Secure Information Sharing Using Role-based Delegation[*]

Gail-Joon Ahn and Badrinath Mohan
University of North Carolina at Charlotte
Charlotte, NC, U.S.A.
{gahn,bmohan}@uncc.edu

Abstract

As computing becomes more pervasive, information sharing occurs in broad, highly dynamic network-based environments. Such pervasive computing environments pose a difficult challenge in formally accessing the resources. The digital information generally represents sensitive and confidential information that organizations must protect and allow only authorized personnel to access and manipulate them. As organizations implement information strategies that call for sharing access to resources in the networked environment, mechanisms must be provided to protect the resources from adversaries. In this paper we seek to address the issue of how to advocate selective information sharing while minimizing the risks of unauthorized access. We integrate a role-based delegation framework to propose a system architecture. We also demonstrate the feasibility of our framework through a proof-of-concept implementation.

Keywords: Information Sharing, Role-based, Delegation

1. Introduction

Several organizations have transited from their old and disparate business models based on ink and paper to a new, consolidated ones based on digital information on the Internet. The Internet is uniquely and strategically positioned to address the needs of a growing segment of population in a very cost-effective way. It provides tremendous connectivity and immense information sharing capability which the organizations can use for their competitive advantage. However, balancing the competing goals of collaboration and security is difficult because interaction in collaborative systems is targeted towards making people, information, and resources available to all who need it, whereas information security seeks to ensure the integrity of these elements while providing it only to those with proper authorization. Furthermore, as computing becomes more pervasive, information sharing occurs in broad, highly dynamic network-based environments. Such pervasive computing environments pose a difficult challenge in formally accessing the resources.

Digital information generally represents sensitive and confidential information that organizations must protect and allow only authorized personnel to access and manipulate them. As organizations implement information strategies that call for sharing access to resources in the networked environment, mechanisms must be provided to protect the resources from adversaries. We seek to address the issue of how to advocate selective information sharing in pervasive computing environments while minimizing the risks of unauthorized access. We integrate a role-based delegation framework [12] to propose a system architecture. We also demonstrate the feasibility of our framework through a proof-of-concept implementation.

The rest of this paper is organized as follows. In section 2 we discuss role-based delegation including details of system architecture. Section 3 overviews other research and related technologies. Section 4 describes implementation details. Section 5 concludes this paper.

2. Role-based Delegation

Ahn et al. [2] recently identified the following issues in collaborative environments. First, selective information sharing is necessary. We are dealing with friends, not enemies, and should provide relevant information expeditiously. Second, the information may be shared across organizational boundaries. Because sharing a resource across organizational boundaries often means authorizing a server to give access to a third party, it implies enabling resource servers to reason about previously unknown third parties. This requirement contrasts with many conventional systems, wherein a server need only reason about the set of users known inside a given organization. Third, it is impossible to fully predicate what data should be shared, when and to whom. And another thing is that a mechanism must

[*]**This work was supported, in part, by funds provided by National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).**

be provided for revoking the sharing when it is no longer needed. All these factors have to be considered in order to formulate the mechanism for information sharing among collaborating organizations.

In order to deal with the aforementioned issues, our work, called FRDIS (A Framework of Role-based Delegation for Information Sharing), leverages the existing models [10, 12]. To illustrate each functional component in our model, we use the role hierarchy example illustrated in Figure 1 and Table 1.

To simplify the discussion of delegation, we assume a user cannot be delegated to a role if the user is already a member of that role. For example, project leader Deloris with role PL1 cannot be delegated the role PO1 or PC1 since he has already been an implicit member of these roles.

2.1. Role Delegation

We first define a new relation called delegation relation (DLGT). It includes sets of three elements: original user assignments UAO, delegated user assignment UAD, and constraints. The motivation behind this relation is to address the relationships among different components involved in a delegation. In a user-to-user delegation, there are four components: a delegating user, a delegating role, a delegated user, and a delegated role. For example, ($Deloris$, PL1, $Cathy$, PL1) means $Deloris$ acting in role PL1 delegates role PL1 to $Cathy$. A delegation relation is one-to-many relationship on user assignments. The delegation relation supports role hierarchies: a user who is authorized to delegate a role r can also delegate a role r' that is junior to r. For example, ($Deloris$, PL1, $Lewis$, PC1) means $Deloris$ acting in role PL1 delegates a junior role PC1 to $Lewis$. A delegation relation is one-to-many relationship on user assignments. It consists of original user delegation (ODLGT) and delegated user delegation (DDLGT). Figure 2 illustrates components and their relations in FRDIS. We assume each delegation relation may have a duration constraint associated with it. If the duration is not explicitly specified, we consider the delegation as permanent unless another user revokes it. The function Duration returns the assigned duration-restriction constraint of a delegated user assignment. If there is no assigned duration, it returns a maximum value.

Table 1. Role Membership

ROLES	DIR	PL1	PL2	PO1	PO2
USERS	John	Deloris	Cathy	Michael David	Mark Lewis

FRDIS has the following components and theses components are formalized from the above discussions.

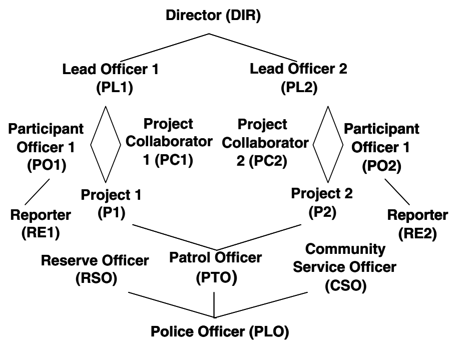

Figure 1. Role Hierarchy and Membership

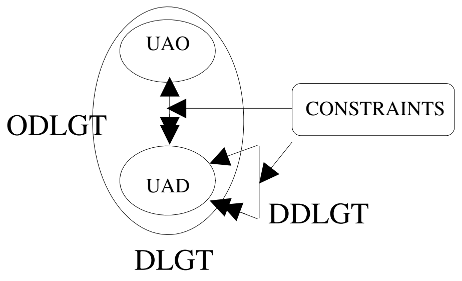

Figure 2. Delegation Relation

- T is a set of duration-restricted constraint.
- DLGT $\subseteq$ UA $\times$ UA is one to many delegation relation. A delegation relation can be represented by (u, r, u', r') $\in$ DLGT, which means the delegating user u with role r delegated role r' to user u'.
- ODLGT $\subseteq$ UAO $\times$ UAD is an original user delegation relation.
- DDLGT $\subseteq$ UAD $\times$ UAD is a delegated user delegation relation.
- DLGT = ODLGT $\cup$ DDLGT.

In some cases, we may need to define whether or not each delegation can be further delegated and for how many times, or up to the maximum delegation depth. We introduce two types of delegation: single-step delegation and multi-step delegation. Single-step delegation does not allow the delegated role to be further delegated; multi-step delegation allows multiple delegations until it reaches the maximum delegation depth. The maximum delegation depth is a natural number defined to impose restriction on the delegation. Single-step delegation is a special case of multi-step delegation with maximum delegation depth equal to one.

Also, we have an additional concept, delegation path (DP) that is an ordered list of user assignment relations generated through multi-step delegation. A delegation path always starts from an original user assignment. We use the following notation to represent a delegation path.

$uao_0 \rightarrow uad_1 \rightarrow uad_i \rightarrow uad_n$

Delegation paths starting with the same original user assignment can further construct a delegation tree. A delegation tree (DT) expresses the delegation paths in a hierarchical structure. Each node in the tree refers to a user assignment and each edge to a delegation relation. The layer of a user assignment in the tree is referred as the delegation depth. The function Prior maps one delegated user assignment to the delegating user assignment; function Path returns the path of a delegated user assignment; and function Depth returns the depth of the delegation path.

Constraints are an important aspect of RBAC and can lay out higher-level organizational policies. In theory, the effects of constraints can be achieved by establishing procedures and sedulous actions of security administrators. In FRDIS, the constraints are enforced by a set of integrity rules that provide management and regulators with the confidence that critical security policies are uniformly and consistently enforced. In the framework, when a user delegates a role, all context constraints that are assigned to the user and anchored to the delegated role are delegated as well.

2.2. Role Revocation

Several different semantics are possible for user revocation. Hagstrom and others [8] categorized revocations into three dimensions in the context of owner-based approach : global and local (propagation), strong and weak (dominance), and deletion or negative (resilience). Barka and Sandhu [3] further identified user grant-dependent and grant-independent revocation (grant-dependency) . Since negative authorization is not considered in FRDIS, we articulate user revocation in the following dimensions: grant-dependency, propagation, and dominance. Grant-dependency refers to the legitimacy of a user who can revoke a delegated role. Grant-dependent revocation means only the delegating user can revoke the delegated user from the delegated role membership. Grant-independent revocation means any original user of the delegating role can revoke the user from the delegated role. Dominance refers to the effect of a revocation on implicit/explicit role memberships of a user. A strong revocation of a user from a role requires that the user be removed not only from the explicit membership but also from the implicit memberships of the delegated role. A weak revocation only removes the user from the delegated role (explicit membership) and leaves other roles intact. Strong revocation is theoretically equivalent to a series of weak revocations. To perform strong revocation, the implied weak revocations are authorized based on revocation policies. However, a strong revocation may have no effect if any upward weak revocation in the role hierarchy fails. Propagation refers to the extent of the revocation to other delegated users. A cascading revocation directly revokes a delegated user assignment in a delegation relation and also indirectly revokes a set of subsequent propagated user assignments. A non-cascading revocation only revokes a delegated user assignment.

Our preliminary study shows grant-dependent revocation for brevity. Suppose the revocation in Figure 3 is weak non-cascading, for $John$ to revoke $Cathy$ from role PL1, it is important to note that only $Cathy$'s membership of role PL1 is changed; other role memberships of $Cathy$ and all the delegated user assignments propagated by $Cathy$ are still valid. If the revoked node is not a leaf node, non-cascading revocation may leave a "hole" in the delegation tree. A solution might be the revoking user takes over the delegating user's responsibility. In this example, $John$ takes over the delegating user's responsibility from $Cathy$, and changes all delegation relations: ($Cathy$, PL1, u, r) $\in$ DLGT to ($John$, DIR, u, r) $\in$ DLGT. In this case, $John$ takes over $Cathy$'s delegating responsibility for $Mark$ and $Lewis$.

2.3. Rule-Based Policy Specification Language

FRDIS defines policies that allow regular users to delegate their roles. It also specifies the policies regarding which delegated roles can be revoked. A rule-based language is adopted to specify and enforce these policies. It is a declarative language in which binds logic with rules. The advantage is that it is entirely declarative so it is easier for security administrator to define policies.

A *rule* takes the form:

$H \leftarrow F1\&F2\&\ldots\&Fn$

where H, F1, F2,. . . , Fn are Boolean functions.

There are three sets of rules in the framework: basic authorization rules specify organizational delegation and revocation policies; authorization derivation rules enforce these policies in collaborative information systems; and integrity rules specify and enforce role-based constraints.

For example, a user-user delegation authorization rule forms as follows:

$can\_delegate(r, cr, n) \leftarrow .$

where r, cr, and n are elements of roles, prerequisite conditions, and maximum delegation depths respectively.

This is the basic user-to-user delegation authorization rule. It means that a member of the role r (or a member of any role that is senior to r) can assign a user whose current membership satisfies prerequisite condition cr to role r (or a role that is junior to r) without exceeding the maximum delegation depth n.

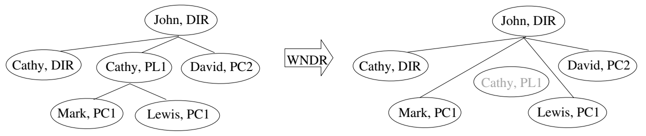

Figure 3. Weak Non-cascading Revocation

A user delegation request is further authorized by the **user-user delegation authorization derivation rule** that takes the form:

$$
\begin{aligned}
&der\_can\_delegate(u, r, u', r', dlg\_opt) \leftarrow \\
&\quad can\_delegate(r'', cr, n)\& \\
&\quad active(u, r, s)\& \\
&\quad delegatable(u, r)\& \\
&\quad senior(r, r'')\& \\
&\quad in(u', cr)\& \\
&\quad junior(r', r'')\& \\
&\quad in(depth(u, r), n).
\end{aligned}
$$

where u and u' are elements of users; r, r', and r'' are elements of roles; cr and s are elements of prerequisite condition and sessions respectively; dlg_opt is a Boolean term, if it is true, then further delegation is allowed. This argument is used as Boolean control of delegation propagation.

This rule means that a user u with a membership of a role r senior to r'' activated in session s can delegate a user u' whose current role membership satisfies prerequisite condition cr to role r' (r' is junior to role r'') without exceeding the maximum delegation depth n. Similar rules are also defined for role-based revocations and are applied to specify constraints.

2.4. Architectural Framework

Our system is designed to provide access control and delegation in collaborative environments. We use the term *Hive* to describe its system architecture. In a bee hive, honeybees collect nectar and pollen and store them for other bees to use to make honey; in the *Hive*, any legitimate users can access resources at anytime and anywhere. To do so, access control services should be provided to users who have access privileges that can be originally assigned by a security officer or can be delegated by other legitimate user(s). The notions described in *Hive* are designed to be utilized within an administrative-directed delegation management architecture. An overview of the preliminary architecture is shown in Figure 4. It consists of a number of services and management agents together with the objects to be managed. The enforcement agents are based on a combination of roles and rules for specifying and interpreting policies. Since delegation and revocation services are only part of a security infrastructure, we choose a modular approach to our architecture that allows the delegation and revocation services to work with current and future authentication and access control services. The modularity enables future enhancements of our approach. The role service is provided by a role server, which is an implementation of the RBAC and *Hive* components. A role server maintains RBAC database and provides user credentials, role memberships, associated permissions, and delegation relations of the system. The rule service is provided by a rule server, which manages delegation and revocation rules. These rules are always associated with a role, which specifies the role that can be delegated. They are implemented as authorization policies that authorize requests from users. The delegation agent is an administrative infrastructure, which authorizes delegation and revocation requests from users by applying derivation authorization rules and processes delegation and revocation transactions on behalf of users. The implementation requirements related to the delegation framework are not only a delegation agent, but also authentication and access control agents. The authentication agent is used to authenticate users during their initial sign-on and supply them with an initial set of credentials. The reference monitor makes access control decisions based on information supplied by the access control agent. In large role-based system, there may be tens or hundreds of delegation and revocation rules. The rule editor is developed to simplify the management of these rules. As a portion of an integrated RBAC administration platform to manage various RBAC and *Hive* components, the rule editor is used to view, create, edit, and delete delegation and revocation rules.

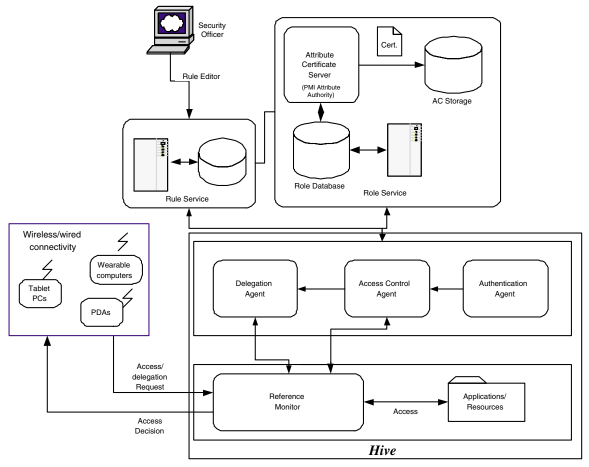

Figure 4. FRDIS Architecture: *Hive*

3. Related Works

Pervasive computing enables users to access services and information at anytime and anywhere. Under such computing environments, the information sharing tends to be very dynamic and often ad hoc. Hence, the traditional management approach is not appropriate to such environments because the workload on a security officer (or a small group of security officers) will be overwhelming. Since the very goal of our research is to enable users to access and selectively share resources in distributed systems, we assume that users can be trusted to exercise their discretions on resources: if Alice explicitly shares a resource with Bob, she trusts Bob to use the resource. We also consider enhancing the scalability of information sharing. Some of the projects that address this issue are UC Baltimore's eBiquity [9], UC Berkely's Ninja project [7], Stanford's Interactive Workspace Project [5], and Matt Blaze's PolicyMaker [4] with the notion of Role-Based Access Control (RBAC). We have found that delegation is one of promising approaches to rectify the above-mentioned issue. There are many definitions and different types of delegation in the literature [1, 6]. In general, it is referred to as the process whereby one active entity in a system authorizes another entity to act on behalf of the former by transferring a set of rights. Through delegation, individual user is trusted and empowered to share resources to which they have access.

3.1. Privilege Management Infrastructure

PMI is based on the ITU-T Recommendation of directory systems specification, which introduced PKI in its earlier version. Public-key certificates are used in PKI while attribute certificates are a central notion of PMI. Public-key certificates are signed and issued by certification authority (CA), while attribute certificates are signed and issued by attribute authority (AA). PMI is to develop an infrastructure for access control management based on attribute certificate framework [11]. Attribute certificates bind attributes to an entity. The types of attributes that can be bound are role, group, clearance, audit identity, and so on. Attribute certificates have a separate structure from that of public key certificates.

PMI consists of four models: general model, control model, delegation model, and roles model. General and control models are required, whereas roles and delegation models are optional. The general model provides the basic entities which recur in other models. It consists of three foundation entities: the object, the privilege asserter, and the privilege verifier. The control model explains how access control is managed when privilege asserters request services on object. When the privilege asserter requests services by presenting his/her privileges, the privilege verifier makes access control decisions based upon the privilege presented, privilege policies, environmental variables, and object methods. The delegation model handles a situation when privilege delegation is necessary. It introduces two additional components: source of authority (SOA) and other attribute authorities (AAs). When delegation is used, SOA assigns privilege to AAs, and AAs delegate privileges to an end-entity privilege asserter. Lastly, PMI roles model also introduces two additional components: role assignment and role specification. Role assignment is to associate privilege asserters with roles, and its binding information is contained in attribute certificate called role assignment attribute certificate. The latter is to associate roles with privileges, and it can be contained in attribute certificate called role specification attribute certificate or locally configured at a privilege verifier's system.

4 Implementation Details

Our implementation leverages *Hive* features and X.509 attribute certificate. We attempt to implement the proof-of-concept prototype implementation of *Hive* on privilege management infrastructure (PMI). PMI provides certificate-based authorization with attribute certificates while public-key infrastructure (PKI) does certificate-based authentication with public-key certificates, so called identity certificates.

Three components are identified for managing attribute certificates: privilege asserter, privilege verifier, and PMI attribute authority. Two different attribute certificates are employed: role assignment attribute certificate (RAAC) for assigning roles to a user and role specification attribute certificate (RSAC) to assign specific permissions to a role. Our implementation is divided into two components. The first component is to build APIs for both a role-based decision making engine and attribute certificates. Those APIs are the core building blocks for constructing an access control policy server and an attribute certificate server. The second component is to implement each entity integrating with APIs. Some of PMI modules in [11] were utilized to construct the above-mentioned components.

We also developed an application working as an access control policy server. This application has been developed in C++. An engine for making access control decisions is a major component in this application. After receiving a valid RAAC and requested objects (with operation type) from the server, the engine extracts permissions from the RSAC and checks if the requested object (with operation type) is in the list of permissions. The programming library, called RBAC API, was developed to facilitate such procedures.

The current work is focused on the multimedia information sharing between a server in *Hive* and handheld devices. A user of a handheld device can communicate directly with the server and also send commands to the server application. The communication channel has been supported by Microsoft Windows Sockets (WinSock). The management of multimedia information is handled by Microsoft Windows Media Control Interface (MCI). We also used a library such as Victor Image processing library for the conversion between BMP files to JPEG files. In order to enhance the performance, our application is capable of zipping files in both the server and handheld devices developed by Visual Studio 6.0 (VC++ version 6.0) and Embedded Visual Tools 3.0 (EVC++ 3.0), respectively.

To support the above-mentioned features, our implementation consists of following modules:

- Communication channel establishment: The communication link is established through the socket connection. It includes creation and listen modes.
- Controlling messages and commands: Both parties need to exchange messages and a client needs to send commands to a server.
- Information sharing between both parties: Once the communication is set up both parties are ready to share information.

5. Conclusion

Sharing information and resources in collaborative environments entails addressing several requirements not raised by traditional single-user environments in part due to the unpredictability of users and the unexpected manners in which users and applications interact in collaborative sessions. In this paper, we have discussed issues of information sharing introducing an architecture, *Hive*. We also attempted to utilize an existing delegation framework and attribute certificates in PMI. In addition, we demonstrated the feasibility of our architecture through a proof-of-concept implementation. Currently we are investigating how this technology can be applied to K-12 education environment supporting some of features.

References

[1] M. Abadi, M. Burrows, B. Lampson and G. Plotkin. A calculus for Access Control in Distributed Systems. *ACM Transaction on Programming Languages and Systems*, Vol.15 No. 4, Sept 1993, pages 706-734.

[2] Gail-J. Ahn, Longhua Zhang, Dongwan Shin and Bill Chu. Authorization Management for Role-based Collaboration. In *IEEE International Conference on System, Man and Cybernetic (SMC2003)*, pages 4128-4214, Washington, DC, October 2003.

[3] E. Barka and R. Sandhu. Framework for role-based delegation model. In *Proceedings of 23rd National Information Systems Security Conference*, pages 101–114, Baltimore, MD, October 16-19 2000.

[4] M. Blaze, J.Feigenbaum, and J.Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996, 1996.

[5] G. Candea and A. Fox. Using Dynamic Mediation to Integrate COTS Entities in a Ubiquitous Computing Environment. In *Proc. Second International Symposium on Handheld and Ubiquitous Computing*, 2000.

[6] M. Gasser and E. McDermott. An Architecture for Practical Delegation a Distributed System. In *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 7-9,1990.

[7] S. D. Gribble et al. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, 2001.

[8] A. Hagstrom, S. Jajodia, F. P. Presicce, and D. Wijesekera. Revocations - a classification. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 44–58, Nova Scotia, Canada, June 2001.

[9] L. Kagal, T. Finin, and A. Joshi. Trust-based Security in Pervasive Computing Environments. *IEEE Computer*, pages 2-5, December 2001.

[10] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[11] D. Shin, Gail-J. Ahn, and S. Cho. Role-based EAM Using X.509 Attribute Certificate. In *Proceedings of Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, King's College, University of Cambridge, UK July 29-31, 2002.

[12] L. Zhang, Gail-J. Ahn and B. Chu. A Rule-Based Framework for Role-Based Delegation and Revocation. *ACM Transactions on Information and System Security*, Vol.6, No.3, August 2003.