Role-Based Access Control (RBAC): Features and Motivations

David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn

National Institute of Standards and Technology
U. S. Department of Commerce
Gaithersburg MD 20899

Abstract

The central notion of Role-Based Access Control (RBAC) is that users do not have discretionary access to enterprise objects. Instead, access permissions are administratively associated with roles, and users are administratively made members of appropriate roles. This idea greatly simplifies management of authorization while providing an opportunity for great flexibility in specifying and enforcing enterprise- specific protection policies. Users can be made members of roles as determined by their responsibilities and qualifications and can be easily reassigned from one role to another without modifying the underlying access structure. Roles can be granted new permissions as new applications and actions are incorporated, and permissions can be revoked from roles as needed.

Some users and vendors have recognized the potential benefits of RBAC without a precise definition of what RBAC constitutes. Some RBAC features have been implemented in commercial products without a frame of reference as to the functional makeup and virtues of RBAC [1]. This lack of definition makes it difficult for consumers to compare products and for vendors to get credit for the effectiveness of their products in addressing known security problems. To correct these deficiencies, a number of government sponsored research efforts are underway to define RBAC precisely in terms of its features and the benefits it affords. This research includes: surveys to better understand the security needs of commercial and government users [2], the development of a formal RBAC model, architecture, prototype, and demonstrations to validate its use and feasibility. As a result of these efforts, RBAC systems are now beginning to emerge. The purpose of this paper is to provide additional insight as to the motivations and functionality that might go behind the official RBAC name.

1. Introduction

The principal motivations behind RBAC are the ability to articulate and enforce enterprise-specific security policies and to streamline the typically burdensome process of security management. RBAC represents a major advancement in flexibility and detail of control from the present-day standards of discretionary and mandatory access control [2][3][4][5]. In many enterprises within industry and civilian government, end users do not "own" the information for which they are allowed access as is often assumed by traditional discretionary access control schemes [6][7]. For these organizations, the corporation or agency is the actual "owner" of system objects and discretion on the part of the users may not be appropriate. With role-based access control, access decisions are based on the roles individual users have as part of an organization. As such, RBAC is often described as a form of non-discretionary access control in the sense that users are unavoidably constrained by the organization's protection policies. In non-classified environments, such policies are not focused on solving the multi-level security problem as is assumed within the existing standard for non-discretionary access control [6]. In contrast, RBAC allows for the specification and enforcement of a variety of protection policies which can be tailored on an enterprise-by-enterprise basis. The policies enforced in a particular stand-alone or distributed system are the net result of the precise configuration of the various components of RBAC. This RBAC framework provides administrators with the capability to regulate who can perform what actions, when, from where, in what order, and in some cases under what relational circumstances.

From a functional perspective, RBAC's central notion is that of *operations* representing actions associated with roles and *users* that are appropriately made members of *roles*. The relationships between users, roles, and operations is depicted in Figure 1. As shown in Figure 1, the use of double arrows indicate a many-to-many relationship. For example, a single user can be associated with one or more roles, and a single role can have one or more user members. Roles can be created for various job positions in an organization. For example, a role can include Teller or Loan Officer in a bank, or Doctor, Nurse, or Clinician in a hospital. The operations that are associated with roles constrain members of the role to a specified set of actions. For example, within a hospital system the role of Doctor can include operations to perform diagnosis, prescribe medication, and order laboratory tests; the role of Researcher can be limited to gathering anonymous clinical information for studies; and the role of Social Worker may be to review patient profiles to flag possible suicidal patients or determine possible abuse cases.

.

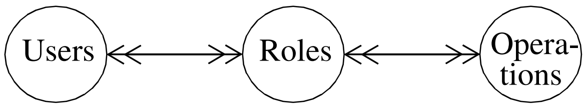

Figure 1. Users, roles, and operations

The association of operations with roles within an enterprise can be in compliance with rules that are self-imposed. For example, a health care provider may decide that the role of Clinician must be constrained to post only the results of certain tests rather than distribute them where routing and human errors can result in a violation of a patient's right to privacy. Operations can also be specified in a manner that can be used in the demonstration and enforcement of laws or regulations. For example, a nurse can be constrained to adding a new entry to a patient's history of treatments rather than being generally able to modify a patient record. A pharmacist can be provided with operations to dispense, but not to prescribe, medication.

Although RBAC does not promote any one protection policy, it has been shown to support several well-known security principles and policies that are important to commercial and government enterprises that process unclassified but sensitive information[2][8][9]. These include: the specification of competency to perform specific tasks; the enforcement of Least Privilege for administrators and general users; and the specification, as well as the enforcement, of conflicts of interest rules, which may entail duty assignment and dynamic and static separation of duties. These policies can be enforced at the time operations are authorized for a role, at the time users are authorized as members of a role, at the time of role activation (e.g., when a role is established as part of a user's active session), or when a user attempts to perform an operation on an object.

To demonstrate the importance of such policies, consider the unconstrained actions of Nicholas Leeson which lead to the bankruptcy of England's oldest investment firm in 1995. In particular, Leeson was allowed to run both the financial derivatives trading operation in Singapore as well as back-office functions where trades were settled. This is a mix of roles that can be – and in this case was – disastrous. In any firm serious about preventing fraud in its operations, this arrangement is flawed. Management at Barings PLC should never have had the same person making and settling trades. Such a conflict of interest policy can be specified centrally by management, administratively implemented, and enforced effectively using the RBAC framework.

In addition to RBAC's commercial relevance, RBAC has the potential to support policies that are essential within classified environments. Such policies can include one-directional information flow by the specification and enforcement of the Simple Security property and the Star-property[1][3][9].

One of RBAC's greatest virtues is the administrative capabilities it supports [3][5][9]. The administration of authorization data is widely acknowledged as an onerous process with a large and reoccurring expense. Under the RBAC framework, users are granted membership into roles based on their competencies and responsibilities. User membership into roles can be revoked easily and new memberships established as job assignments dictate. With RBAC, users are not granted permission to perform operations on an individual basis, but operations are associated with roles. Role association with new operations can be established as well as old operations deleted as organizational functions change and evolve. This basic concept has the advantage of simplifying the understanding and management of privileges: roles can be updated without having to update the privileges for every user on an individual basis.

Another administrative advantage of RBAC is that system administrators control access at a level of abstraction that is natural to the way enterprises typically conduct business. This is achieved by statically and dynamically regulating users' actions through the establishment and definition of roles, role hierarchies, relationships, and constraints. Thus, once an RBAC framework is established, the principal administrative actions are the granting and revoking of users into and out of roles. This is in contrast to the more conventional and less intuitive process of attempting to administer lower level access control mechanisms directly (e.g., access control lists (ACLs), capabilities, or type enforcement entities) on an object-by-object basis.

For distributed systems, another benefit is that RBAC administrator responsibilities can be divided among central and local protection domains, that is, central protection policies can be defined at an enterprise level while leaving protection issues that are of local concern at the

1. The Simple Security Property states that a subject (i.e., a process executing on a user's behalf) must not be allowed to read from storage repositories that are at a higher sensitivity level than the subject's current sensitivity level. The Star Property states that a subject must not be allowed to write to storage repositories that are at a lower sensitivity level than the subject's maximum sensitivity level allowed for reading.

organizational unit level. For example, within a distributed health care system, operations that are associated with health care providers may be centrally specified and pertain to all hospitals and clinics, but the granting and revoking of memberships into specific roles may be specified by administrators at local sites.

2. RBAC features and supporting policies

RBAC policies are described in terms of users, subjects, roles, role hierarchies, operations, and protected objects. To perform an operation on an object controlled under RBAC, a user must be active in some role. Before a user can be active in a role, that user must first have been authorized as a member of the role by a security administrator.

RBAC provides administrators with the capability to place constraints on role authorization, role activation, and operation execution. These constraints have a variety of forms. Constraints include cardinality and mutual exclusivity rules which can be applied on a role-by-role basis. In addition, constraints can be placed on the authorization of an operation to a role and on operations being performed on objects (i.e., time and location constraints).

The following subsections define RBAC entities and provide a precise definition for several representative constraints.

2.1 Users, roles, and operations

Within the RBAC framework, a *user* is a person, a *role* is a collection of job functions, and an *operation* represents a particular mode of access to a set of one or more protected RBAC *objects*. As shown in Figure 2, a *subject* represents an active user process with the single arrow denoting a one-to-many relationship.

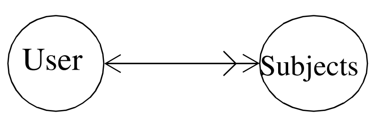

Figure 2. User and subjects

The following functions describe the mappings among users, subjects, and roles:

subject-user(s:subject) = the user associated with subject "s." (eq. 1)

authorized-roles(s:subject) = {the roles associated with subject "s"}. (eq. 2)

role-members(r:role)={the users authorized for role "r"}.(eq. 3)

user-authorized-roles(u:user) = {the roles associated with user "u"}. (eq. 4)

Note that the user associated with a subject is determined by a unique user identifier. Each subject is mapped to one individual user and possibly many roles.[2] RBAC also requires that if *authorized-roles*(s) = R and *subject-user*(s) = u, then "u" must be associated with the set of roles "R." This is better described as follows:

Assumption 1 (Consistent Subject) The consistent subject assumption is satisfied only if :

$$\forall\ s{:}subject,\ u{:}\ user,\ R,\ r{:}roles{:}\ \ subject\text{-}user(s) = u \ \wedge\ authorized\text{-}roles(s) = R \ \wedge\ u \in role\text{-}members(r) \ \Rightarrow\ r \in R. \qquad (eq.\ 5)$$

□

The type of operations and the objects that RBAC controls is dependent on the type of system in which it will be implemented. For example: within an operating system, operations might include read, write, and execute; within a database management system, operations might include insert, delete, append, and update; and within a transaction management system, operations would take the form of and exhibit all the properties of a transaction. The set of objects covered by the RBAC system include all of the objects accessible by the RBAC operations. However, not all file system and system objects need to be included in an RBAC scheme. For instance, access to infrastructure objects such as synchronization objects (e.g., semaphores, pipes, message segments) and temporary objects (e.g., temporary files and directories) may not necessarily be controlled within the RBAC protected object set.

An operation represents a unit of control that can be referenced by an individual role that is subject to regulatory constraints within the RBAC framework. It is important to note the difference between a simple mode of access and an operation. An operation can be used to capture security-relevant details or constraints that cannot be determined by a simple mode of access[2]. These details can be in terms of both method and granularity of access.

To demonstrate the importance of an RBAC operation, consider the differences between the access needs of a teller and an accounting supervisor in a bank. An enterprise defines a teller role as being able to perform a savings

2. The user identifier is relevant for auditing, but with an RBAC authorization scheme the role identifier(s) are what determine access.

deposit operation. This requires read and write access to specific fields within a savings file. An enterprise may also define an accounting supervisor role that is allowed to perform correction operations. These operations require read and write access to the same fields of a savings file as the teller. However, the accounting supervisor may not be allowed to initiate deposits or withdrawals but only perform corrections after the fact. Likewise, the teller is not allowed to perform any corrections once the transaction has been completed. The difference between these two roles is the operations that are executed by the different roles and the values that are written to the transaction log file.

To demonstrate the importance of granularity of control, consider the need of a pharmacist to access a patient's record to check for interactions between medications and to add notes to the medication section of the patient record. Although such operations may be necessary, the pharmacist should not be able to read or alter other parts of the patient record.

As shown in Figure 3, operations are administratively associated with objects as well as with roles.

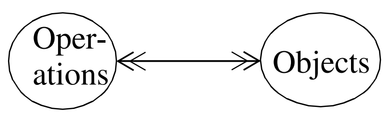

Figure 3. Operations and objects

When authorizing user membership into a role, the user is implicitly provided with the potential to execute the operations that are associated with the role. Each operation is referenced by a unique identifier. The notion of an operation, as well as the relationships between roles, operations, and objects are described by the following functions:

role-operations(r:roles) = {the operations that are associated with role "r"}. (eq. 6)

operation-objects(op:operation) = {the authorized objects for which the operation "op" can be applied}. (eq. 7)

2.2 Roles and role hierarchies

Roles can have overlapping responsibilities and privileges, that is, users belonging to different roles may need to perform common operations. Furthermore, within many organizations there are a number of general operations that are performed by all employees. As such, it would prove inefficient and administratively cumbersome to specify repeatedly these general operations for each role that gets created. To improve efficiency and provide for the natural structure of an enterprise, RBAC includes the concept of *role hierarchies*. A role hierarchy defines roles that have unique attributes and that may "contain" other roles, that is, that one role may implicitly include the operations, constraints, and objects that are associated with another role. Role hierarchies are a natural way of organizing roles to reflect authority and responsibility, and competency. An example of a role hierarchy is shown in Figure 4. In this example, the role Specialist "contains" the roles of Doctor and Intern. This means that members of the role Specialist are implicitly associated with the operations, constraints, and objects of the roles Doctor and Intern without the administrator having to explicitly list the Doctor and Intern attributes. The most powerful roles are represented at the top of the diagram with the less powerful roles being represented at the bottom, i.e., the roles on the top of the diagram contain the greatest number of operations, constraints, and objects. As shown in Figure 4, not all roles have to be related. The roles Cardiologist and Rheumatologist are not hierarchically related but they can contain some or all of the same roles.

Role hierarchies can be represented as ancestor relationships. The immediate parent relationship can be represented as an ordered pair $((R_{i+1},R_i), >)$, where R_{i+1} is the immediate parent and R_i the child and ">" is a transitive relation "contains." Thus, $R_{i+1} > R_i$, implies, R_{i+1} contains R_i. As shown in Figure 4, the role Specialist is the immediate parent of the role Doctor, and the role Intern is an ancestor of the role Specialist (i.e., the role Specialist "contains" the role Intern). However, the role Cardiologist is not an ancestor of the role Rheumatologist. Because roles are contained by other roles through the "contains" relationship, granting membership in a role implies membership to all the roles that role "contains."

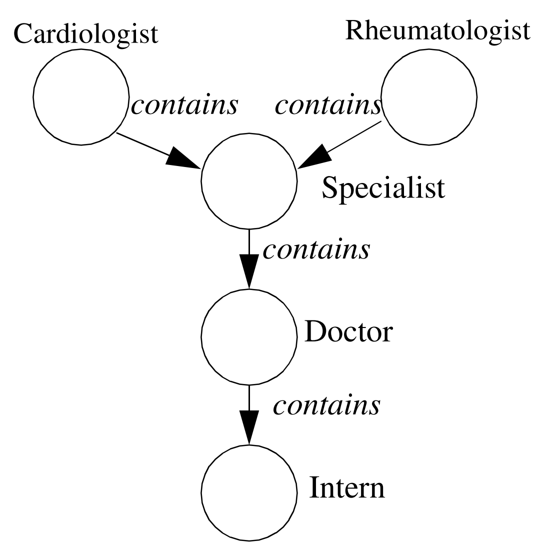

Figure 4. Example of a role hierarchy

This role hierarchy can be described as:

Rule 1 (Role Hierarchy) If a subject is authorized to access a role and that role contains another role, then the subject is also allowed to access the contained role:

$$\forall \; s{:}subject,\; r_{i,j}{:}roles:$$
$$r_j \in authorized\text{-}roles(s) \wedge r_j > r_i \Rightarrow r_i \in authorized\text{-}roles(s). \qquad \text{(eq. 8)}$$

□

2.3 Role authorization

The association of a user with a role can be subject to the following:

• the user can be given no more privilege than is necessary to perform his/her job;

• the role in which the user is gaining membership is not mutually exclusive with another role for which the user already possesses membership; and

• the numerical limitation that exists for role membership cannot be exceeded.

The first property in intended to ensure adherence to the principle of Least Privilege [2]. The principle of Least Privilege requires that a user be given no more privileges than necessary to perform his/her job function. Ensuring least privilege requires identifying the user's job functions, determining the minimum set of privileges required to perform that function, and restricting the user to a domain with those privileges and nothing more. In non-RBAC implementations, this is often difficult or costly to achieve. For example, in a capability-based system, someone assigned to a job category may be allowed more privileges than one needs because of the inability of capability-based systems to tailor access based on various attributes or constraints. Since many of the responsibilities overlap between job categories, maximum privilege for each job category could cause unlawful access. RBAC can be configured so that only those operations that need to be performed by members of a role are granted to the role, and these operations and roles can be subject to organizational policies or constraints. In the cases where operations overlap, hierarchies of roles can be established. In the past, careful auditing has been used to justify the granting of greater access. For example, it may seem sufficient to allow physicians to have access to all patient data records if their access is monitored sufficiently. However, this would entail much more auditing and monitoring than would be necessary with a better defined access control mechanism. With RBAC, constraints can be placed on physician access so that, for example, only those records that are associated with a particular physician can be accessed.

The second property listed above is intended to preserve a policy of Static Separation of Duty or conflict of interest. This means that by virtue of a user being authorized as a member of one role, the user is not authorized as a member of a second role. For example, a user that is authorized to be a member of the role Teller in a bank may not be allowed to be a member of the role Auditor of the same bank. That is, the roles Teller and Auditor are mutually exclusive.

The policy of Static Separation of Duty can be centrally specified and can then be uniformly imposed on specific roles. The mutually exclusive roles for a given role and the Static Separation of Duty property can be specified as follows:

mutually-exclusive-authorization(r:roles) = {the list of roles that are mutually exclusive with role "r"}. (eq. 9)

Rule 2 (Static Separation of Duty) A user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership:

$$\forall \; u{:}user,\; r_{i,j}{:}roles : i \neq j :$$
$$u \in role\text{-}members(r_i) \wedge u \in role\text{-}members(r_j)$$
$$\Rightarrow r_i \notin mutually\text{-}exclusive\text{-}authorization(r_j) \qquad \text{(eq. 10)}$$

□

The third property listed above which can be preserved under the granting of user membership to roles is the Cardinality property. Some roles can only be occupied by a certain number of employees at any given period of time. For example, consider the role of Manager. Although other employees may act in that role, only one employee may assume the responsibilities of a manager at any given time. A user can become a new member of a role as long as the number of members allowed for the role is not exceeded. The number of users allowed for a role and the existing number of users associated with a role is specified by the following two functions:

membership-limit (r: roles) = the membership limit ($\geq$ 0) for role "r." (eq. 11)

number-of-members(r:roles) = N ($\geq$ 0) the number of existing members in role "r." (eq. 12)

Role capacity can now be described as:

Rule 3 (Cardinality) The capacity of a role cannot be exceeded by an additional role member:

$$\forall \; r{:}roles:$$

$$membership\text{-}limit(r) \geq number\text{-}of\text{-}members(r). \quad \text{(eq. 13)}$$

□

2.4 Role activation

Each subject is a mapping of a user to one or possibly many roles. A user establishes a session during which the user is associated with a subset of roles for which the user has membership. A user's role authorization (which is a consequence of role membership) is a necessary but not always sufficient condition for a user to be permitted to perform an operation. Other organizational policy considerations or constraints may need to be taken into account that pertain to authorizing users to perform operations.

Role activation provides the context for which these organizational policies can be applied. As such, RBAC requires a user to first be authorized as being active in a role before a user can perform an action.

Depending on the organizational policy under consideration, checks are applied in terms of the role which is being proposed for activation, the operation which is being requested for execution, and/or the object which is being accessed. That is, a role can be activated if:

- the user is authorized for the role being proposed for activation;
- the activation of the proposed role is not mutually exclusive with any other active role(s) of the user;
- the proposed operation is authorized for the role that is being proposed for activation;
- the operation being proposed is consistent within a mandatory sequence of operations; and

The following functions enable subjects to execute RBAC operations and define the active roles for a subject:

exec: (s: subject, op: operation) = {TRUE iff subject "s" can execute operation "op," otherwise it is FALSE}. (eq. 14)

active-roles(s:subject) = {the current list of active roles for subject "s"}. (eq. 15)

The specification that a subject's proposed active role must be in the authorized role set for that subject is stated by the following property:

Rule 4 (Role Authorization) A subject can never have an active role that is not authorized for that subject:

$$\forall\ s{:}subject,\ op{:}\ operation:\ active\text{-}roles(s) \subseteq authorized\text{-}role(s). \quad \text{(eq. 16)}$$

□

Once it is determined that a role is part of the authorized role set for the subject, the operation can be executed provided that the role is active. Even though a role may be in the role set, there may be certain organizational policies (such as dynamic separation of duty described below) that precludes the role from being activated. This provides the context from which other checks are made and is specified by the following rule:

Rule 5 (Role Execution) A subject can execute an operation only if the subject is acting within an active role:

$$\forall\ s{:}subject,\ op{:}\ operation:\ exec(s,op) \Rightarrow active\text{-}roles(s) \neq \emptyset. \quad \text{(eq. 17)}$$

□

RBAC also provides administrators with the capability to enforce an organization-specific policy of Dynamic Separation of Duty. Static Separation of Duty provides an enterprise with the capability to address potential conflicts of interest issues at the time a user's membership is authorized for a role. However, in some organizations it is permissible for a user to be a member of two roles which do not constitute a conflict of interest when acted in independently, but introduce policy concerns when allowed to be acted in simultaneously.

For example, a static policy could require that no individual who has the role of Payment Initiator can also have the role of Payment Authorizer. Although such an approach may be adequate for some organizations, for others it may prove too rigid, making the cost of separation greater than the loss that might be expected. The objective behind Dynamic Separation of Duty is to allow more flexibility in operations. Dynamic Separation of Duty places constraints on the simultaneous activation of roles, so for example, an individual user can be authorized for both the roles Payment Initiator and Authorizer, but can dynamically assume only one of these roles at the same time.

The mutually exclusive roles for the proposed active role is specified by the following function:

mutually-exclusive-activation(r:roles) = {the list of active roles that are mutually exclusive with the proposed role "r"}. (eq. 18)

The RBAC Dynamic Separation of Duty rule is defined as:

Rule 6 (Dynamic Separation of Duty) A subject can become active in a new role only if the proposed role is not mutually exclusive with any of the roles in which the subject is currently active:

$$\forall \; s{:}subject,\; r_{i,j}{:}roles : i \neq j :$$
$$r_i \in active\text{-}roles(s) \wedge r_j \in active\text{-}roles(s)$$
$$\Rightarrow r_i \notin mutually\text{-}exclusive\text{-}activation(r_j). \quad \text{(eq. 19)}$$

□

The specification that a subject can perform an operation only if the operation is authorized for the subject's proposed active role is provided by the following property:

Rule 7 (Operation Authorization) A subject can execute an operation only if the operation is authorized for the role in which the subject is currently active:

$$\forall \; s{:}subject,\; op{:}\; operation \; \exists \; r{:}roles :$$
$$exec(s,op) \Longrightarrow r \in active\text{-}roles(s) \wedge$$
$$op \in role\text{-}operations(r). \quad \text{(eq. 20)}$$

□

2.5 Operational separation of duty

RBAC can be used by a system administrator to enforce a policy of Operational Separation of Duty. Operational Separation of Duty can be a valuable approach at deterring fraud [14]. This is based on the idea that fraud can occur if collaboration exists between various job-related capabilities within a critical business function. For example, the function of purchasing an item might involve the following operations: authorizing the purchase order; recording the arrival of the invoice; recording the arrival of the item; and, finally, authorizing payment [14].[3] If each of these operations is performed by different roles, the likelihood of fraud can be diminished. By allowing one user to perform all operations, fraud may occur.

Operational Separation of Duty requires that for all the operations associated with a particular business function, no single user can be allowed to perform all of these operations. Therefore, the failure of one role to perform as expected can be detected by the organization. In RBAC terms, the Operational Separation of Duty policy can be enforced when roles are authorized for individual users and when operations are assigned to roles.

3. Execution of a single operation of a business function proceeds only upon the successful completion of the previous operation.

Operational Separation of Duty can be specified with the following function and property:

function-operations(f:function) = {the set of all operations required for a business function "f"}. (eq. 21)

Rule 8 (Operational Separation of Duty) A role can be associated with an operation of a business function only if the role is an authorized role for the subject and the role had not been assigned previously to all of the other operations:[4]

$$\forall \; s{:}subject,\; r{:}role,\; f{:}function :$$
$$\neg \; (function\text{-}operations(f) \subseteq \bigcup_{r \in ur(u)} role\text{-}operations(r)). \quad \text{(eq. 22)}$$

□

2.6 Accessing objects

To ensure enforcement of enterprise policies for RBAC objects, subject access to RBAC objects must be controlled. The following function is used to determine if a subject can access an RBAC object:

access (s: subject, o: object) = {TRUE iff the subject can access the object, otherwise it is FALSE}. (eq. 23)

With the Role Authorization and Role Execution properties defined above (Rules 4 and 5), the Operation Access Authorization property defined below ensures that a subject's access an RBAC object can only be achieved through authorized operations by authorized active roles.

Rule 9 (Object Access Authorization) A subject can access an object only if the role is part of the subject's current active role set, the role is allowed to perform the operation, and the operation to access the object is authorized:

$$\forall \; s{:}subject,\; o{:}\; object :$$
$$access(s,o) \Longrightarrow \exists \; r{:}\; roles,\; op{:}operation :$$
$$r \in active\text{-}roles(s) \wedge op \in role\text{-}operations(r)$$
$$\wedge \; o \in operation\text{-}objects(op). \quad \text{(eq. 24)}$$

□

3. Conclusion

The principal motivations behind RBAC are the ability to express and enforce enterprise-specific security policies and streamline the typically burdensome process of security management. As shown above, RBAC is a framework of

4. For this property, *user-authorized-roles(u)* is represented by *ur(u).*

policy rich mechanisms, and its configuration is dependent on organizational policies. This allows RBAC to be adaptable to any organizational structure and means of conducting business. Also, the policies implemented under RBAC can evolve over time as enterprise and organizational structure and security needs change. RBAC provides greater productivity on the part of security administrators, resulting in fewer errors and a greater degree of operational security.

Currently, the National Institute of Standards and Technology (NIST) is conducting research in the area of RBAC. To date three independently developed efforts on RBAC are underway at NIST: a Small Business Innovation Research (SBIR) program with Dr. Ravi Sandhu of George Mason University and Seta Corporation to help define RBAC and its feasibility, an effort with NSA's R23 Research and Engineering group and Dr. Virgil Gligor of the University of Maryland to create a formal model and implement RBAC on a policy-independent Mach microkernel-based operating system being developed by R23 called Synergy [6][10], and a Advanced Technology Program (ATP) effort being led by John Barkley of NIST to demonstrate how RBAC can be used for a health care system. Although these research efforts have shown great promise and continues to generate enthusiasm within the research and vendor communities, RBAC remains a long way from reaching its full potential as a commercially viable technology. This could only be achieved through further research and consensus on the part of researchers, vendors, and the user community.

Acknowledgments

The authors would like to thank Dr. Virgil Gligor, NSA R23 group, Dr. Ravi Sandhu, and the Seta Corp. for their efforts in making RBAC a reality. The authors would also like to thank Bruce Aldridge and John Barkley of NIST for their assistance in reviewing this paper.

4. References

[1] Oracle Corporation, *ORACLE7 Server SQL Language Reference Manual*, 778-70-1292, December 1992.

[2] David F. Ferraiolo, Dennis M. Gilbert, Nickilyn Lynch, "An Examination of Federal and Commercial Access Control Policy Needs," *Proceedings of the 16th NIST-NSA National Computer Security Conference*, Baltimore, MD, 20-23 September 1993.

[3] Ravi S. Sandhu, et al., *Role-Based Access Control Models,* unpublished journal article.

[4] Imtiaz Mohammed and David M. Ditts, "Design for Dynamic User Role-Based Security," *Computers and Security,* 1994.

[5] David F. Ferraiolo and Richard Kuhn, "Role-Based Access Control," *Proceedings of the 15th NIST-NSA National Computer Security Conference*, Baltimore, MD, 13-16 October 1992.

[6] Department of Defense, *Trusted Computer Security Evaluation Criteria*, DoD 5200.28-STD, 1985.

[7] National Computer Security Center, *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, September 1987.

[8] Deborah Hamilton, "Application Layer Security Requirements of a Medical Information System," *Proceedings of the 15th NIST-NSA National Computer Security Conference*, Baltimore, MD, 13-16 October 1992.

[9] Hal L. Feinstein, et al., *Final Report: Small Business Innovation Research (SBIR): Role-Based Access Control: Phase 1*, McLean, VA, SETA Corporation, January 20, 1995.

[10] Virgil Gligor, *RBAC Security Policy Model, Preliminary Draft Report,* R23 Research and Development Department of the National Security Agency, April 1995.

[11] Daniel F. Stern, "A TCB Subset for Role-Based Access Control," *Proceedings of the 15th NIST-NSA National Computer Security Conference*, Baltimore, MD, 13-16 October 1992.

[12] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, W.R. Shockley, *The Seaview Formal Security Policy Model, Final Report*, Rome Air Development Center, Volume 2, February 1989.

[13] R. W. Baldwin, "Naming and Grouping Privileges to Simplify Security Management in Large Databases," *Proceedings of the IEEE Symposium on Computer Security and Privacy*, 1990.

[14] D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the IEEE Symposium on Computer Security and Privacy*, April 1987.

[15] Virgil Gligor, J. Huskamp, S. Welke, C. Linn, and W. T. Mayfield, *Traditional Capability-Based Systems: An Analysis of Their Ability to Meet the Trusted Computer Security Evaluation Criteria,* IDA Paper P-1935, October 1986.

[16] D. E. Bell and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, Technical Report. ESD-TR-73-278, Volume 1, The MITRE Corporation, Bedford, MA, March 1973.

[17] S. H. von Solms and Isak VanderMerve, "The Management of Computer Security Profiles Using a Role-Oriented Approach," *Computers and Security,* 1994.