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.

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. $RBAC_M$

2.3 Authorized Role Subsetting

With an ACL mechanism, if a user is a member of a group, then that user is a member of that group for every and all sessions established. In other words, a user is a member of a group at all times and in all circumstances. With the $RBAC_0$ Model[5], each session may be associated with a different ARS. In addition, since an ARS may be a proper subset of a user's authorized roles, a user may not be able to act in all authorized roles in a given session. In other words, a user may not be a *member* of a role at all times and in all circumstances.

If the choice of roles to be used in a session is completely at the discretion of the user, then the concept that each session may be associated with a different ARS is probably acceptable to most security administrators. For example, a user who is also an administrator benefits from having a window open for the user role and, on the same display, a window open for the administrator role. This may help reduce the possibility of user error, e.g., an illegal user action attempted in the administration window is likely carried out without error, whereas, the same illegal user action attempted in the user window causes an error.

However, if the choice of ARS for a session is constrained, as is the case with the Dynamic Separation of Duty feature of the NIST Model[2], the idea that users on their displays may have one window open with one ARS and another window open with a different ARS may not be consistent with a security administrator's idea of "constrained choice." For example, a bank teller is also an account holder at the bank where employed. Bank policy is to constrain employees who are also account holders from being active simultaneously in both their employee and account holder roles. The bank does not consider the situation where two windows, one open for the role teller and one open for the role account holder, are simultaneously open on the same display to be consistent with this policy.

One solution is to restrict a user to a single session at a time. This approach does not increase the processing required during the Session phase. The indication that a user has a session active becomes part of user security attributes which must be referenced during the Session phase anyway.

Another solution is to restrict a user to a single session on a given workstation or to a single session on several workstations at a given location. These approaches are variations of the requirement that a single unique ARS is associated with a given *set* of sessions rather than just a single session. In particular, a session set could consist of all sessions in which case there would be a single unique ARS associated with the user at all times.

From an implementation point of view, this requirement for a single ARS for a session set is significant. It means that:

- the single unique ARS for each user's session set must be available as part of the user's security attributes when a new session for that user is established, and
- whenever a user's ARS changes for any session in a session set, that change must be reflected in all sessions within the session set.

Compare the actions necessary during Session processing for a system implementing the semantic where there can be a different ARS for each session to the actions necessary during Session processing for a system implementing the semantic where the ARS must be the same for all sessions in a session set. First, consider the semantic where there can be a different ARS for each session. When a session is created for a given user or when an ARS is changed for a given user's existing session:

1. the given user's security attributes are obtained,
2. an ARS is created based on constraints and/or user choice, and
3. each subject in the new or existing session is updated with the ARS created.

Now, consider the semantic where there must be the same ARS for all sessions within a session set. When a session is created for a given user or when an ARS is changed for a given user's existing session, in addition to the three actions listed above:

4. the user's session set to which the new or existing session belongs is identified.
5. each subject in each session of the identified session set is updated with the ARS created, and
6. the identified session set in the given user's security attributes is updated with the ARS created.

Thus, to support the semantic where there must be the same ARS for all sessions within a session set, in addition to updating user attributes during Administration processing and updating the ARS for each subject in the current session during Session processing, the user security attributes and the subject security attributes for all subjects in a session set must be updated during the Session phase. An ACL mechanism usually only requires that user security attributes be updated during Administration processing and that subject security attributes for the subjects in the current session be updated during Session processing.

---

and the role hierarchy, Static Separation of Duty, and Cardinality features of the NIST Model[2] can also be implemented as a privileged administration tool in a system that supports POSIX.1. However, POSIX.1 does not support ACL_G and has limited flexibility in associating groups with file permissions.

For example, in Windows NT, when a different ARS is permitted for each Session, there is no significant difference between the processing required during the Session phase for $RBAC_0$ and for ACLs. To implement authorized role (i.e., group) subsetting in Windows NT where each session can have a different ARS, the logon process (*Net Logon*) must be modified. Subject security attributes (called an access token) for the subject associated with the user must be created indicating that the user is a member of only those groups in the ARS.

However, when the same ARS is required for all sessions in a session set, the processing required by $RBAC_0$ during Session processing is significantly greater as compared to ACLs. On Windows NT, a Domain consists of several servers one of which is distinguished as the Primary Domain Controller (PDC) and the others are known as Backup Domain Controllers (BDC). A user logs into any server and subjects (processes) are initiated which act on behalf of that user. In order for each subject in a session set to be updated with a newly created or modified ARS, every subject in the session set in which the new ARS is created or modified must be located on each of the servers in the Domain and the ARS in the subject security attributes of each of those subjects must be updated to reflect the new ARS.

In addition to updating the ARS (i.e., the access token) in all subjects of a session set, the session set's ARS in the user security attributes must be updated as well. Windows NT is an example of the Cached User Information approach to ensuring that user security attributes are kept up-to-date. User security attributes are changed on the PDC and the PDC communicates changes to each of the BDCs. When the ARS is the same for all sessions in a session set, the ARS for the session set in user security attributes must be kept up-to-date throughout the domain. Thus, when an ARS changes for any session in a session set, that ARS must be updated for that session set on the PDC and then communicated to each of BDCs. For anything but a small domain, such activity can seriously impact network throughput and PDC/BDC performance.

2.4 Role Hierarchies

The capability for one role to inherit another role is a common feature of RBAC models, e.g., the Sandhu $RBAC_1$ Model[5], the NIST Model[2], the SQL3 RBAC Model[3]. A role hierarchy is a strict partial ordering[9] (i.e., like "$<$", asymmetric and transitive) on the set of roles. One can think of role inheritance as the capability for one role to be authorized for (or "included in") another role. SQL3 implements role hierarchies in this manner.

The equivalent concept with ACLs is the capability for one group to be a member of another group[3]. In other words, "role a inherits role b" is equivalent to "role a is authorized to perform role b" or "group a is a member of group b." The processing required for role hierarchies during the Session phase is approximately the same for the equivalent feature in ACLs. However, during the Administration phase, the processing may be greater for RBAC since some RBAC Administration tools permit role hierarchies to be managed graphically.

3 Summary

The ability to express access control policy using the very simple RBAC Model $RBAC_M$ is no different from ACL_G which is supported by many ACL mechanisms, e.g., PASC P1003.1e[7] and Windows NT[4]. An RBAC mechanism consisting of $RBAC_M$ and the role hierarchy, Static Separation of Duty, and Cardinality features of the NIST Model[2] can usually be implemented on a system that supports ACL_G. Such an implementation only requires the development of administration tools that only require processing during the Administration phase. An implementation of a simple RBAC model, i.e., $RBAC_M$, $RBAC_0$[5], $RBAC_1$[5], can require approximately the same amount of processing as that required by an ACL implementation. The only significant difference occurs when the RBAC model includes an Authorized Role Subsetting feature with the semantic that all sessions within a set of sessions must have the same ARS.

References

[1] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrilla, and D.R. Kuhn. Role Based Access Control for the World Wide Web. In *20th National Information System Security Conference*. NIST/NSA, 1997.

[2] D. Ferraiolo, J. Cugini, and D.R. Kuhn. Role Based Access Control: Features and Motivations. In *Annual Computer Security Applications Conference*. IEEE Computer Society Press, 1995.

[3] ISO/IEC 9075, (Working Draft) Database Language SQL - Part 2: Foundation. Document ISO/IEC JTC1/SC21 N10489, July 1996.

[4] Karanjit S. Siyan. *Windows NT Server 4 Professional Reference*. New Riders Publishing, Indianapolis, Indiana, 1996.

[5] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role Based Access Control Models. *IEEE Computer*, 29(2), February 1996.

[3] Windows NT Local Groups can contain Domain Groups as members. However, Windows NT Domain Groups cannot have Local Groups or other Domain Groups as members.

[6] Schumann Software Systems Inc. World Wide Web URL - http://www.schumannsoftware.com/.

[7] Draft Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface: Protection, Audit, and Control Interfaces [C Language]. Portable Applications Standards Committee (PASC), IEEE Computer Society.

[8] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) Amendment 1: Realtime Extension [C Language]. IEEE Std 1003.1b-1993.

[9] Patrick Suppes. *Axiomatic Set Theory.* Van Nostrand, Princeton, New Jersey, 1960.

A Equivalence of $RBAC_M$ and ACL_G in Specifying Access Control Policy

Definitions:

Let:

$U = \{u_1, u_2, \ldots, u_{nu}\}$ - a finite set of users
$P = \{p_1, p_2, \ldots, p_{np}\}$ - a finite set of permissions defining permitted operations on objects

role$\in \wp(P)$ - a role is an element in the power set of P, i.e., the set of all subsets of P

$RBAC_M P_{\{U,P\}}$ - an $RBAC_M$ access control policy on U and P, is a function $f_{R\{U,P\}}$ on $U \times \wp(\wp(P))$ ($f_{R\{U,P\}}$ is a function which maps $u_i \in U$ to a set of roles)

group$\in \wp(U)$ - a group is an element in the power set of U

$GP_{\{U,P\}}$ - an ACL_G access control policy on U and P, is a function $f_{G\{U,P\}}$ on $\wp(U) \times \wp(P)$ ($f_{G\{U,P\}}$ is a function which maps a set of users, i.e., a group, to a set of operations)

Equivalence:

Any access control policy which can be specified using ACL_G ($GP_{\{U,P\}}$) can be specified using $RBAC_M$ ($RBAC_M P_{\{U,P\}}$), i.e.,

$$\forall U, P, f_{G\{U,P\}}, \exists f_{R\{U,P\}} \ni (G \in \wp(U), RS = \{PS \mid u \in G, f_{G\{U,P\}}(G) = PS\} \Longrightarrow f_{R\{U,P\}}(u) = RS)$$

The equivalent $RBAC_M$ access control policy specification may be constructed from the ACL_G access control policy specification by initializing the function $f_{R\{U,P\}}$ to the empty set and repeating the following procedure for each user $u \in U$:

1. Initialize the role set RS to the empty set.
2. For each group G, if the user u is a member of group G, add the set of operations PS, where $PS = f_{G\{U,P\}}(G)$, to the role set RS.
3. Add the mapping $u \longrightarrow RS$ to $f_{R\{U,P\}}$.

Any access control policy which can be specified using $RBAC_M$ ($RBAC_M P_{\{U,P\}}$) can be specified using ACL_G ($GP_{\{U,P\}}$), i.e.,

$$\forall U, P, f_{R\{U,P\}}, \exists f_{G\{U,P\}} \ni (PS \in \{PS \mid u \in U, PS \in f_{R\{U,P\}}(u)\} \Longrightarrow G_{PS} = \{u \mid PS \in f_{R\{U,P\}}(u)\}, f_{G\{U,P\}}(G_{PS}) = PS)$$

The equivalent ACL_G access control policy specification may be constructed from the $RBAC_M$ access control policy specification by initializing the function $f_{G\{U,P\}}$ to the empty set and repeating the following procedure for each role PS such that PS is a role for some user u, i.e., PS is an element of the set $\{PS \mid u \in U, PS \in f_{R\{U,P\}}(u)\}$:

1. Initialize the group G_{PS} to the empty set.
2. For each user u, if u has the role PS, i.e., $PS \in f_{R\{U,P\}}(u)$, add the user to the group G_{PS}.
3. Add the mapping $G_{PS} \longrightarrow PS$ to the function $f_{G\{U,P\}}$.