Proceedings of 3rd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, October 22-23, 1998

Role Activation Hierarchies*

Ravi Sandhu
Laboratory for Information Security Technology and
Information and Software Engineering Department
George Mason University

Abstract

The concept of a role hierarchy (that is, partial order) is often included in role-based access control (RBAC) models and systems. In current practice the same hierarchy is typically used for two distinct purposes. Members of a senior role in the hierarchy inherit permissions from juniors. We call this the *usage* (or permission-usage) aspect of role hierarchies. Membership in a senior role also authorizes users to activate junior roles. For purpose of least privilege a user may choose to activate only a junior role on a particular occasion, leaving the senior roles dormant. We call this the *activation* (or role-activation) aspect of role hierarchies.

In this paper we introduce and motivate the idea that there are useful situations where these two hierarchies should not be identical, and the activation hierarchy should extend the inheritance hierarchy. In particular we explore RBAC with respect to read-write access, and its relationship to traditional lattice-based access control or LBAC (also known as mandatory access control). More generally, we consider roles that are required to have dynamic separation of duty.

1 Introduction

Role based access control (RBAC) has emerged as a familiar alternative to classical discretionary and mandatory access controls [SCFY96]. Several models of RBAC have been published and several commercial implementations are available. A common aspect of RBAC is the use of role hierarchies (partial orders) to simplify management of authorizations.

In current practice the same hierarchy is typically used for two distinct purposes. Members of a senior role in the hierarchy inherit permissions from juniors. We call this the *usage* (short for permission-usage) aspect of role hierarchies. We also refer to this as the permission inheritance hierarchy.

Membership in a senior role also authorizes users to activate junior roles. For purpose of least privilege a user may choose to activate one or more junior roles on a particular occasion, leaving the senior roles dormant. We call this the *activation* (or role-activation) aspect of role hierarchies. It should be mentioned that not all RBAC models support role activation, but at the same time it is quite common.

The central contribution of this paper is introduction and motivation of the idea that there are useful situations where usage and activation hierarchies should not be identical. As we will argue an activation hierarchy that extends the usage hierarchy is useful when there are roles in dynamic mutual exclusion. The same user can belong to such roles but cannot activate them simultaneously. We were led to this idea by exploring the relationship between RBAC and traditional lattice-based access control or LBAC (also known as mandatory access control). Our analysis also reveals a close connection between RBAC and LBAC which has not been previously recognized in the literature.

The rest of the paper is organized as follows. Sections 2 and 3 respectively review RBAC and LBAC models. Section 4 discusses how RBAC with read and write permissions can be simulated in LBAC. Section 5 discusses the converse construction and shows how the separation of activation and usage hierarchies is useful in this context. Section 6 argues that this separation is useful whenever we have roles in dynamic mutual exclusion (such as in LBAC, for example). Section 7 formally defines the intuitive concepts discussed so far and explores the relationship of activation hierarchies

*This work is partially supported by grant CCR-9503560 from the National Science Foundation at the Laboratory for Information Security Technology at George Mason University.

All correspondence should be addressed to Ravi Sandhu, ISE Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22030, `sandhu@isse.gmu.edu`, `www.list.gmu.edu`.

to AND-OR roles [Gui95, Gui97]. Section 8 concludes the paper.

2 The RBAC96 Model

This section gives a brief review of the RBAC96 model introduced by Sandhu et al [SCFY96, San97]. Figure 1 illustrates the most general model in this family. For simplicity we use the term RBAC96 to refer to the family of models as well as its most general member.

The top half of figure 1 shows (regular) roles and permissions that regulate access to data and resources. Intuitively, a user is a human being or an autonomous agent, a role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role, and a permission is an approval of a particular mode of access to one or more objects in the system or some privilege to carry out specified actions. The bottom half shows administrative roles and permissions. These are not used in this paper and are included only for sake of completeness.

Roles are organized in a partial order or hierarchy, so that if $x > y$ then role x inherits the permissions of role y, but not vice versa. In such cases, we say x is senior to y. By obvious extension we write $x \geq y$ to mean $x > y$ or $x = y$. Each session relates one user to possibly many roles. The idea is that a user establishes a session (e.g., by signing on to the system) and activates some subset of roles that he or she is a member of.

Like most other RBAC models, RBAC96 has a single hierarchy for usage of permissions (via permission inheritance) and for role activation (in sessions). When a senior role is activated the permissions of all junior roles can be used in that session. At the same time a user assigned to a senior role may activate sessions in which only some of the junior roles are activated.

The use of a single hierarchy for both permission-usage and role-activation purposes is used by almost all existing RBAC models that support role-activation.[1] As we will see in this paper there are good reasons to separate these two aspects of role hierarchies. For consistency we will require that the role activation hierarchy is a superset of the permission usage hierarchy.

As a motivating example, consider a situation where there are two roles Cashier and Manager in a retail store. The Manager role can override and correct errors which the Cashier role is not able to do. A Manager can also serve as a Cashier, but both roles cannot be invoked by a single user at the same time. From the activation viewpoint we would like the Manager role to be senior to the Cashier role, so that a user who is a Manager does not need to be explicitly enrolled in the Cashier role. From the permission-usage viewpoint, the dynamic separation of duty between Cashier and Manager precludes Manager being senior to Cashier. If activation and usage hierarchies are identical, we have an impasse (as noted by Kuhn [Kuh97]). Separating these two hierarchies allows us to resolve this impasse gracefully.

[1] We will return to the treatment of this issue in existing RBAC models in section 6.

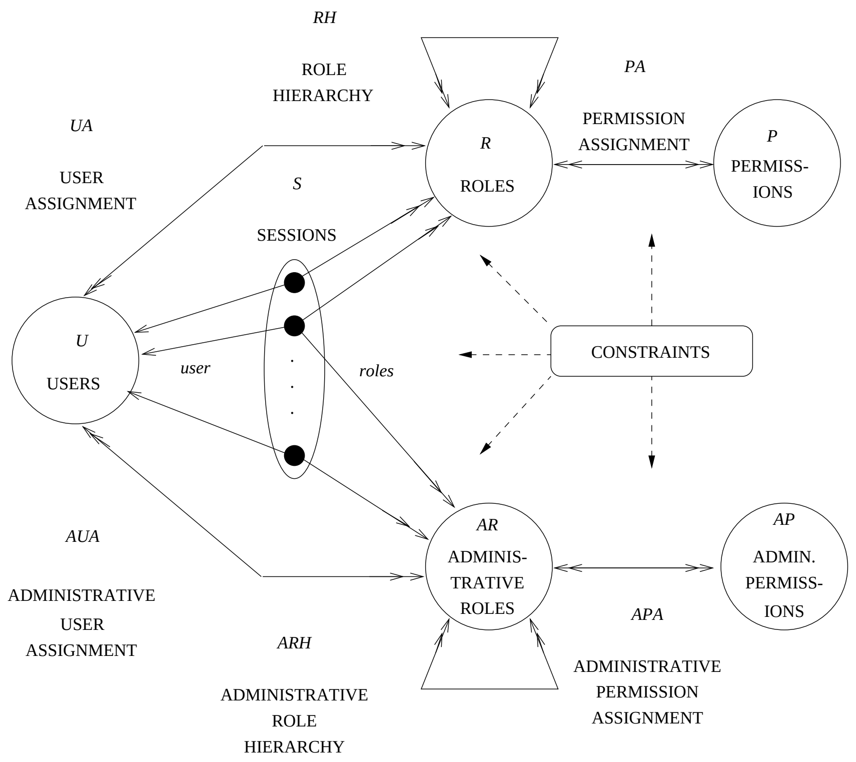
$\star$-property variation	subject s can write object o only if
liberal $\star$-property	$\lambda(s) \leq \lambda(o)$
trusted liberal $\star$-property	$\lambda_w(s) \leq \lambda(o)$
strict $\star$-property	$\lambda(s) = \lambda(o)$
trusted strict $\star$-property	$\lambda_w(s) \leq \lambda(o) \leq \lambda_w(r)$

Table 1: Variations of $\star$-property in LBAC

3 LBAC Models

We were led to the idea of separating these two hierarchies while exploring the relationship between LBAC and RBAC. In this section we identify some commonly recognized variations of LBAC. LBAC is concerned with enforcing one directional information flow in a lattice of security labels [San93] (possibly with exceptions allowed for trusted subjects). LBAC is also known as mandatory access control or MAC. Each subject and object carries a label which we denote by the symbol λ. The security labels form a lattice structure with a partially ordered dominance relation $\geq$ and a least upper bound operator. For read access we have the familiar simple security rule: subject s can read object o only if $\lambda(s) \geq \lambda(o)$.

Simple security for read access is required in all variations of LBAC. For write access there are several variations of the $\star$-property as shown in table 1. The liberal $\star$-property comes from the original formulation of the Bell-LaPadula model [BL75]. In many systems the strict $\star$-property is stipulated to prevent integrity or covert channel problems due to writing up. The trusted liberal $\star$-property was defined by Bell [Bel87]. In this case each subject and object has two labels, λ_r and λ_w with $\lambda_w \leq \lambda_r$ so that simple-security is applied relative to λ_r and liberal $\star$-property to λ_w. We similarly define the trusted strict $\star$-property as shown. The relationship to the strict $\star$-property is easier to see by writing it as $\lambda(s) = \lambda(o) = \lambda(s)$, and then comparing with the trusted strict $\star$-property. Both the strict and trusted strict $\star$-properties adhere to the principle that a subject



- U, a set of users
  R and AR, disjoint sets of (regular) roles and administrative roles
  P and AP, disjoint sets of (regular) permissions and administrative permissions
  S, a set of sessions

- $UA \subseteq U \times R$, user to role assignment relation
  $AUA \subseteq U \times AR$, user to administrative role assignment relation

- $PA \subseteq P \times R$, permission to role assignment relation
  $APA \subseteq AP \times AR$, permission to administrative role assignment relation

- $RH \subseteq R \times R$, partially ordered role hierarchy
  $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
  (both hierarchies are written as $\geq$ in infix notation)

- $user : S \rightarrow U$, maps each session to a single user (which does not change)

  $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

  session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$

- there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.

Figure 1: Summary of the RBAC96 Model

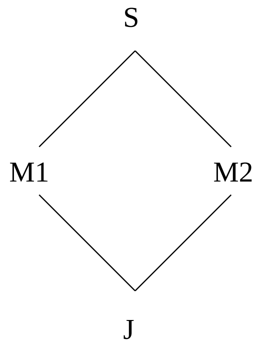


Figure 2: A Role Hierarchy

cannot write what it cannot read.

Like traditional RBAC hierarchies, LBAC also couples label-activation and permission-usage in a single lattice. A user cleared to a high sensitivity in the lattice can activate subjects with lesser sensitivity. Thus a Top-Secret user can activate an Unclassified subject. The read permission is inherited upwards in the security lattice. For the liberal $\star$-property the write permission is inherited downwards, whereas for the strict $\star$-property there is no inheritance of write.

4 Simulating Read-Write RBAC in LBAC

We now consider how RBAC can be simulated in LBAC.[2] In general, RBAC allows for abstract permissions such as credit and debit operations on an account. Both operations require read and write access to the account balance. Since LBAC only considers read and write operations, it is unable to distinguish these. In such cases RBAC cannot be reduced to LBAC. So we limit our scope to RBAC with read and write operations only.

Consider the RBAC hierarchy shown in figure 2. S is the seniormost role and inherits permissions (both read and write) from M1, M2 and J. In particular S can read and write whatever J can, and then some more. M1 and M2 inherit from J, while J being juniormost does not inherit permissions from any other role. A user who is a member of S can create a session in which, say, only J is activated. As discussed earlier this hierarchy serves both purposes of usage and activation.

Suppose we try to simulate this RBAC hierarchy in LBAC.[3] Neither the liberal nor strict $\star$-properties give us the RBAC behavior. Inheritance of read permissions is the same in all these cases, but inheritance of write is very different. In RBAC there is no difference between read and write inheritance. In LBAC with liberal $\star$-property write inheritance is exactly opposite to read inheritance, so J inherits the write power of M1, M2 and S while S inherits nothing. In LBAC with strict $\star$-property there is no write inheritance.

It turns out there is actually a simple construction for solving this problem. Let us use the given RBAC hierarchy of figure 2 as a lattice with trusted strict $\star$-property with following assignment of read and write labels.

role	λ_r	λ_w
S	S	J
M1	M1	J
M2	M2	J
J	J	J

This results in exactly the same read and write inheritance as the original RBAC hierarchy. The construction obviously generalizes to arbitrary read-write RBAC hierarchies.[4]

This is an interesting fact that indicates a strong connection between RBAC and the trusted strict $\star$-property down to system low. We can consider read-write RBAC to be an extreme variation of strict LBAC with trusted write-down to system low. In hindsight, this correspondence can be traced to the different motivations for RBAC and LBAC. RBAC has been largely driven by consideration of authority to users, whereas LBAC is much more concerned with Trojan Horses.

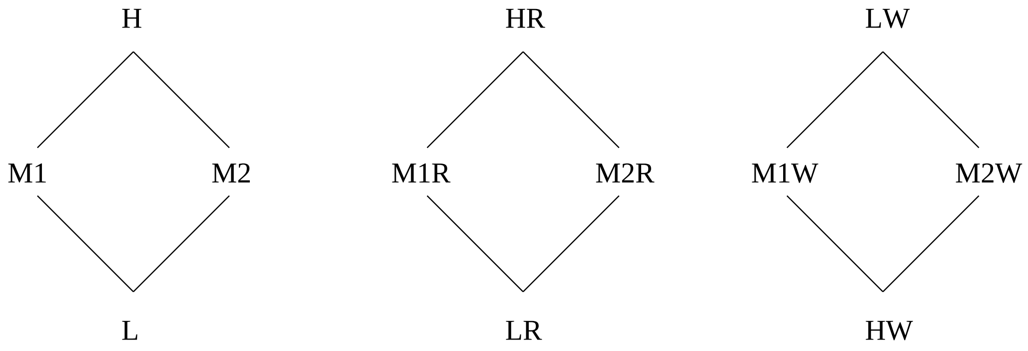
5 Simulating LBAC in RBAC

The simulation of LBAC in RBAC has been considered previously by Nyanchama and Osborn [NO96] and by Sandhu [San96]. The Nyanchama-Osbron construction

[2] Understanding the relationship between different models is a fundamental activity of computer science. It has theoretical significance because such results show the underlying unity between models that at first thought appear to be quite different. It has practical utility because systems which implement one model can then also be used to support other models. Computer science has numerous examples of such results particularly in the area of automata and formal languages. Development of such results in the access control arena can be similarly beneficial.

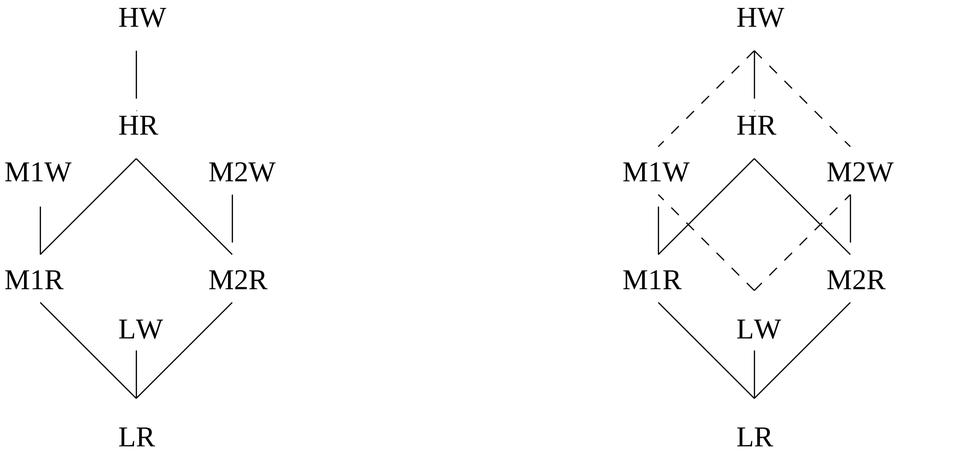
[3] By suitable construction of lattices and modifications to LBAC rules, it is possible to accommodate many read-write configurations that at first sight do not seem to be compatible with LBAC information flow [Bry97, Fol92, San93]. Some of these constructions are intended to handle very general situations and can result in fairly complex lattices. Our objective here is find an "intuitive" and "natural" construction. If we naively use the RBAC hierarchy as a lattice we get completely different read write properties.

[4] Of course, if the RBAC hierarchy is not a lattice the LBAC hierarchy will also be a partial order which is not a lattice.



(a) A lattice (b) Dual role simulation in RBAC

Figure 3: Simulating a lattice using dual read and write roles



(a) Private Write Roles (b) Separate Activation Hierarchy

Figure 4: Simulating strict ⋆-property

does not make use of role hierarchies. Sandhu's constructions show how different LBAC variations, such as in table 1, can be simulated using role hierarchies in RBAC96. Sandhu's construction is shown in figure 3 for the liberal $\star$-property. The original lattice is shown on the left. For each lattice label we need two corresponding read and write roles as shown on the right with both read and write going up in the role hierarchy. The suffixes R and W respectively identify read and write roles. Appropriate constraints are required to ensure that only matched read and write roles are activated in a session. Similar constructions for other variations of LBAC are also given in [San96]. For the strict $\star$-property there is no write hierarchy and the write roles are all incomparable.

In this section we explore the possibility of simpler constructions relating LBAC to RBAC. As we have observed there is a strong connection between RBAC and LBAC with trusted strict $\star$-property. In fact if we are given a lattice with trusted strict $\star$-property we can enforce the identical controls using the lattice as a role hierarchy.

This raises the question of what happens if we have the strict $\star$-property (with no trusted write-down). Consider the lattice of figure 3(a). We can attempt to simulate it using the role hierarchy of figure 4(a). We have separate read and write roles. Each write role is senior to its read role, but only reads are inherited upwards in the hierarchy. Such roles, which have no ancestors, are called private roles [SCFY96]. Constraints are imposed so that users can only be assigned to the write roles (for instance, by requiring the maximum cardinality of read roles to be zero), and only write roles can be activated. Also only one write role can be activated at a time. The role hierarchy of figure 4(a) achieves the effect of strict $\star$-property with respect to permission usage, but not with respect to role activation. A user assigned to M1W can activate the role M1W and will inherit the write permissions of M1W and the read permissions of M1R and LR in that session. However, that user is not automatically authorized to activate LW in another session.

This leads us to suggest that permission-usage and activation hierarchies should be separated. In figure 4(b) we show the activation hierarchy in dashed lines coexisting with the usage hierarchy in solid lines. With this separation a user assigned to M1W can invoke a session with role LW. Similarly, a user assigned to HW can invoke any one of the junior write roles in a session.

6 Dynamic Separation of Duties

In the previous section we have seen how to simulate LBAC with the strict $\star$-property in RBAC using private roles and an enhanced activation hierarchy which extends the permission inheritance hierarchy. We now interpret this construction in terms of dynamic separation of duty. One of the constraints used in the construction was that only one write role can be activated in any session, although in different sessions a user may invoke different write roles. Such a requirement is often called dynamic separation of duties or run-time separation of duties [FCK95, FB97, Kuh97, SZ97].

In general a separate activation hierarchy is useful in dealing with roles that are in dynamic separation of duties. If the roles are not in dynamic separation of duty, we can allow senior roles to inherit from them. With reference to figure 4(b) an activation hierarchy allows users assigned to HW to invoke any one of HW, M1W, M2W or LW with dynamic separation. Suppose the dynamic separation was not required. In that case we could convert the dashed lines to solid ones and simply have a single hierarchy as traditionally done. The net effect would be to have LBAC with trusted strict $\star$-property down to system low.

This separation of activation and usage hierarchies also allows us to resolve an impasse that was noted by Kuhn [Kuh97]. Kuhn observes that it is not possible to have a role A which is senior in the inheritance hierarchy to two or more roles, say B and C, that are in dynamic separation of duty. Dynamic separation of duties is different from static separation only if there are some users who are able to activate B and C (in different sessions). There is no means to assign these users to a common senior role A because activation of A violates dynamic separation of duty with respect to B and C.[5] Thus the common users must be explicitly made members of B and C. This goes against the basic motivation of RBAC to reduce administrative complexity. By bringing in a distinct activation hierarchy that extends the inheritance hierarchy we can successfully resolve this impasse.

This leads us to assert the following principle.

> An activation hierarchy can extend beyond the permission-inheritance hierarchy to roles that are stipulated to have dynamic separation of duty.

[5] We could constrain A so that it cannot be activated, but this is not a general solution.

7 Formal Definitions and Relation to AND-OR Roles

The formal definitions for RBAC96 were summarized earlier in figure 1. We formally define the activation hierarchy, written $\succeq$, to be an extension of the inheritance hierarchy, written $\geq$, as follows.

Definition 1 The activation hierarchy $\succeq$ is a partial order on the set of roles R and on the set of administrative roles AR, which extends the inheritance hierarchy $\geq$ (so that $\geq$ is a subset of $\succeq$). We write x$\hat{\succeq}$y to mean that x$\succeq$y and x$\not\geq$y. □

In terms of RBAC96 we need to modify the following requirement concerning the roles activated in a session.

> $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

Since role activation is governed by the activation hierarchy, this requirement is recast in terms of $\succeq$ as follows.

> $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \succeq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

Note that the following requirement regarding permission inheritance in a session remains unchanged.

> session s_i has the permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$

With these changes the modified model, which we call ERBAC96 (extended RBAC96), has an activation hierarchy that extends the inheritance hierarchy.

With reference to figure 4(b) the dashed lines indicate the $\hat{\succeq}$ relation, that is roles which are related by the activation hierarchy but not by the inheritance hierarchy. Since the four write roles are in dynamic mutual exclusion we stipulated the constraint that only one of these can be activated in a session. Following the general approach of RBAC96 we do not make this constraint part of our basic model but leave it to be introduced explicitly as needed.

In figure 4(b) the roles related by $\hat{\succeq}$ are maximal roles with respect to the inheritance hierarchy (that is, they have no seniors with respect to $\geq$). Figure 5(a) shows a different situation where B is not a maximal role. A user who is a member of role A can activate various combinations of roles in a single session as follows: A, AD, AE, ADE, BD, BE, BDE, D, E, and DE.[6] Figure 5(b) shows a situation where roles A and C inherit permissions from D and E, but B does not. At the same time members of B do have ability to activate D or E. This is an acceptable situation in ERBAC96.[7]

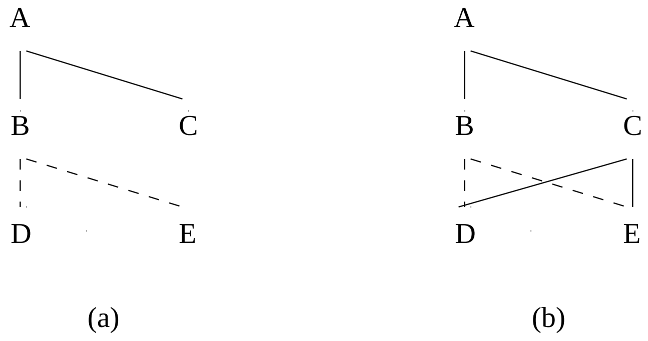


Figure 5: Activation hierarchies

Relationship to AND-OR Roles

Guiri [Gui95, Gui97] has proposed an activation hierarchy based on AND and OR roles. In context of figure 5(a), A is an AND role consisting of AND(B,C). OR roles in Guiri's model are really exclusive-OR roles because exactly one of them can be activated. Let us interpret B as an OR role consisting of OR(D,E). This means that if B is activated one of D or E must be activated.[8] In this case if A is activated, one of D or E must also be activated.

Guiri's AND-OR model is easily simulated in ERBAC96. AND roles correspond to the inheritance hierarchy and OR roles to the activation hierarchy, with the requirement that if a role is activated all OR roles junior to it in the activation hierarchy must have exactly one alternative activated. Guiri's model can thus be interpreted as a special case of ERBAC96 with an activation hierarchy that extends the inheritance hierarchy in a particularly constrained manner.

8 Conclusion

In this paper we have shown that it is useful to have a separate role activation hierarchy which extends the permission-usage hierarchy. In most RBAC models there is a single hierarchy that serves both purposes.

[6]If D and E are stipulated to be in dynamic mutual exclusion the combinations having both of them will not be allowed.

[7]However, if D and E are stipulated to be in dynamic mutual exclusion they cannot have common seniors in the inheritance hierarchy and this situation will be prohibited.

[8]Guiri's model also includes null roles, so OR(D,E,null) means that at most one of D or E can be activated but activation is not mandatory.

Distinguishing the two hierarchies is useful when roles in dynamic separation of duties need to have common seniors in the activation hierarchy, but cannot have common seniors in the permission-usage hierarchy. Separate hierarchies are therefore called for in models that support dynamic separation of duties.

While exploring these issues we have observed a close connection between LBAC and RBAC. We can think of read-write RBAC as LBAC with trusted strict $\star$-property down to system low. Conversely we can view LBAC with strict $\star$-property as a form of read-write RBAC with dynamic separation of duties with respect to write roles.

References

[Bel87] D.E. Bell. Secure computer systems: A network interpretation. In *Proceedings of 3rd Annual Computer Security Application Conference*, pages 32–39, 1987.

[BL75] D.E. Bell and L.J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford, MA, March 1975.

[Bry97] Cyrian Bryce. Security engineering of lattice-based policies. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 195–207, Rockport, Mass., June 1997.

[FB97] David Ferraiolo and John Barkley. Specifying and managing role-based access control within a corporate intranet. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 77–82. ACM, Fairfax, VA, November 6-7 1997.

[FCK95] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48, New Orleans, LA, December 11-15 1995.

[Fol92] Simon Foley. Aggregation and separation as non-interference properties. *The Journal Of Computer Security*, 1(2):159–188, 1992.

[Gui95] Luigi Guiri. A new model for role-based access control. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 249–255, New Orleans, LA, December 11-15 1995.

[Gui97] Luigi Guiri. Role-based access control: A natural approach. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[Kuh97] D. Richard Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 23–30. ACM, Fairfax, VA, November 6-7 1997.

[NO96] Matunda Nyanchama and Sylvia Osborn. Modeling mandatory access control in role-based security systems. In *Database Security VIII: Status and Prospects*. Chapman-Hall, 1996.

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[San96] Ravi S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In Elisa Bertino, editor, *Proc. Fourth European Symposium on Research in Computer Security*. Springer-Verlag, Rome, Italy, 1996. Published as *Lecture Notes in Computer Science, Computer Security–ESORICS96.*

[San97] Ravi Sandhu. Rationale for the RBAC96 family of access control models. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[SCFY96] 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.

[SZ97] R. Simon and M. Zurko. Separation of duty in role-based environments. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 183–194, Rockport, Mass., June 1997.