Comparing Simple Role Based Access Control Models and Access Control Lists

John Barkley
National Institute of Standards and Technology
Gaithersburg MD 20899
(301) 975-3346
jbarkley@nist.gov

August 11, 1997

Abstract

The RBAC metaphor is powerful in its ability to express access control policy in terms of the way in which administrators view organizations. The functionality of simple Role Based Access Control (RBAC) models are compared to access control lists (ACL). A very simple RBAC model is shown to be no different from a group ACL mechanism from the point of view of its ability to express access control policy. RBAC is often distinguished from ACLs by the inclusion of a feature which allows a session to be associated with a proper subset of the roles (i.e., groups in ACL terms) authorized for a user. Two possible semantics for this feature are described: one which requires a similar amount of processing as that required by ACLs, and another which requires significantly more processing than that required by ACLs. In addition, the capability to define role hierarchies is compared to an equivalent feature in ACLs.

1 Introduction

This paper compares simple Role Based Access Control (RBAC) models and access control lists (ACL). RBAC has several advantages over ACLs. Even a very simple RBAC model affords an administrator the opportunity to express an access control policy in terms of the way that the organization is viewed, i.e., in terms of the roles that individuals play within the organization. With RBAC, it is not necessary to translate a natural organizational view into another view in order to accommodate an access control mechanism.

RBAC '97 Fairfax Va USA

In addition, most RBAC models have features which most ACLs do not. In particular, some RBAC models[5][2][3] have role hierarchies where one role can *inherit* another. Much of this paper has been derived from the experiences of the NIST team which implemented RBAC on the World Wide Web (RBAC/Web)[1] for Unix and Windows NT servers[1].

This Introduction describes some concepts needed to discuss access control mechanism implementation. Section 2 briefly describes simple RBAC models and compares them to ACLs. Section 2.2 describes how a very simple RBAC model is no different from an ACL mechanism which supports groups from the point of view of its ability to express access control policy. Section 2.3 discusses the implementation implications of associating sessions with a proper subset of a user's authorized roles. Section 2.4 describes how hierarchies are sometimes implemented in ACLs.

1.1 Implementation Environment

RBAC models are typically independent of the environment in which they may be implemented. For example, RBAC can be embedded in operating systems or database systems, or implemented at the application level. RBAC implementations discussed in this paper assume a network environment. All of the objects which are controlled by RBAC are spread among several servers connected by a network.

Access control mechanisms require that security attributes be kept about users and about objects. User security attributes consist of things like the groups to which the user belongs and the roles authorized for the user. Object security attributes generally consist of the permissions required to perform operations on the object. Access control mechanisms compare user security attributes and object security attributes in order to determine access.

Usually (although not always), object security attributes are kept with the object (e.g., in the header of a file) and the object resides on a single server. Consequently, when an object is accessed, its security attributes are quickly obtained once the object has been located. Changes in object security attributes need only be made at a single location.

However, in a network environment, the up-to-date values of user security attributes must be available to all servers. If user security attributes are kept on a single server, then that single server must be accessed across the network whenever user security attributes are

[1] **Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.**

required by another server. If a copy of user security attributes is kept on each server, then when user security attributes change, those changes must be made on each server.

1.2 Processing Phases

This paper compares the processing required for simple RBAC models and ACLs from the perspective of the processing phases associated with most access control mechanisms:

Administration The Administration phase consists of creating and maintaining user and object security attributes. Administration tools are usually privileged applications. Administration usually occurs the least often of the three phases.

Session The Session phase consists of establishing, changing the characteristics of, and removing sessions. A *session* is a set of processes, called *subjects*, which act on behalf of a user. Session establishment involves authenticating the user, creating one or more subjects, and associating user security attributes with each subject. The Session phase usually occurs more often than the Administration phase and less often than the Enforcement phase.

Enforcement The Enforcement phase consists of comparing the user security attributes associated with the subject (i.e., the subject security attributes) to object security attributes in order to grant or deny access. The Enforcement phase occurs every time a subject attempts to access an object and is usually the most frequently occurring phase of the three.

1.3 Sessions and Up-to-date User Information

The Administration phase defines the rules under which subjects access objects so that when a user is authenticated to a system during the Session phase, a subject is created which accesses objects in the name of the user during the Enforcement phase. Up-to-date values of user security attributes defined during the Administration phase must be available in order for subject security attributes to be created or modified during Session processing. In a network environment, implementations ensure that the Session phase has up-to-date values of user security attributes by one of the following basic approaches:

Uncached User Information User information including security attributes is kept on a single server and that server is accessed during Session processing.

Cached User Information User information including security attributes is kept on each server and Session processing takes place on each server without having to access any other server.

Uncached User Information guarantees that user security attributes used for Session processing are always up-to-date but requires a network communication whenever a session is established. In addition, when a failure occurs on the network or on the server where user attributes are located, no sessions can be processed. On the other hand, Cached User Information does not require network communication when a session is established but does require that the cache be kept consistent with up-to-date user information whenever user information, including user security attributes, changes. Most implementations use Cached User Information on the assumption that the Administration phase occurs much less frequently than the Session phase. Consequently, network use is reduced and servers' overall throughput increased.

2 Implementing Simple RBAC Models

This section briefly describes simple RBAC models and compares their functionality to ACL mechanisms. The ACL mechanism of Windows NT[4] is used as an example to illustrate the comparison.

In general, RBAC and ACL mechanisms require approximately the same amount of processing during the Enforcement phase. However, because of its increased functionality, RBAC can require more processing during the Administration and Session phases. Processing during the Administration phase is usually limited by the ability of an administration tool to respond in a timely manner to requests from the administrator. Significant additional processing during the Session phase and especially during the Enforcement phase can seriously impact the throughput of the entire network of servers.

2.1 Minimal RBAC Model

Sandhu et al.[5] define the $RBAC_0$ Model as the minimal set of characteristics required for an access control mechanism to be considered an RBAC mechanism. The $RBAC_0$ Model has the following components:

1. users, roles, operations, and sessions;
2. role/operation association (many to many);
3. user/role association (many to many);
4. user/session association (one to many, i.e., users may have multiple sessions but a session is only associated with a single user);

5. session/role set association (one to one, i.e., a user session is associated with a subset of the roles authorized for the user and this subset, often referred to as the *active role set* (ARS), may change during the session's lifetime).

During a session, the user may successfully perform any operations permitted by the roles in the current ARS. The ARS may or may not be a proper subset of the set of authorized roles.

Some would require an access control mechanism to have only the first four components of the $RBAC_0$ Model in order to be considered RBAC[6]. In this paper, the $RBAC_M$ ("M" for "Minimal") Model is defined as having the first four components of the $RBAC_0$ Model. In addition, any feature of an RBAC model which provides the capability for the ARS to be a proper subset of the set of authorized roles is referred to as an *Authorized Role Subsetting* feature.

2.2 $RBAC_M$ vs. ACLs

ACLs typically associate an object with a list of users and groups. Associated with each user or group in an ACL for an object is a set of operations which may be performed on that object. An operation on the object may be performed by a user if that user or a group to which that user belongs is listed in the ACL associated with the object and that operation is associated with that user or that group. PASC P1003.1e[7] (formerly know as POSIX.6) and Windows NT[4] are examples of specifications which define ACL mechanisms.

Consider an ACL mechanism, ACL_G, where only groups are permitted as entries in the ACL. ACL_G may have an arbitrary number of groups and there are no restrictions on a user's membership in any group or several groups. To describe an access control policy using ACL_G, an administrator:

1. Creates groups of individuals according to their responsibilities (i.e., every member of a group has the same responsibilities).

2. Associates with these groups the permissions necessary for the individuals in the group to carry out their responsibilities.

To describe an access control policy using $RBAC_M$, an administrator:

1. Creates roles based on the responsibilities necessary to meet the goals of the organization.

2. Associates these roles with the permissions necessary to carry out these roles.

3. Associates these roles to individuals.

ACL_G is equivalent to $RBAC_M$ from the point of view that any access control policy described using ACL_G can be described by $RBAC_M$ and any access control policy described by $RBAC_M$ can be described by ACL_G. This equivalence can be shown by creating a one-to-one correspondence between the groups in the access control policy described using ACL_G and the roles in the access control policy described using $RBAC_M$. Given an access control policy described using ACL_G, the equivalent access control policy can be constructed using $RBAC_M$ by associating a role with the user if that user is a member of the group which maps to that role. Conversely, given an access control policy described using $RBAC_M$, the equivalent access control policy can be constructed using ACL_G by making the user a member of the group which maps to the role if that user is associated with that role. Appendix A provides a more precise description of how these constructions can be accomplished. Note that:

- The functions in Appendix A used to define an access control policy based on $RBAC_M$ or ACL_G express the capability of $RBAC_M$ and ACL_G as a means to represent access control policy. Moreover, these functions mirror the actions taken by an administrator to create the access control policy using $RBAC_M$ or ACL_G.

- The constructions used to show the equivalence only depend on user/role, role/permission, user/group, and group/permission associations in the access control policy representations. How the permissions are represented is immaterial to the constructions as long as the permission representations are the same in both $RBAC_M$ and ACL_G.

Windows NT is an example of a network operating system which supports a group ACL mechanism that includes the functionality of ACL_G. It is possible to implement $RBAC_M$ in such an environment by simply creating tools to administer the RBAC metaphor using the ACL mechanism provided. To accomplish this, the groups of ACL_G become the roles of $RBAC_M$. Such tools can usually be implemented as privileged applications that:

- only require processing during the Administration phase (see section 1.2), and

- require no change to existing privileged system or kernel processes which provide processing during the Session and Enforcement phases.

Not only is it usually possible to implement $RBAC_M$ in such a manner given an ACL mechanism which supports ACL_G, it is also usually possible to implement the role hierarchy, Static Separation of Duty, and Cardinality features of the NIST Model[2][2].

[2]POSIX.1[8] has a *Supplementary Groups* feature. $RBAC_M$