Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies

SYLVIA OSBORN
The University of Western Ontario
and
RAVI SANDHU and QAMAR MUNAWER
George Mason University

Access control models have traditionally included mandatory access control (or lattice-based access control) and discretionary access control. Subsequently, role-based access control has been introduced, along with claims that its mechanisms are general enough to simulate the traditional methods. In this paper we provide systematic constructions for various common forms of both of the traditional access control paradigms using the role-based access control (RBAC) models of Sandhu et al., commonly called RBAC96. We see that all of the features of the RBAC96 model are required, and that although for the mandatory access control simulation, only one administrative role needs to be assumed, for the discretionary access control simulations, a complex set of administrative roles is required.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Management, Security

Additional Key Words and Phrases: Role-based access control, mandatory access control, lattice-based access control, discretionary access control

Osborn's research was supported by the Natural Sciences and Engineering Research Council of Canada. The research of Sandhu and Munawer was partially supported by the National Science Foundation, USA.
Authors' addresses: S. Osborn, Dept. of Computer Science, The University of Western Ontario, London, ON N6A 5B7, Canada; email: sylvia@csd.uwo.ca; R. Sandhu and Q. Munawer, Laboratory for Information Security Technology, MS4A4, Information and Software Engineering Department, George Mason University, Fairfax, VA 22030; email: sandhu@gmu.edu.

1. INTRODUCTION

Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access controls (see, for example, Proceedings of the ACM Workshop on Role-Based Access Control, 1995-2000). In RBAC, permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles can be created for the various job functions in an organization and users then assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed.

An important characteristic of RBAC is that by itself it is policy neutral. RBAC is a means for articulating policy rather than embodying a particular security policy (such as one-directional information flow in a lattice). The policy enforced in a particular system is the net result of the precise configuration and interactions of various RBAC components as directed by the system owner. Moreover, the access control policy can evolve incrementally over the system life cycle, and in large systems it is almost certain to do so. The ability to modify policy to meet the changing needs of an organization is an important benefit of RBAC.

Traditional access control models include mandatory access control (MAC), which we shall call lattice-based access control (LBAC) here [Denning 1976; Sandhu 1993], and discretionary access control (DAC) [Lampson 1971; Sandhu and Samarati 1994; 1997]. Since the introduction of RBAC, several authors have discussed the relationship between RBAC and these traditional models [Sandhu 1996; Sandhu and Munawer 1998; Munawer 2000; Nyanchama and Osborn 1994; 1996]. The claim that RBAC is more general than all of these traditional models has often been made. The purpose of this paper is to show how RBAC can be configured to enforce these traditional models.

Classic LBAC models are specifically constructed to incorporate the policy of one-directional information flow in a lattice. This one-directional information flow can be applied for confidentiality, integrity, confidentiality and integrity together, or for aggregation policies such as Chinese Walls [Sandhu 1993]. There is nonetheless strong similarity between the concept of a security label and a role. In particular, the same user cleared to, for example, Secret, can on different occasions login to a system at Secret and Unclassified levels. In a sense the user determines what role (Secret or Unclassified) should be activated in a particular session.

This leads us naturally to ask whether or not LBAC can be simulated using RBAC. If RBAC is policy neutral and has adequate generality it should indeed be able to do so, particularly since the notion of a role and the level of a login session are so similar. This question is theoretically significant because a positive answer would establish that LBAC is just one instance of RBAC, thereby relating two distinct access control models that have been developed with different motivations. A positive answer is also practically significant, because it implies that the same Trusted Computing Base can be configured to enforce RBAC in general and LBAC in particular. This addresses the long held desire of multilevel security advocates that technology which meets needs of the larger commercial marketplace be applicable to LBAC. The classical approach to fulfilling this desire has been to argue that LBAC has applications in the commercial sector. So far this argument has not been terribly productive. RBAC, on the other hand, is specifically motivated by needs of the commercial sector. Its customization to LBAC might be a more productive approach to dual-use technology.

In this paper, we answer this question positively by demonstrating that several variations of LBAC can be easily accommodated in RBAC by configuring a few RBAC components.[1] We use the family of RBAC models recently developed by Sandhu et al. [1996; 1999] for this purpose. This family is commonly called the RBAC96 model. Our constructions show that the concepts of role hierarchies and constraints are critical to achieving this result.

Changes in the role hierarchy and constraints lead to different variations of LBAC. A simulation of LBAC in RBAC was first given by Nyanchama and Osborn [1996]; however, they do not exploit role hierarchies and constraints and cannot handle variations so easily as the constructions of this paper.

Discretionary access control (DAC) has been used extensively in commercial applications, particularly in operating systems and relational database systems. The central idea of DAC is that the owner of an object, who is usually its creator, has discretionary authority over who else can access that object. DAC, in other words, involves owner-based administration of access rights. Whereas for LBAC, we do not need to discuss a complex administration of access rights, we will see that for DAC, the administrative roles developed in Sandhu et al. [1999] are crucial. Because each object could potentially be owned by a unique owner, the number of administrative roles can be quite large. However, we will show that the role administration facilities in the RBAC96 model are adequate to build a simulation of these sometimes administratively complex systems.

The rest of this paper is organized as follows. We review the family of RBAC96 models due to Sandhu et al. [1996] in Section 2. This is followed by a quick review of LBAC in Section 3. The simulation of several LBAC variations in RBAC96 is described in Section 4. This is followed by a brief discussion in Section 5 of other RBAC96 configurations which also satisfy LBAC properties. Section 6 introduces several major variations of DAC. In Section 7 we show how each of these variations can be simulated in

[1]It should be noted that RBAC will only prevent overt flows of information. This is true of any access control model, including LBAC. Information flow contrary to the one-directional requirement in a lattice by means of so-called covert channels is outside the purview of access control per se. Neither LBAC nor RBAC addresses the covert channel issue directly. Techniques used to deal with covert channels in LBAC can be used for the same purpose in RBAC.

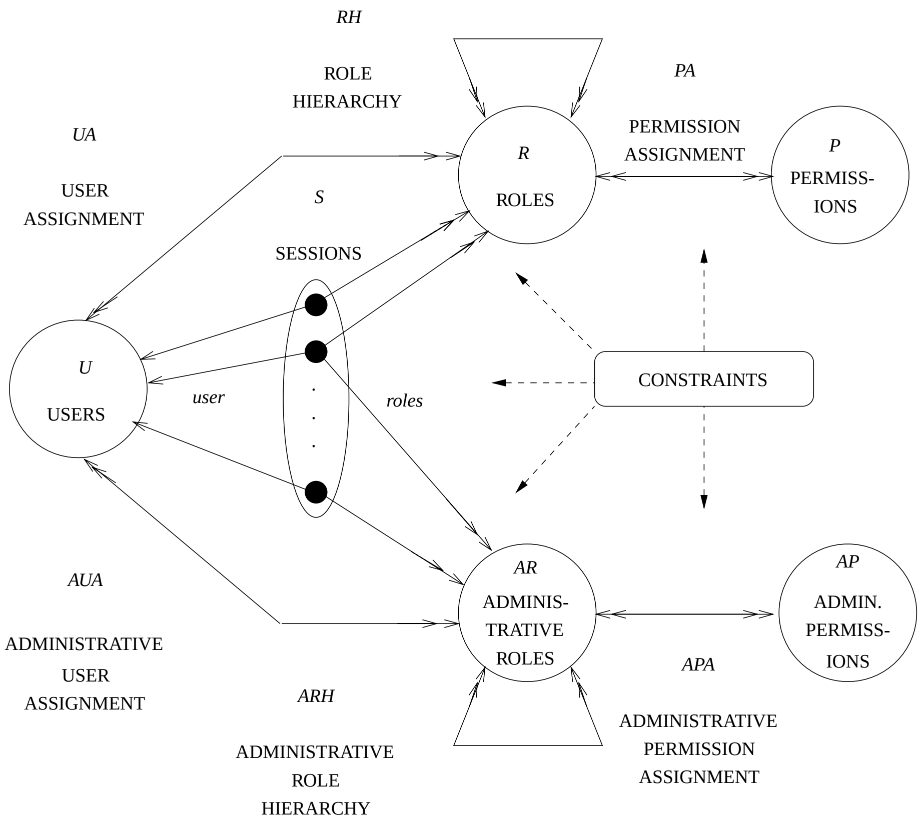

Fig. 1. The RBAC96 model.

RBAC96. Section 8 summarizes the results. Preliminary versions of some of these results have appeared in Sandhu [1996], Sandhu and Munawer [1998], Nyanchama and Osborn [1996], and Osborn [1997].

2. RBAC MODELS

A general RBAC model including administrative roles was defined by Sandhu et al. [1996]. It is summarized in Figure 1. The model is based on three sets of entities called users (U), roles (R), and permissions (P). 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.

The *user assignment* (UA) and *permission assignment* (PA) relations of Figure 1 are both many-to-many relationships (indicated by the double-headed arrows). A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permissions, and the same permission can be assigned to many roles. There is a partially ordered *role* *hierarchy* RH, also written as $\geq$, where $x \geq y$ signifies that role x inherits the permissions assigned to role y. In the work of Nyanchama and Osborn [1994; 1996; 1999], the role hierarchy is presented as an acyclic directed graph, and direct relationships in the role hierarchy are referred to as *edges*. Inheritance along the role hierarchy is transitive; multiple inheritance is allowed in partial orders.

Figure 1 shows a set of sessions S. Each session relates one user to possibly many roles. Intuitively, a user establishes a session during which the user activates some subset of roles that he or she is a member of (directly or indirectly by means of the role hierarchy). The double-headed arrow from a session to R indicates that multiple roles can be simultaneously activated. The permissions available to the user are the union of permissions from all roles activated in that session. Each session is associated with a single user, as indicated by the single-headed arrow from the session to U. This association remains constant for the life of a session. A user may have multiple sessions open at the same time, each in a different window on the workstation screen for instance. Each session may have a different combination of active roles. The concept of a session equates to the traditional notion of a *subject* in access control. A subject (or session) is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions active at the same time.

The bottom half of Figure 1 shows administrative roles and permissions. RBAC96 distinguishes roles and permissions from administrative roles and permissions respectively, where the latter are used to manage the former. Administration of administrative roles and permissions is under control of the chief security officer or delegated in part to administrative roles. The administrative aspects RBAC96 elaborated in Sandhu et al. [1999] are relevant for the DAC discussion in Section 6. For the purposes of the LBAC discussion, we assume a single security officer is the only one who can configure various components of RBAC96.

Finally, Figure 1 shows a collection of *constraints*. Constraints can apply to any of the preceding components. An example of constraints is mutually disjoint roles, such as purchasing manager and accounts payable manager, where the same user is not permitted to be a member of both roles.

The following definition formalizes the above discussion.

Definition 1. The RBAC96 model has the following components:

- 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

- $PA \subseteq P \times R$, a many-to-many permission to role assignment relation

  $APA \subseteq AP \times AR$, a many-to-many permission to administrative role assignment relation

- $UA \subseteq U \times R$, a many-to-many user to role assignment relation
  $AUA \subseteq U \times AR$, a many-to-many user to administrative role assignment relation
- $RH \subseteq R \times R$, a partially ordered role hierarchy
  $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
  (both hierarchies are written as $\geq$ in infix notation)
- $user : S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session's lifetime),
  $roles : S \rightarrow 2^{R \cup AR}$ 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.

3. LBAC (OR MAC) MODELS

Lattice based access control is concerned with enforcing one directional information flow in a lattice of security labels. It is typically applied in addition to classical discretionary access controls, but in this section we will focus only on the MAC component. A simulation of DAC in RBAC96 is found in Section 7. Depending upon the nature of the lattice, the one-directional information flow enforced by LBAC can be applied for confidentiality, integrity, confidentiality and integrity together, or for aggregation policies such as Chinese Walls [Sandhu 1993]. There are also variations of LBAC where the one-directional information flow is partly relaxed to achieve selective downgrading of information or for integrity applications [Bell 1987; Lee 1988; Schockley 1988].

The mandatory access control policy is expressed in terms of security labels attached to subjects and objects. A label on an object is called a *security classification*, while a label on a user is called a *security clearance*. It is important to understand that a Secret user may run the same program, such as a text editor, as a Secret subject or as an Unclassified subject. Even though both subjects run the same program on behalf of the same user, they obtain different privileges due to their security labels. It is usually assumed that the security labels on subjects and objects, once assigned, cannot be changed (except by the security officer). This last assumption, that security labels do not change, is known as *tranquillity*. (Non-tranquil LBAC can also be simulated in RBAC96 but is outside the scope of this paper.) The security labels form a lattice structure as defined below.

Definition 2. **(Security Lattice)** There is a finite lattice of security labels $\mathcal{SC}$ with a partially ordered dominance relation $\geq$ and a least upper bound operator.

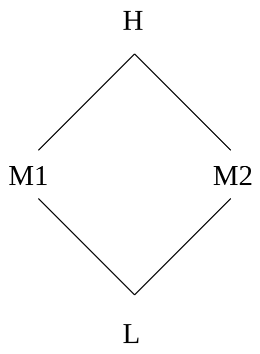

Fig. 2. A partially ordered lattice.

An example of a security lattice is shown in Figure 2. Information is only permitted to flow upward in the lattice. In this example, H and L respectively denote high and low, and M1 and M2 are two incomparable labels intermediate to H and L. This is a typical confidentiality lattice where information can flow from low to high but not vice versa.

The specific mandatory access rules usually specified for a lattice are as follows, where λ signifies the security label of the indicated subject or object.

Definition 3. **(Simple Security Property)** Subject s can read object o only if $\lambda(s) \geq \lambda(o)$.

Definition 4. **(Liberal *-property)** Subject s can write object o only if $\lambda(s) \leq \lambda(o)$.

The *-property is pronounced as the star-property. For integrity reasons sometimes a stricter form of the *-property is stipulated. The liberal *-property allows a low subject to write a high object. This means that high data may be maliciously or accidently destroyed or damaged by low subjects. To avoid this possibility we can employ the strict *-property given below.

Definition 5. **(Strict *-property)** Subject s can write object o only if $\lambda(s) = \lambda(o)$.

The liberal *-property is also referred to as write-up and the strict *-property as non-write-up or write-equal.

In variations of LBAC, the simple-security property is usually left unchanged as we will do in all our examples. Variations of the *-property in LBAC whereby the one-directional information flow is partly relaxed to achieve selective downgrading of information or for integrity applications [Bell 1987; Lee 1988; Schockley 1988] will be considered later.

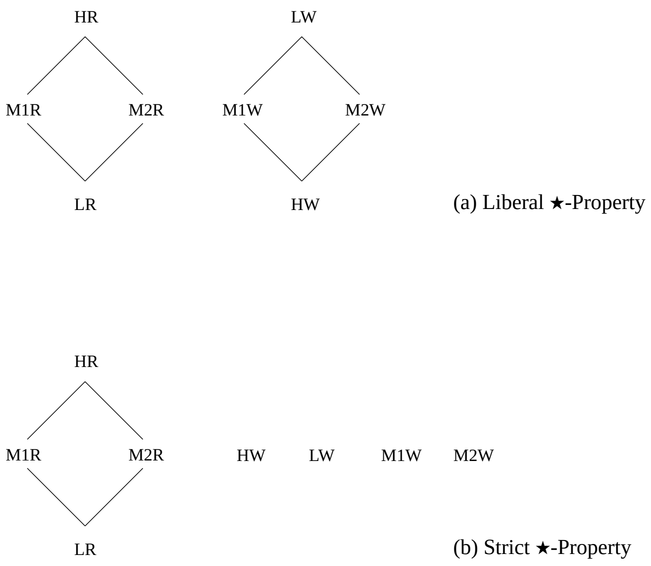

Fig. 3. Role hierarchies for the lattice of Figure 2.

4. CONFIGURING RBAC FOR LBAC

We now show how different variations of LBAC can be simulated in RBAC96. It turns out that we can achieve this by systematically changing the role hierarchy and defining appropriate constraints. This suggests that role hierarchies and constraints are central to defining policy in RBAC96.

4.1 A Basic Lattice

We begin by considering the example lattice of Figure 2 with the liberal *-property. Subjects with labels higher up in the lattice have more power with respect to read operations but have less power with respect to write operations. Thus, this lattice has a dual character. In role hierarchies subjects (sessions) with roles higher in the hierarchy always have more power than those with roles lower in the hierarchy. To accommodate the dual character of a lattice for LBAC we will use two dual hierarchies in RBAC96, one for read and one for write. These two role hierarchies for the lattice of Figure 2 are shown in Figure 3(a). Each lattice label x is modeled as two roles xR and xW for read and write at label x respectively. The relationship among the four read roles and the four write roles is shown on the left and right hand sides of Figure 3(a), respectively. The duality between the left and right lattices is obvious from the diagrams.

To complete the construction we need to enforce appropriate constraints to reflect the labels on subjects in LBAC. Each user in LBAC has a unique security clearance. This is enforced by requiring that each user in RBAC96 is assigned to exactly two roles xR and LW. An LBAC user can login at any label dominated by the user's clearance. This requirement is captured in RBAC96 by requiring that each session has exactly two matching roles yR and yW. The condition that x $\geq$ y, that is the user's clearance dominates the label of any login session established by the user, is not explicitly required because it is directly imposed by the RBAC96 construction. Note that, by virtue of membership in LW, each user can activate any write role. However, the write role activated in a particular session must match the session's read role. Thus, both the role hierarchy and constraints of RBAC96 are exploited in this construction.

LBAC is enforced in terms of read and write operations. In RBAC96 this means our permissions are read and writes on individual objects written as (o,r) and (o,w) respectively. An LBAC object has a single sensitivity label associated with it. This is expressed in RBAC96 by requiring that each pair of permissions (o,r) and (o,w) be assigned to exactly one matching pair of xR and xW roles respectively. By assigning permissions (o,r) and (o,w) to roles xR and xW, respectively, we are implicitly setting the sensitivity label of object o to x.

4.2 The General Construction

Based on the above discussion we have the following construction for arbitrary lattices (actually the construction works for partial orders with a lower-most security class). Given $\mathcal{SC}$ with security labels {$L_1 \ldots L_n$}, and partial order $\geq_{LBAC}$, an equivalent RBAC96 system is given by:

*Construction 1. (Liberal *-Property)*

- $R = \{L_1R \ldots L_nR, L_1W \ldots L_nW\}$
- RH which consists of two disjoint role hierarchies. The first role hierarchy consists of the "read" roles $\{L_1R \ldots L_nR\}$ and has the same partial order as $\geq_{LBAC}$; the second partial consists of the "write" roles $\{L_1W \ldots L_nW\}$ and has a partial order which is the inverse of $\geq_{LBAC}$.
- $P = \{(o, r), (o, w) \mid o$ is an object in the system$\}$
- Constraint on UA: Each user is assigned to exactly two roles xR and LW where x is the label assigned to the user and LW is the write role corresponding to the lowermost security level according to $\geq_{LBAC}$
- Constraint on sessions: Each session has exactly two roles yR and yW
- Constraints on PA:
  (o,r) is assigned to xR iff (o,w) is assigned to xW
  (o,r) is assigned to exactly one role xR such that x is the label of o

THEOREM 1. *An RBAC96 system defined by Construction 1 satisfies the Simple Security Property and the Liberal *-Property.*

PROOF. (a) Simple Security Property: Subjects in the LBAC terminology correspond to RBAC96 sessions. For subject s to read o, (o,r) must be in the permissions assigned to a role, either directly or indirectly, which is among the roles available to session s, which corresponds to exactly one user u. For u to be involved in this session, this role must be in the UA for u (either directly or indirectly). Let $\lambda(u) = z$ and $\lambda(s) = y$. By the constraints on PA given in Construction 1, (o,r) is assigned directly to exactly one role xR, where $x = \lambda(o)$, and by the construction of RH, is inherited by roles yR such that $y \geq_{LBAC} x$. For s to be able to read o, it must have one of these yR in its session. By the definition of roles in a session from Definition 1, any role junior to zR can be in a session for u, i.e., $z \geq_{LBAC} y$. In other words, a session for u can involve one reading role yR such that $z \geq_{LBAC} y$. Therefore, the RBAC96 system defined above allows subject s to read object o if $\lambda(u) \geq_{LBAC} \lambda(s)$ and $\lambda(s) \geq_{LBAC} \lambda(o)$, which is precisely the Simple Security Property.

(b) Liberal *-Property: Each user, u, is assigned by UA to xR, where x is the clearance of the user. According to LBAC, the user can read data classified at level x or at levels dominated by x. It also means that the user can start a session at a level dominated by x. So, if a user cleared to say level x, wishes to run a session at level y, such that $x \geq_{LBAC} y$, the constraints in Construction 1 allow the session to have the two active roles yR and yW. Because every user is assigned to LW, it is possible for every user to have a session with yW as one of its roles. The structure of the two role hierarchies means that if the yW role is available to a user in a session, the user can write objects for which the permission (o,w) is in yW. By construction of the role hierarchy, the session can write to level y or levels dominated by y. In LBAC terms, the subject, s, corresponds to the session, and within a session a write can be performed if (o,w) is in the permissions of a role, which by the construction is only if $\lambda(o) \geq_{LBAC} \lambda(s)$. This is precisely the Liberal *-Property. □

4.3 LBAC Variations

Variations in LBAC can be accommodated by modifying this basic construction in different ways. In particular, the strict *-property retains the hierarchy on read roles but treats write roles as incomparable to each other as shown in Figure 3(b) for the example of our basic lattice.

*Construction 2. (Strict *-Property)* Identical to Construction 1 except RH has a partial order among the read roles identical to the LBAC partial order, and no relationships among the write roles.

THEOREM 2. *An RBAC96 system defined by Construction 2 satisfies the Simple Security Property and the Strict *-Property.*

The proof of this and subsequent similar results is omitted.

Next we consider a version of LBAC in which subjects are given more power than allowed by the simple security and *-properties [Bell 1987]. The basic idea is to allow subjects to violate the *-property in a controlled manner. This is achieved by associating a pair of security labels λ_r and λ_w

with each subject (objects still have a single security label). The simple security property is applied with respect to λ_r and the liberal *-property with respect to λ_w. In the LBAC model of Bell [1987], it is required that λ_r should dominate λ_w. With this constraint, the subject can read and write in the range of labels between λ_r and λ_w, which is called the *trusted range*. If λ_r and λ_w are equal, the model reduces to the usual LBAC model with the trusted range being a single label.

The preceding discussion is remarkably close to our RBAC constructions. The two labels λ_r and λ_w correspond directly to the two roles xR and yW we have introduced earlier. The dominance required between λ_r and λ_w is trivially recast as a dominance constraint between x and y. This leads to the following construction:

*Construction 3. (Liberal *-Property with Trusted Range)* Identical to Construction 1 except

—Constraint on *UA*: Each user is assigned to exactly two roles xR and yW such that $x \geq y$ in the original lattice

—Constraint on sessions: Each session has exactly two roles xR and yW such that $x \geq y$ in the original lattice

Lee [1988] and Schockley [1988] have argued that the Clark-Wilson integrity model [Clark and Wilson 1987] can be supported using LBAC. Their models are similar to the above except that no dominance relation is required between x and y. Thus, the write range may be completely disjoint with the read range of a subject. This is easily expressed in RBAC96 as follows.

*Construction 4. (Liberal *-Property with Independent Write Range)* Identical to Construction 3 except $x \geq y$ is not required in the constraint on *UA* and the constraint on sessions.

A variation of the above is to use the strict *-property as follows.

*Construction 5. (Strict *-Property with Designated Write)* Identical to Construction 2 except

—Constraint on *UA*: Each user is assigned to exactly two roles xR and yW

—Constraint on sessions: Each session has exactly two roles xR and yW

Construction 5 can also be directly obtained from Construction 4 by requiring the strict *-property instead of the liberal *-property. Construction 5 can accommodate Clark-Wilson transformation procedures as outlined by Lee [1988] and Schockley [1988]. (Lee and Schockley actually use the liberal *-property in their constructions, but their lattices are such that the constructions are more directly expressed in terms of the strict *-property.)

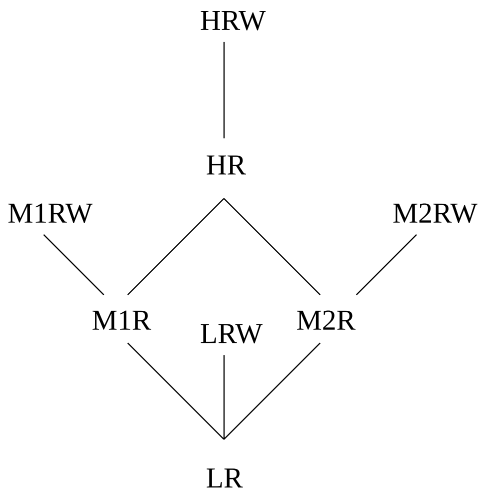

Fig. 4. Alternate role hierarchy for Strict *-property.

5. EXTENDING THE POSSIBLE RBAC CONFIGURATIONS

In the previous section, we looked at specific mappings of different kinds of LBAC to an RBAC system with the same properties. In this section we examine whether or not more arbitrary RBAC systems which do not necessarily follow the constructions in Section 4 still satisfy LBAC properties. In order to do this, we assume that all users and objects have security labels, and that permissions involve only reads and writes.

In the previous discussion, all constructions created role hierarchies with disjoint read and write roles. This is not strictly necessary; the role hierarchy in Figure 4 could be the construction for the strict *-property with the following modifications:

- Constraint on *UA*: Each user is assigned to all roles, xRW such that the clearance of the user dominates the security label x

- Constraint on sessions: Each session has exactly one role: yRW

- Constraints on *PA*:
  (o,r) is assigned to xR iff (o,w) is assigned to xRW
  (o,r) is assigned to exactly one role xR

Nevertheless, the structure of role hierarchies which do map to valid LBAC configurations is greatly restricted, as the examples in Osborn [1997] show. For example, a role with permissions to both read and write a high data object and a low data object cannot be assigned to a high user as this would allow write down, and cannot be assigned to a low user, as this would allow read up. If a role had only read permissions for some objects classified at M1, and other objects classified at M2 (cf. Figure 2), a subject cleared at H could be assigned to this role.

As far as the read operation is concerned, a subject can have a role r in its session if the label of the subject dominates the level of all o such that (o,r) is in the role. Since the least upper bound is defined for the security lattice, this can always be determined. Similarly, for write operations, if a greatest lower bound is defined for the security levels, then the Liberal *-property is satisfied in a session if the security level of the subject dominates the greatest lower bound of all o such that (o,w) is in the role. If such a greatest lower bound does not exist, such a role should not be in any user's UA. (If it could be determined that $\lambda(s) \leq \lambda(o)$ for all o such that (o,w) is in the role, then this $\lambda(s)$ would be a lower bound, and then a greatest lower bound would exist.)

We introduce the following two definitions to capture the maximum read level of objects in a role, and the minimum write level if one exists.

Definition 6. The *r-level* of a role r (denoted r-level(r)) is the **least upper bound** (lub) of the security levels of the objects for which (o,r) is in the permissions of r.

Definition 7. The *w-level* of a role r (denoted w-level(r)) is the **greatest lower bound** (glb) of the security levels of the objects o for which (o,w) is in the permissions of r, if such a glb exists. If the glb does not exist, the *w-level* is undefined.

The following theorem follows from these definitions.

THEOREM 3. *An RBAC96 configuration satisfies the simple security property and the Liberal *-Property if all of the following hold:*

- *Constraint on Users:* $((\forall u \in U)[\lambda(u) \text{ is given}])$
- *Constraints on Permissions:*
  $P = \{(o, r), (o, w) \mid o \textit{ is an object in the system}\}$
  $((\forall o \in P)[\lambda(o) \text{ is given}])$
- *Constraint on UA:*
  $((\forall r \in UA)[w\text{-}level(r) \textit{ is defined}])$
  $((\forall (u, r) \in UA)[\lambda(u) \geq r\text{-}level(r)])$
  $((\forall (u, r) \in UA)[\lambda(u) \leq w\text{-}level(r)])$
- *Constraint on Sessions:* $((\forall s \in sessions)[\lambda(s) \leq \lambda(u)])$

An example showing a possible role hierarchy is given in Figure 5, where the underlying security lattice contains labels {unclassified, secret, top secret}and roles are indicated by, for example, (ru,rs) meaning the permissions in the role include read of some unclassified and some secret object(s) (each role may have permissions inherited because of the role hierarchy). The roles labeled ru1 and ru3 at the bottom have read access to distinct objects labeled unclassified; ru2 inherits the permissions of ru1 and has additional read access to objects at the unclassified level. The role labeled (ru,ws) contains permission to read some unclassified objects and write

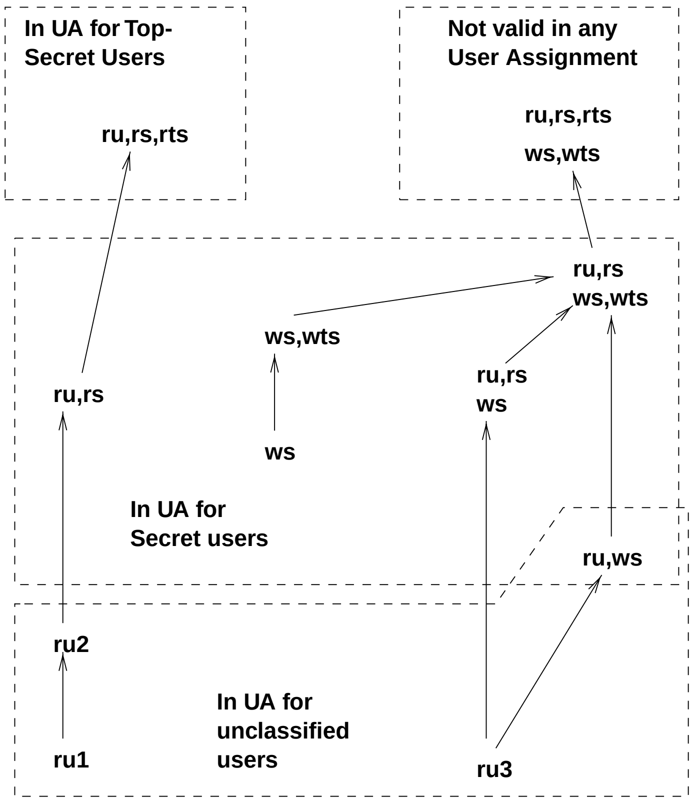

Fig. 5. A role hierarchy and its user assignments.

some secret objects. This role could be assigned in UA to either unclassified users or to secret users. Notice the role at the top of the role hierarchy, labeled (ru,rs,rts,ws,wts). This role cannot be assigned to any user without violating either the Simple Security Property or the Liberal *-Property. Note that if this role is deleted from the role hierarchy, we have an example of a role hierarchy which satisfies the Simple Security Property and the Liberal *-Property, and which does not conform to any of the constructions of Section 4.

An RBAC96 configuration satisfies the strict *-property if all of the above conditions hold, changing the Constraint on Sessions to:

- *Constraint on Sessions:* $((\forall s \in sessions)[\lambda(s) = \lambda(u)])$.

6. DAC MODELS

In this section, we discuss the DAC policies that will be considered in this paper. The central idea of DAC is that the owner of an object, who is usually its creator, has discretionary authority over who else can access that object. In other words the core DAC policy is owner-based administration of access rights. There are many variations of DAC policy, particularly concerning how the owner's discretionary power can be delegated to other users and how access is revoked. This has been recognized since the earliest formulations of DAC [Lampson 1971; Graham and Denning 1972].

Our approach here is to identify major variations of DAC and demonstrate their construction in RBAC96. The constructions are such that it will be obvious how they can be extended to handle other related DAC variations. This is an intuitive, but well-founded, justification for the claim that DAC can be simulated in RBAC.[2]

The DAC policies we consider all share the following characteristics.

- The creator of an object becomes its owner.
- There is only one owner of an object. In some cases ownership remains fixed with the original creator, whereas in other cases it can be transferred to another user. (This assumption is not critical to our constructions. It will be obvious how multiple owners could be handled.)
- Destruction of an object can only be done by its owner.

With this in mind, we now define the following variations of DAC with respect to granting of access.

(1) **Strict DAC** requires that the owner is the only one who has discretionary authority to grant access to an object and that ownership cannot be transferred. For example, suppose Alice has created an object (Alice is owner of the object) and grants read access to Bob. Strict DAC requires that Bob cannot propagate access to the object to another user. (Of course, Bob can copy the contents of Alice's object into an object that he owns, and then propagate access to the copy. This is why DAC is unable to enforce information flow controls, particularly with respect to Trojan Horses.)

(2) **Liberal DAC** allows the owner to delegate discretionary authority for granting access to an object to other users. We define the following variations of liberal DAC.

   (a) **One Level Grant:** The owner can delegate grant authority to other users but they cannot further delegate this power. So Alice being the owner of object O can grant access to Bob who can grant access to Charles. But Bob cannot grant Charles the power to further grant access to Dorothy.

   (b) **Two Level Grant:** In addition to a one-level grant the owner can allow some users to further delegate grant authority to other users. Thus, Alice can now authorize Bob for two-level grants, so Bob can grant access to Charles, with the power to further grant access to Dorothy. However, Bob cannot grant the two-level grant authority to Charles. (We could consider n-level grant but it will be obvious how to do this from the two level construction.)

[2]A formal proof would require a formal definition of DAC encompassing all its variations, and a construction to handle all of these in RBAC96. This approach is pursued in Munawer [2000].

(c) **Multilevel Grant:** In this case the power to delegate the power to grant implies that this authority can itself be delegated. Thus Alice can authorize Bob, who can further authorize Charles, who can further authorize Dorothy, and so on indefinitely.

(3) **DAC with Change of Ownership:** This variation allows a user to transfer ownership of an object to another user. It can be combined with strict or liberal DAC in all the above variations.

For revocation, we consider two cases as follows.

(1) **Grant-Independent Revocation:** Revocation is independent of the granter. Thus Bob may be granted access by Alice but have it revoked by Charles.

(2) **Grant-Dependent Revocation:** Revocation is strongly tied to the granter. Thus if Bob receives access from Alice, access can only be revoked by Alice.

In our constructions, we will initially assume grant-independent revocation and then consider how to simulate grant-dependent revocation. In general, we will also assume that anyone with authority to grant also has authority to revoke. This coupling often occurs in practice. Where appropriate, we can decouple these in our simulations because, as we will see, they are represented by different permissions.

These DAC policies certainly do not exhaust all possibilities. Rather these are representative policies whose simulation will indicate how other variations can also be handled.

7. CONFIGURING RBAC FOR DAC

To specify the above variations in RBAC96 it suffices to consider DAC with one operation, which we choose to be the read operation. Similar constructions for other operations such as write, execute and append, are easily possible.[3] Before considering specific DAC variations, we first describe common aspects of our constructions.

7.1 Common Aspects

The basic idea in our constructions is to simulate the owner-centric policies of DAC using roles that are associated with each object.

7.1.1 *Create an Object.* For every object O that is created in the system we require the simultaneous creation of three administrative roles and one regular role as follows.

[3]More complex operations such as copy can be viewed as a read of the original object and a write (and possibly creation) of the copy. It can be useful to associate some default permissions with the copy. For example, the copy may start with access related to that of the original object or it may start with some other default. Specific policies here could be simulated by extending our constructions.

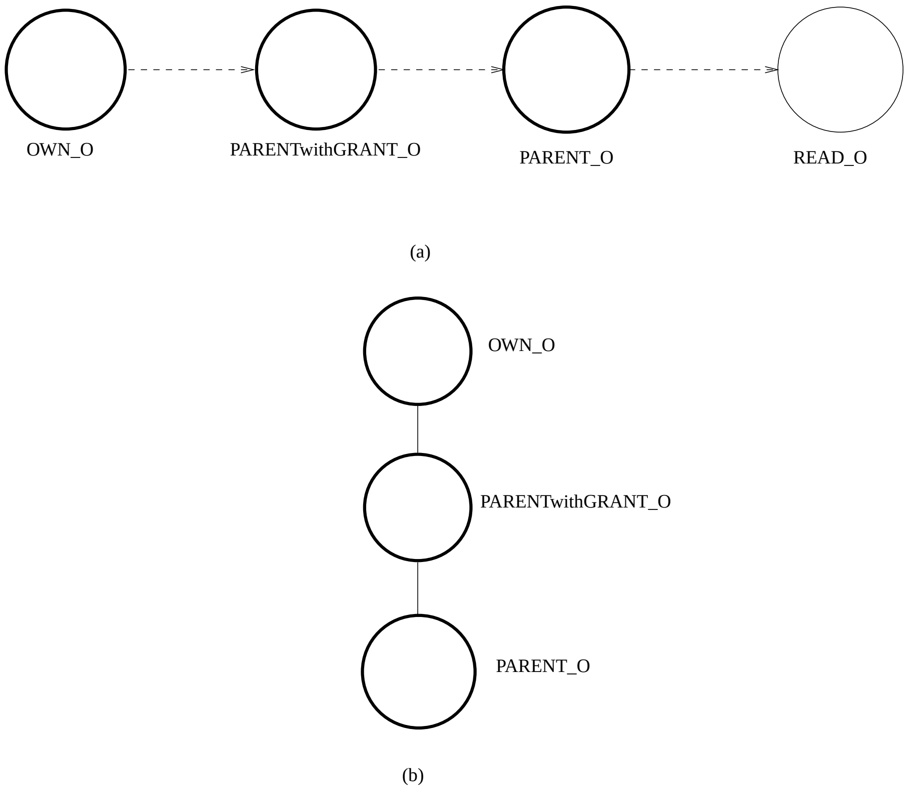

Fig. 6. (a) Administration of roles associated with an object; (b) Administrative role hierarchy.

—Three administrative roles in *AR*: OWN_O, PARENT_O and PARENT-withGRANT_O

—One regular role in *R*: READ_O

Role OWN_O has privileges to add and remove users from the role PARENTwithGRANT_O which in turn has privileges to add and remove users from the role PARENT_O The relationship between these roles is shown in Figure 6. In Figure 6, administrative roles are shown with darker circles than regular roles. In Figure 6(a), the dashed right arrows indicate that the role on the left contains the administrative permissions governing the role on the right. Figure 6(b) shows the administrative role hierarchy, with the senior role above its immediate junior, connected by an edge. For instance role OWN_O has administrative authority over roles PARENT-withGRANT_O as indicated in Figure 6(a). In addition due to the inheritance via the role hierarchy of Figure 6(b) OWN_O also has administrative authority over PARENT_O and READ_O.

In addition, we require simultaneous creation of the following eight permissions along with creation of each object O.

—canRead_O: authorizes the read operation on object O. It is assigned to the role READ_O.

—destroyObject_O: authorizes deletion of the object. It is assigned to the role OWN_O.

—addReadUser_O, deleteReadUser_O: respectively authorize the operations to add users to the role READ_O and remove them from this role. They are assigned to the role PARENT_O.

—addParent_O, deleteParent_O: respectively authorize the operations to add users to the role PARENT_O and remove them from this role. They are assigned to the role PARENTwithGRANT_O.

—addParentWithGrant_O, deleteParentWithGrant_O: respectively authorize the operations to add users to the role PARENT_O and remove them from this role. They are assigned to the role OWN_O.

These permissions are assigned to the indicated roles when the object is created and thereafter they cannot be removed from these roles or assigned to other roles.

7.1.2 *Destroy an Object*. Destroying an object O requires deletion of the four roles namely OWN_O, PARENT_O, PARENTwithGRANT_O and READ_O and the eight permissions (in addition to destroying the object itself). This can be done only by the owner, by virtue of exercising the destroyObject_O permission.

7.2 Strict DAC

In strict DAC, only the owner can grant/revoke read access to/from other users. The creator is the owner of the object. By virtue of membership (via seniority) in PARENT_O and PARENTwithGRANT_O, the owner can change assignments of the role READ_O. Membership of the three administrative roles cannot change, so only the owner will have this power. This policy can be enforced by imposing a cardinality constraint of 1 on OWN_O and of 0 on PARENT_O and PARENTwithGRANT_O.

This policy could be simulated using just two roles OWN_O and READ_O, and giving the addReadUser_O and deleteReadUser_O permissions directly to OWN_O at creation of O. For consistency with subsequent variations we have introduced all required roles from the start.

7.3 Liberal DAC

The three variations of liberal DAC described in Section 6 are now considered in turn.

7.3.1 *One-Level Grant*. The one-level grant DAC policy can be simulated by removing the cardinality constraint of strict DAC on membership in PARENT_O. The owner can assign users to the PARENT_O role who in turn can assign users to the READ_O role. But the cardinality constraint of 0 on PARENTwithGRANT_O remains.

7.3.2 *Two-Level Grant.* In the two level grant DAC policy the cardinality constraint on PARENTwithGRANT_O is also removed. Now the owner can assign users to PARENTwithGRANT_O who can further assign users to PARENT_O. Note that members of PARENTwithGRANT_O can also assign users directly to READ_O, so they have discretion in this regard. Similarly the owner can assign users to PARENTwithGRANT_O, PARENT_O or READ_O as deemed appropriate. (N-level grants can be similarly simulated by having N roles, PARENTwithGRANT_O^{N-1}, PARENTwithGRANT_O^{N-2}, . . . , PARENTwithGRANT_O, PARENT O.)

7.3.3 *Multilevel Grant.* To grant access beyond two levels we authorize the role PARENTwithGRANT_O to assign users to PARENTwithGRANT_O. We achieve this by assigning the addParentWithGrant_O permission to the role PARENTwithGRANT_O when object O is created. As per our general policy of coupling grant and revoke authority, we also assign the deleteParentWithGrant_O permission to the role PARENTwithGRANT_O when O is created. This coupling policy is arguably unreasonable in the context of grant-independent revoke, so the deleteParentWithGrant_O permission could be retained only with the OWN_O role if so desired. For grant-dependent revoke the coupling is more reasonable.

7.4 DAC with Change of Ownership

Change of ownership can be easily accomplished by suitable redefinition of the administrative authority of a member of OWN_O. Recall that change of ownership in this context means transfer of ownership from one user to another. Thus the OWN_O role needs a permission that enables this transfer to occur and this permission can only be assigned to this role. A member of OWN_O can assign another user to OWN_O but at the cost of losing their own membership.

7.5 Multiple Ownership

Multiple ownership can also be accommodated by removing the cardinality constraint on membership in the OWN_O role. Since all members of OWN_O have identical power, including the ability to revoke other owners, it would be appropriate with grant-independent revoke to distinguish the original owner. Alternately, we can have grant-dependent revoke of ownership.

7.6 Grant-Dependent Revoke

So far, we have considered grant-independent revocation where revocation is independent of granter. Now finally we consider how to simulate grant-dependent revoke in RBAC96. In this case, only the user who has granted access to another user can revoke the access (with possible exception of the owner who is allowed to revoke everything).

Specifically, let us consider the one level grant DAC policy simulated earlier by allowing members of PARENT_O role to assign users to the READ_O role. To simulate grant-dependent revocation with this one level

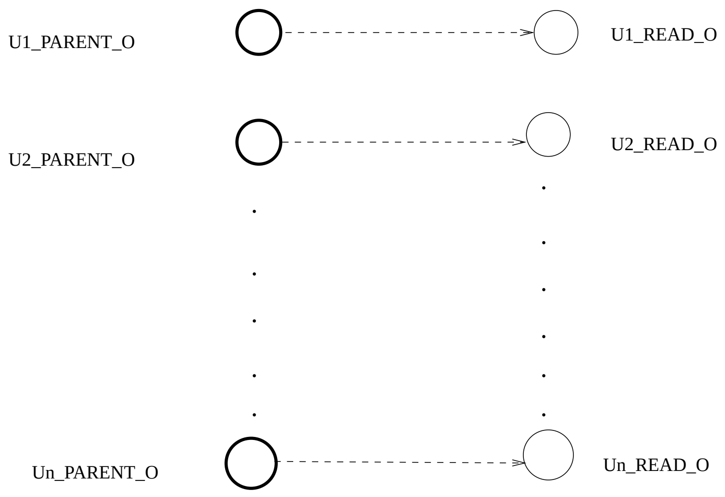

Fig. 7. Read_O roles associated with members of PARENT_O.

grant policy, we need a different administrative role U_PARENT_O and a different regular role U_READ_O for each user U authorized to do a one-level grant by the owner. These roles are automatically created when the owner authorizes user U. We also need two new administrative permissions created at the same time as follows.

- addU_ReadUser_O, deleteU_ReadUser_O: respectively authorize the operations to add users to the role U_READ_O and remove them from this role. They are assigned to the role U_PARENT_O.

Ui_PARENT_O manages the membership assignments of Ui_READ_O role as indicated in Figure 7 for user Ui. U_PARENT_O has a membership cardinality constraint of one. Moreover, its membership cannot be changed. Thus, user U will be the only one granting and revoking users from U_READ_O. The U_READ_O role itself is assigned the permission can-Read_O at the moment of creation. As before all of this enforced by RBAC96 constraints. We can allow the owner to revoke users from the U_READ_O role by making U_PARENT_O junior to OWN_O in the administrative role hierarchy. Simulation of grant-dependent revocation can be similarly simulated with respect to the PARENT_O and PARENTwith-GRANT_O roles. Extension to multiple ownership is also possible.

8. CONCLUSIONS

We have shown that the common forms of LBAC and DAC models can be simulated and enforced in RBAC96 with systematic constructions. All of the components of the RBAC96 model shown in Figure 1 were required to carry out these simulations. Users and permissions are essential to express any access control model. The Role Hierarchy is important in the LBAC simulation. The Administrative Role Hierarchy is essential in the enforcement of DAC policies, as is the administrative user to role assignment

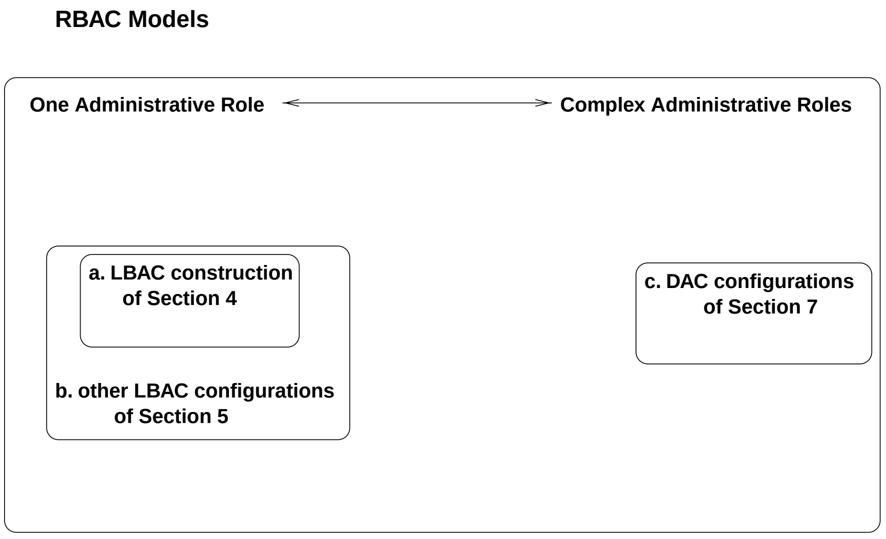
RBAC Models
One Administrative Role
Complex Administrative Roles
a. LBAC construction of Section 4
b. other LBAC configurations of Section 5
c. DAC configurations of Section 7

Fig. 8. Containment of models.

relation. We observe however that the permissions that have been granted to users in a DAC system can give an arbitrarily rich role hierarchy, as was noted in a conversion of relational database permissions to role graphs by Osborn et al. [1997]. Constraints play a role in all of the constructions. It is important to note that the LBAC simulation assumes a single administrative role, whereas the DAC simulation requires a large number of administrative roles, which are dynamically created and destroyed.

We can represent some of our findings using the Venn diagram in Figure 8. The area on the left of the figure indicates that in this subset of RBAC96 configurations, there is no need for administrative roles except for an assumed single administrative role. On the right, the administrative part of the RBAC96 model is fully utilized. Part (a) represents the subset of possible RBAC96 configurations which are built by Constructions 1 and 2. Area (b) shows that there are other configurations not built by these two constructions which still satisfy LBAC properties. Part (c) of the diagram represents in general the RBAC96 configurations built by the various constructions in Section 7. Note that these latter constructions all fall in the region where the administrative roles of the RBAC96 model are being fully utilized.

Future work should now focus on what happens in the rest of the RBAC96 Models not included in the areas constructed in this paper. Models for decentralized role administration which fall in between these extremes have been proposed by Sandhu et al. [1999]. These models allow for large numbers of administrative roles but this number is expected to be much smaller than the number of objects in the system.

In conclusion, then, we have shown with various systematic constructions how to simulate and enforce traditional LBAC and DAC access control models in RBAC96.

REFERENCES

BELL, D. 1987. Secure computer systems: A network interpretation. In *Proceedings on 3rd Annual Computer Security Application Conference*. 32–39.

CLARK, D. AND WILSON, D. 1987. A comparison of commercial and military computer security policies. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA, May). 184–194.

DENNING, D. E. 1976. A lattice model of secure information flow. *Commun. ACM 19*, 2, 236–243.

GRAHAM, G. AND DENNING, P. 1972. Protection-principles and practice. In *Proceedings on AFIPS Spring Joint Computer Conference*. AFIPS Press, Arlington, VA, 417–429.

LAMPSON, B. 1974. Protection. In *Proceedings of the 5th Symposium on Information Sciences and Systems* (Princeton, NJ, Mar.). 437–443.

LEE, T. 1988. Using mandatory integrity to enforce "commercial" security. In *Proceedings of IEEE Symposium on Security and Privacy* (Oakland, CA). 140–146.

MUNAWER, Q. 2000. Administrative models for role-based access control. Ph.D. Dissertation.

NYANCHAMA, M. AND OSBORN, S.. 1994. Access rights administration in role-based security systems. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*. Elsevier North-Holland, Inc., Amsterdam, The Netherlands, 37–56.

NYANCHAMA, M. AND OSBORN, S. 1996. Modeling mandatory access control in role-based security systems. In *Database Security VIII: Status and Prospects*. Chapman and Hall, Ltd., London, UK, 129–144.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Secur. 1*, 2 (Feb.), 3–33.

OSBORN, S. 1997. Mandatory access control and role-based access control revisited. In *Proceedings of the Second ACM Workshop on Role-based Access Control* (RBAC '97, Fairfax, VA, Nov. 6–7), C. Youman, E. Coyne, and T. Jaeger, Chairs. ACM Press, New York, NY, 31–40.

OSBORN, S., REID, L. K., AND WESSON, G. J. 1997. On the interaction between role-based access control and relational databases. In *Proceedings of the Tenth Annual IFIP TC11/WG11.3 International Conference on Database Security: Volume X: Status and Prospects* (Como, Italy, July 22–24, 1996), P. Samarati and R. S. Sandhu, Eds. Chapman and Hall, Ltd., London, UK, 275–287.

SANDHU, R. S. 1993. Lattice-based access control models. *IEEE Computer 26*, 11, 9–19.

SANDHU, R. 1996. Role hierarchies and constraints for lattice-based access controls. In *Proceedings of the Conference on Computer Security* (ESORICS 96, Rome, Italy), E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds. Springer-Verlag, New York, NY, 65–79.

SANDHU, R. S., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur. 1*, 2 (Feb.), 105–135.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29*, 2 (Feb.), 38–47.

SANDHU, R. AND MUNAWER, Q. 1998. How to do discretionary access control using roles. In *Proceedings of the Third ACM Workshop on Role-Based Access Control* (RBAC '98, Fairfax, VA, Oct. 22–23), C. Youman and T. Jaeger, Chairs. ACM Press, New York, NY, 47–54.

SANDHU, R. AND SAMARATI, P. 1994. Access control: Principles and practice. *IEEE Commun. Mag. 32*, 9, 40–48.

SANDHU, R. S. AND SAMARATI, P. 1997. Authentication, access control and intrusion detection. In *The Computer Science and Engineering Handbook*, A. B. Tucker, Ed. CRC Press, Inc., Boca Raton, FL, 1929–1948.

SCHOCKLEY, W. 1988. Implementing the Clark/Wilson integrity policy using current technology. In *Proceedings of the 11th National Computer Security Conference* (NIST-NCSC, Baltimore, Maryland, Oct.17-20). National Institute of Standards and Technology, Gaithersburg, MD, 29–37.

Received: November 1999; revised: April 2000; accepted: May 2000