Attribute Support for Inter-Domain Use

Mary Ellen Zurko
Laboratory for Computer Science
Massachusetts Institute of Technology and
Digital Equipment Corporation
Littleton, MA, 01460

Abstract

This paper describes the User Attribute Service (UAS), a tool providing the storage and management of application-specific per-user security attributes for applications running in a distributed environment. The UAS provides for the security and integrity of attribute-to-user bindings, as well as the secrecy of those bindings, if the application or user requests it. Four goals of the UAS are support of Least Privilege, local control and autonomy, instantiation of trust relationships, and psychological acceptability. Mechanisms to group and enable privilege attributes support the Least Privilege principal at the user request level. Functions are designed to enhance the usability of the UAS within and across domains by attribute holders and security managers.

1 Introduction

Due to the limited pool of trust technology engineers [NRC91], there is a need for building blocks or tools to support authorization in a distributed world. Our research concentrates on the use of attributes by distributed applications. Most applications associate information, called *attributes*, with user identities. Attributes provide a way for applications to rapidly identify classes of users. Attributes defined by an application are termed *application-specific*. Attributes associated with a user's identity are termed *per-user attributes*. Attributes used as input to authorization decisions concerning a user are called *security attributes*. This paper will discuss application-specific per-user security attributes, and will use the simple term attributes to refer to them.

1.1 Distributed World Model

In our basic model of the world, application instances (called *applications* throughout) use attributes in their authorization decisions. Applications are associated with their attributes via a one-to-many mapping. Applications are grouped into domains, so that the universe of application is partitioned into a set of domains. Each domain has a security manager, who manages the attributes in the domain. Managing the attributes entails maintaining the application-to-attributes association, and granting attributes to principals, for them to use within the domain. The association between principals and attributes is many-to-many. Management and use of any attribute is bounded by its domain.

When this model is fleshed out with a detailed design, each principal is identified via an authentication identity. All the authentication identities of a single user, and the attributes associated with the identities, are grouped together to represent the information applicable to a single user within a domain. Users are expected to call applications in multiple domains, and so may hold attributes in multiple domains. The principals for a user may be authenticated by diverse authentication authorities. The spheres of the influence of authentication authorities are orthogonal to the domains that we use to group and manage attributes.

The User Attribute Service (UAS) is the tool which provides the definition, storage, and management for application-specific per-user security-relevant attributes in a distributed environment. It provides the necessary functions for use by three user populations: applications, attribute holders, and attribute managers. This paper focuses on the aspects of the UAS designed to support use between the domains of distributed applications by attribute holders and attribute security managers.

0-8186-2850-2/92 $3.00 © 1992 IEEE

1.2 Goals

There are four main goals of the attribute support in this paper: Least Privilege, local control and autonomy, explicit instantiation of trust relationships, and psychological acceptability. We discuss each in turn below.

1.2.1 Least Privilege

Some attributes consistently authorize the user to perform actions otherwise forbidden. These attributes are called *privileges*. We call attributes which are used solely to bar their holder from particular actions *prohibitions*. Attributes may be of either, or both, types. The Principle of Least Privilege [SS75] states that users should only have available sufficient privilege to accomplish the task at hand, to reduce the damage caused by user error or malfunctioning software. The UAS provides support for this principle by allowing users to disable their privileges, while forbidding them to do so with their prohibitions.

Giving users the ability to enable and disable privileges also allows users to exert some control over what is being said in their behalf. However, this ability increases the burden on the user to use the mechanisms appropriately. Mechanisms to enhance usability can ameliorate some of this burden.

1.2.2 Local Control and Autonomy

We provide two abstractions for local control and autonomy. One, as discussed above, is the enabling and disabling of privileges. Our other abstraction, domains, provides for local control and autonomy for applications and their security manager. A domain contains a set of systems which are willing trust a single authority for the infrastructure needed to manage their Automated Security Policies [Ste91]. A domain might consist of the systems managed by a cost center in a company, or a laboratory in a university. These organizations may have more than one person responsible for implementing all or part of the security policy, and only have consistent Organizational Security Policies [Ste91], which define high level issues such as user privacy and the identification and protection of sensitive information.

In allowing user-to-user delegation of authority in our system, we consider the conflicting need for local control of attribute holders and security managers. For example, the same person may take on the role of both group manager and intimate friend. While many applications or object owners would allow a different person standing in as group manager to perform the actions for a group manager, those relying on the accessor being their intimate friend will not wish to allow delegation. Generally, some privileges and prohibitions are tightly bound to the identity of the holder. We will see one example of a security policy where attributes must not be delegatable, for that reason.

1.2.3 Trust Relationships

We divide trust (or distrust) relationships, which require explicit instantiation, into two categories based on the relationship between the authorities involved. The first involves complementary (or cooperating) authorities, where one authority must rely on another to accomplish its job. For example, the attribute manager must rely on an authentication authority to identify a principal. Different levels of trust may be associated with each authentication authority, either by some universally recognized objective measure, or by each attribute manager individually. How much an attribute manager trusts an authentication authority will be reflected in how much power (in the form of attributes) a principal authenticated by the authority is allowed.

Similarly, the user must rely on an authentication authority to provide her with an authentication identity. Having more local knowledge, users may have different ideas about how much they trust their authenticators.

The second relationship involves similar authorities, where one authority wishes to rely on another to accomplish the activity entrusted to it. User-to-user delegation, as mentioned above, is one form. Another is allowing an attribute manager to offload some portion of its job to another attribute manager, in another domain (the issue of redundancy within domains is not considered in this paper). This trust is best conferred explicitly, in part so that it can be exhibited to security managers and users, to make the security policy visible.

1.2.4 Psychological Acceptability

The final goal, psychological acceptability [SS75], is approached via user-centered design of security functions and interfaces. It is a prerequisite to the appropriate and secure use of the security features by humans and their artifacts. Attributes themselves provide a way for applications to provide a mnemonic name for an aspect of their authorization decisions, making application security decisions more obvious to both users and security managers. We also provide

tools to facilitate user customization and help ease the tension between control and confusion. In that context, we explore some simple sharing mechanisms.

Grouping objects can also help users make sense out of complex worlds. Our attribute management service provides a variety of grouping mechanisms: users grouped by attribute, users can group privileges to enable, authentication identities are grouped in a record identifying the user who they represent, attributes are grouped by their application, and domains group applications' security policies.

By grouping security policy information within domains, the user can interact with a limited number of well-defined places for their security information. While the domain is a useful unit for security management, users in a distributed system will also work across domains. The mechanism that groups enabled privileges allows users to do so by task, across domains.

1.3 Structure of This Paper

For the examples in this paper, we consider a set of distributed applications implementing a purchasing task. Purchasing an item consists of three steps or procedures: authorizing a purchase order, recording that the item has arrived, and authorizing payment. Authorizing payment is itself broken down into two procedures: writing the check and updating the cash account. Since our example is a distributed system, each of the above steps may actually run in a different domain. In fact, some steps will run in more than one domain.

The applications supporting the purchasing example use the Clark and Wilson model [CW87] for their access control decisions. The Clark and Wilson model calls for a certifier for each procedure (called a Transformation Procedure or TP), who verifies that the TP maintains the integrity of data items (called Constrained Data Items or CDIs). Certifiers also grant the ability to run their TPs on particular CDIs. Certifiers are not allowed to run the TPs they verify.

In the next section, we give an overview of UAS functions, and the communication and key management protocol it uses. We then discuss associating a list of authentication identities with a user's record, a basic grouping mechanism. Next we discuss restricting the use of attributes to particular authentication authorities; offloading the management of attributes to a UAS in a foreign domain; and restricting the use of privilege attributes to particular authentication identities and augmenting the prohibitions associated with identities. These abilities instantiate trust relationships between the authorities involved. Finally we discuss defining Named Attribute Sets which span domains; and delegating attributes between users.

2 Overview of UAS Functions

Each UAS provides attribute storage and management for the set of instantiations of applications which define its domain. The responsibilities of the security manager are divided into two roles. General UAS operations such as UAS installation, new application installation, and creating user records are performed by a *UAS security manager*. Applications must trust the UAS and its manager[s], since they can subvert application security by redefining the applications attributes, and thus altering the database of users and their attributes. In addition, the UAS allows applications to specify *attribute security managers* at a fine granularity. Attribute security managers grant, revoke, and view the attributes they manage.

The attribute management tool must maintain the integrity of the association of attributes to user identities and, when desired, maintain the privacy of application and user information. Attribute holders can group attributes with *Named Attribute Sets* (NASs), and enable and disable the privilege attributes in those groups. This feature is transparent to the applications; to them, a user either currently holds or does not hold a privilege.

Attributes may also constrain the user from taking particular actions. Attributes of this type are called *prohibitions* in this paper. They cannot be disabled by their holders. Not all attributes are purely of one type or the other. The UAS provides for *binatured attributes*, which the application uses as either a privilege or a prohibition, depending on its policy and the context of its use. When an application inquires about a user's attributes, a binatured attribute will always appear in the list of the user's prohibitions, and will appear in the list of the user's privileges if it is enabled.

2.1 User Requests

NASs are associated with a user request by the inclusion of a signed Enabled NAS Token (ENT) as part of the request. An ENT contains:

- user authentication identity
- unique request ID
- timeout — a suitably long value makes this almost optional.

- NAS[s] enabled (may be empty) — the intersection of the privilege attributes the user holds and the privilege attributes in the NASs are considered enabled for the request.

The ENT is signed by the user's session or delegation key, and sent as part of the request to which it applies. When an application receives a request, it passes the request's ENT to its local UAS as part of a call asking to view the attribute holder's attributes for that application. The application decides when to call the UAS for user attribute information. Requests can be batched for efficiency, if it is appropriate.

2.2 UAS Communication and Keys

UAS communication requires the options of authentication, privacy, and integrity. UAS communication must be integrity protected and authenticated to users and applications (and occasionally other UASs) so that they may trust the attribute information provided, and trust that the information in the request is the information the UAS put there. Node or application authentication, such as provided by DSSA [GGKL90] or Kerberos tickets [KN91], is required for UAS authentication.

In return, users, and some applications, must be authenticated to the UAS. The UAS trusts its local authentication authority to verify the authentication of principals. Each authentication authority will provide this information via a secure channel, such as a certificate encrypted with the authority's private key. The security of the channel depends on the strength of the encryption algorithm and key management.

The privacy of requests to UASs is assured via privacy-encrypted channels. Applications that require confidentiality of their attribute must be able to communicate via these channels. Applications may use their node's authentication identity as their own. Session keys will be provided by the principal initiating the request.

The integrity of all information returned by the UAS, to both users (attribute holders and attribute security managers) and applications, is protected by a hash, such as a Message Authentication Code (MAC), signed by the UAS. Both DSSA and Kerberos provide such services to assure the integrity of communications.

3 Associating Principals With a User Record

In a distributed system, a single user may have multiple authentication identities. While in some cases a user may have no need for more than one, their multiplicity is due to the lack of a single, global authentication authority. We expect this to continue to be the case, particularly with individuals associated with multiple organizations. The UAS allows multiple authentication identities to be associated with a single user record. This feature supports consistent attributes among a user's identities, which enhances the transparency of a user's security management. If no further restrictions are applied (see below), a user may login from any of her authentication identities, and run the same tasks in the same manner. Any of a user's identities can be used to access information about the user's attributes.

This feature also allows an attribute security manager to grant privileges and prohibitions to all of the user's identities with a single request. Adding a new authentication identity to a user record grants all the attributes of that user record to that principal.

Adding and removing authentication identities and user records can only be accomplished by the UAS security manager. If the UAS could check that a new authentication identity belonged to the same user as all other authentication identities in a user record, anyone could add an identity to a user record without damage. Such a check might be based on all authentication authorities having a standardized format for the person's full legal name and birthdate and place. However, since adding an identity grants all the record's privileges to it, it is an extremely powerful and currently uncheckable operation. As such, it is restricted to the UAS's security managers. This is not the most useful and usable option, since a local UAS security manager cannot be expected to know a priori if a principal represents a particular user. She may have to contact the foreign authentication authority's manager to verify the information.

Removing an identity can do damage if an application applies fewer or different prohibitions to a principal unknown to its UAS than those associated with some principal known to its UAS. This is the case with the Chinese Wall model [BN89], where a user who has read no information is not barred from reading any information, while a user that has accessed information has restrictions. For security and consistency, removing, like adding, is restricted to the UAS security manager.

Once an authentication identity is added to a user record, it is known to a UAS, and the UAS will respond to application queries about that identity. This may occur before the attribute security managers have a chance to grant attributes, including prohibitions. This may cause a window when a new user record is created with its first identity, and it is not sufficiently restricted via prohibitions. To close this window, the UAS provides a default template for all new user records created by the UAS security manager. Attribute security managers can give this template each application's most restrictive prohibitions, so that no new user has any access she should not have before the attribute security manager is able to set up her attributes properly.

Some sites may also want to recognize all of a user's authentication identities as that user under a single consistent name. This can be accomplished by associating a pseudo-identity attribute with a user record, and associating all of a user's authentication identities with that record. The pseudo-identity can be checked as if it were the user's identity. It would be fed to the application for the purposes of authorization. Such an identity should not be used for auditing or accountability. Since the site would probably not want to allow the user to disable the pseudo-identity, it would be a prohibition or binatured attribute.

4 Restricting Attributes by Authentication Authority

The user record grouping, which associates all the attributes a user holds with all principals that represent the user, is useful for security management. In some cases, an attribute security manager may want to restrict a privilege attribute's use to the most trusted authentication authorities, or to associate a prohibition attribute with particular authentication authorities with loose security management.

In order to discourage fragmentation of a user's identities and attributes, the UAS allows an attribute security manager to restrict an attribute's use by authentication authorities. Authentication authority is the only level of granularity consistently provided by current distributed authentication services. These authorities may be DSSA Certification Authorities, or Kerberos servers. Attribute security managers can restrict particularly powerful privileges to principals authenticated by authorities with a high degree of trust.

A similar feature in ECMA is the ability to designate an authentication as strong or weak [ECM91]. However, attribute security managers might not want to rely on an authentication authority they view as untrustworthy to tell them they have provided strong authentication. Per-authority restriction is not a substitute for authentication strength testing, since a single authority may allow both passwords and smart cards. However, an attribute security manager may take a wider range of information into account when designating the authorities to which an attribute applies, including the quality of the security tools and security management at that site.

When an application asks about the privileges enabled by an ENT, the list of privileges returned is filtered by the UAS based on the authentication authority of the authentication identity in the ENT. Only privileges which may be used from that authority are included. Prohibitions, instead of being explicitly included in a list of authorities, are explicitly exempted from a list of authorities. If an authentication identity from an authority previously unknown to the attribute security manager is added to a user's record, the default will be fail-safe; the prohibition will apply. A prohibition not valid for the ENT's authority is not returned to the application (unless a mechanism to allow users to override this is provided). A binatured attribute require two such lists; one from which it may be exercised as a privilege, and one from which it is excepted as a prohibition.

The authentication authority of an ENT is determined in the process of checking the signature on it. In the case of DSSA public delegation keys, the principal's long-term public key vouches for the principal's short-term delegation key, and the principal's Certification Authority vouches for its long-term key.

5 Trusting Foreign UASs to Manage an Attribute

While the sphere of a UAS is a single domain, the definition of the scope of the domain of a UAS is extremely flexible. It is composed of those applications which call a particular UAS for their attributes. The number of applications can be very large or very small, and concentrated or dispersed. Two applications on the same node may use two different UASs. Performance considerations will probably cause most UASs to be "close" to their applications, when measured in terms of network performance. Several domains may want to centralize and share the management of one or more attributes. The ability to refer to a particular attribute and its holders in a foreign UAS allows or-

ganizations to place limited trust in a shared service for very specific attribute management.

This remote reference is accomplished with an *Attribute Translation*. The attribute translation identifies which attributes in a UAS are actually managed by a foreign UAS. Since the local UAS transfers the security management of the attributes with translations to a foreign UAS, only a local attribute security manager can provide or remove its translation. The scope of trust is well defined; for each translated attribute, a particular UAS is trusted to provide information about it for local use. The UAS named in the translation defines who holds the attribute, as well as other attribute constraints such as conflicting attributes and authentication authority restrictions. For a particular attribute, the foreign UAS which manages it may in turn put that attribute on its Attribute Translation List. Cycles are illegal and disallowed.

6 Associating Attributes with Authentication Identity

Attribute holders may further restrict the power of one of their principals by restricting the privileges or augmenting the prohibitions associated with its authentication identity. For each privilege attribute a user holds, the user may specify a subset of the user's authentication identities which may enable that privilege. This prevents malfunctioning authentication code from a not-very trustworthy domain from producing an ENT which enables extremely powerful privileges. It may also protect the user from mistakes. The request to extend or shrink the list of authentication identities that can enable a privilege attribute must come from a principal whose authentication identity may already enable it. The UAS does not allow an empty list.

Users may also specify authentication identities associated with a prohibition they hold. This feature has more limited usefulness. Prohibitions are associated with all of the user's identities except for the ones from authentication domains exempted from that prohibition by the attribute's security manager. Attribute holders may override this exemption by specifying those identities in the list of identities associated with the prohibition. This may be confusing to the user unless carefully presented because, unlike privileges, prohibitions are associated with authentication identities **not** on the list as well as with those on the list. If a static coupling existed between authentication identities and authentication authorities, the UAS could maintain an exact list of the identities to which each prohibition applies. Because the coupling between authentication identities and authorities is loose, this feature may be confusing and/or practically unused. Binatured attributes have two lists, one for privilege identities and one for prohibition identities.

7 Named Attribute Sets Across Domains

Named Attribute Sets (NASs) are used to group and enable attributes across domains as well as across applications. They provide the distributed task-level grouping for the user. An NAS may hold attributes for one or more applications. They also support Least Privilege by allowing the user to only enable the privileges needed by the current request.

An NAS in a UAS contains a group of attributes in that domain. Any user may create one. An NAS in a UAS contains the following information:

- name of owner
- users of the NAS
- attributes locally associated with that NAS

Attributes in a new domain are added to an NAS by producing a two-way link between the new piece and any other domain that contains a piece of the NAS. Thus, an NAS can be thought of as a doubly linked tree between UASs, where each of the *nodes* of the tree is the portion of the NAS in that domain. We will refer to the piece of an NAS contained in a single UAS as a node. The two-way links allow operations on an entire NAS to be started at any domain which contains one of its nodes.

An additional field is needed in the NAS node definition. An NAS node also contains:

- neighbor UASs in the NAS

Figure 1 provides a simplified illustration of this structure. It shows only one attribute per domain, all attributes of the same type, and no other users of the NAS besides the owner. The owner and attribute fields of the NAS are represented textually, while the neighbors of each node are represented by arrows. More complex NASs will have a variety of attributes in each domain.

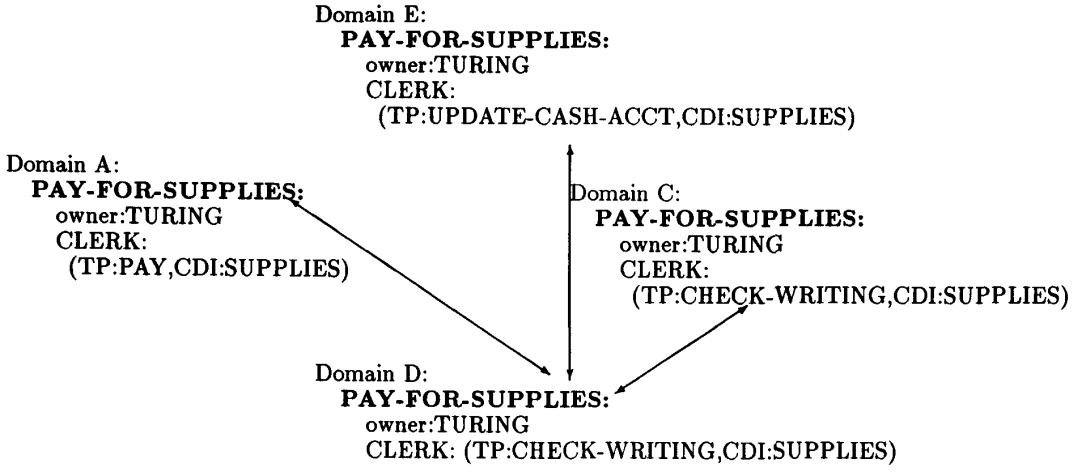

Figure 1: PAY-FOR-SUPPLIES NAS

7.1 Cross-domain NAS Operations

Users can view the contents of any NAS they can use by issuing a request to any UAS containing a part of it. A simple tree walk, using the node at that UAS as the root, finds all attributes in an NAS, if all relevant UASs are accessible. If some UAS containing part of the NAS is not accessible, any branch of the NAS defined through that domain is truncated, and cannot be viewed. This information is reported to the user. An entire NAS can be deleted or revoked (see below) with a single request, via a tree walk.

An NAS, named in an ENT in a request that is still being processed, can be revoked from that request. Revoking an NAS from a request has the effect of revoking the privileges in that NAS, unless any of the privileges in the revoked NAS are named in another NAS in the same ENT. The user must be able to name the request ID and authentication identity in her ENT to revoke an NAS from it. This can be facilitated by a tool which keeps a table of outstanding ENTs with their contents and context. A timeout can also be specified on the revocation. If specified, the UAS may remove the revocation from its revocation table. The timeout provided by the user should be the time after which the signature on the ENT would no longer be valid, so that the ENT would be unusable anyway, or the timeout in the ENT itself, if there was one.

When a user wishes to revoke an NAS from a request, she merely needs to send the revocation information to the UAS holding that NAS. If the NAS is contained in more than one UAS, the user may send the revocation to any UAS which contains a piece of the NAS. The UAS will then forward the revocation on to the neighbor UASs containing the NAS, who will do the same, until the revocation has been sent to all nodes of the NAS. If the user is worried about the request completing in a particular domain before the revocation has propagated, she can send the revocation directly to the UAS in that domain. If UASs containing the NAS are not currently available, the request for change is resent until it is acknowledged.

Continuing the purchasing example, we show user TURING using his PAY-FOR-SUPPLIES NAS, in Figure 2. His request cascades through multiple applications [Sol88], as well as multiple domains.

The sequence of actions is as follows:

1. The user TURING runs the PAY TP in Domain A, passing it the ENT [TURING, PAY-FOR-SUPPLIES, request-id].

2. PAY passes the ENT in its call to the local UAS, requesting all CLERK and CERTIFIER attributes with PAY in them.

3. PAY receives the attribute (TP:PAY CDI:supplies), and authorizes the run.

4. PAY calls the CHECK-WRITING TP in Domain C, including the ENT for TURING's PAY-FOR-SUPPLIES.

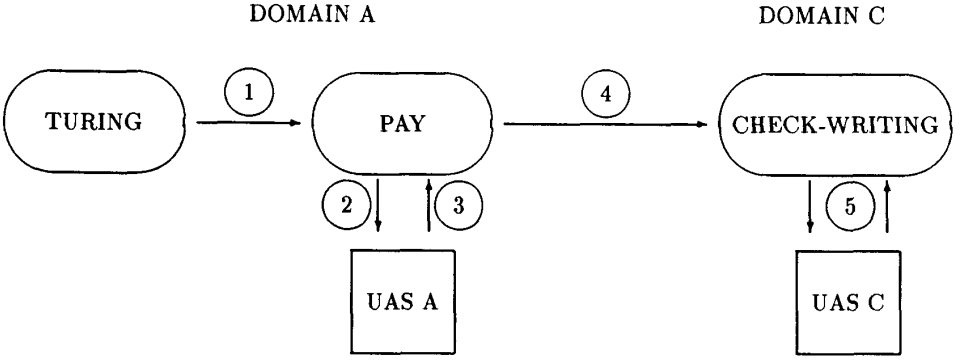

Figure 2: Cascaded request with NAS

5. CHECK-WRITING calls Domain C's UAS, with the PAY-FOR-SUPPLIES ENT, requesting and receiving TURING's attributes for CHECK-WRITING.

7.2 Sharing and Copying NASs

According to Mackay [Mac91], mechanisms for sharing facilitate user customization. Since the enforcement of Least Privilege relies on user customization, tools to support this enhance security.

Since NASs are the primary structure for customization, they can be shared explicitly, or by viewing and copying the NASs of others. Explicitly sharing of an NAS entails placing the users who will share that NAS on the list of users who can enable it. However, this places a burden on the owner to maintain the NAS for others, which may discourage this form of sharing.

NASs can also be made PUBLIC. This places no restrictions on who can use the NAS. It has no owner, and the list of users may be used for informational purposes only. It can be viewed and copied by anyone. A copied NAS replicates the attribute and neighbor information at each node to the new NAS, with the copier as owner, and an empty users list. As with other whole-NAS operations (see above), the NAS is tree-walked to send requests for the update to all other UASs involved.

8 Delegating to Other Users

A delegation token can be used to allow one user to delegate a particular set of privileges to another user for a limited amount of time. This token is called a Delegated NAS Token (DNT). This delegation token must be signed by the delegater, to prevent tampering and vouch for the delegation. It is signed with a key of the user's which will last at least as long as the delegation, such as a user's private key in a public/private key pair. It contains the following information:

- authentication identity being delegated to.
- authentication identity of the delegater.
- delegation ID. Like the request ID, this uniquely defines a delegation.
- time-out for the delegation. As with ENTs, this provides a timeout on the delegation, as well as allowing UASs to flush a revoked delegation at that time.
- forwardable bit. This bit indicates whether the delegate is allowed to delegate this token to another user or identity.
- NAS[s] containing the privileges being delegated

To use the delegation, the delegation token must be inserted into an ENT of the delegate, after the list of enabled NASs. If the forwardable bit is set, the original delegate may delegate one or more of the NASs in the DNT to some other user. Such forwarded delegations must be accompanied by a chain of DNTs, linking the requester back to the original delegater.

For simplicity, the links in the chain must match exactly. That is, the authentication identity of the last delegater must match the authentication identity

of the delegate of the immediately preceding delegation. In addition, the authentication identity of the requester must match the delegate in the final DNT in each DNT chain.

When asked for the prohibitions associated with an ENT containing DNTs, the UAS will return the union of the prohibitions of that application held by the delegate and the delegater (and all interim delegaters). Privileges enabled by a DNT must be named in the NASs of the DNT naming the requester as delegate, and held by the original delegater.

Privileges restricted to particular authentication identities may only be delegated by those identities. This restriction only applies to the first DNT in the chain. While this allows users to delegate from an authentication identity which can wield a privilege to one that cannot, they would only be working around a restriction they imposed upon themselves.

Privileges restricted to particular authentication authorities must be used (via an ENT) by a delegate authenticated by one of those authorities, and the original DNT from the privilege holder must be signed by a principal authenticated by one of those authorities. Only a delegation with the forwardable bit set will contain intermediate DNTs whose domains are not checked. As with discretionary access control, the attribute holder trusts the original delegate with determining who may subsequently use the delegation. Only identities authenticated and affiliated with authentication authorities trusted by the attribute's security manager may actually make use of the privilege. Prohibitions work in much the same way; any prohibitions associated with the authentication identity and authority of the original delegate as added to the prohibitions of the authentication identity who signed the ENT when an ENT with DNTs is used.

Privileges which cannot be delegated are not returned as enabled even when named in a valid NAS in a valid DNT. Privileges whose use also involves a prohibition should never be delegatable. As an example, consider an implementation of the Chinese Wall model [BN89]. Company dataset privileges should never be delegatable. If they were, a user could make one ENT containing the delegation to read information about a particular company, and another ENT without the delegation to access information about some other company in the same conflict of interest class. The conflict of interest class prohibition on the delegater would apply to the first legal read with the DNT, but not the second, now illegal read, of the delegate.

9 Conclusion

In this paper we have discussed some of the features necessary to supporting application-specific per-user security attributes in a distributed environment. Our goals were Least Privilege, local control and autonomy, explicit instantiation of trust relationships, and psychological acceptability. The chunking mechanism of domains eases the problems of understanding, implementing, and remembering security policy for both security managers and users (attribute holders). It provides a local association between an application and its users. However, both user populations will want to work across domains as well. The UAS provides security managers with the ability to associate multiple authentication identities with a single user record in a domain, and the ability to restrict the use of both privileges and prohibitions based on the authentication domain vouching for the user's process. A local attribute security manager may also delegate the management of an attribute to a foreign domain. Attribute holders may further restrict their own privilege use, based on authentication identity, and expand prohibition use as well. Finally, they are provided with Named Attribute Sets to group, enable, and delegate attributes.

Acknowledgments

Discussions with Karen Sollins greatly improved my thinking on the system's model and goals. This research was supported by Digital Equipment Corporation's Graduate Engineering Education Program (GEEP). Additional funding was provided by the Defense Advanced Research Project Agency, monitored by the National Aeronautic and Space Administration under contract No. NAG2-582, and by the National Science Foundation under grant NCR-881418.

References

[BN89] David F.C. Brewer and Michael J. Nash. The Chinese Wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[CW87] David D. Clark 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.

[ECM91] ECMA. Security application—authentication and privilege attribute. April 1991. draft standard.

[GGKL90] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, 1990.

[KN91] John Kohl and B. Clifford Neuman. The kerberos™ network authentication service. June 1991. INTERNET-DRAFT.

[Mac91] Wendy E. Mackay. Triggers and barriers to customizing software. In *CHI '91 Conference Proceedings*, pages 153–160, 1991.

[NRC91] National Rearch Council (NRC). *Computers at Risk: Safe Computing in the Information Age*. National Academy Press, 1991.

[Sol88] Karen R. Sollins. Cascaded authentication. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 156–165, April 1988.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1975.

[Ste91] Daniel F. Sterne. On the buzzword "security policy". In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 219–231, May 1991.