rules defined by MAP: **set** rules and **range** rules. The two kinds of rules treat the group memberships differently.

Set rules. Set rules support the standard notion of groups and roles. In a set rule, the groups are treated as an unordered list. The relationships that can be required are:

- AND - The principal's label must contain all groups in the intersection of the rule's groups and the object's groups. So, if the rule specifies groups (G1, G2, G3) and the object's groups are (G1, G3) then the principal's label must contain both G1 and G3.
- OR - The principal's label must contain at least one of the groups in the intersection of the rule's groups and the object's groups. Using the example from the preceding bullet, the principal's label would have to contain either G1 or G3.

Range rules. Range rules support the notion of levels in security policies. In a range rule, the groups are treated as an ordered list. The ordering is defined by the order in which the group names are input at rule creation time, much like the way the parts of an enumeration type are assigned values in C or C++. This means that group names do not have a predetermined numeric value, and so could have different list positions and so different values in different rules. This potential source of inconsistency came from our decision to use unaltered DCE groups. The list of groups in a range rule may not be empty.

Range rules work by computing and comparing sub-ranges of group lists. Given a group ordering defined by a specific rule, two intersections are computed: the intersection between the rule's groups and the object's groups, and the intersection between the rule's groups and the groups in the principal's label. If the intersection between the rule's groups and the object's groups is empty, then the rule is not applied. Non-empty intersections are interpreted as a sub-range of the groups in the rule's ordered list of groups. For example, if a range rule defines the group order (G1, G2, G3, G4, G5) and the intersection of this list with the object's groups is (G2, G4), then the rule is applied to the sub-range (G2, G3, G4), the G3 "gap" being ignored.

Range rules allow the following relationships to be used:

- >= The high end of the principal's sub-range is greater than or equal to the high end of the object's sub-range
- <= The low end of the principal's sub-range is greater than or equal to the low end of the object's sub-range
- = The principal's sub-range is identical to the object's sub-range
- SUBSET - The principal's sub-range fully encompasses the object's sub-range
- SUPERSET - The principal's sub-range is a sub-range of the object's sub-range

B. Results

The MAP prototype confirmed the potential of the user-centered approach while pointing out pitfalls that should be avoided in the Adage system.

1) Advantages

Flexibility. Using a rules-based system allows enormous flexibility in the kinds of policies that can be defined and enforced, while allowing those policies to be expressed with tools more powerful and user-friendly than ACLs. For example, a common Multi-Level Secure (MLS) policy is based on the Bell and LaPadula model [3] and requires the Simple Security Property (read up) and the *-Property (write down) to be enforced. This policy can be easily expressed by defining groups to represent the various security classes and then rules that represent the two access properties. For example, suppose the groups Unclassified, Confidential, Secret, and TopSecret are defined. If both properties are enforced for all objects, then this could be represented as a pair of server-wide range rules at the root level of the server namespace, one for read access and one for write access, such as (owner information has been omitted from the table):

Name	Scope	Relation	Perm	Groups
Simple Security	/	>=	R	Unclassified, Confidential, Secret, TopSecret
Star Property	/	<=	W	Unclassified, Confidential, Secret, TopSecret

The Biba integrity model could also be enforced by adding two similar rules. Furthermore, the portion of the namespace protected by each policy can be controlled by setting the scope of the rules accordingly. So, objects in one branch could be protected by these strict security policies, while others in a different branch were left more open.This flexibility can overcome problems with inflexible models such as Role Based Access Control (RBAC) [9]. Ferraiolo et. al point out that RBAC will not work for all objects in

the system (such as temp files), but offer no solution to this problem.

Accessible Graphical User Interface. Extending the DCE-Web Secure Local Proxy (SLP) let us use World Wide Web HTML forms as our user interface. This forms interface is familiar to most users and accessible through widely deployed Web browsers. In addition, it was easy to write forms to allow the creation and viewing of a label or a rule. The existence of the SLP gave us an infrastructure for serving our forms and invoking DCE, saving development time and providing an integrated view of the Web for our users.

Object Groups and Labels. Allowing objects to be labeled and collected into groups fixed a basic asymmetry between how principals and objects were treated in DCE. Labeling and grouping objects according to sensitivity or function is a natural thing to do, and relying on namespace partitions to accomplish this (by placing different kinds of objects in different places) is unnecessarily restrictive and difficult to manage. Explicitly adding object groups and labels allows objects with similar labels to be protected similarly no matter where they are.

2) Pitfalls

Attachment of Labels and Rules to Namespace. The ways that access was computed in the MAP prototype were complex and often obscured exactly how any given object was protected. This difficulty stemmed from the fact that the protections on an object were ultimately defined by its position in the namespace rather than by the kind of object it was. While it is occasionally useful to construct parts of the namespace to reflect the security policy, requiring this tight coupling is too constraining. This difficulty showed up in several ways:

- It was not possible to specify exceptions to rules within a namespace branch. Another way to say this is that the granularity of protection was not fine enough. Because rules were attached to the namespace, all objects beneath a rule's point of attachment were governed by the rule. There was no easy way to alter the policy governing some subset of objects in that branch. If the connection between rules and the namespace were broken, then finer grained control could be obtained more easily (by having the scope of rules be governed by the labels on objects no matter where they were, rather than by their position in the namespace, for example).

- It was hard to know exactly in which groups a given object was and what rules would be applied in any given access decision. Part of this problem could be solved with facilities designed to present this information (see next pitfall). The tight coupling between label and rule scope and object namespace forced users to specify this information in a fragmented and sometimes unintuitive way.

The lesson here is that object labels and groups and the rules that protect an object should be determined purely by what kind of object something is and not by where it is. Security policy is naturally organized around information content, not namespace placement.

Inadequate Interface. An engine like MAP is designed as a basis for a higher-level user interface. The MAP user interface was good enough for prototype testing, but it was clear that a simple forms interface would be inadequate for a full system. The forms interface was awkward to use and failed to give the user sufficient context to know if the information entered was correct or not. In particular there was no way to see a high-level view of rules and labels for a given principal, object, or group. This made it hard to know exactly what effect a new rule might have.

Clarity of effect is crucial to the implementation of a coherent security policy. Even with a design that offers clear relationships among rules, labels, and groups, the overall scheme of protection that is actually in force must be easy to perceive. This can only be done through a carefully thought-out GUI that facilitates rule, label, and group definition and perception of the overall protection structure that is in force. The minimal requirements on this GUI would be:

- Integrated management of rules, labels, and groups
- Convenient data entry of complex structures like rules
- Consistency checking between current rules and ones being entered
- High-level overviews of rules, labels, and groups and their effects
- Convenient querying of current policy constraints (how is a given object or group of objects protected?)

User-centered security is as much about user interface as about security mechanisms. A coherent, consistent GUI is itself a security tool, not just window dressing.

Complex Internal Structure. One final area that we felt needed improvement was the complexity of the actual data structures used to implement the design (see the appendix for details). While not user-visible, these are important because it made the code hard to work with and (we conjecture) would make a production system built along the same lines difficult to debug and maintain.

The main problems were the visibility (to the programmer) of the complex structures used to hold the rule and label information. C has little provision for encapsulation and data hiding, and it was easy to make mistakes in the way the structures were referenced. Bugs often had to be tracked back through several levels of pointers and structure fields.

We believe that these problems can be alleviated by using an object-oriented design approach and implementation language. The complex structures required to implement rules, labels, and groups seem to be a good match for encapsulation and data hiding. The Adage system will be written in C++ and effort will be put into the design to limit the areas of code where the internals are visible.

V. Future Work

We are continuing to work on our vision of user-centered security in the Adage project (Authorization for Distributed Applications and Groups) [29]. We are in the process of designing and implementing Adage, so there are no results to report at this time. Adage is specifically conceived to overcome the usability problems with authorization mechanisms for distributed applications in use today. The first of these usability problems is that the applica-

tions unnecessarily export the underlying data structure as the user model. The user metaphor for Access Control Lists (ACLs) is the ACL data structure; for system masks it is the system mask. The user is given a rudimentary formatted display of the information in the data structure (or perhaps just a literal display of its values) and must learn the algorithm that the computer software will use to evaluate that data structure in order to understand what access control policy is actually instantiated. This problem is starting to be addressed. GUI ACL editors provide a simplified display, graphics, and contextual help. Some even support rudimentary user queries about the access control policy, such as "What is my access to this object?" and "What is user X's access to this object?" [8]

A large gap remains between security mechanisms and a user's or site's security policy, stated in natural language. By analogy, ACLs are the assembly language of security policy. They are a complex, low-level language. Only an expert in a particular implementation of ACLs can hope to program it correctly the first time. ACLs have the added disadvantage of being difficult to test without making changes on a live system. One component of Adage will be a higher-level authorization language that begins to close the gap between security mechanisms and site security policies. It will come with a visual builder that allows site security administrators to build up an authorization policy from visible policy pieces. Furthermore, these policies can be shared with other domains. The primitives supported by this language will support a wide range of user and application policies, because they will be based on security policies actually in use [2][19] and on interviews with security administrators.

One insight that Adage shares with current work on roles is that within organizations it is natural to think about both users and objects in terms of how they relate to each other and what place they fill within the organizational structure. Adage will use groupings to reflect these intuitions. It will use groupings of objects and of actions to more easily refer to objects and actions in a security policy. Groups of users and their roles will receive particular attention. Adage will provide an infrastructure for defining the relationships and restrictions on groups and roles that will allow it to support models from both the security and groupware literature. For example, two groups can be restricted to have no membership overlap, to support static separation of duty in policies such as [19]. Users taking on the role of Chair can be restricted to those users in a particular group.

Adage will continue the work in user-centered trust models by modeling common trust dimensions such as amount of trust (How much do I trust you? How much do I distrust you?) and type of trust (What do I trust you for?). Adage will apply this trust model to services whose information is used as input to authorization decisions (such as authentication servers and group membership servers). This will allow an enterprise to articulate a trust policy and have it apply to all its authorization decisions. In addition, the model will allow trusted services to introduce other trusted services, forming chains of trust where the amount of trust degrades over hops, much as real-life trust does.

VI. Conclusions

When considering the security of systems and applications in their context of use, it is clear that the security mechanisms need to be appropriately used to maintain their effectiveness. Mechanisms and models that are confusing to the user will be misused. Additionally, in contexts such as the home market where the user makes the buy decision about all software, security applications that are difficult to use are unlikely to be deployed. Therefore, this paper considers user-centered security as an appropriate goal for secure systems. We have reflected on the usability problems of secure systems in the past, and provide three categories for work in user-friendly security:

- Applying usability testing and techniques to secure systems
- Developing security models and mechanisms for user-friendly systems (such as groupware)
- Considering user needs as a primary design goal at the start of secure system development

We gathered together the work in usable secure systems from the security and CHI literature as an aid to future development, and sorted it into these categories. We believe the third category will yield the richest developments and we are following that approach in our Adage work. We discussed our early prototype user-centered authorization engine and our current direction towards a user-centered authorization language and trust model. We found that a rule-based authorization engine provides the flexibility to support user-centered authorization, but more work is needed on the interface and concepts presented to the user. We hope to hear of other work in user-centered security, as we expect the need for user-friendly security to grow more acute over time.

Acknowledgments

The MAP work was funded by the DoD. The Adage work is funded by ARPA under contract F30602-95-C-0293 (This paper is Approved for Public Release Distribution Unlimited). This paper has benefited from feedback from Marty Hurley, Rich Salz, Howard Melman, Charlie Brooks, Keith Loepere, Scott Meeks, and our anonymous reviewers.

Bibliography

[1] Berners-Lee, Tim, Robert Cailiau, Ari Luotonen, Henrik Frystk Nielsen, and Arthur Secret. "The World-Wide Web", Communications of the ACM, August 1994

[2] Baldwin, Robert W. "Naming and Grouping Privileges to Simplify Security", in Proceedings of the 1990 IEEE Symposium on Security and Privacy, pages 116-132, 1990

[3] Bell, D. E. and L. J. LaPadula. Secure Computer Systems: Unified Exposition and Multics, Technical Report ESD-TR-75-306, The MITRE Corp., March 1976.

[4] Blaze, Matt, Joan Feigenbaum, and Jack Lacy. "Decentralized Trust Management," in Proceedings of IEEE Conference on Security and Privacy, 1996.

[5] Clark, David D. and David R. Wilson. "A Comparison of Commercial and Military Computer Security Policies", in Proceedings of the 1987 IEEE Symposium on Security and Privacy, pages 184-195, April 1987.

[6] Clement, Andrew. "Privacy Considerations in CSCW," in Proceedings of CSCW '92.

[7] DCE-Web home page, http://www.osf.org/www/dceweb/index.html

[8] Digital Equipment Corporation. Visual ACL Editor Online Help. Available: Digital Equipment Corporation, Littleton, MA. 1995.

[9] Ferraiolo, Janet A. Cugini, and D. Richard Kuhn, "Role-Based Access Control (RBAC): Features and Motivation," in Proceedings of Eleventh Annual Computer Security Applications Conference, December 11-15, 1995.

[10] Fish, Robert S. and Robert E. Kraut. "Networking for Collaboration: Video Telephony and Media Conferencing," in Proceedings of ACM CHI '94.

[11] Foley, Simon and Jeremy Jacob. "Specifying Security for CSCW Systems," in Proceedings of 8th IEEE Computer Security Foundations Workshop, June 13-15, 1995.

[12] Garfinkel, Simson. PGP: Pretty Good Privacy, O'Reilly and Associates, Inc., 1995.

[13] Gasser, Morrie. "Building a Secure Computer System," Van Nostrand Reinhold Company, New York, 1988.

[14] Hill, Will and Loren Terveen. "New Uses and Abuses of Interaction History: Help Form the Research Agenda," in Proceedings of ACM CHI '94.

[15] Karat, Clare-Marie. "Iterative Usability Testing of a Security Application," Proceedings of the Human Factors Society 33rd Annual Meeting, 1989.

[16] Kling, Rob. "Controversies About Privacy and Open Information in CSCW," in Proceedings of CSCW '92.

[17] Lewontin, S. "The DCE-Web: Securing the Enterprise Web," http://www.osf.org/www/dceweb/papers/Secure_Enterprise.html, 1995.

[18] Linn, J. "Privacy-Enhanced Electronic Mail: From Architecture to Implementation", in Proceedings, IFIP TC11 Seventh International Conference on Information Security (IFIP/Sec '91), Brighton, UK, 15-17 May 1991.

[19] Nash, M. and Poland, K., "Some Conundrums Concerning Separation of Duty," in Proceedings of 1990 IEEE Symposium on Security and Privacy, May 1990.

[20] Norman, Donald A., "The Design of Everyday Things", Doubleday, 1988.

[21] Mosteller, William S. and James Ballas. "Usability Analysis of Messages from a Security System," Proceedings of the Human Factors Society 33rd Annual Meeting, 1989.

[22] Nielsen, Jacob. "Usability Engineering," AP Professional, 1995.

[23] Rosenberry, Ward, David Kenney and Gerry Fisher. "Understanding DCE," O'Reilly & Associates, Inc., 1992.

[24] Rubinstein, Richard and Harry Hersh, "The Human Factor: Designing Computer Systems for People," Digital Press, 1984.

[25] Saltzer, Jerome H. and Michael D. Schroeder. "The Protection of Information in Computer Systems", in Proceedings of the IEEE, 63(9), 1975.

[26] Shen, HongHai and Prasun Dewan. "Access Control for Collaborative Environments," Proceedings of CSCW '92.

[27] Wixon, Dennis, Karen Holtzblatt, and Stephen Knox. "Contextual Design: An Emergent View of System Design", in CHI '90 Conference Proceedings.

[28] Zurko, Mary Ellen. "Attribute Support for Inter-Domain Use", in Proceedings of The Computer Security Foundations Workshop V, June 1992.

[29] Zurko, Mary Ellen. Adage home page, http://www.osf.org/www/adage/index.html.

[30] Zurko, Mary Ellen. MAP home page, http://www.osf.org/www/map/index.html.

A. MAP Data Definitions

This section contains the DCE Interface Data Language (IDL) definitions of the MAP rules and labels.

The rule structure had five fields:

```
typedef struct map_rule_s_t
{
mapname        *name;
/* the name of the rule */
URL_list_t     *scope;
/* where the rule should be
applied */
user_or_group  owner_type;
/* owned by a principal or
group? */
uuid_t           owner;
rule_body_t      body;
} map_rule_t;
```

The body of the rule was itself a structure that contained:

```
typedef struct rule_body_s_t
{
relation_t    relationship;
/* such as AND, OR, <=,
   etc.*/
sec_acl_permset_t
   permissions;
/* permission granted */
group_list_t
    *groups;
/* groups covered by the
   rule */
} rule_body_t;
```

The label structure contained:

```
typedef struct map_label_s_t
{
mapname        *name;
URL_list_t     *scope;
/* where the label applies*/
user_or_group  owner_type;
/* owned by a principal or
   group? */
uuid_t           owner;
group_list_t   *groups;
} map_label_t;
```