CERIAS Tech Report 2003-31

A FRAMEWORK FOR ROLE-BASED ACCESS CONTROL IN GROUP COMMUNICATION SYSTEMS

Ninghui Li and Cristina Nita-Rotaru

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

A Framework for Role-Based Access Control in Group Communication Systems

Ninghui Li and Cristina Nita-Rotaru
Department of Computer Sciences and CERIAS
Purdue University
250 N. University Street
West Lafayette, IN 47907-2066
{ninghui, crisn}@cs.purdue.edu
FAX: (765) 496-3181

Abstract

In addition to basic security services such as confidentiality, integrity and data source authentication, a secure group communication system should also provide authentication of participants and access control to group resources. While considerable research has been conducted on providing confidentiality and integrity for group communication, less work focused on group access control services. In the context of group communication, specifying and enforcing access control becomes more challenging because of the dynamic and distributed nature of groups and the fault tolerance issues (i.e. withstanding process faults and network partitions).

In this paper we analyze the requirements access control mechanisms must fulfill in the context of group communication and define a framework for supporting fine-grained access control in client-server group communication systems. Our framework combines role-based access control mechanisms with environment parameters (time, IP address, etc.) to provide policy support for a wide range of applications with very different requirements. While policy is defined by the application, its efficient enforcement is provided by the group communication system.

Keywords: role-based access control, group communication systems, distributed systems, fault-tolerance

1 Introduction

Collaboration is seen by many as the driving force behind the progress of our civilization. The extraordinary development of the Internet in recent years has pushed the meaning of collaboration to a new level, allowing people to share ideas and information, communicate effectively and coordinate activities, thus providing support for education, business and personal activities. Examples of such applications include: phone and video conferencing, white-boards, distance-learning applications, games, shared instrument control, as well as command-and-control systems. Although these collaborative applications provide different functionalities, they have in common the need for a communication infrastructure that provides efficient dissemination of messages to multiple parties

(often organized in groups based on a common interest), efficient synchronization mechanisms that allow for coordination and last, but not least, security services. Group communication systems provide such services needed by collaborative applications. Examples of group communication systems include: ISIS [9], Horus [28], Transis [3], Totem [5], RMP [35], Rampart [27], SecureRing [19], Ensemble[31] and Spread [7].

Basic security services needed in such a dynamic peer group setting are largely the same as in point-to-point communication. The minimal set of security services that should be provided by any secure group communication system include: client authentication and access control as well as group key management, data integrity and confidentiality.

While considerable research has been conducted to design scalable and fault-tolerant group key management protocols [4, 30, 36], and to provide confidentiality and integrity [2, 6, 25, 32] for groups, less work focused on the access control services. When group communication systems are used as a common platform by several applications, each with their own security needs, there is an obvious need to control who can join a group, who can send/receive messages, etc. How to provide a flexible access control framework to accommodate the needs of diverse applications, in a group context is a challenging problem because of the dynamic nature of groups, the multi-party aspect and the fault-tolerance issues.

Existing work in addressing access control for group communication focuses on either application flexibility or efficiency of enforcement, at the expense of the other. Thus, on one hand, we have systems that provide flexible application policy by leaving policy specification and enforcement to the application, while on the other hand we have systems that enforce coarse-grained access control policies, ignoring application specific requirements. What is desirable is the best of both worlds: allowing an application to specify its own policies, while having the underlying group communication system to make the enforcement in an efficient manner.

In addition, most existing work in providing access control for groups employs traditional access control schemes such as Access Control Lists (ACL's). These schemes make authorization decisions based on the identity of the requester. However, in decentralized or multi-centric environments, the resource owner and the requester often are unknown to one another, making access control based on identity ineffective or very expensive to maintain. We adopt an approach in which the operations a client is allowed to perform depends on the roles the client is playing in the group, and authenticated attributes of the client are used to determine which roles the client can play in a group.

Our contribution In this paper we focus on defining and providing flexible and fine-grained access control services for a group communication system. More specifically, our contributions are:

- We investigate the requirements access control mechanisms must fulfill in the context of group communication systems and show why identity-based schemes do not provide enough flexibility to support a large class of collaborative applications.

- We design a fine-grained access control framework for group communication systems, based on ideas in Role-Based Access Control [33, 15] and RT [22], a Role-based Trust-management language. We define a set of basic group operations that can be controlled and enforced in our framework. The claim is that any application specific policy can be reduced to a combination of the basic group operations, and instead of

having each individual application to implement and enforce the policy, our framework allows an application to define its specific policies that are translated to basic operations that the group communication systems will enforce in an efficient manner.

- We analyze what are the implications of process (servers and clients) failures and network partitions on the life cycle of a group policy in general, and of an access control policy in particular, and suggest how these issues be addressed.

Roadmap We provide background in group communication systems and access control in Sections 1.1 and 1.2 and discuss the failure and trust models we use in Section 2. In Section 3 we present in details the components for a group policy, while in Section 4 we analyze the implications of process failures and network partitions on the life cycle of the policy. We overview related work in Section 5. Finally, we summarize our work and suggest future work directions in Section 6.

1.1 Client-Server Group Communication Systems

Group communication systems (GCS) are distributed messaging systems that enable efficient communication between a set of processes logically organized into groups and communicating via multicast in an asynchronous environment. More specifically they provide two services: group membership and reliable and ordered message delivery. The membership service provides all members of a group with information about the list of current connected and alive group members (also referred to as a view) and notifies the members about every group change. A group changes because of several reasons. In a fault-free network, group changes are caused by members voluntarily joining or leaving the group. In a faulty environment, group changes can also be caused by faults and network changes. For example, processes may crash or be disconnected, network partitions may occur, preventing members from communicating. When faults are healed, group members can communicate again. All the above events can also trigger changes in group memberships.

The reliable and ordered message delivery includes several services: FIFO – messages are delivered to recipients in the order they were sent by the sender; CAUSAL – messages are delivered to destination in causal order and AGREED – messages are delivered in total order. The strongest service is SAFE delivery, that provides both ordering and reliability guarantees, messages are delivered in total order and delivered to recipients unless they crash.

Group communication systems have been built around a number of different architectural models, such as peer-to-peer libraries, 2- or 3-level middle-ware hierarchies, modular protocol stacks, and client-server. To improve performance, modern group communication systems use a client server architecture where the expensive distributed protocols are run between a set of servers, providing services to numerous clients. In this architecture the client membership service is implemented as a “light-weight” layer that communicates with a “heavy-weight” layer asynchronously using a FIFO buffer. In such a model, an application that uses the group communication system as a communication infrastructure is linked with the group communication client library, thereby getting access to the membership service and the message delivery service provided by the servers. As many group

communication systems are built around a client-server architecture, in this paper we focus on systems using this architecture.

1.2 Access Control

Access control techniques, which govern whether one party can access resources and objects controlled by another party, are useful in protecting the confidentiality, integrity, and availability of information. In security-sensitive GCS-based applications, a number of group operations may need access control, for example, joining a group, sending a message, receive a message, etc.

Role-based access control (RBAC) [15, 16, 33] is an emerging approach to access control that is attracting a great deal of attention both from the research community and from the industry. In RBAC, permissions are associated with roles, and users are granted membership in appropriate roles, thereby acquiring the roles' permissions. In the context of an organization, roles are created for the various job functions and users are assigned and revoked role memberships based on their responsibilities and qualifications. Roles add a level of indirection between users and permissions, thus simplifying the management of the many-to-many relationships between users and permissions.

Traditional access control schemes make authorization decisions based on the identity of the requester. However, in decentralized or multi-centric environments, the resource owner and the requester often are unknown to one another, making access control based on identity ineffective. For example, although a certificate authority may assert that the requester's name is John Q. Smith, if this name is unknown to the access mediator, the name itself does not aid in making an authorization decision. What is needed is information about the rights, qualifications, responsibilities and other characteristics assigned to John Q. Smith by one or more authorities, as well as trust information about the authorities themselves.

Trust management [12, 10, 11, 14, 13, 23, 22, 21] is an approach to access control in decentralized distributed systems with access control decisions based on policy statements made by multiple principals. In trust management systems, statements that are maintained in a distributed manner are often digitally signed to ensure their authenticity and integrity; such statements are called *credentials* or *certificates*. A key aspect of trust management is delegation: a principal may transfer limited authority over one or more resources to other principals.

RT [22, 23, 21] is a family of Role-based Trust-management languages, combining the strengths of Role-Based Access Control (RBAC) [33] and trust-management (TM) systems. A central organizing concept in RT is the notion of *roles* (or *attributes*). We will use the term *attribute* in this paper, to avoid confusion with RBAC roles. An attribute is named by a principal (i.e., a user) and an *attribute term*, which consists of an attribute name and zero or more parameters. For example, StateU.$student(name = "JohnSmith", category = "FullTime")$ is an attribute defined by principal StateU. Each attribute has a value that is a set of principals who have the attribute. Only the principal StateU can define who have the above attribute.

2 Trust and Failure Models

In this section, we discuss the trust model and the failure model we are using in this paper.

Trust Model In client-server group communication systems, a relatively small number of servers running distributed protocols (that maintain membership of groups, reliability and ordering) provide service to numerous clients. In this architecture, a hierarchy of two layers exist, one for the clients (i.e., client libraries running on users' machines) and one for the servers. Clients obtain service by connecting to servers. A trust model has to define the trust relationships within each layer as well as between layers (i.e. do clients trust servers or not).

Given this environment, several trust models are possible, ranging from a model where no entity trusts any other entity for any operation, both within a layer and between layers, to an optimistic model where servers and clients trust each other completely.

The trust model impacts where access control is enforced. If each client is trusted to enforce access control decisions, then access control can be implemented in client libraries and servers can be oblivious to access control enforcement. However, this trust assumption is questionable. As client libraries run on users' machines and may be tampered with by the users, trusting the clients is equivalent to trusting the users and the end hosts, in which case access control is not necessary anymore. Enforcing access control by the clients is also quite inefficient, because every client needs to maintain the access control policy (potentially several of them is he is a member of more than one group) and every message needs to be delivered to the client library and filtered there.

Moreover, in a model where servers are not trusted, a user can trust only the local client library to enforce access control for other parties. This often involves using expensive cryptographic protocols, e.g., secure multiparty computation. Although this model defends against attacks even when servers may be subverted, it is too inefficient to be used in large group communication systems.

In this paper, we adopt the following trust model:

- Servers trust each other: in a client-server group communication system, servers themselves are part of a special group, and a membership protocol is run to detect server faults or network partitions. In order for the system to be bootstrapped correctly, at least a list with potential legitimate servers should be provided to all servers. This is a system administrator's task and less an application task. We assume for this paper that there is a form to authenticate a server when he comes up and verify if he is on the authorized configuration list. In this case, an access control list is an acceptable solution since in general the way these systems are used is first define a servers' configuration that best matches the application performance requirements, and the number of servers is relatively small. Once authenticated and authorized all servers trust each other.

- Clients trust servers to enforce the access control policy. The assumption is acceptable because, in the client-server GCS architecture, clients already trust servers to maintain group membership and to transport, order and deliver group messages, so it seems natural to trust them also for enforcing the access control policy. Furthermore, this will allow for a more efficient enforcement since in numerous cases, the decision can be made locally, diminishing the communication overhead.

- Clients are not trusted (either by the other clients or by servers). Therefore, compromising one client does not compromise the security of the whole system.

We believe that this model achieves a desirable tradeoff between security and the efficiency and simplicity of the access control model.

Failure Model Our model considers a *distributed system* that is composed of a group of processes executing on several computers and coordinating their actions by exchanging messages [8]. The message exchange is conducted via asynchronous multicast and unicast. Messages can be lost or corrupted. We assume that message corruption is masked by a lower layer.

Clients obtain service by connecting to a set of servers distributed on several machines. Clients can connect locally or remotely. We consider that both clients and servers can fail. When a server crashes, all the clients that are obtaining group communication services through that server, will crash too, i.e. they are not redirected to other servers.

Due to network events (e.g., congestion or outright failures) the network can be split into disconnected subnetwork fragments. At the group communication layer, this is referred to as a *partition*. A network partition splits the servers and can potentially split several client groups in different components. While processes are in separate disconnected components they cannot exchange messages.

When a network partition is repaired, the disconnected components merge into a larger connected component, this is referred at the group communication layer as a *merge*. First servers are merged, which in turn can trigger several client groups to be merged.

Byzantine (arbitrary) process failures are not considered in this work.

3 A Policy Model for Access Control in Group Communication Systems

In this section, we study the requirements for specifying access control policies in group communication systems and propose a policy model for doing so. We note that besides policies specifying authentication and access control requirements, there are other kinds of policies in the context of GCS, such as the specification of the encryption algorithm, key length, how often to refresh keys, etc. These kinds of policies have been studied in previous work [17], and are out of the scope of this work.

Our goal is to design a policy model that is flexible enough such that diversified application policies can be specified in this model, and, at the same time, the policy model can be efficiently implemented by the GCS. The basic approach we use is as follows. For any group there is a number of basic operations that can be performed, by principals (entities) based on their role, in a given context. *This mapping between group operations and roles, in a given context defines the access control policy for that group*. This way, instead of having every individual application to deal with access control issues, we can have applications defining specific policies that we can translate to these basic operations, that the GCS knows how to enforce in an efficient manner.

The rest of this section is organized as follows. We begin by considering an example scenario and discussing the various possible access control policies in Section 3.1. In Section 3.2, we describe the group operations that are subjected to access control. We analyze the use of roles in group policies in Section 3.3. We present the policy model in Section 3.4. In Section 3.5 we describe how a policy specified in the model is enforced.

A challenging aspect in the context of group communication systems is maintaining the policy while dealing with dynamic membership determined not only by actions initiated by clients (group members), but also by faults and recoveries. We discuss this in Section 4.

3.1 An Example Scenario

Consider a virtual-classroom application implemented using a GCS. Multiple courses exist in the application. Each course has multiple sessions, each of which is represented by a virtual classroom, implemented as a group. For each course, there are instructors (some courses may have more than one instructors), TA's, and students.

Now consider the life cycle of one virtual classroom (i.e., a group). A classroom should be created only by an authorized user; thus a policy controlling the creation of groups must exist before the creation of a group. We call such a policy, a *template policy*. Each course has a template policy. Since template policies exist outside the context of any group and can be viewed as resources not specific to GCS, standard access control techniques are used to control the creation and modification of template policies. In the simplest case, only the GCS administrator is allowed to create or modify template policies.

A template policy determines, among other things, who can create a group based on the policy. One possible group creation rule is that only the instructors of a course are allowed to create a classroom for the course. An alternative rule is that a TA may also create a classroom. One may also allow the course instructor to delegate to another user, e.g., a guest lecturer, the authority to create a classroom.

After the classroom/group is created, a *group policy* needs to be created. We assume that a group policy is created by copying the template policy. This group policy may then be tailored to suit the need of the current classroom session. Only authorized users are allowed to change the group policy.

Various users may join the classroom in different roles, e.g., instructor, TA, student. Only authorized users should be allowed to join these roles. For joining as a student, different rules are desirable for different cases. One possible rule is that only students who are enrolled in the class may join. Then it may be desirable to allow the instructor or the TA's to admit additional students in special cases. Another rule is that only full-time students registered in the school or the university may join. Yet another rule is that only students who are connecting from certain IP addresses may join (e.g., when taking an exam). Some courses may even allow everyone to join.

Several kinds of communication may be going on simultaneously in the classroom, and they should be subjected to different access control rules. For example, one kind of communication is public; this includes lecture delivered by the instructor, public questions asked by a student and the answers to those questions by the instructor or another member of the classroom. Some classrooms may allow any student to freely ask questions, other classrooms may require approval of the instructor before a student asks a question publicly. There may be private communications between a student and the TA's, e.g., students may be allowed to ask questions privately to the TA's and these questions are answered by one of the TA's. Another use of such private communication channels is for students to submit their answers to a quiz given in class. There may also be private communication between the instructor and the TA's. We model the different kinds of communication within a group by message types. One can also think that the instructor may be allowed to eject a student from the classroom.

We note that group communication systems provide support for such a collaborative application. For example, the Spread [7] group communication system allows for subgroup communication and also for unicast communication within a group, it also allows for any member to be both a sender and a receiver and can also distinguish between different type of communication (messages), while providing different reliability and ordering communication services. In addition, confidentiality and integrity of the data is provided.

3.2 Operations in Groups

From the above scenario description, we can extract the sensitive operations that need access control. The following operations are not performed within the context of a group, they precede the group creation and are not subjected to a group policy or a template policy.

1. *create a group template policy*.
2. *modify a group template policy*.

A comprehensive list of basic operation that apply to a group and are the object of access control is presented below:

1. *create a group*.
2. *modify a group policy*.
3. *join a group*.
4. *send a message of a given type*.
5. *receive a message of a given type*.
6. *eject a user from a group*.
7. *destroy a group*.

The above list does not include the operation of leaving a group, because it is impossible to prevent a client from leaving a group if that is what the client desires. For example any client can effectively leave a group by closing the connection with the server. Therefore, it does not make sense to put access control by leaving a group.

We allow separate control for joining a group, sending a message, and receiving a message to provide support for a wide-range of applications. For example in an application used in military settings, some members of a group may be only allowed to send messages to the group, but not receiving messages. In one such scenario, clients can be wireless and reporting information, and it is desirable to limit the information they receive and store to minimize the damage caused in case of compromise. For some other application group members are only allowed to receive messages, but not sending messages, for example in a conference setting or in the virtual classroom scenario, students auditing the class or just sitting in the class with instructor's permission, can just listen to the lecture.

3.3 Roles in Groups

One approach to specify and enforce access control is to use Access Control Lists (ACL's). Under this approach, a group has an ACL, which includes a set of users and the operations they are allowed to carry out. Such an approach is appropriate when the number of principals and operations is small and static. In general, ACL's have the following disadvantages. First, ACL's can get very large. For example, if every registered student in a

university is allowed to join a classroom, then the ACL would be simply too long. Second, the ACL often duplicates information maintained in other places and its use in a dynamic distributed system will require maintaining its consistency across several sites which can be very difficult. For example, suppose that only students enrolled in the class may join, then the ACL needs to duplicates the registered student lists. When students are adding and dropping courses, maintaining the ACL's accuracy at all servers is difficult and prone to introduce inconsistency in the system.

From the scenario described in Section 3.1, it is clear that what operations a user is allowed to carried out depends upon the roles that the user is playing in a group. For example, although a user may be the instructor of a course, in a guest lecture session she may be playing a TA or a student role.

We distinguish between two kinds of roles: *system roles* and *application roles*. System roles are predefined by the GCS; they exist in every group and have predefined meanings in terms of operations they are allowed to carry out. The following are system roles our framework supports.

- *(group) creator*: this role has at most one member, identifying the user that is the original creator of the group, i.e., the first member of the group. Because of failures, a group's creator role may be empty.

- *(group) controller*: this role has exactly one member, who has full control over a group, including changing the policy for the group and destroying a group.

  When a user creates a group, it is automatically made the creator and the controller of the group. We differentiate the group creator from the group controller for several reasons. First, the creator of a group may want to transfer the controller responsibilities to another member of the group; for example, a TA may create a classroom before the instructor comes and then, after the instructor joins, transfer the role to the instructor. Second, even when the group creator is the original controller, it may crash or leave the group, in which case another member needs to assume the group controller role.

- *(group) member*: any user who joins a group is automatically a member of this role.

Each group may also have a set of application-specific roles, for example, in the virtual classroom scenario, the following application roles may be needed: instructor, TA, student, auditor.

After a user joins a group, in addition to operations such as sending a message or receiving a message, the user may also perform the following operations related to roles: *assume a role*, *drop a role*, *appoint another user to a role*, or *remove another user from a role*. We allow a client to drop a role at its will; however, the other three operations are subjected to access control.

The access control policy of the group defines the operations each role is allowed to carry out. In other words, a group access control policy maps each role to a set of operations. At any time, a user in a group plays a set of roles. When a user is about to perform an action, the roles that the user is playing are used to determine whether the action should be authorized or not.

3.4 A Model for Access Control Policies in GCS

If access control is to be enforced, then the clients need to be authenticated first. Several authentication mechanisms are commonly used. A GCS may provide a username/password based authentication mechanism or may use an external authentication system such as Kerberos [20, 26]. The client may connect with the server through TLS/SSL [1] with client authentication, in which case the client's public key and X.509 [29] Distinguished Name are available. Another solution is having the client to use certificates that document attributes of the clients, e.g., certificates in trust management systems such as RT.

Sometimes what operations a client is allowed to carry out depends on more than the roles of the client; environmental factors may also have effect. For example, a student may be allowed to attend a lecture if she is registered for the class and if the student joins the "class group" in a particular time frame, after the lecture started, she cannot join the group.

To accommodate the diversified authentication methods and the effect of environmental factors in access control, we introduce the notion of contexts. The GCS maintains a *client context* for each connected client and a *group context* for each group. A *group context* consists of a set of name/value pairs, in ways similar to Unix environmental variables. A group context provides environmental information such as current time and group state information (e.g., lecture has began in a classroom). The client context is similar to a group context; it stores information specific to a client, such as the IP address from which the client is connecting and the result of authentication (e.g., authenticated attributes of the client). The combination of roles and context can accommodate a wide range of applications with very diverse policy requirements.

In the following we give our model of group access control policies, an example policy is given in Figure 1. In the model each group policy has a mapping from each role to a set of permissions. In addition, each group policy also has rules controlling who can assume a role and who can remove other users from a role. Each role has a set of *admission rules*, which control who can join as a member of that role and a set of *removal rules*, which control who can remove a user from a role.

Definition 3.1 (Role-Based Group Access Control Model) A group policy consists of the following elements:

- A set T of message types. These are the types of messages in the group.
- A set of V of variables in the group context, and their values.

  T and V determines the set P of permissions: $P = \{\langle\text{send,t}\rangle, \langle\text{receive,t}\rangle \mid t \in T\} \cup \{(set, v) \mid v \in V\}$. A permission $\langle$send,t$\rangle$ authorizes one to send messages of type t to the group. A permission $\langle$receive,t$\rangle$ authorizes one to receive messages of type t. A permission $\langle$set,v$\rangle$ authorizes one to set the value of the variable v in the group context.
- A set AR of application roles.

  The set, R, of all roles is defined to be: $R = AR \cup \{creator, controller, member\}$.
- A binary relation $PA \subseteq R \times P$, which determines the permissions each role has.

- An admission policy AP, which maps each role to a set of admission rules, each rule has the form $\langle c, q, ar \rangle$, where c is a constraint on the group context variables, q is a constraint on the client context, and ar is an approval requirement. When a client tries to assume a role r, all the admission rules are checked one by one until one of them approves the admission. An admission rule $\langle c, q, ar \rangle$ approves the admission if the condition c is satisfied by the current group context, the client satisfies the qualification q, and the approval requirement ar has been met.

  An approval requirement ar has one of the following forms:

  - $true$; this requirement is always met.
  - $vote(r, m, f)$, where $r \in R$ is a role, m is an integer, and f is a rational number in the range $[0..1]$. To meet this requirement, a voting is called among members of the r role, if $n \geq m$ votes are received and among them, $\lceil fn \rceil$ votes are yes; then the requirement is met.
  - $votef(r, f_1, f_2)$, where f_1 and f_2 are both rational numbers in the range $[0..1]$. When the role r has n members, this is equivalent to $vote(r, m = \lceil nf_1 \rceil, f_2)$. This specifies that a certain percentage of members in a role have to vote.

- A removal policy RP, which maps each role to a set of removal rules, each rule has the form $\langle c, ar \rangle$, where c is a constraint on the group context variables and ar is an approval requirement. This rule means that if c is satisfied and the approval requirement is met, then a client can be removed from the role.

In Figure 1 we provide a sample policy for a virtual classroom of the course "CS555". There are two types of messages: $lecture$ and $question$. Lecture messages are sent to everyone in the group, but only the instructor and the TA's can receive question messages. The only state variable in the group context is $ongoing$, which indicates whether the class has begun and is still ongoing. Only the instructor is allowed to set this state variable, i.e., to start the class. We assume that the Registrar issues assertions that who are the instructors, TA's, and registered students for the class. These assertions may be in the form of digital certificates signed by public keys, or information stored in the databases. These assertions are processed by the authentication mechanism, and the results, i.e., authenticated attributes are put in the client context. The admission policy consists of (condition, qualification, approval requirement). Our admission rules for joining the instructor role or the TA role is straightforward. For joining the student role, two admission rules apply. Before the class begins, anyone registered in the class can join with no approval required. Anyone who is a student (but may not be registered in the class) can join the role with the approval the instructor. The approval requirement "vote(instructor, 1, 1)" means that one member of the instructor role has to vote and approve this. We allow an instructor or a TA to be the group creator. The removal policy has only one rule specifying that the instructor can remove any entity from a student role. This allows an instructor to eject a student from the class.

3.5 Enforcing Access Control in Group Communication Systems

When enforcing access control in group communication systems it is very important who is making the access control decision and who is enforcing it. Remember that in such an architecture, service to clients (organized in groups based on common interest) is provided by a set of servers. Many groups can exist in the system.

Message Types: $\{lecture, question\}$
Group Context: {ongoing $\in$ {true, false}}
Roles: {Instructor, TA, Student}

Group Policy: {
    Instructor: {(send, $\{lecture, question\}$, ongoing=true), (receive, $\{lecture, question\}$, ongoing=true)}
    TA: {(receive, $\{lecture, question\}$, ongoing=false), send, $\{lecture\}$, ongoing=false)}
    Student: {(receive, $\{lecture\}$, ongoing=false), (send, $\{question\}$, ongoing=false)}
}

Admission Policy: {
    Instructor: (true, Registrar.instructor(course='CS555'), true)
    TA: (true, Registrar.TA(course='CS555'), true)
    Student: (ongoing=false, Registrar.student(course='CS555'), true),
        (true, Univ.student(), vote(instructor,1,1)

    Group Creator: (true, Instructor $\vee$ TA, true)
    Group Controller: (true, Instructor $\vee$ TA, true)
}
Removal Policy: {
    student: true vote(instructor,1,1)
}

Figure 1. Specifying a Group Policy for a Virtual Classroom Application

One solution is to have access control enforced by group members (clients). Although this approach seems appealing because in fact access control policies are group specific, it decreases the scalability of the system since each group must perform its own enforcement mechanism, sometimes requiring running distributed protocols (e.g. for voting-based schemes). Additionally, when access control is performed by clients, access restrictions such as dropping messages and requests at the receiver are more difficult to provide.

As clients are already trusting the servers for maintaining group membership and delivering and ordering correcting information, the security model is not weakened by requiring the servers to also perform the access control enforcement, the potential benefit being increased scalability and more flexibility of the operations that can be enforced. Based on group's policy, servers must first reach a decision, if access is granted or not, and then enforce that decision. We distinguish between two general approaches:

- local decision: in this case only one server is required to make a decision. For example: when a client requests access to a group during a join operation, the server can make the access control decision locally based on the client's role, group name and group policy.

- distributed (collaborative) decision: in this case the policy requires more than one server to collaborate in order to reach a decision, using for example a voting mechanism, such as a given percentage of group members of a certain role have to approve. This requires a complete view of all the members of all roles of a group, information available to the servers.

Besides decision reaching, another important aspect is who is enforcing the operation. For most of the opera-

tions, the enforcement can be done locally by the server that makes the authorization decision. For other group operations, for example group destroying, the server enforcing the decision can be a different one that the one making the decision. Below we describe in details how enforcement is performed on each of the group operations we presented in Section 3.2.

Client connection A GCS specifies a set of authentication methods that it supports. When a client connects to a server, it may choose to authenticate using a subset of these methods. The server creates a client context for the client and stores the result of these authentication methods in the client context.

Create a group One fundamental operation is creating a group. For an existent group one can assume that a group policy is already in place, and any operation attempted on the group is verified if it complies with the policy. The more interesting problem is how the group policy relates with the group creation, which brings us to a chicken-and-egg problem, what is created first, the group or the policy.

One approach is to assume that the group creator defines the group policy, in other words, group creator is first. Because of the asynchronous communication, several members might think that they are the group creators, which will make correct implementation of this method very difficult without using a central policy server. In addition, a policy defined by the group creator obviously cannot control who can create the group in the first place.

The approach that we use is to assume that the group policy exists before the group is created. When a client creates a group, it specifies the template policy the group is based on. The server then uses the admission rules for the *creator* role in the template to determine whether the client can create the group. In other words, we treat this as a request to assume the *creator* role. After the server determines that the client is authorized to create the group, the group is created, and the client who creates it is made a member of the creator role and of the controller role. Once the creation of the group was allowed, the server can announce it to the rest of the servers.

Modify a group policy In our model, only the group controller is allowed to modify a group policy. For simplicity, we assume that changing the policy does not affect existing memberships. That is, if a client should not a member of the group under the new policy, we do not automatically eject the client from the group. Of course, the controller can always eject such users explicitly.

Destroy a group Our policy model allows only the group controller to destroy a group. When the controller destroys a group, this information is sent to all the servers; they then delete all information about that group and deliver a notification to all other members group that the group was destroyed.

Assume a role Whenever a client tries to assume a role. The admission rules for the role is used, as described in the policy model. The server handling locally the request can check the group context and the client context against the role's admission rules and accept or reject the request. In this case, both the decision and the enforcement are performed locally, if no voting scheme is specified in the policy. A voting may be called if necessary.

Appoint another client to a role This requires the agreement of the client being appointed. After the agreement has been obtained, this is treated as the client trying to assume the role, with the approval (a 'yes' vote) of the appointer already obtained.

Join a group When a client tries to join a group, it specifies also the role the client wishes to assume. This is then treated as the an operation to assume the role.

Send a message of a give type to a group If a client asks a server to send a message of a given type, the roles that the client is a member of are checked. If any of the roles has the permission to send a message of the type; the server sends the message, otherwise the server will drop the message locally.

Receive from a group a message of a given type Unlike the send enforcement that it can be done locally by the server the sender is connected to, enforcing access control for receive is more difficult.

One solution is to require all servers to maintain for each group not only the members of the groups but also their roles and check at send if it needs to forward or not messages to a particular server. This method makes sense only if communication is point-to-point, otherwise, the server will multicast the message anyway. Also, this method does not provide complete enforcement since in some cases, another check must be performed at delivery.

Best approach seems to be to have the servers providing service to the receivers to make the check, which in turn requires each server to know the roles of all clients connected to it. In this case the decision and enforcement occur independently at each receiver which increases the networking traffic by multicasting messages that will only be rejected when received, as well as increases the processing load on all of the machines.

Remove another client from a role This operation should be performed by the server to which the client that is to be removed from a role is connected. The removal rules for the role are checked one by one. If the client initiating the removal by itself does not satisfy the approval requirement, a voting is called. If the operation is successful, the server removes the client's role membership.

Eject a member from a group If all role memberships of a client are removed, the client is ejected from the group.

This operation may require the collaboration of all the servers, since all must to remove the information about the ejected member for that specific group. At the group level a new membership notification will be delivered to the remaining group members informing them about the member(s) who was (were) ejected.

We distinguish between two types if ejection: in one the client is just ejected from a particular group, while a more severe operation will cut any interaction of the client with the system by disconnecting the network communication channel.

Our framework is designed to be very flexible by controlling both send and receive. However, we note that many applications policies can be expressed as either policies on send, or on receive. Whenever, appropriate,

policies should be defined with enforcement on send because this can be more efficiently provided. Enforcing the access control at the receivers adds complexity and loads the networks with traffic that can be avoided.

4 Life Cycle of an Access Control Policy

In the previous section we described how a fine-grained access control policy for group communication can be defined and enforced, in a model where faults do not happen. Unfortunately, this is not the case in the real world where processes can crash, computers can fail, network mis-configurations can happen, or just the network overload can create unusual latencies that can be perceived as network partitions. In this section we examine the life cycle of the policy when the group changes are caused not only by actions voluntarily initiated by clients as dictated by the nature of the application, but also by faults.

The life cycle of a policy is defined by the policy creation and subsequent updates. As described in the previous section we assume that based on an application policy's specifications a group template is generated. The creation and revision of a group template is handled by the administrator of a GCS. Based on the template, a group policy is created when a client allowed to create groups, creates a group based on the template.

An access control policy can be static, in other words it can never change during the life of the group, or it can be dynamic, in which case can suffer changes. In case of dynamic policies, a policy reconciliation must be performed in many cases. As shown in [24], policy reconciliation cannot always be solvable, in which case the question is what happens to the group. For example, current group members that do not satisfy the policy anymore can be excluded from the group. This task can be taken by the group controller. Note that even in the case of static policies, policy reconciliation cannot be avoided when several groups need to be merged together.

We now discuss what happens when two or more group need to be merged. If the groups to be merged have the origins in the same group – in other words they are the result of a network partition that separated a group – and if the group policy is static, the groups should in fact have the same policy so no reconciliation will be necessary. What needs to be addressed is who will become the new group controller, since each policy specifies the same group creator of the original group, but different controllers.

Another case is when groups with the same name were created independently in partitioned components. Some systems uniquely identify groups based only on the group name, so they will try to merge the groups, which, can possibly have different policies. Again, there is no guarantee that a reconciliation is possible. In case a reconciliation is not possible, the servers can decide to destroy the group and inform all clients that the group was destroyed because a policy reconciliation was not possible. If systems identify groups not only by name, then groups created independently in partitioned components will be interpreted as different groups and no merge and policy reconciliation will be performed.

From the previous scenarios it is apparent that the policy framework should specify and provide support for the selection of a new group controller. There are several events that can drive such a need:

- a client or server crashed: The client that crashed was the group controller, or the server that crashes was serving the group controller [1].

[1] Our failure model assumes that clients are not redirected when the server they are connected to crashes, so all the clients connected to that server will fail too.

```
GroupName: CS555

GroupPolicy: ....

FailurePolicy {
        Client:
            PotentialClientGroupController: Professor, TA
        Server
            PotentialServerGroupController: Random, Hash
        Reconciliation:
            Action: DestroyGroup
}
```

Figure 2. Specifying a Failure Group Policy

- a network partition occurred: The group controller will end up only in one network component, while the other components will need to select a new group controller.
- a network merge occurred and policy reconciliation was possible: In this case the new merge group will have to select one of the groups to be merged controller, as the new group controller.

While we want the system to make decisions, we still want the application to specify the policy. Defining how failures should be handled can be defined by the application, of course some default policies can be used, in case an application does not want to deal with it. Faults can affect clients as well as servers, so a failure handling policy should be defined for both.

Below we argue why a failure handling policy is required for both clients and servers. Consider the case of selecting a new group controller. If a group controller already exists, changing the group controller can be achieved by a simple role delegation. In case a group is merged, several legitimate group controllers will exist (one for each subgroup), the "oldest" controller will be selected as the new group controller.

An interesting case is when the group controller failed and there is no authority that can perform the role delegation. In this case we can define an extension of the role of the client as a group controller to the server that he connected to, so the server can temporarily take over the role of group controller and just select deterministically select (acting as a delegator) a new group controller from a list provided by the application. If the application did not provide such a list, this can be perceived as a fatal failure and the server can just decide destroying the group.

Now, consider that the server itself crashed. In this case, the set of servers must decide which one of them will take over the task of selecting the new group controller. This can be done in several ways, the easiest is for example to select deterministically any of the servers (let's say the first). If the application wants to restrict this to a particular set of servers (main campus servers for example), it can provide an ordered set of potential take-over servers or a percentage if a voting policy is desired.

In Figure 2 we provide an example for the group "CS555". Any group member with the role of Instructor or TA can be selected to act as a group controller, in case of client failures. In case of server crashes, only servers $Random$ and $Hash$ can take over. In case of servers we use their names because the access control for servers is not based on roles (see Section 2).

To summarize, the application should specify in case of failures how a new group controller should be chosen and what action should be taken and by whom, in case a reconciliation is not possible. This information will translate into the group's template and then, at group creation time, in the group policy and cannot be changed by anyone, including the group controller.

5 Related Work

There are two major research directions on providing secure communication in the context of groups. The first one aims to provide security services for IP-Multicast and reliable IP-Multicast. Research in this area assumes a model consisting of one sender and many receivers and focuses on high scalability of the protocols. In such systems, receiver access control is considered, but not sender access control. For example in [18], an external access control server with a list of group members is used. Clients contact the server that performs authentication and authorization based on PKI and certificates. Our focus is not on this type of groups, but on peer groups.

The second major direction is represented by peer-groups. Group communication systems fall into this category since they consider a communication model where any member can be both a sender and a receiver. There are several group communication systems that considered access control. The Ensemble secure group communication system [31, 32] assumes the 'fortress' model where an attack can come only from outside. The system uses a symmetric-key based key distribution scheme and uses Access Control List (ACL) as access control mechanism. The ACL is treated as replicated data within the group.

In [2] access control in groups is provided by using an authorization service, Akenti [34], which relies on X509 [29]. The method used is to have all group members registering with the authorization service off-line to obtain a membership certificate signed by the Akenti server, and then when the group membership changes, every member verifies the membership certificate and the personal certificate of every member. The approach relies on identity for access control and provides a coarse granularity for access control.

Relevant to our work, but somehow orthogonal is the Antigone [25, 17] framework. Antigone provides a policy framework that allows flexible application-level group security policies in a more relaxed model than the one usually provided by group communication systems. Policy flavors addressed by Antigone include: re-keying, membership awareness, application message and process failure policies. Every group is arbitrated by a session leader, a model justified by the fact that groups in Antigone are not peer-groups. Moreover, although failures (as process crashes) are detected, the system does not recover from them, so Antigone does not deal with recovering from process crashes and network partitions.

Most of the systems described above provide access control based on identity of participants and do not discuss how failures can affect the enforcement of policies. As oppose to above described schemes our approach is not identity-based. Instead, we take advantage of role-based access control [33, 15] and RT [22], a family of Role-based Trust-management languages, to define a fine-grained access control framework for group communication systems, while discussing the implications of recovering from failures over the access control framework.

6 Conclusions and Future Work

In this paper we have analyzed the requirements access control mechanisms must fulfill in the context of group communication and defined a framework for supporting fine-grained access control for groups. Our framework combines role-based access control mechanisms with environment parameters (time, IP address, etc.) to provide policy support for a wide range of applications with very different requirements. In order to provide both flexible policy and efficient enforcement, we use the group communication servers to decide and enforce access control. We identify the set of all possible group operations that can be controlled and define the group policy as a mapping between roles and operations using context as constraints. In addition, we suggest a way in which failure policy can also be specified by the application.

Several things remain to be addressed in future work. One is to provide a "user-friendly" interface for our framework so that policies can be generated in an automatic way, based on some user specifications. In this paper, we have chosen to focus only on access control; in the future we would like to investigate in an integrated manner all aspects of group policies.

Our final goal is to have an operational system that implements the framework. We are currently in the process of integrating the RT library with the Spread group communication system. We intend to use the resulted system to experiment with several collaborative applications having different group behavior (membership dynamics, type of communication, group sizes) and different access control policies, to assess the power and limitations of the framework.

References

[1] *The TLS Protocol Version 1.0*. Number 2246 in RFC. T. Dierks and C. Allen, 1999. http://www.faqs.org/rfcs/rfc2246.html.

[2] D. A. Agarwal, O. Chevassut, M. R. Thompson, and G. Tsudik. An integrated solution for secure group communication in wide-area networks. In *Proceedings of the 6^{th} IEEE Symposium on Computers and Communications*, Hammamet, Tunisia, July 2001.

[3] Yair Amir, Danny Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pages 76–84, 1992.

[4] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, John Schultz, Jonathan Stanton, and Gene Tsudik. Secure group communication using robust contributory key agreement. In *To appear in Transactions on Parallel and Distributed Systems*, September 2003.

[5] Yair Amir, L. E. Moser, P. M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4):311–342, November 1995.

[6] Yair Amir, Cristina Nita-Rotaru, Jonathan Stanton, and Gene Tsudik. Scaling secure group communication systems: Beyond peer-to-peer. In *the 3^{rd} DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C., April 2003.

[7] Yair Amir and Jonathan Stanton. The Spread wide area group communication system. Technical Report 98-4, Johns Hopkins University, Center of Networking and Distributed Systems, 1998.

[8] Keneth P. Birman. *Building Secure and Reliable Network Applications*. Manning, 1996.

[9] Keneth P. Birman and Robert V. Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.

[10] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.

[11] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems. In *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–210. Springer, 1999.

[12] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.

[13] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[14] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. IETF RFC 2693, September 1999.

[15] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, April 2003.

[16] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role–based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, August 2001.

[17] H. Harney, A. Colegrove, and P. McDaniel. Principles of policy in secure groups. In *Network and Distributed Systems Security*, San Diego, CA, February 2001.

[18] Paul Judge and Mostafa Ammar. Gothic: A group access control architecture for secure multicast and anycast. In *INFOCOM*, 2002.

[19] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, pages 317–326, January 1998.

[20] John Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service (Version 5). RFC-1510, September 1993.

[21] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages (PADL 2003)*, pages 58–73. Springer, January 2003.

[22] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[23] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, February 2003.

[24] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. In *IEEE Symposium on Security and Privacy*, pages 73–87, Oakland, CA, May 2002.

[25] Patrick McDaniel, Atul Prakash, and Peter Honeyman. Antigone: A flexible framework for secure group communication. In *Proceedings of the 8th USENIX Security Symposium*, pages 99–114, August 1999.

[26] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, pages 33–38, September 1994.

[27] Michael K. Reiter. Secure agreement protocols: reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, November 1994.

[28] R. V. Renesse, K.Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39:76–83, April 1996.

[29] ITU-T Rec. X.509 (revised). *The Directory - Authentication Framework*. International Telecommunication Union, 1993.

[30] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. In *Proceedings of ISOC Network and Distributed Systems Security Symposium*, February 2000.

[31] Ohad Rodeh, Ken Birman, and Danny Dolev. Using AVL trees for fault tolerant group key management. Technical Report 2000-1823, Cornell University, Computer Science; Tech. Rep. 2000-45, Hebrew University, Computer Science, 2000.

[32] Ohad Rodeh, Ken Birman, and Danny Dolev. The architecture and performance of security protocols in the Ensemble Group Communication System. *ACM Transactions on Information and System Security*, 4(3):289–319, August 2001.

[33] 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.

[34] Mary R. Thompson, Abdelilah Essiari, and Srilekha Mudumbai. Certificate-based authorization policy in a PKI environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.

[35] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop*, LNCS, page 938, September 1994.

[36] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98*, pages 68–79, 1998.