

Securing Resources in Collaborative Environments: A Peer-to-peer Approach

Karlo Berket, Abdelilah Essiari, Mary R. Thompson
Lawrence Berkeley National Laboratory
1 Cyclotron Rd, MS 50B-2239, Berkeley CA 94720*
{kberket, aessiari, mrthompson}@lbl.gov

Abstract

We have developed a security model that facilitates control of resources by autonomous peers who act on behalf of collaborating users. This model allows a gradual build-up of trust. It enables secure interactions among users that do not necessarily know each other and allows them to build trust over the course of their collaboration. This paper describes various aspects of our security model and describes an architecture that implements this model to provide security in pure peer-to-peer environments.

KEYWORDS: security, authorization, peer-to-peer

1. Introduction

Applications based on peer-to-peer (P2P) architectures have become popular because they allow information to flow freely between distributed components. However, this very feature holds back the acceptance of these applications by the corporate and scientific communities. In these communities, the information flow needs to be controlled. For example, a group of collaborating scientists would like to share the initial findings of their research within their group, but do not want these findings available to the general audience until they have had a chance to verify them.

Providing security in P2P environments is difficult due to the distributed and autonomous nature of the peers. There are two major challenges: efficiently establishing authenticated, encrypted communication channels between the peers and distributing the authentication and authorization enforcement for shared resources. It is also desirable that the provider of the resource can securely control its access without causing a significant burden on either the provider or the user of that resource. We have developed security mechanisms for establishing secure connections between peers and for distributing and verifying signed authorization policy [1]. Through the

application of these mechanisms peers can establish spontaneous trust relationships, and securely share and access resources in a straightforward manner. We implemented and applied our security mechanisms in a secure P2P information sharing application, scishare [2].

Our previous paper [1] concentrated on the mechanisms of establishing authenticated and encrypted channels between the peers. This paper describes the authorization architecture that is required to implement distributed authorization decisions in a P2P environment. The remainder of this paper is organized as follows. First, we present other work related to authorization in P2P environments. We introduce a collaborative application model for a P2P environment in Section 3. Then, we discuss the particular authorization challenges of this environment in Section 4. In Section 5, we present a security model that addresses the authorization challenges of this environment. Then, we present the authorization architecture in Section 6. Finally, we present our conclusions and future work in Section 7.

2. Related Work

A number of authorization systems have been developed to provide access control to shared resources in distributed environments [3][4][5][6]. While these systems do not address highly dynamic or purely decentralized collaborations, they provide some useful concepts. CAS [5] is a system that uses the new IETF standard X.509 proxy certificates [7] to delegate access rights from a central server to a user. We intend to use a similar mechanism to delegate rights from one user to another in order to facilitate temporary access to resources. Both Akenti [6] and VOMS [3] use attribute certificates to assign attributes to users in a verifiable way. PRIMA [4] introduces the idea of privilege management. A privilege represents the rights that a user has to a resource and can be stored in a verifiable certificate. Each of these systems has some dependence on central servers to provide the authorized access information and each depends on a few specified trusted Certification Authorities (CA) to provide X.509 public key certificates.

Akenti provides an example of decentralized access control by permitting most policy to be held in signed

* The authors are supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This document is report LBNL-58867, Disclaimer available at <http://www-library.lbl.gov/disclaimer>.

certificates. This allows multiple stakeholders to write policy for resources. The Akenti policy decision engine can then gather and verify the policy and attribute information. Allowing a new user to access resources still requires that the user have an X.509 certificate from a trusted CA. Furthermore, the user must be explicitly granted some attribute or added to the policy files, which is not optimal for spontaneous collaborations.

The following P2P applications each provide some access control over their resources. Groove [8][9] allows a small group of collaborators to form spontaneous shared spaces in which they exchange information. It essentially implements a simple public key infrastructure (PKI) without certificates to build trust among users. Groove provides mechanisms for simplifying the building of collaborations and has automated a number of protocols to hide the key management issues from the users. It also provides support to gradually build trust through its built-in invitation protocol. Groove does not support fine-grained access control to individual resources.

Waste [10] is a secure file-sharing system that provides security of information within a small, trusted group of peers. It secures all communication at the transport-level using Transport Layer Security (TLS) [11] and builds a PKI web of trust between the trusted peers. Waste assumes that all of the trusted peers are equal. In addition, peers are forced into trust relationships that are commutative and associative. Thus, any peer that is allowed into the system has full access to all the information in the system.

LionShare [12][13] is an Internet2 project that aims to facilitate legitimate file sharing among individuals and educational institutions around the world. It promises secure file-sharing capabilities for the easy exchange of image collections, video archives, large data collections, and other types of academic information. The LionShare security model is based on three requirements: files should not be shared anonymously, an anonymous search facility should exist, and file owners must have the ability to control access to their files using verifiable attributes.

3. Application Model

We consider an application that consists of a set of distributed application components. Each component provides identical communication and security interfaces, and is referred to as a peer, P_i . Each peer, P_i , has a *user identity*, U_i associated with it. The user identity may represent the person who is using this application component, the organization that has deployed this application component, etc. A single user identity may be associated with multiple application components, but not vice versa.

The set of all peers forms a peer group G . The peers communicate by exchanging messages. A peer, P_i , can send a message to any other peer, P_j , or to the peer group, G . Communication between two peers, P_i and P_j , is

direct, i.e. we do not consider the scenario where a third peer, P_k , acts as proxy, forwarding messages for the communication. Messages sent to the peer group G are intended for every peer in G . Each message is received by a proper subset of the intended recipients. The peers form a closed group in the sense that only peers can send messages to the group.

A peer, P_i , may disconnect from G at any time. A disconnected peer does not receive or send any messages within G while it is disconnected. Peers may disconnect from G frequently and for prolonged periods of time. The likelihood of any given peer, P_i , staying connected to group G for the entire lifetime of G is assumed to be negligible.

A peer, P_i , may provide a set of resources, $R(P_i)$, to group G . A resource may be a file, a scientific instrument, a chat room, etc. The holder of the user identity, U_i , governs access to a resource, R in $R(P_i)$. The peer, P_i , enforces access to this resource, R . In order to gain access to a resource a user must have their user identity authenticated by the peer and meet the authorization requirements for access to that resource.

4. Authorization Challenges

The application model described in Section 3 provides unique challenges in authorization. The peer-based model treats all application components equally; whether they behave as clients, servers, certificate authorities, etc. The dynamic membership of the group does not allow us to assume that any peer will be highly available. Thus, the traditional notions of highly available certificate authorities and authorization servers are not applicable. Their roles need to be filled by the individual peers, working in concert when appropriate.

The first authorization challenge arises from the collaborative application requirements to support the rapid admission of new users. New collaborators need to be provided basic access to the application as soon as possible, i.e. immediately. This requirement makes infeasible the standard approach of requiring a central administrator to authorize each new user.

Revocation of access rights is also a major challenge in such an environment. Typical solutions, such as the Online Certificate Status Protocol (OCSP) [14] assume that an authoritative server is always available. Certificate Revocation Lists (CRLs) [15] relax this requirement a bit, but still assume that a fresh CRL can be obtained from an authoritative server at regular intervals. The Simple Certificate Validation Protocol (SCVP) [16] allows for both of these revocation methods. SCVP allows a client to delegate certificate path construction and certificate path validation (e.g. making sure that none of the certificates in the path are revoked) is performed according to a validation policy, which contains one or more trust

anchors. It allows simplification of client implementations and use of a set of predefined validation policies.

The application model also places a stronger security role on the end-users, who are providers of resources. In this model, it is very likely that the end-users are determining the authorization requirements for accessing resources. Thus, the authorization interfaces presented to the average user need to be clear and concise.

5. Security Model

We have designed a security model to meet these challenges. The underlying assumptions of our model are: Authentication is based on user identity (e.g. an X.509 credential), rather than peer identity (e.g. IP address); Each user must have some authentication token that can be recognized by all the other peers on which authorization rules can be based; All communication between peers takes place over authenticated encrypted channels; Each provider is entirely responsible for setting and enforcing access to the service or resource it provides.

We use X.509 public-key certificates [15] as the authentication token for the user identities. This allows us to leverage existing support for the X.509 public key infrastructure (PKI). The X.509 PKI provides scalable key management, works with widely available TLS implementations, e.g. OpenSSL[17], to provide secure connections, and provides keys that can be used to digitally sign authorization and attribute assertions.

In order to allow new users to easily join an existing collaboration, we automatically provide a self-signed X.509 certificate, called a *pseudo-certificate*, to users the first time they use the system. These credentials allow users that do not already have a credential from a trusted Certification Authority (CA) to authenticate into the collaboration. A pseudo-certificate contains a Distinguished Name (*DN*) for the user, which is self-selected and may not be globally unique and a public key that, due to the sparseness of the key space, is assumed to be unique. Thus, pseudo-certificates can be assumed to be unique and can be used as an authentication token. Peers can therefore build trust based on pseudo-certificates. All users of collaborative applications using this communication model are thus able to immediately and securely participate in the collaboration. They do not have to wait for an administrator to grant them appropriate credentials just to cross this threshold.

Note that a CA is considered trusted if the relying party has read and agreed with the CA's certification policy and has decided to trust the binding between the DN in the certificate and the individual holding the private key. In the case of pseudo-certificates there is no vetting that the name honestly represents the individual holding the private key. The only assurance is that the same key will represent the same individual each time it is used. Thus all trust in the holder of these keys must be done individually on the basis of behavior or out-of-band

information. The handling of access rights revocation in our model is based on the type of resource being accessed, as well as the user type. Our model makes a clear distinction between users with pseudo-certificates and those with CA-signed certificates. We use the term *authenticated user* to be a user who is known to possess the private key that corresponds to the pseudo or CA-signed public-key certificate he has presented and the term *validated user*¹ to be an authenticated user whose certificate can be traced back to an accepted CA. The most significant difference between authentication and validation is that in the latter process each certificate in the chain is checked to see if it has been revoked. This process usually requires access to some central service provided by the CA.

We classify the resources shared within a collaborative application as *low-value* or *high-value*. The difference between low-value and high-value resources is whether all the credentials used in establishing authorization need to be validated as well as authenticated.

Access rights are often granted to users based on the trust placed in the holder of the authentication token. In longer-term collaborations, the trust relationships between users may vary over time. Thus, it is important to allow the resource providers to grant and change access rights for users based on this trust relationship. In our security model, resource access policies can range from complete access to any user to specific access for individual validated users. Since all users in our model must be authenticated a peer is able to capture the certificates of all of the users it interacts with. The trust relationship with these users can then be measured based on off-line information such as recommendations from other users or user behavior during previous interactions. Thus, a user can easily join a collaboration and gradually be granted access rights.

The policy aspects of our security model are strongly influenced by our experience with the Akenti authorization system [6]. Akenti was designed to use *distributed* authorization policy, in contrast to the more common approach of having all policy local to the resource or alternatively on a central trusted server. The access policy consists of *policy rules*, which are stated in terms of *attributes* of a user or resource. Policy rules and attribute assertions are signed documents (certificates) that can be stored in a distributed manner and are gathered from a set of locations and verified at the time the authorization decision is made. This approach matches our need to support both local and peer shared policy. An attribute is simply some characteristic of a user, such as identity, group membership, a role or a clearance level. A policy rule states what attributes a user must have to get a specific type of access to a resource.

¹ A validated user corresponds to a trusted user in the PKI literature. We reserve the word trust to refer to the level of privileges or authorizations that a user has, in other words, what actions they are trusted to do.

6. Authorization Architecture

In this section we present a flexible authorization architecture that implements the presented security model. This architecture is suitable for applications that involve sharing of high-value resources, as well as general collaborations where users only need a reasonable sense of security. We have divided the authorization system into five components: a *Validation Manager* that manages, discovers, and validates X.509 identities, an *Attribute Manager* that generates and looks up attribute assertions, a *Policy Manager* that creates and shares policy rules, a *Resource Manager* that manages the protected resources, and a *Delegation Manager* that manages the delegation of access rights. At the end of this section we present an example of how these components interact.

VALIDATION MANAGER

The *Validation Manager* (VM) allows resource owners to designate trusted Certification Authorities (CA). The VM stores the set of CA X.509 public key certificates that are trusted by the user. The user may also provide the location of the public directories that contain the X.509 identity certificates issued by a CA and the location of the certificate revocation lists (CRL) maintained by a CA.

The VM also checks whether a certificate belongs to a validated user. A certificate passes this test if its certificate chain is valid, i.e. the signature of each certificate can be verified using a public key contained in the chain, if no entity in the chain has been revoked, and if one of the CAs in the chain has been stored at the VM. The VM does not assume that the whole certificate chain is presented to it with the certificate. It locates any missing certificates by checking public directories and P2P resource discovery. Pseudo-certificates can never be validated since there is no way to tell if a pseudo-certificate has been revoked.

In summary, the Validation Manager performs the following operations: manage the list of trusted CAs. (add, remove, edit); validate entity (users, attribute issuers, authorization policy issuers) certificates; search for X.509 certificates/CRLs using server-based directories or P2P discovery; provide X.509 certificates/CRLs in response to P2P queries.

ATTRIBUTE MANAGER

The *Attribute Manager* (AM) can support a number of attributes, such as group, role, organization or licensing. Without loss of generality, we will concern ourselves with the *group* attribute in order to best explain what this component does. A group is simply a set that has a name and one or more owners (group authorities). Groups can be private (used and stored at only one peer) or public (named sets visible to other users).

A user belongs to a private group if that user's DN or public key is listed in that group's set. A user belongs to a public group if that user has a 'valid' X.509 attribute certificate [18] for that group signed by one of the group's attribute authorities. We note that 'valid' does not mean 'validated'. A valid certificate can pass all the tests (expiration, signature, etc.) without validating the attribute authority with the Validation Manager. Our model allows such certificates to be acceptable when used during access decisions for *low-value* resources. This allows un-validated users (including pseudo-users) to manage and belong to groups.

A user can be removed from a private group simply by removing the user from that group's set. Removing a user from a public group requires the use of a revocation mechanism much like the one used for X.509 identity certificates.

In summary, the Attribute Manager performs the following operations: allow users to create public and private user attributes; define and manage names of public attributes so that users can reference them; allow users to search for public user attributes using P2P queries; provide public user attribute information (attribute names, users, owners, attribute certificates and CRLs) in response to P2P queries; check whether a user satisfies an attribute.

POLICY MANAGER

The *Policy Manager* (PM) manages and checks policy rules. A policy rule has a name, one or more owners, a condition, and zero or more actions. Like attributes, policy rules can be private (stored at the resource to which it applies) or public (named and visible to other peers). Public rules are necessary to support multiple stakeholders who may not have login privileges at the resource site. Rules can be combined using Boolean operators 'and' and 'or'. Combined rules are given names and are marked as public or private. A rule composed of at least one private rule must also be private. A user that satisfies a rule's condition is granted the rights listed in the rule.

Public rules can be opaque or transparent. Opaque rules are evaluated at the creator's site and require the creator be online. Transparent rules are evaluated by each involved entity and are used to secure access to common resources (shared spaces, chat rooms) that are implemented in a decentralized manner, e.g. on top of a decentralized secure group communication scheme such as SGL [19]. In such systems, every peer must be able to evaluate the policy rule when a member joins or leaves. The group configuration (members, session key, etc.) stays consistent across the peers as they all arrive to the same decision. This decentralized approach allows, among other things, subgroups of users to persist during network partitions.

In summary, the Policy Manager performs the following operations: allow users to create public and private rules; allow users to search for public rules using

P2P queries; provide public rules in response to P2P queries; define and manage names of public rules so that users can reference them; check whether a user satisfies a rule.

RESOURCE MANAGER

The *Resource Manager* (RM) allows users to register their protected resources into a local database and to map a combined or single policy rule to one or more resources. Resources can be marked as high-value or low-value. When enforcing access to high-value resources, the RM makes sure that all authorization elements are validated. While this is the rigorous approach to handling distributed policy, we have found in practice that many P2P resources do not require that level of security. Thus for low-value resources, real-time checks for revocation are skipped and policies are allowed to specify holders of pseudo-certificates as authorized users and attribute authorities. If a relying party has reason to believe that a pseudo-certificate has been compromised or no longer trusts the holder of such a certificate, he must take action to remove any access granted to this user from his authorization policies.

In summary, the Resource Manager performs the following operations: allow users to map policies to resources; allow users to mark resources as high or low value; search for resources owned by others using P2P queries; provide resources' information in response to P2P queries; determine users' access rights to resources.

ACCESS DECISION PROCESS

Figure 1 shows how the presented components interact during an access decision. A user U1 is attempting to access a resource R owned by user U2. The Resource Manager, at U2, determines the mapped policy P and the sensitivity level L. If L is high-value then U1 must be validated and RM calls on the Validation Manager to perform this check. RM then hands these arguments together with U1's name to the Policy Manager who evaluates the policy P. In this example, P is private and P's condition simply requires a user to belong to group G. Since P is private and thus owned by U2, the PM does not need to interact with the VM to check whether U2 is validated or not. The PM checks with the Attribute Manager to see if U1 belongs to group G. Since G is public the AM gets the necessary attribute certificates and calls on the VM to validated G's authority. If everything succeeds then the actions listed in P are returned to the application.

DELEGATION MANAGER

Delegation of access rights to other users provides a quick and a simple way of allowing temporary and restricted access to common resources such as shared spaces and chat rooms. The benefits are huge since users can now be invited to join collaborations without having to go through the long process of obtaining the necessary attribute credentials.

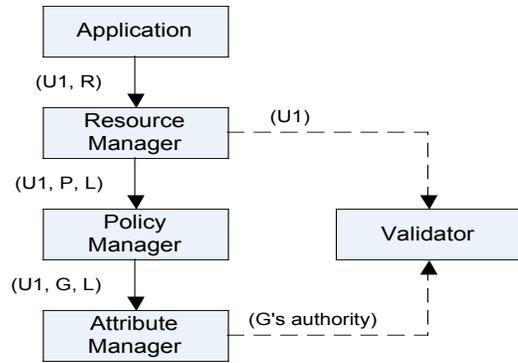


Figure 1: Access decision steps.

The *Delegation Manager* (DM) uses a delegation protocol similar to the one in [5]. At the end of this protocol a certificate in the form of an X.509 proxy certificate [7] is generated. These certificates tie a user to a list of access rights and can be used as authentication tokens to SSL/SGL. The verification process of these capabilities is rather simple. In the case of an invitation, the signing party must have an invite capability. In the case of an escort, the signing party must have an escort capability and the guest can only use the resource if the signing party is present. Rejecting a user is also straightforward and can be used to deny access to someone else's guest. We do not address the revocation of capability certificates as they are meant to have short lifetimes. Figure 2 shows an example of user U1 inviting user U2 to resource R.

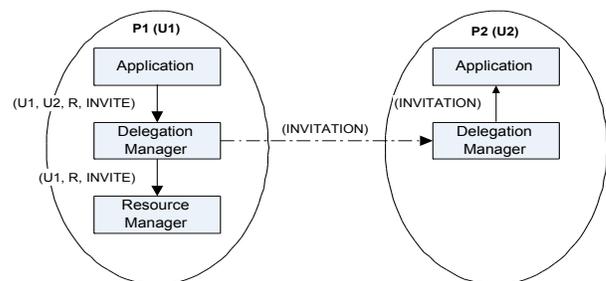


Figure 2: Invitation example.

In summary, the Delegation Manager performs the following operations: allow users to invite, escort, or reject guests; allow users to request invitations and escorts.

7. Conclusion

We have introduced a flexible security solution for pure P2P environments. By using a P2P resource discovery service to discover public groups and policy rules the overall system is tolerant to policy issuers being offline or unreachable. The introduction of pseudo-certificates allows users to easily join a collaborative application. Our model enforces authenticated access to public resources, which allows users to meet each other and facilitates the building of trust relationships. The use of resource sensitivity levels allows casual collaborations to be secured without requiring that all users have X.509 public key certificates from trusted CAs. In the future, we plan to look into caching and duplication mechanisms to provide a greater range of sensitivity levels.

Our file sharing application has already used many of these concepts to provide users with a simple and a secure way of sharing files. We plan to re-engineer it to adopt its P2P resource discovery service for locating access control information if such information is not available from central servers. We also plan to apply our security implementation to a secure chat application. The results of these two deployments should help us further evaluate the usefulness of the various components of our security model.

8. References

All referenced web pages were verified to be correct on September 8, 2005.

- [1] K. Berket, A.Essiari and A. Muratas, "PKI-Based Security for Peer-to-Peer Information Sharing," *Proceedings of the Fourth IEEE International Conference on Peer-to-Peer Computing*, Zurich, Switzerland, Aug. 25-27, 2004.
- [2] "scishare", <http://dsd.lbl.gov/scishare/>.
- [3] R. Alfieri, et.al. "Managing Dynamic User Communities in a Grid of Autonomous Resources", *Proceedings of Conference for Computing in High Energy and Nuclear Physics*, La Jolla March 24-28, 2003
- [4] M. Lorch, D. Kafura, "The PRIMA Grid Authorization System" , *Journal of Grid Computing*, 2(3), 2004, 279-298
- [5] L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke, "A Community Authorization Service for Group Collaboration", *Proceedings of the IEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
- [6] M.Thompson, A. Essiari, S. Mudumbai, "Certificate-based Authorization Policy in a PKI Environment", *ACM Transactions on Information and System Security (TISSEC)*, 6(4), 2003, 566-588.
- [7] S. Tuecke, D. Engert, I. Foster, V. Welch, M. Thompson, L. Pearlman, and C. Kesselman, "Internet X.509 Public Key Infrastructure Proxy Certificate Profile", RFC 3820, 2004.
- [8] A. Oram, editor, *Peer-to-peer: Harnessing the power of disruptive technologies* (O'Reilly & Associates, Inc., Sebastopol, CA, 2001).
- [9] "Groove, Networks", <http://www.groove.net/>.
- [10] "Waste", <http://waste.sourceforge.net/>.
- [11] Dierks, T. and C. Allen, "The TLS Protocol", RFC 2246, January 1999.
- [12] "LionShare project," <http://lionshare.its.psu.edu/main/>.
- [13] "LionShare: Connecting and Extending Peer-to-Peer Networks," LionShare White Paper Draft, October 2004, <http://lionshare.its.psu.edu/main/info/docspresentation/LionShareWP.pdf>.
- [14] M. Myers, R. Ankey, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP", RFC 2560, 1999.
- [15] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, and W. Ford, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Internet Draft, April 2005.
- [16] T. Freeman, R. Housley, A. Malpani, D. Cooper, and T. Polk, "Simple Certificate Validation Protocol (SCVP)", Internet Draft, February 2005.
- [17] OpenSSL, <http://www.openssl.org/>.
- [18] S. Farrel, R. Housley, "An Internet Attribute Certificate Profile For Authorization", RFC 3281, 2002.
- [19] D.A. Agarwal, O. Chevassut, M.R. Thompson, G. Tsudik, "An Integrated Solution for Secure Group Communication in Wide-Area Networks", *Proceedings of the 6th IEEE Symposium on Computers and Communications*, Hammamet, Tunisia, July 3-5, 2001, pp 22-28.
- [20] D. Agarwal, M. Lorch, M. Thompson, and M. Perry, "A New Security Model for Collaborative Environments," *Proceedings of the Workshop on Advanced Collaborative Environments*, Seattle, WA, June 22, 2003.