Role-Based Access Control Models

Ravi S. Sandhu
George Mason University and SETA Corporation

Edward J. Coyne
Hal L. Feinstein
Charles E. Youman
SETA Corporation

Security administration of large systems is complex, but it can be simplified by a role-based access control approach. A family of increasingly sophisticated models shows how RBAC works.

Starting in the 1970s, computer systems featured multiple applications and served multiple users, leading to heightened awareness of data security issues. System administrators and software developers alike focused on different kinds of access control to ensure that only authorized users were given access to certain data or resources. One kind of access control that emerged is role-based access control (RBAC).

A role is chiefly a semantic construct forming the basis of access control policy. With RBAC, system administrators create roles according to the job functions performed in a company or organization, grant permissions (access authorization) to those roles, and then assign users to the roles on the basis of their specific job responsibilities and qualifications (see sidebar "Role-based access control terms and concepts").

A role can represent specific task competency, such as that of a physician or a pharmacist. A role can embody the authority and responsibility of, say, a project supervisor. Authority and responsibility are distinct from competency. A person may be competent to manage several departments but have the responsibility for only the department actually managed. Roles can also reflect specific duty assignments rotated through multiple users—for example, a duty physician or a shift manager. RBAC models and implementations should conveniently accommodate all these manifestations of the role concept.

Roles define both the specific individuals allowed to access resources and the extent to which resources are accessed. For example, an operator role might access all computer resources but not change access permissions; a security-officer role might change permissions but have no access to resources; and an auditor role might access only audit trails. Roles are used for system administration in such network operating systems as Novell's NetWare and Microsoft's Windows NT.

The particular combination of users and permissions brought together by a role tend to change over time. The permissions associated with a role, on the other hand, are more stable; they tend to change less often than the people who fill the job function that role represents. Therefore, basing security administration on roles rather than on permissions is simpler. Users can be easily reassigned to different roles as needs change. Similarly, as a company acquires new applications and systems, roles can have new permissions granted and existing permissions revoked.

This article explains why RBAC is receiving renewed attention as a method of security administration and review, describes a framework of four reference models we have developed to better understand RBAC and categorize different implementations, and discusses the use of RBAC to manage itself. Our framework separates the administration of RBAC from its access control functions.

NEEDS ADDRESSED BY ROLES

A recent study of 28 organizations by the National Institute of Standards and Technology[1] (NIST) demonstrates that RBAC addresses many different needs in the commercial and government sectors. Access control requirements were found to be determined by a need for customer, stockholder, and insurer confidence; personal information privacy; prevention of unauthorized financial asset distribution and unauthorized long-dis-

0018-9162/96/$5.00 © 1996 IEEE

tance telephone calls; and adherence to professional standards. Moreover, the study found that many organizations

- based access control decisions on "the roles that individual users take on as part of the organization";
- preferred to centrally control and maintain access rights that reflect the organization's protection guidelines; and
- viewed their access control needs as unique, believing that commercially available products lacked adequate flexibility.

RBAC is attracting strong interest in the standards arena. Roles are being considered as part of the emerging SQL3 standard for database management systems on the basis of the implementation of roles in Version 7 of Oracle. Roles have also been incorporated in the commercial security profile of the "common criteria" draft.[2] RBAC is also in tune with prevailing technology and business trends. Numerous software products, for example, directly support some form of RBAC, and others support closely related concepts, such as user groups, through which roles can be implemented.

REASONS TO USE RBAC

Renewed interest in RBAC has focused on general support at the application level. Traditionally, specific applications have had to encode RBAC internally, with existing operating systems and environments offering little application-level RBAC support. This is beginning to change; however, the challenge is to identify sufficiently flexible yet easy-to-use application-independent facilities to support many applications with minimal customization.

Although RBAC's usefulness is widely acknowledged, there is little agreement on what RBAC means. As a result, RBAC is open to interpretation by researchers and system developers. Sophisticated variations of RBAC include the capability to establish relations between roles, between permissions and roles, and between users and roles. For example, two roles can be established as mutually exclusive—the same user is not allowed to assume both. Roles can also acquire inheritance relations, whereby one role inherits permissions assigned to a different role. These role-role relations can enforce security policies, including separation of duties and delegation of authority. Previously, these relations would have required application-software encoding; with RBAC, they can be specified once for a security domain.

With RBAC, role-permission relationships can be predefined, which makes it simple to assign users to the predefined roles. The NIST study[1] indicates that permissions assigned to roles, unlike user membership in roles, tend to change relatively slowly. The study also found it desirable to let administrators confer and revoke user membership in existing roles without authorizing these administrators to create new roles or change role-permission assignments. One reason for this finding is that assigning users to roles typically requires less technical skill than assigning permissions to roles. Without RBAC, it can also be difficult to determine what permissions have been authorized for what users.

Access control policy is embodied in RBAC components such as role-permission, user-role, and role-role relationships. These components collectively determine whether a particular user is allowed access to a certain piece of system data. RBAC components can be configured directly by the system administrator or indirectly by appropriate roles as delegated by the system administrator. The policy enforced in a given system results from the specific configuration of RBAC components as directed by the system administrator. Because the access control policy can, and usually does, change over the system life cycle, RBAC offers a key benefit through its ability to modify access control to meet changing organizational needs.

Role-based access control terms and concepts

Access—A specific type of interaction between a subject and an object that results in the flow of information from one to the other.[1]
Access control—The process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network).[2]
Administrative role—A role that includes permission to modify the set of users, roles, or permissions, or to modify the user assignment or permission assignment relations.
Constraint—A relationship between or among roles.
Group—A set of users.
Object—A passive entity that contains or receives information.[1]
Permissions—A description of the type of authorized interactions a subject can have with an object.[2]
Resource—Anything used or consumed while performing a function. The categories of resources are time, information, objects, or processors.[1]
Role—A job function within the organization that describes the authority and responsibility conferred on a user assigned to the role.
Role hierarchy—A partial order relationship established among roles.
Session—A mapping between a user and an activated subset of the set of roles the user is assigned to.
Subject—An active entity, generally in the form of a person, process, or device, that causes information to flow among objects or changes the system state.[1]
System administrator—The individual who establishes the system security policies, performs the administrative roles, and reviews the system audit trail.
User—Any person who interacts directly with a computer system.[1]

References

1. US Department of Defense, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, Washington, D.C., US Department of Defense, Dec. 1985.
2. US Department of Defense, National Computer Security Center, *Glossary of Computer Security Terms*, NCSC-TG-004-88, Ft. Meade, Md., National Computer Security Center, Oct. 21, 1988.

Although the RBAC concept is policy neutral, it directly supports three well-known security principles:

- *Least privilege*: Only those permissions required for the tasks performed by the user in the role are assigned to the role.
- *Separation of duties*: Invocation of mutually exclusive roles can be required to complete a sensitive task, such as requiring an accounting clerk and an account manager to participate in issuing a check.
- *Data abstraction*: Instead of the read, write, execute permissions typically provided by the operating system, abstract permissions, such as credit and debit for an account object, can be established.

Two caveats: RBAC cannot enforce the way these principles are applied. Theoretically, a system administrator could configure RBAC to violate these principles. Also, the degree to which data abstraction is supported will be determined by the implementation details.

RBAC is not a panacea for all access control issues. More sophisticated methods are required to deal with situations that control operation sequences. For example, where a purchase requisition requires various steps before the purchase order can be issued, RBAC does not attempt to directly control the permissions for such an event sequence. Other forms of access control can be layered on top of RBAC for this purpose. (See Mohammed and Dilts[3] and Thomas and Sandhu.[4]) We regard operation sequence control to be outside the scope of RBAC, although RBAC can be a foundation on which to build such controls.

ROLES AND RELATED CONCEPTS

Many access control systems commonly provide groups of users as the access control unit. A major difference between groups and roles is that groups are typically treated as a collection of users but not as a collection of permissions. A role, serving as an intermediary, is both a collection of users *and* a collection of permissions.

In Unix, because group membership is defined in two files (/etc/passwd and /etc/group), it is easy to determine the users belonging to a particular group. Permissions are granted to groups on the basis of permission bits associated with individual files and directories. Determining the permissions granted to a particular group generally requires a traversal of the entire file system tree. It is easier, therefore, to determine a group's membership than its permissions. Moreover, the assignment of permissions to groups is highly decentralized. Essentially, the owner of any Unix file system subtree can assign permissions for that subtree to a group. Although Unix groups are different from our concept of roles, in certain situations Unix groups can implement roles.

Groups versus roles

To illustrate the qualitative nature of the group-versus-role distinction, let's consider a hypothetical system in which it takes twice as long to determine group membership as to determine group permissions. Let's assume that group permissions and membership can be changed only by the system administrator. In this example, the group mechanism closely resembles our role concept.

Our example suggests that (1) it should be roughly as easy to determine role membership as role permissions, and (2) control of role membership and role permissions should be relatively centralized in a few users. Many mechanisms claiming to be role based have neither of these characteristics.

Roles and compartments

A question we're frequently asked concerns the relationship of roles to compartments. Compartments are part of the security label structure used in the classified defense and government sectors[5] and are based on the "need to know," which has a semantic connotation regarding the information available under a compartment label analogous to the semantic connotation of role. This idea essentially underlies the apparent similarity of compartments and roles. However, compartments are used for the specific policy of one-directional information flow in a lattice of labels, whereas roles are not confined to any single policy.

Discretionary and mandatory access

A long-standing distinction between discretionary and mandatory access controls, respectively known as DAC and MAC, emerged from defense security research. MAC controls access on the basis of security labels attached to users (more precisely, subjects) and objects.[5] DAC controls access to an object on the basis of an individual user's permissions and/or denials. Typically, the object's owner is another user, who establishes the permissions and/or denials. RBAC is an independent component of access control, coexisting with MAC and DAC when appropriate. In such a case, access is allowed only if permitted by RBAC, MAC, and DAC. In other cases, we expect that RBAC will exist by itself.

A related issue is whether RBAC itself is a discretionary or a mandatory mechanism. The answer depends on how the terms are defined and on the nature and configuration of permissions, roles, and users in an RBAC system. Our understanding of mandatory means that individual users have no choice regarding which permissions or users are assigned to a role. Discretionary signifies that individual users make these decisions. Recall that by itself, RBAC is policy neutral; however, individual RBAC configurations can support a mandatory policy, while others can support a discretionary policy.

A FAMILY OF REFERENCE MODELS

To explore RBAC's various dimensions, we have defined a family of four conceptual models. Figure 1a shows the model relationships and Figure 1b portrays their essential characteristics. $RBAC_0$, as the base model at the bottom, is the minimum requirement for an RBAC system. Advanced models $RBAC_1$ and $RBAC_2$ include $RBAC_0$, but $RBAC_1$ adds role hierarchies (situations where roles can inherit permissions from other roles), whereas $RBAC_2$ adds constraints (which impose restrictions on acceptable configurations of the different components of RBAC). $RBAC_1$ and $RBAC_2$ are incomparable to one another. The consolidated model, $RBAC_3$, includes $RBAC_1$ and $RBAC_2$ and, by transitivity, $RBAC_0$.

Researchers and developers can compare their systems and models with our reference models. The four models can also serve to guide product development and customer evaluation.

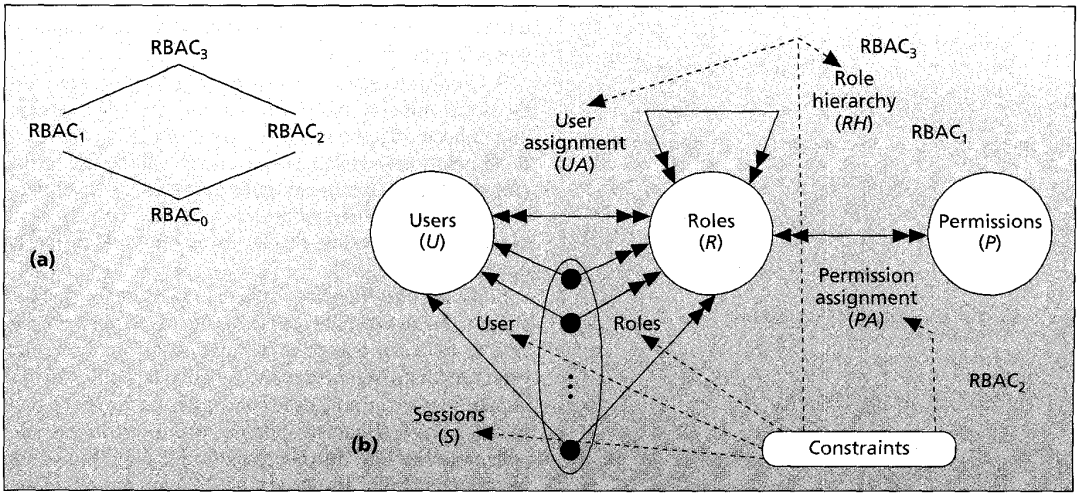

Figure 1. A family of role-based access control models. $RBAC_0$, as the base model at the bottom, is the minimum requirement for an RBAC system. Advanced models $RBAC_1$ and $RBAC_2$ include $RBAC_0$, but $RBAC_1$ adds role hierarchies, whereas $RBAC_2$ adds constraints. The consolidated model, $RBAC_3$, includes $RBAC_1$ and $RBAC_2$ and, by transitivity, $RBAC_0$.

For the following discussion of the models, we assume that a single system administrator is the only one authorized to configure the various sets and relations of the models. A more sophisticated management model is discussed later.

Base model—$RBAC_0$

In Figure 1b, the base model $RBAC_0$ consists of everything except role hierarchies and constraints. Four entities are shown: users (*U*), roles (*R*), permissions (*P*), and sessions (*S*).

USERS AND ROLES. For simplicity in our model, *user* is a human being. A *role* is a named job function within the organization that describes the authority and responsibility conferred on a member of the role.

PERMISSIONS. A *permission* is an approval of a particular mode of access to one or more objects in the system. The terms *authorization*, *access right*, and *privilege* are also used in the literature to denote a permission. Permissions are always positive and confer on their holder the ability to perform an action in the system. *Objects* are data objects or resource objects represented by data in the computer system. Our conceptual model accommodates many interpretations for permissions, from those where access is permitted to an entire subnetwork, to those where the unit of access is a particular field of a particular record. Some access control literature discusses negative permissions, which deny rather than confer access. However, we consider access denial to be a constraint rather than a negative permission.

The nature of a permission depends largely on system type and implementation; thus, a general access-control model must treat permissions somewhat as uninterpreted symbols. Because each system type protects objects of the abstraction it implements, an operating system, for example, protects files, directories, devices, and ports through operations such as read, write, and execute. A relational database management system protects relations, tuples, attributes, and views through operations such as select, update, delete, and insert. An accounting application protects accounts and ledgers through operations such as debit, credit, transfer, create account, and delete account.

Permissions can apply to single objects or to many, and they can be as specific as read access to a particular file or as generic as read access to all files belonging to a particular department. The manner in which individual permissions are joined into a generic permission so that they can be assigned as a single unit is highly implementation dependent.

Figure 1b shows *user assignment* (*UA*) and *permission assignment* (*PA*) relations; both are many-to-many and both are key to RBAC. A user can belong to 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. Ultimately, the user exercises permissions. The role's position as an intermediary to let a user exercise a permission provides greater control over access configuration and review than does a direct relationship between users and permissions.

SESSIONS. Users establish *sessions* during which they may activate a subset of the roles they belong to. Each session maps one user to possibly many roles. The double-headed arrow from the session to *R* in Figure 1b indicates that multiple roles are 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* in Figure 1b. This association remains constant for a session's duration. The concept of a session equates to the traditional notion of subject in the access control literature.

A user might have multiple sessions open simultaneously, each in a different window on a workstation screen.

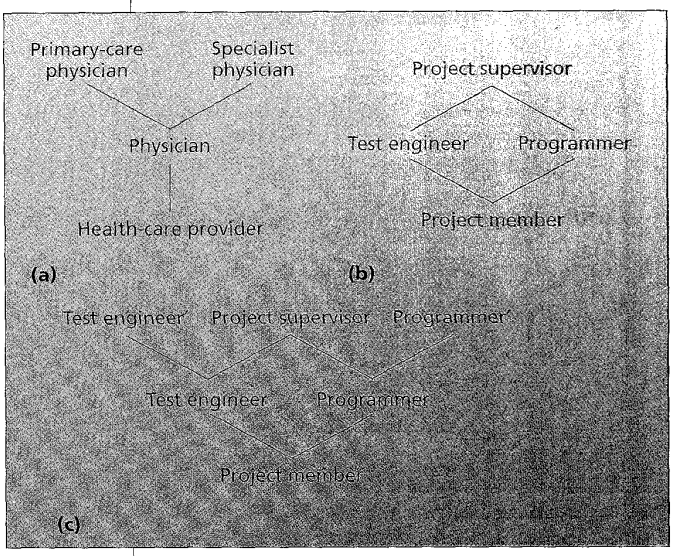

Figure 2. Examples of role hierarchies.

Each session might combine different active roles. This $RBAC_0$ feature supports the least-privilege principle. A user belonging to several roles can invoke any subset of them that enables tasks to be accomplished in a session. Thus, a user who is a member of a powerful role can normally keep this role deactivated and explicitly activate it when needed. (All constraints are discussed in the $RBAC_2$ subsection.) In the $RBAC_0$ model, the user's discretion alone determines which roles are activated in a given session. This model also lets roles be dynamically activated and deactivated during a session.

The formal definition of $RBAC_0$ follows.

Definition 1—The $RBAC_0$ model has the following components:

- U, R, P, and S (users, roles, permissions, and sessions, respectively);
- $PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation;
- $UA \subseteq U \times R$, a many-to-many user-to-role assignment relation;
- $user : S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session's lifetime); and
- $roles : S \rightarrow 2^R$, a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r \mid (user(s_i), r) \in UA\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (p, r) \in PA\}$.

Each role would likely be assigned at least one permission, and each user at least one role. The model does not require this, however.

As noted earlier, $RBAC_0$ treats permissions as uninterpreted symbols because permissions are implementation and system dependent. Our framework requires that permissions apply to data and resource objects and not to the components of RBAC itself. Permissions to modify the sets U, R, and P and relations PA and UA are called administrative permissions and are discussed in the section on issues in role administration.

Sessions are under the control of individual users. As far as the model is concerned, a user can create a session and choose to activate some subset of the user's roles. Roles active in a session can be changed at the user's discretion. The session terminates at the user's initiative. (Some systems will terminate a session if it is active for too long. Strictly speaking, this is a constraint and properly belongs in $RBAC_2$.)

Some authors[6] consider duties, in addition to permissions, to be an attribute of roles. A duty is a user's obligation to perform one or more tasks that are generally essential for an organization to function smoothly. In our view, duties are an advanced concept that does not belong in $RBAC_0$. We feel that incorporation of duties in access control models requires further research and at present is not incorporated in our advanced models. One approach might treat duties as it does permissions. Another approach to incorporating duties would have as its basis new access control paradigms such as task-based authorization.[4]

Role hierarchies—$RBAC_1$

The next model in our framework, $RBAC_1$, introduces role hierarchies (RH), as indicated in Figure 1. Role hierarchies are invariably discussed along with roles in the literature[7-10] and are commonly implemented in systems that provide roles.

Hierarchies are a natural means for structuring roles to reflect an organization's lines of authority and responsibility (see Figure 2). By convention, more powerful (senior) roles are shown toward the top of these diagrams and less powerful (junior) roles toward the bottom.

In Figure 2a, the junior-most role is that of health-care provider. The physician role is senior to health-care provider and thereby inherits all permissions from health-care provider. The physician role can have permissions besides those it inherited. Permission inheritance is transitive, so in Figure 2a, for example, the primary-care physician role inherits permissions from both the physician and health-care-provider roles. Primary-care physician and specialist physician both inherit permissions from the physician role, but each will have different permissions directly assigned to it. Figure 2b illustrates multiple inheritance of permissions, where the project supervisor role inherits from both test engineer and programmer roles.

Mathematically, these hierarchies are partial orders. A partial order is a reflexive, transitive, and antisymmetric relation. Inheritance is reflexive because a role inherits its own permissions, transitivity is a natural requirement in this context, and antisymmetry rules out roles that inherit from one another and would therefore be redundant.

The formal definition of $RBAC_1$ follows.

Definition 2—The $RBAC_1$ model has the following components:

- U, R, P, S, PA, UA, and $user$ are unchanged from $RBAC_0$;
- $RH \subseteq R \times R$ is a partial order on R called the role hierarchy or role dominance relation, also written as $\geq$; and

- *roles* : S → 2^R is modified from $RBAC_0$ to require $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA]\}$.

A user can establish a session with any combination of roles junior to the user's own roles. Similarly, the permissions in a session are those directly assigned to the session's roles plus those assigned to junior roles.

LIMITED INHERITANCE. Sometimes it is useful to limit the scope of inheritance. As an example, let's look at the hierarchy of Figure 2b, where the project supervisor role is senior to both the test engineer and programmer roles. It's entirely reasonable that test engineers might want to keep some permissions private to their role and prevent their inheritance by project supervisors. For example, access to incomplete work in progress, although appropriate for test engineers, might not be appropriate for the senior role. This situation can be accommodated by defining a new role, test engineer′, and relating it to test engineer (see Figure 2c). The private permissions of test engineers can be assigned to the test engineer′ role. Test engineers are assigned to the test engineer′ role and inherit permissions from the test engineer role; these permissions are also inherited upward by the project supervisor role. Test engineer′ permissions, however, are not inherited by the project supervisor role. We call test engineer′ an example of a *private role*; Figure 2c shows a second example of a private role, that of programmer′.

Private roles are achieved in some systems by blocking upward inheritance of certain permissions, but this technique prevents the hierarchy from accurately depicting permission distribution. It is preferable to introduce private roles and keep the hierarchical role relationship intact.

PRIVATE SUBHIERARCHY. Figure 3 shows how a private role subhierarchy can be built. The hierarchy of Figure 3a has four task roles, *T*1, *T*2, *T*3, and *T*4, all of which inherit permissions from the common project-wide role *P*. Project supervisors are assigned to role *S*. Tasks *T*3 and *T*4 are a subproject with *P*3 as the subproject-wide role and *S*3 as the subproject supervisory role. Role *T*1′ in Figure 3b is a private role for members of task *T*1. Suppose the subproject of Figure 3a comprising roles *S*3, *T*3, *T*4, and *P*3 requires a private subhierarchy within which private permissions of the project are shared without inheritance by *S*. The entire subhierarchy is replicated as shown in Figure 3b. The permissions inheritable by *S* are appropriately assigned to *S*3, *T*3, *T*4, and *P*3, whereas the private ones are assigned to *S*3′, *T*3′, *T*4′, and *P*3′, allowing their inheritance within the subproject only. As before, members of the subproject team are directly assigned to *S*3′, *T*3′, *T*4′, or *P*3′. The system's private roles are clearly seen here; this assists in access review to determine the nature of the private permissions.

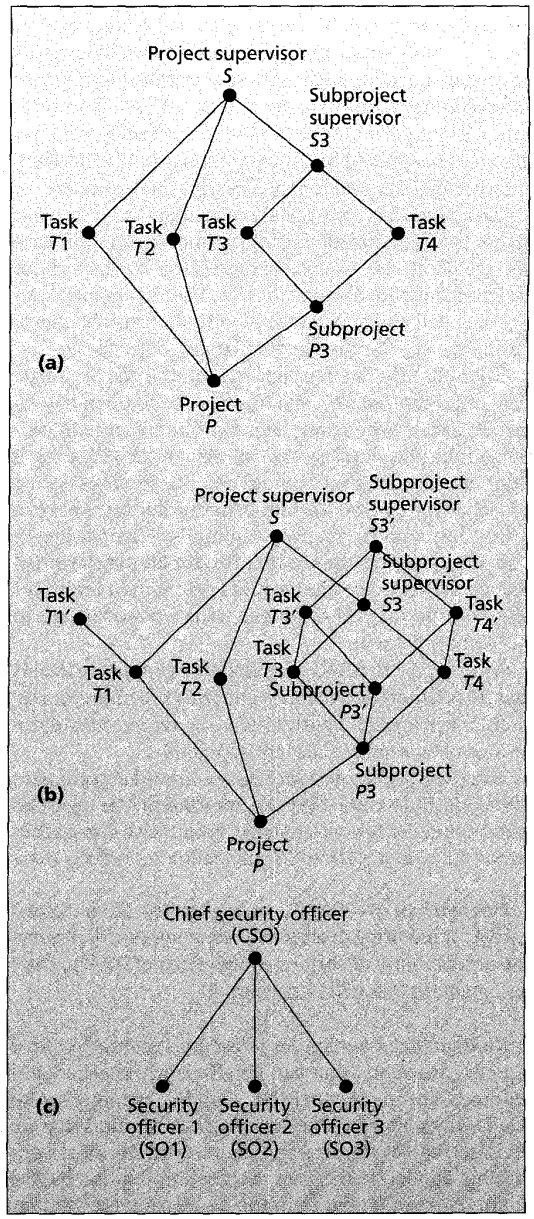

Figure 3. Role hierarchies for a project. The hierarchy of Figure 3a has four task roles, *T*1, *T*2, *T*3, and *T*4, all of which inherit permissions from the common project-wide role *P*. Role *S* at the top of the hierarchy is intended for project supervisors. Tasks *T*3 and *T*4 are a subproject, with *P*3 as the subproject-wide role and *S*3 as the subproject supervisory role. Role *T*1′ in Figure 3b is a private role for members of task *T*1. Figure 3c shows the administrative hierarchy of the security officer (system administrator) role.

Constraints model—$RBAC_2$

The third reference model in our framework, $RBAC_2$, introduces constraints, as shown in Figure 1b. Although we have called our models $RBAC_1$ and $RBAC_2$, there isn't really an implied progression. Either constraints or role hierarchies can be introduced first (indicated by the incomparable relation between $RBAC_1$ and $RBAC_2$ in Figure 1a).

Constraints are an important aspect of RBAC and are

sometimes argued to be the principal motivation behind RBAC. A common example is that of mutually disjoint organizational roles, such as those of purchasing manager and accounts payable manager. Generally, the same individual is not permitted to belong to both roles, because this creates a possibility for committing fraud. This well-known, time-honored principle is *separation of duties*.

Constraints are a powerful mechanism for laying out higher level organizational policy. Once certain roles are declared mutually exclusive, there's less concern about assigning individual users to roles. User assignment can be delegated and decentralized without fear of compromising the organization's overall policy objectives.

As long as RBAC's management is centralized in a single system administrator, constraints are simply a convenience, because the same effect could be achieved by caution on the part of the system administrator. However, if RBAC management is decentralized, constraints become a mechanism by which senior system administrators can restrict users' ability to exercise administrative privileges. This lets the chief system administrator lay out the broad scope of what is acceptable and make it mandatory for other system administrators and users who participate in RBAC management.

With respect to $RBAC_0$, constraints can apply to the *UA* and *PA* relations and the *user* and *roles* functions for sessions. When applied, constraints are predicates that return a value of "acceptable" or "not acceptable."

Intuitively, constraints are better viewed according to their kind and nature; they can, for example, be regarded as sentences in a formal language. Because we discuss constraints informally, the following definition reflects that.

Definition 3—$RBAC_2$ is unchanged from $RBAC_0$ except for requiring that there be constraints to determine the acceptability of various components of $RBAC_0$. Only acceptable values will be permitted.

RBAC implementation considerations generally call for simple constraints that can be efficiently checked and enforced. Fortunately, in $RBAC_1$ simple constraints can go a long way, and we next discuss some constraints that we feel are reasonable to implement. Because most constraints applied to the user assignment relation have a counterpart that applies to the permission assignment relation, we discuss constraints on these two components in parallel.

MUTUALLY EXCLUSIVE ROLES. The most common RBAC constraint is mutually exclusive roles. The same user can be assigned to at most one role in a mutually exclusive set. This supports separation of duties, which is further ensured by a mutual exclusion constraint on permission assignment.

The dual constraint on permission assignment can provide additional assurance for separation of duties but has received hardly any mention in the literature. This dual constraint requires that the same permission be assigned to at most one role in a mutually exclusive set. For example, consider two mutually exclusive roles, accounts manager and purchasing manager. Mutual exclusion in terms of *UA* specifies that one individual cannot belong to both roles. Mutual exclusion in terms of *PA* specifies that the same permission—to issue checks, for instance—cannot be assigned to both roles. Normally, such a permission would be assigned to the accounts manager role. The mutual exclusion constraint on *PA* would prevent the permission from being inadvertently or maliciously assigned to the purchasing manager role. More directly, exclusion constraints on *PA* limit the distribution of powerful permissions. For example, it may not matter whether role *A* or role *B* receives signature authority for a particular account, but what does matter is that only one of the two roles receives this permission.

More generally, various combinations of roles can be prohibited. For example, a user might belong to both a programmer role and a tester role on different projects, but within the same project this would be unacceptable. Similarly, various combinations of permissions can be prohibited.

CARDINALITY. Another user assignment constraint is a maximum number of members in a role. Only one person can fill the role of department chair; similarly, the number of roles an individual user can belong to could also be limited. These are *cardinality constraints*, which can be correspondingly applied to permission assignment to control the distribution of powerful permissions. Minimum cardinality constraints, on the other hand, may be difficult to implement. For example, if a role requires a minimum number of members, it would be difficult for the system to know if one of the members disappeared and to respond appropriately.

PREREQUISITE ROLES. The concept of *prerequisite roles* is based on competency and appropriateness, whereby a user can be assigned to role *A* only if the user already is assigned to role *B*. For example, only users who are already assigned to the project role can be assigned to the testing role in that project. The prerequisite (project) role is junior to the new (test) role. In practice, prerequisites between incomparable roles are less likely to occur.

The dual constraint on permission assignment applies more at the role end of the *PA* relation. For consistency, permission *p* might be assigned to a role only if that role already possesses permission *q*. For instance, in many systems permission to read a file requires permission to read the directory in which the file resides. Assigning the former permission without the latter would be incomplete.

OTHER CONSIDERATIONS. User assignment constraints are effective only if suitable external discipline is maintained in assigning user identifiers to human beings. If the same individual is assigned two or more user identifiers, separation and cardinality controls break down. A one-to-one correspondence between user identifiers and human beings is required. The situation with permission constraints is similar. If the same operation is sanctioned by two different permissions, the RBAC system cannot effectively enforce cardinality and separation constraints.

Constraints also apply to sessions and to the *user* and *roles* functions associated with a session. A user may belong to two roles but cannot be active in both at the same time. Other session constraints limit the number of ses-

Other approaches to managing access control

Containment hierarchy

In one approach to access control management, the International Organization for Standardization (ISO) has developed security-management-related standards and documents, described in the top-level System Management Overview document.[1] The ISO model is object-oriented and includes a hierarchy based on containment (a directory contains files and a file contains records). Roles could be integrated into the ISO approach.

Propagation of access rights

There is a long tradition of models for access rights propagation, where the right to propagate rights is controlled by special control rights. Among the most recent and most developed of these is Sandhu's typed access matrix model.[2] While it's often difficult to analyze the consequences of even simple rules for rights propagation, these models indicate that simple primitives can be composed to yield flexible and expressive systems.

Negotiated authority

To manage RBAC, Moffet and Sloman[3] have defined an elaborate model based on role domains, owners, managers, and security administrators. Authority is not controlled or delegated from a single central point, but is negotiated between independent managers who have only a limited trust in each other.

References

1. ISO/IEC 10040, *Information Technology—Open Systems Interconnection–Systems Management Overview*, Int'l Organization for Standardization/Int'l Electrotechnical Commission, 1992, Geneva, Switzerland.
2. R.S. Sandhu, "The Typed Access Matrix Model," *Proc. IEEE Computer Soc. Symp. Research in Security and Privacy*, IEEE CS Press, Los Alamitos, Calif., Order No. 2825, 1992, pp. 122-136.
3. J.D. Moffett and M.S. Sloman, "Delegation of Authority," in *Integrated Network Management II*, I. Krishnan and W. Zimmer, eds., Elsevier Science Publishers B.V., North-Holland, 1991, pp. 595-606.

sions a user can have active at the same time. Correspondingly, the number of sessions to which a permission is assigned can be limited.

A role hierarchy can be considered a constraint in that a permission assigned to a junior role must also be assigned to all senior roles, or a user assigned to a senior role must also be assigned to all junior roles. In a sense, $RBAC_1$ is redundant and subsumed by $RBAC_2$. However, the existence of role hierarchies should be recognized accordingly; they're reduced to constraints only when redundant permission or user assignments are introduced. Preferably, hierarchies are supported directly rather than indirectly with redundant assignment.

Consolidated model—$RBAC_3$

$RBAC_3$ provides both role hierarchies and constraints, as it combines $RBAC_1$ and $RBAC_2$. Combining both concepts raises several issues, which we explore next.

CONSTRAINTS ON ROLE HIERARCHIES. Constraints can apply to the role hierarchy itself, as indicated by the dashed arrow to *RH* in Figure 1b. The role hierarchy must be a partial order (a constraint intrinsic to the model). Additional constraints can limit the number, if any, of senior or junior roles that a given role may have. Two or more roles can also be constrained to have no common senior (or junior) role. Such constraints are useful where the authority to change the role hierarchy has been decentralized but the chief system administrator wants to restrict the manner in which changes are made.

INTERACTIONS. Subtle interactions arise between constraints and hierarchies. Suppose that test engineer and programmer roles are declared mutually exclusive in the context of Figure 2b. The project supervisor role violates this mutual exclusion constraint. Such a violation by a senior role may or may not be acceptable. The model should therefore accommodate both possibilities.

A similar situation concerns cardinality constraints. Suppose that a user can be assigned to at most one role. Does an assignment to the test engineer role in Figure 2b violate this constraint? In other words, do cardinality constraints apply only to direct membership, or do they also carry on to inherited membership?

PRIVATE ROLES. Let's look at Figure 2c to see how constraints affect private roles. The test engineer′, programmer′, and project supervisor roles can be declared mutually exclusive, and because these have no common senior role, there's no conflict. In general, private roles will not share common seniors with other roles because they are maximal hierarchical elements; thus, private roles can be mutually exclusive without causing conflict.

A maximum cardinality constraint of zero members can be declared for the nonprivate roles. Test engineers must then be assigned to the test engineer′ role. The test engineer role offers a way to share permissions with the project supervisor role.

ISSUES IN ROLE ADMINISTRATION

Our discussions so far have assumed that all RBAC components are directly controlled by a single system administrator, yet in large systems the number of roles can exceed hundreds or thousands. Managing these roles and their interrelationships is a formidable task that is often highly centralized and delegated to a small team of security administrators. Because RBAC's key advantage is that it simplifies permission administration, the next step is to see how RBAC might be used to manage itself. We believe that the use of RBAC to aid in managing RBAC will be a decisive factor in RBAC's overall success. (For different views on access control management, see sidebar "Other approaches to managing access control.")

Our management model for RBAC is illustrated in Figure 4, where the constraints apply to all components. The top half of this figure is essentially the same as Figure

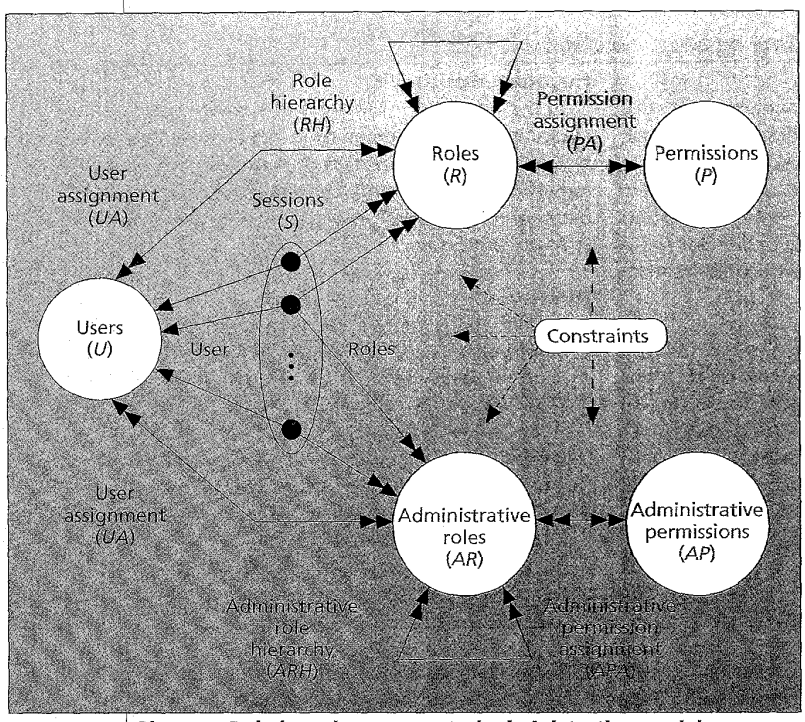

Figure 4. Role-based access control administrative model.

1b. Except for administrative roles and administrative permissions, the bottom half of the figure mirrors the top half. Our intent is for administrative roles *AR* and administrative permissions *AP* to be respectively disjoint from the regular roles *R* and permissions *P*. The model shows that permissions can be assigned only to roles and that administrative permissions can be assigned only to administrative roles; this is a built-in constraint.

The top half of Figure 4 can range in sophistication across $RBAC_0$, $RBAC_1$, $RBAC_2$, and $RBAC_3$. The bottom half can similarly range in sophistication across $ARBAC_0$, $ARBAC_1$, $ARBAC_2$, and $ARBAC_3$, where the first A denotes administrative. Generally, the administrative model will be simpler than the RBAC model itself. Thus $ARBAC_0$ can be used to manage $RBAC_3$, but there seems to be no point in using $ARBAC_3$ to manage $RBAC_0$.

Constraints can cut across both top and bottom halves of Figure 4. We have already described the built-in constraint regarding administrative and regular permissions; however, if administrative roles are mutually exclusive with respect to regular roles, we will have a situation in which system administrators can manage RBAC but not use any privileges themselves.

How will the administrative hierarchy be managed? Theoretically, a second-level administrative hierarchy could be built to manage the first-level one, and so on, but this is unnecessary in our opinion. The administrative hierarchy's administration can be handled by a single chief system administrator—a reasonable arrangement for either a single organization or a single administrative unit within an organization. Our model does not directly address the issue of how these units interact.

Administrative authority in RBAC is the ability to modify the user assignment, permission assignment, and role hierarchy relations. In a management model, the permissions that authorize these administrative operations must be explicitly defined. The precise nature of these permissions depends on the implementation, but they are generally alike.

A major management model issue is how to establish the scope of the administrative authority vested in administrative roles. To illustrate this, look at the hierarchies in Figure 3a. The administrative hierarchy of Figure 3c shows a single chief security officer (system administrator) role, which is senior to the three security officer roles SO1, SO2, and SO3.

The scoping issue essentially concerns how the tasks in Figure 3a might be managed by the security officers in Figure 3c, and we'll assume the chief security officer (CSO) can manage all tasks. Suppose SO1 manages task *T*1. We do not want SO1 to automatically inherit management of the junior role *P*, so SO1's scope can be limited to *T*1. Similarly, SO2's scope can be limited to *T*2. We'll assume SO3 can manage the entire subproject (*S*3, *T*3, *T*4, and *P*3), which means SO3's scope is bounded by *S*3 at the top and *P*3 at the bottom.

Usually, each administrative role is mapped to the subset of the role hierarchy it manages. There are, however, other aspects of management to be scoped—for example, SO1 may be able to add users only to the *T*1 role, while their removal requires the CSO to act. The permissions and users that the administrative role manages also need to be scoped, and changes in the role hierarchy itself must be controlled. For example, because SO3 manages the subhierarchy between *S*3 and *P*3, SO3 could be authorized to add additional tasks to that subproject.

OUR FAMILY OF RBAC MODELS systematically spans the spectrum from simple to complex. These models provide a common frame of reference for related research and development. We've shown, through a management model, that RBAC can be used to control itself. This supports our position that RBAC is policy neutral rather than a model of a specific security policy.

Many research problems must be solved if RBAC's potential is to be fulfilled. One is to develop a systematic approach to RBAC configuration design and analysis, although progress has been reported.[8,9,11] Because another problem is the lack of information about constraints with respect to RBAC, a constraints categorization and taxonomy would be useful. Also lacking is a formal notation for stating and enforcing constraints, along with some measure of enforcement difficulty. The ability to reason about constraints and analyze the net effect of an RBAC configuration in terms of higher level policy objectives is an important, open research area. The management aspects of RBAC need further work. Development of a systematic method-

ology that deals with the design and analysis of role hierarchies, constraints, and RBAC management in a unified framework is yet another challenging research goal.

Many of these open issues and problems are intertwined and will require an integrated approach to be satisfactorily resolved. ■

Acknowledgments

We are grateful to David Ferraiolo and Janet Cugini of the National Institute of Standards and Technology (NIST) for useful comments while this work was in progress. We also thank the anonymous reviewers, whose comments and suggestions have significantly improved the article. This work is funded in part by contracts 50-DKNA-4-00122 and 50-DKNB-5-00188 from the NIST. The work of Ravi Sandhu is also supported by grant CCR-9503560 from the National Science Foundation.

References

1. D.F. Ferraiolo, D.M. Gilbert, and N. Lynch, "An Examination of Federal and Commercial Access Control Policy Needs," *Proc. NIST-NCSC National Computer Security Conf.*, 1993, Nat'l Inst. Standards and Technology, Gaithersburg, Md., pp. 107-116.
2. Common Criteria Editorial Board, *Common Criteria for Information Technology Security Evaluation*, draft, Version 1.0, Nat'l Inst. Standards and Technology, Gaithersburg, Md., Jan. 1996.
3. I. Mohammed and D.M. Dilts, "Design for Dynamic User-Role-Based Security," *Computers & Security*, 1994, Vol. 13, No. 8, pp. 661-671.
4. R. Thomas and R.S. Sandhu, "Conceptual Foundations for a Model of Task-Based Authorizations," *Proc. IEEE Computer Security Foundations Workshop 7*, IEEE Press, Piscataway, N.J., June 1994, pp. 66-79.
5. R.S. Sandhu, "Lattice-Based Access Control Models," *Computer*, Vol. 26, No. 11, Nov. 1993, pp. 9-19.
6. D. Jonscher, "Extending Access Controls with Duties—Realized by Active Mechanisms," in *Database Security VI: Status and Prospects*, B. Thuraisingham and C.E. Landwehr, eds., Elsevier North-Holland, 1993, pp. 91-111.
7. D. Ferraiolo and R. Kuhn, "Role-Based Access Controls," *Proc. 15th NIST-NCSC Nat'l Computer Security Conf.*, Nat'l Inst. Standards and Technology, Gaithersburg, Md., 1992, pp. 554-563.
8. M.-Y. Hu, S.A. Demurjian, and T.C. Ting, "User-Role Based Security in the ADAM Object-Oriented Design and Analyses Environment," in *Database Security VIII: Status and Prospects*, J. Biskup et al., eds., Elsevier North-Holland, 1995, pp. 333-348.
9. M. Nyanchama and S. Osborn, "Access Rights Administration in Role-Based Security Systems," in *Database Security VIII: Status and Prospects*, J. Biskup et al., eds., Elsevier North-Holland, 1994, pp. 37-56.
10. S.H. von Solms and I. van der Merwe, "The Management of Computer Security Profiles Using a Role-Oriented Approach," *Computers & Security*, Vol. 13, No. 8, 1994, pp. 673-680.
11. E.B. Fernandez, J. Wu, and M.H. Fernandez, "User Group Structures in Object-Oriented Database Authorization," in *Database Security VIII: Status and Prospects*, J. Biskup et al., eds., Elsevier North-Holland, 1995.

Ravi S. Sandhu *is professor and associate chairman of Information and Software Systems Engineering at George Mason University, Fairfax, Virginia; director of the Laboratory for Information Security Technology at GMU; and a member of the senior staff at SETA Corporation, McLean, Virginia. His principal research and teaching interests are in information and systems security. Sandhu received PhD and MS degrees from Rutgers University, New Jersey, and BTech and MTech degrees from IIT Bombay and Delhi, India, respectively. He has consulted and extensively published on computer security. Sandhu chairs ACM's Special Interest Group on Security Audit and Control.*

Edward J. Coyne *is a principal scientist at SETA Corporation where, as part of a team, he developed an RBAC demonstration prototype. He is an expert on computer and communications security and has worked on relevant National Security Agency programs. Previously, at the Mitre Corporation, Coyne supported the National Computer Security Center in security evaluation of commercial computer systems. He received a PhD in computational linguistics from Georgetown University, Washington, D.C., in 1977; an MA in linguistics from American University, Washington, D.C., in 1972; and an MA in science and public policy in 1965 and a BS in astronomy in 1963, both from Case Institute of Technology, Cleveland, Ohio.*

Hal L. Feinstein *is a senior telecommunications specialist with SETA Corporation and is experienced in all aspects of information security, including policy development and risk analysis, cryptography and communications security, and Defense Department trusted computer system evaluation criteria analysis techniques. Previously, at the Mitre Corporation, Feinstein was a member of the company's networking center, where he supported the Defense Information System Agency's Defense Data Network Project Management Office to solve protocol-level problems. Feinstein was also an evaluator for the National Security Agency's National Computer Security Center. He received a BS in computer science from State University of New York at Potsdam in 1974.*

Charles E. Youman *has been a member of the senior technical staff at SETA Corporation since 1991. Previously, at the Mitre Corporation, Youman analyzed host security requirements for the Defense Department's trusted computer system evaluation criteria for the worldwide Military Command and Control System Information System. For the Federal Bureau of Investigation, Youman was a certified information systems auditor in a project to modernize the National Crime Information Center. He received an MS in administration, systems management, from George Washington University, Washington, D.C., in 1976 and a bachelor of engineering degree in information engineering from Vanderbilt University, Nashville, Tennessee, in 1969. He was awarded the certified information systems auditor designation by the Information Systems Audit and Control Association in 1979.*

Readers can contact Ravi Sandhu at the ISSE Department, MS 4A4, George Mason University, Fairfax, VA 22030; phone (703) 993-1659, fax (703) 993-1638, e-mail sandhu@isse.gmu.edu.

Howard Rubin, Computer*'s software metrics area editor, coordinated the review of this article and recommended it for publication. His e-mail address is 71031.377@compuserve.com.*