INHERITANCE PROPERTIES OF ROLE HIERARCHIES

W.A. Jansen
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
wjansen@nist.gov

Abstract: *Role Based Access Control (RBAC) refers to a class of security mechanisms that mediate access to resources through organizational identities called roles. A number of models have been published [1, 2 ,3] that formally describe the basic properties of RBAC. One feature of these models is the notion of a role hierarchy, which represents the relationship among roles that are defined in terms of other roles and inherit basic capabilities from them. This paper explores some interesting characteristics of role hierarchies and how they affect basic RBAC properties such as separation of duty*.

Keywords: *Role Based Access Control, Formal Models, Role Hierarchy*

Introduction

A role is an organizational identity that defines a set of allowable actions for an authorized user. Role Based Access Control (RBAC) mechanisms rely on role constructs to mediate a user' s access to computational resources. Typically, the roles within an organization often relate to other roles in terms of their capabilities. Allowing administrators to define roles with respect to other roles, improves efficiency and consistency, especially in organizations that have a large number of roles. The overall set of capability relationships is called a role hierarchy, which can be represented as a directed acyclic graph, where each node represents a role and each arrow between roles represents the "is defined in terms of" relationship. Not all implementations of RBAC include role hierarchies. While many RBAC models do include them [1, 2, 3], they are silent on how basic capabilities are upheld among role hierarchies.

This paper addresses some of the issues surrounding role hierarchies by beginning with a simple RBAC model. Properties of the basic RBAC model are organized along two themes: static properties and dynamic properties. Static properties deal mainly with constraints on role membership, while dynamic properties deal with constraints on role activation [4]. With this perspective, role hierarchies are introduced and several new properties are derived from the basic model. The reader is assumed to be somewhat familiar with RBAC concepts and models.

Model Elements

The main components of the original model are User, Subject, Role, Operation, and Object. These components and the relationships between them are illustrated in Figure 1(a), where a single headed arrow represents a one-to-many, binary relationship between model components, and a double headed arrow represents a many-to-many, binary relationship. The components are also defined below, where "$\wp$" used to represent a subset of the indicated set.

u : User
User = the set of people, both trusted (e.g., administrators) and untrusted, who use the system.

x, y : Subject
Subject = the set of active entities of the system, operating within roles on behalf of individual users.

i, j, k : Role
Role = the set of named duties or job functions within an organization.

op : Operation
Operation = the set of access modes or types permitted on objects of the system.

o : Object
Object = the set of passive entities within the system, protected from unauthorized use.

Permission: $\wp$(Operation × Object)
p, q : Permission
permission = a set of ordered operation/object pairs, <op, o>, where op is an operation that can be applied to object o.

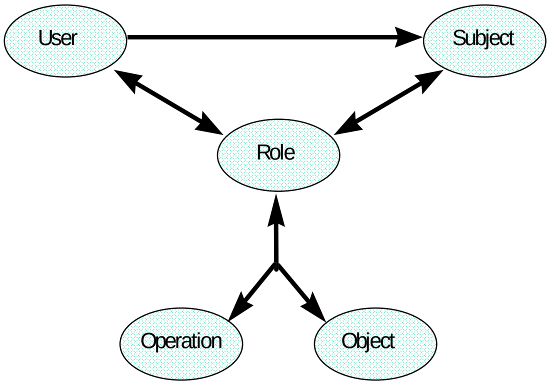

Figure 1(a): Model Components

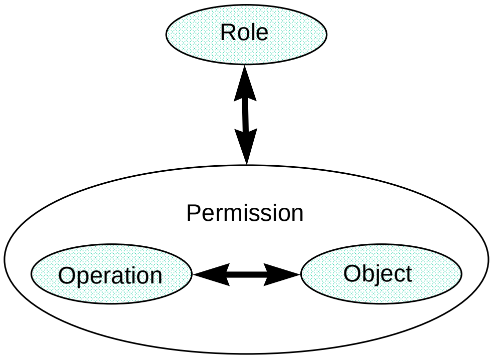

Figure 1(b): Permission Refinement

For notational and conceptual purposes, the ternary relationship between Role, Operation, and Object is refined into a pair of binary relations: one between operations and objects, referred to as Permission; the other between Role and Permission. The reformulation is shown in Figure 1(b), where Permission is used to designate a set of Operation/Object pairs associated with Role elements. The use of Permission conforms with the notion of privilege or permission found in present day information systems [2]. Permission can represent a broad range of access controls ranging from basic read/write/execute rights on files to more extensive administrator rights on operating systems, depending on the context (e.g., an operating system, a database management system, or an application).

Mappings & Relations

Specific mappings further refine the general relationships among the components of the model given in Figures 1(a) and 1(b), and are used to express properties of the model. The set selected closely models the actions of an administrator in assigning permissions to roles and roles to users. However, other equally effective alternatives for the set of mappings exist. The mappings for the basic model are given below, where "↑" is used to represent the power set of the exponent.

> authorized-roles: User → 2↑Role
> authorized-roles[u] = the set of roles authorized for user u.
>
> authorized-permissions: Role → 2↑Permission
> authorized-permissions[i] = the set of permissions authorized for role i.
>
> active-user: Subject → User
> active-user[x] = the user u associated with the subject x.
>
> active-roles: Subject → 2↑Role
> active-roles[x] = the set of roles in which a subject x is active.

Static Properties

Static properties refer to properties of the model that do not involve either the Subject component or mappings from Subject to other basic components (vis., active-user and active-roles). As their name implies, static properties apply early, at role authorization time, and are maintained through role activation. Hence, they are the most fundamental constraints and relationships expressed in the model, and also the strongest. Static properties include cardinality, separation of duty, and operational separation of duty. Static properties are defined in terms of the mappings and relations below.

> membership-limit: Role → N
> membership-limit[i] = the maximum number of users that may be authorized a role; the default value is the total number of system users.
>
> authorized-members: Role → N
> authorized-members[i] = the number of users authorized a given role; i.e.; |{ u ∋ i∈authorized-roles[u] }|, where the cardinality of the set is expressed by the pair of bars "| |" delimiting the defined set.

SSD: $\wp(\text{Role} \times \text{Role})$
SSD = the symmetric set of role pairs <i, j> involved in a Static Separation of Duty (SSD) relationship (i.e., where i and j are mutually exclusive of one another for authorization to the same user); for a symmetric set <i, j> is a member iff <j, i> is also a member.

Mutex-permission: $\wp(\text{Permission} \times \text{Permission})$
mutex-permission = the symmetric set of permission pairs <p, q> mutually exclusive of one another for authorization to an individual role or to the set of roles authorized any user.

SOSD: $\wp(\text{Role} \times \text{Role})$
SOSD = the symmetric set of role pairs <i, j> involved in a Static Operational Separation of Duty (SOSD) relationship, with respect to the permissions in Mutex-permission; i.e., $\forall i \forall j \forall p \forall q\ \text{SOSD} = \{\langle i, j\rangle \ni p \in \text{authorized-permissions}[i] \wedge q \in \text{authorized-permissions}[j] \wedge \langle p, q\rangle \in \text{Mutex-permission}\}$.

Static Cardinality: The number of users authorized a role at any one time cannot exceed the capacity (i.e., membership limits) of the role. For example, a role with a capacity of one would be used exclusively by any single user assigned to it. In terms of the mappings defined, the cardinality of the set of users who are authorized the same role must be less than or equal to the membership limit of that role.

$$\forall i\ \text{authorized-members}[i] \leq \text{membership-limit}[i]$$

Static Separation of Duty: In many organizations, responsibilities are split among multiple roles to make collusion more difficult. A group of roles may be designated through the Static Separation of Duty (SSD) property as mutually exclusive of one another with regard to role authorization. That is, a user may be authorized to only one of the distinct roles so designated. SSD involving multiple roles is expressed pairwise, using the SSD relation. If for example i, j, and k are such roles, then <i, j>, <j, i>, <i, k>, <k, i>, <j, k>, <k, j> are members of SSD.

$$\forall i \forall j \forall u\ i \in \text{authorized-roles}[u] \wedge j \in \text{authorized-roles}[u] \rightarrow \langle i, j\rangle \notin \text{SSD}$$

Static Operational Separation of Duty: The rationale behind SOSD is that business tasks are composed of number of operations, only a subset of which a single user may perform. SOSD is enforced by using permissions to represent allowable subsets of operations on objects involved in business tasks, and designating a group of permissions as mutually exclusive of one another with respect to the roles authorized any single user. Mutually exclusive permissions ensure that no single user may be authorized one or more roles having permissions involved in an SOSD relationship. SOSD is expressed among multiple permissions through pairwise specification of members in a mutual exclusion set, Mutex-permission, which in turn determines the membership of the SOSD relation.

$$\forall i \forall j \forall u \forall p \forall q\ \ i \in \text{authorized-roles}[u] \wedge j \in \text{authorized-roles}[u] \wedge p \in \text{authorized-permissions}[i] \wedge q \in \text{authorized-permissions}[j] \rightarrow \langle p, q\rangle \notin \text{Mutex-permission}$$

or alternatively

$\forall i \forall j \forall u\ \ i \in \text{authorized-roles}[u] \wedge j \in \text{authorized-roles}[u] \qquad \langle i, j\rangle \notin \text{SOSD}$

Dynamic Properties

Dynamic properties complement static properties and refer to properties of the model that involve either Subject or mappings from Subject to other basic components (i.e., active-user and active-roles). Dynamic properties are in a sense weaker than similar static properties, since they apply at role activation time rather than at role authentication time. Weaker doesn't mean undesirable. Instead, it offers an additional degree of flexibility desirable in many contexts. Dynamic properties are used in conjunction with static properties to maintain additional constraints and relationships on the activities that can occur when a role is active (i.e., a subject is active in an authorized role on behalf of a user). Dynamic properties include role activation, cardinality, separation of duty, and operational separation of duty, and utilize the mappings and relations below.

exec: Subject × Operation × Object → {True, False}
exec[x, op, o] = True iff subject x can perform an operation op on object o; otherwise, False.

active-membership-limit: Role → N
active-membership-limit[i] = the maximum number of users that may be active in a role.

active-members: Role → N
active-members[i] = the number of users active in a given role; i.e.; $|\{\ u \ni \forall x\ i \in \text{active-roles}[x] \wedge u = \text{active-user}[x]\}|$.

DSD: $\wp$(Role × Role)
DSD = the symmetric set of role pairs <i, j> involved in a Dynamic Separation of Duty (DSD) relationship (i.e., where i and j mutually exclusive of one another for activation by the same user).

Mutex-perm: $\wp$(Permission × Permission)
mutex-perm = the symmetric set of permission pairs <p, q> mutually exclusive of one another for activation by the same user, simultaneously within different roles.

DOSD: $\wp$(Role × Role)
DOSD = the symmetric set of role pairs <i, j>, involved in a Dynamic Operational Separation of Duty (DOSD) relationship with respect to the permissions in Mutex-perm; i.e., $\forall i \forall j \forall p \forall q$ DOSD = $\{\langle i, j\rangle \ni p \in \text{authorized-permissions}[i] \wedge q \in \text{authorized-permissions}[j] \wedge \langle p, q\rangle \in \text{Mutex-perm}\}$.

Role Activation: A subject cannot be active in a role that is not authorized for its associated user. In general, the active roles of a subject must be a subset of the authorized roles for the user associated with the subject (i.e., active-roles[s] ⊆ authorized-roles[subject-user[s]])

∀x∀i i∈active-roles[x]   i∈authorized-roles[active-user[x]]

Permitted Action: A subject can perform an operation on an object if, and only if, the subject is acting within an active role authorized that permission.

∀x∀op∀o exec[x, op, o] ≡ ∃i (i∈active-roles[x] ∧ p∈authorized-permissions[i] ∧
<op, o>∈p)

Dynamic Cardinality: The number of users active in a role at any one time cannot exceed the dynamic capacity (i.e., active-membership-limit) of the role. This rule, though more difficult to implement than Static Cardinality, seems to be much more desirable, since the role capacity is maintained at activation time as opposed to authorization time. For example, a role with a dynamic capacity of one would allow at most a single role instance to be active at any time, ensuring consecutive use of the role's capabilities by any assigned users.

∀i active-members[i]| active-membership-limit[i]

Dynamic Separation of Duty: A group of roles may be designated as mutually exclusive of one another with regard to role activation, ensuring that at any one time a user may be active in only one of the distinct roles so designated. DSD is a memoryless property insofar as no history of activation is kept for a user. Although DSD roles are prevented from being activated simultaneously by a user, they may be activated consecutively, negating its usefulness in some environments.

∀x∀y∀i∀j i∈active-roles[x] ∧ j∈active-roles[y] ∧ active-user[x]=active-user[y]
<i, j>∉DSD

Dynamic Operational Separation of Duty: A group of permissions may be designated as mutually exclusive of one another with regard to the roles activated by a subject on behalf of any single user. As with DSD, this property is memoryless and of limited usefulness in some environments.

∀x∀y∀i∀j∀p∀q i∈active-roles[x] ∧ j∈active-roles[y] ∧ active-user[x]=active-user[y] ∧
p∈authorized-permissions[i] ∧ q∈authorized-permissions[j]   <p, q>∉Mutex-perm

or alternatively

∀x∀y∀i∀j i∈active-roles[x] ∧ j∈active-roles[y] ∧ active-user[x]=active-user[y]
<i, j>∉DOSD

Role Hierarchy

In order to facilitate administration of access control privileges and constraints, a role may be defined in terms of one or more other roles, with additional characteristics added to distinguish the new role further. A role defined this way is said to contain the roles that comprise its baseline, since it automatically takes on or inherits their collective characteristics as the basis for the new role being defined. Containment is similar to inheritance in object-oriented systems, whereby the properties and constraints of a containing role are inclusive of the properties and constraints of any contained role. Containment is also recursive; one role can contain other roles, which contain others, etc., as illustrated in Figure 2(a). By definition, a role cannot contain itself.

Besides facilitating role administration, containment permits the substitution of role instances. For example, if role A contains role B, then instances of role A are treated as instances of role B for the purpose of access control. In Figure 2(b), users active within instances of Role A have the same capabilities as if they were active within instances of Role B, namely the access allowed through Permission 1 and Permission 2. In addition, users active within Role A also possess an additional capability, access allowed by Permission 3.

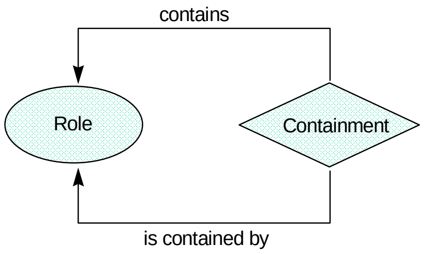

Figure 2(a): Containment Relation

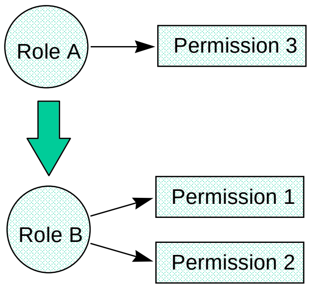

Figure 2(b): Inheritance View of Containment

Containment can be characterized through the notion of effective roles. The effective roles of any given role include that role plus the set of roles contained by that role. For any role, the effective role set represents the capabilities afforded a user authorized the role. In the example above, the effective roles for Role A are Role A and Role B. At times it is also useful to consider the effective roles associated with a given user (or subject), which is the set of roles authorized (active) that user (subject) plus all roles contained by any authorized (active) role. For example, assume there is a role C in addition to the roles defined above, and a user is authorized for both Roles A and C. The effective roles for that user would be Roles A, B, C, and any roles contained by Role C.

Role Hierarchy and its properties are formally defined in terms of the following relation and mapping:

Contains: $\wp(\text{Role} \times \text{Role})$
contains = the set of ordered role pairs <i, j> having a containment relation, written as $i \geq j$, where role i is said to contain role j, or alternatively, role j is said to be contained by i.

effective-roles: Subject ➞ 2↑Role
effective-roles[x] = the union of the set of active roles for a subject, x, together with the set of roles contained by each active role; i.e., {j ∋ j∈active-roles[x]∨ i∈active-roles[x] ∧ i≥j}.

Role Hierarchy: The containment relation defines an irreflexive and transitive relation on Roles, forming a quasi ordering of the elements in the set. The containment relation can also be shown to be antisymmetric. The quasi ordering of Roles is referred to as a role hierarchy.

∀i ¬(i≥i)   (irreflexive)
∀i∀k i≥j ∧ j≥k   i≥k   (transitive)

Role Hierarchy Implications

The introduction of role hierarchy affects the way some of the basic properties are applied across related roles. One example already discussed is effective roles and their relationship to authorized roles. In general, one would expect that a containing role accumulates not only the capabilities of contained roles, but also any constraints and separation of duty relationships. The nature of this form of inheritance is described in the properties below, which regulate these aspects of the role hierarchy.

Permitted Action (modified for hierarchies): With the containment property, the range of operations authorized for a subject is expanded to include those privileges associated with all effective roles. That is, a subject can perform an operation on an object if, and only if, the subject is acting within an effective role authorized that permission.

∀x∀op∀o exec[x, op, o] ≡ ∃i (i∈effective-roles[x] ∧ p∈authorized-permissions[i] ∧ <op, o>∈p)

Cardinality Inheritance: Cardinality constraints, both static and dynamic, are inherited by containing roles. A containing role must be assigned a membership limit less than or equal to that of any contained role.

∀i∀j i≥j   (membership-limit[i]   membership-limit[j]) ∧ (active-membership-limit[i]   active-membership-limit[j])

This property is more easily understood and accurately represented with the redefinition of the authorized-members and active-members functions given below. The original statement of cardinality properties holds under the new definitions.

authorized-members[i] = the number of users authorized a given role or a role that contains the given role; i.e.; |{u ∋ ∃j ((j≥i ∨ j=i) ∧ j∈authorized-roles[u]) }|

active-members[i] = the number of users active in a given role or in a role that contains the given role; i.e.; |{ u ∋ ∀x∃j ((j≥i ∨ j=i) ∧ j∈active-roles[x] ∧ u=active-user[x]) }|

Separation of Duty Hierarchical Consistency: A separation of duty (SD) relationship cannot exist between roles that have a containment relation between them or are contained by another role in common (i.e., a common heir exists). The rationale behind this property is that, by definition, an instance of a containing role is treated the same as an instance of any contained role (i.e., the effective roles of the containing role include the contained role); therefore, the conflict of interest asserted by an SD relationship cannot exist without contradicting the behavior intended by the containment relation. This property holds for each of the separation of duty properties defined, and is expressed in summary fashion below.

$$\forall i \forall j\ (i \quad j \vee \exists k\ (k \quad i \wedge k \quad j) \qquad \langle i, j\rangle \notin DSD \wedge \langle i, j\rangle \notin SSD \wedge \langle i, j\rangle \notin SOSD \wedge \langle i, j\rangle \notin DOSD)$$

Separation of Duty Inheritance: SD relationships are inherited by containing roles. If one role contains another role that has an SD relationship with a third role, then the containing role also has an SD relationship with the third role. This property must hold since a contradiction occurs if the effective roles for the containing role include, in addition to the contained role, the third role. This property holds for each of the separation of duty properties defined.

$$\forall i \forall j \forall k\ i \quad j \wedge \langle j, k\rangle \in SSD \qquad \langle i, k\rangle \in SSD$$

$$\forall i \forall j \forall k\ i \quad j \wedge \langle j, k\rangle \in DSD \qquad \langle i, k\rangle \in DSD$$

$$\forall i \forall j \forall k\ i \quad j \wedge \langle j, k\rangle \in SOSD \qquad \langle i, k\rangle \in SOSD$$

$$\forall i \forall j \forall k\ i \quad j \wedge \langle j, k\rangle \in DOSD \qquad \langle i, k\rangle \in DOSD$$

Summary

Table 1 summarizes the basic properties of the RBAC model specified in this paper, including those properties that are new or affected with the introduction of role hierarchies. For each property indicated by the row and column heading, the entry in the table indicates the presence (i.e., ✓) or absence (i.e., ✗) of a property. Note that some properties (e.g., role activation) by their very nature have only a dynamic variant. For properties involving role hierarchies, the two columns are collapsed into a single column, since these properties generally apply to both static and dynamic variants.

Table 1: Summary of RBAC Properties

Property	**Static**	**Dynamic**
Role Activation	✗	✓
Permitted Action	✗	✓
Cardinality	✓	✓
Separation of Duty	✓	✓

Property	Static	Dynamic
Operational Separation of Duty	✓	✓
Role Hierarchy	✓	
Permitted Action (modified for hierarchies)	✗	✓
Cardinality Inheritance	✓	
Separation of Duty Hierarchical Consistency	✓	
Separation of Duty Inheritance	✓	

Acknowledgments

Many thanks to David Ferraiolo, whose earlier work and subsequent discussions provided the stimulus for this paper.

References

[1] Role-Based Access Control (RBAC): Features and Motivations, David Ferraiolo et Ali., Computer Security Applications Conference, December 1995.

[2] Role-Based Access Control Models, Ravi S. Sandhu et Ali., IEEE Computer, February 1996.

[3] Access Rights Administration in Role-based Security Systems, M. Nyanchama & S. Osborn, in Database Security VIII: Status and Prospects, Elsevier Science B.V. North-Holland, 1994

[4] Separation of Duty in Role-Based Environments, Richard T. Simon & Mary Ellen Zurko, Proceedings of the Second New Security Foundations Workshop, June 1997.