An Integrity Enforcement Application Design and Operation Framework in Role-Based Access Control Systems: A Session-Oriented Approach

HyungHyo Lee, BongNam Noh
Dept. of Computer Science, Chonnam National University, Kwangju, Korea
hlee@athena.chonnam.ac.kr, bongnam@chonnam.ac.kr

Abstract

Role-Based Access Control(RBAC) policy is being widely accepted not only as an access control policy but as a flexible permission management framework in various commercial environments. RBAC simplifies the process of security management by assigning permissions to roles not directly to individual users. As security administrators can design and manage security policies by changing the configuration of RBAC components to meet their organization's own security needs, RBAC is called policy-neutral and has ability to articulate enterprise-specific security policies. While most researches on RBAC are for defining, describing model in formal method and other important properties such as separation of duty, little work has been done on how applications should be designed and then executed in automated information systems based on RBAC security model. In this paper, we describe important, dynamic features of a session that can be used as a vehicle for building applications, and present a basic framework for session-oriented integrity enforcement application design and operation applicable to commercial environments.

1. Introduction

As computer and information systems around the world are getting connected via various communication channels, information security is becoming one of most important issues in every organization. *Trusted Computer System Evaluation Criteria*(TCSEC) contains security features and assurances, exclusively derived, engineered and rationalized based on DOD security policy[5]. It was to meet a major security objective - preventing the unauthorized observation of classified information[6]. The TCSEC specifies two types of access controls: Discretionary Access Control(DAC) and Mandatory Access Control(MAC). MAC is used for multi-level secure military systems, but its usage in other applications is rare. DAC has been perceived are being technically correct for commercial and civilian government security needs. MAC is based on rules or axioms for deciding users' access request to objects. Those rules are based on the security levels: *clearance level* for users and *classification level* of objects, assigned by security administrator[3]. DAC permits the granting and revoking of access privileges to be left to the discretion of the individual users, especially the owner of objects.

While civilian governments and corporations are concerned with protecting the secrecy of information as in military environments, many of these organizations have been greater concern for integrity[4]. Each organization has unique security requirements, and many of them are difficult to meet by traditional MAC and DAC controls. In addition, MAC is too rigid for commercial environments, and DAC can not provide controlled or centralized access control capability that almost organizations need.

But in *Role-Based Access Control*(RBAC), access control decisions are determined by the roles individual users take on as part of an organization. Also, in RBAC, access permissions are assigned not to individual users but to roles, and access permission can not pass to other users at users' discretion. This greatly simplifies the process of permission management in complex and ever changing contemporary commercial environments.

In order to uphold the integrity properties of information system where RBAC is applied, several separation duty policies are defined[7]. They differ in several criteria such as enforcement phase(static or dynamic), flexibility, and the target which separation of duty applied(role or object), etc. Separation of duty is to minimize the likelihood of collusion that can be made among the users who have privileges to perform operations on objects pertinent to integrity properties. Researches on separation of duty to date are concerned with what the exact meaning and definition of separation of duty are, and the relationships among them[4,7,10].

While most researches on RBAC are for defining and describing model and other important properties such as separation of duty in formal method, little work has been done on how applications should be designed and executed in automated systems upholding security policies on the other hand. In this paper, we describe important features and properties of session as a vehicle for building

applications and present a framework for session-oriented integrity enforcement application design and operation.

In section 2, key features and major components of RBAC model are described. In order to reduce the ambiguity and incorrectness, a formal description for RBAC model and various separation of duty properties are also reviewed. We revisit the concept and features of session needed to build integrity preserving application design and operation in section 3. Section 4 concludes the features of the proposed framework for flexible and effective business application design and development.

2. Role-Based Access Control

As mentioned above, MAC is based on rules imposed by security domain authority that can be automatically enforced and is most common in military and other classified environments. In contrast, DAC has much greater flexibility for providing access rights to resources to owners of those resources. But none of them solely is adequate for the commercial environment where both flexible and rather centralized control are equally important.

RBAC is another type of security policy which is particularly not only valuable in contemporary commercial environments but every other environment such as operating systems and database systems[2]. RBAC is concerned more with access to functions and information than strictly with access to information as in MAC and DAC policy[1,10]. Sandhu et al.[11] define the family of RBAC models, and Ferraiolo et al.[7] developed formal RBAC model as well as its architecture and prototype. RBAC model is considered policy-neutral and a means for articulating policy rather than embodying a particular security policy[13,14]. In this section, we briefly summarize the features and components of RBAC model.

2.1. Features and Components of RBAC Model

RBAC has recently received considerable attention as a promising alternative to traditional DAC and MAC[6,11]. In RBAC, permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies the management of permissions[14]. Although the RBAC concept is policy-neutral, it directly supports three well-known security principles such as least privilege, separation of duties, and data abstraction.

Figure 1 shows the components of RBAC model[11]. A *role* is a named job function within the organization that describes both the equality and responsibility conferred on a member of the role. *Role hierarchy*(RH), a partial order relationship among roles, is a natural means for both

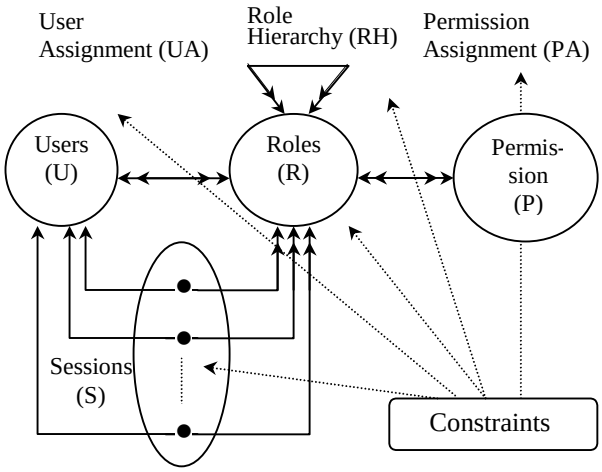

Figure 1: Components of RBAC Model

structuring roles to reflect an organization's lines of authority and responsibility. *User* is either a human being or process that interacts directly with a computer system. *Permission* is an approval of a particular access to one or more objects in the system. Permission is sometimes expressed in the term such as authorization, access right and privilege. Permissions can apply to single object or to many, and they can be as specific as read access to a particular file or as generic as read access to a particular department. Objects are data objects or resources of system represented by data in the computer system. *Sessions* map one user to possibly many roles. Sessions are established by users, and during which users may activate a subset of roles they belong to.

Both *user assignment*(UA) and *permission assignment*(PA) relations are many-to-many and important components of RBAC. The role's position as an intermediary to let user exercise permission provides greater control over access configuration than does a direct relationship between users and permissions. *Constraints* can apply to any of the preceding components. Examples of constraints are separation of duties, cardinality and prerequisite roles, etc.

2.2. Formal Description of RBAC

Formal methods are to reduce misunderstandings among different groups of software developers and across different stages of the development process. Formal methods utilize specification techniques to precisely state and analyze the behaviors of system. In this paper, we use Z as a formal specification method to describe both properties and components of RBAC. Z is based on typed set theory and the schema construct[15]. Schemas are used to capture both static and dynamic aspects of system being modeled. At first, we declare the following basic types, functions and state schemas for RBAC model.

[***USERS***, ***OPERATIONS, OBJECTS***, ***CONSTRAINTS***]
USERS: a set of user or process identifiers,
OPERATIONS: a set of operations,
OBJECTS: a set of objects that contain information,
CONSTRAINTS: a set of constraints.

users_of_role: Role → ℙ ***USERS***
roles_of_user: ***USERS*** → ℙ Role
user_of_session: Session → ***USERS***
roles_of_session: Session → ℙ Role
active_roles: Session → ℙ Role
mutex_roles: Role ∪ ℙ Role → ℙ Role

Permission
operations: ***OPERATIONS***
objects: ℙ ***OBJECTS***

Role
name: ***STRING***
users: ℙ ***USER***
permissions: **P** Permission

Session
name: ***STRING***
user: ***USERS***
roles: ℙ Role
∀ r : Role | r ∈ roles • user ∈ users_of_role(r)

A set of active roles for a specific session is a subset of roles that are authorized to the user of that session. Whether all of the roles for session become active or not when that session is activated depends on the policy of organization.

2.3. Separation of Duty

While secrecy of information is the key concern in specialized environments, integrity may be the most important property in commercial environments. To ensure the integrity of information, *separation of duty*(SOD), as a security principle, is introduced. The purpose of SOD is to prevent and minimize those chances of collusion, assigning users with the integrity-related conflicting interests or *mutually exclusive roles* to separate users[9].

SOD can be either static or dynamic. Static SOD requirements can be simply satisfied by the assignment of individuals to roles. Static SOD requires that the same user can not be made member of mutually exclusive roles. But such a policy is too restrictive for commercial domain, another more flexible SOD policy is needed. Dynamic SOD allows one user to be a member of mutually exclusive roles, but places constraints on the simultaneous activation of roles. The objective behind dynamic SOD is to allow more flexibility in operations, and checking the compliance with the dynamic SOD requirement can be determined only during system operation. But, it is more difficult to show given transaction complies with the dynamic SOD requirement.

Another important SOD is operational SOD. Operational SOD requires that for all the operations associated with a particular business function, no single user can be allowed to perform all of these operations. Clark and Wilson, Nash and Poland, and Simon and Zurko introduce the wide variety of SOD and describe the properties of each SOD[3,4,10]. Gligor et al. formally define various SOD properties and establish their relationship within a formal model of RBAC[9].

Static, dynamic and operational separation of duty policies are represented formally as follows:

StaticSOD
$r_1?, r_2?$: Role
$r_1? \in mutex\_roles(r_2?) \wedge r_2? \in mutex\_roles(r_1?) \wedge$
$users\_of\_role(r_1?) \cap users\_of\_role(r_2?) = \varnothing$

DynamicSOD
s?: Session
∀ r : Role | r ∈ s?.roles •
mutex_roles(r) ∩ s?.roles = ∅

Both static and dynamic SOD policies could be used to maintain the integrity of information systems effectively if applied appropriately.

Before we define the operational SOD in formal method, the concept of business function or application also should be clarified. As each session should have only one user in RBAC model, two of more users or sessions are grouped together to execute integrity-relevant applications. In other words, application can be described as inter-related sessions with some constraints for preserving integrity properties of organization. Like role membership(static) and session(dynamic), application also is another target of SOD policy applied.

Application
name: ***STRING***
initiators: ℙ Role
sessions: ℙ Session
constraint: ***CONSTRAINTS***

OperationalSOD
app?: Application
¬ (∀ s : Session; ∀ r : Role; ∃ u: ***USERS*** |
s ∈ app?.sessions, r ∈ roles_of_session(s) •
u ∈ users_of_role(r))

But, as the constraint of operational SOD is too weak and does not consider the meaning and the relationship among sessions of application, there is a possibility that the integrity may be compromised. In section 3, in order to prevent such an integrity violation, we revisit and

extend the concept of session in commercial environments and propose additional strict separation of duty for upholding the integrity of applications.

3. Session-Oriented Application Design and Operation Framework

As noted earlier, we define business function or application as inter-related sessions and some constraints for preserving integrity properties. Despite the several important features session has, little work has been done about how it can be used as a vehicle for building applications, and when user and roles should be assigned to each session, and in which way sessions are related each other, etc. Furthermore, in order to session-oriented application design and operation is implemented in computerized and automated system, the concept and features of session should be explored. Traditional meaning of session is a logical duration of interaction between system and user from user login to user logout. But, in automated information processing environments, session can be invoked and assigned to user dynamically within the security policy stipulated by security administrator. This, *dynamic* and extended view of session, made RBAC model be a useful in workflow and distributed systems.

As session in RBAC model plays an important role in our research, we define and describe the important features of sessions.

Reusable, Sharable Unit of Application

In real world, a unit of job performed by single user may be used or sharable in several, independent applications. Such a job function can be mapped to session of RBAC model, and session is a building block for applications in RBAC systems.

Enforcement of Least Privilege Principle

As RBAC model permits a user to be a member of several roles, privileges authorized to a user are the union of the privileges of each role to which a user is assigned. But session activates the only needed roles, a subset of authorized roles to given user to execute the part of application function, the possibility of privilege misuse is minimized.

$\forall$ u : ***USERS***, $\forall$ s: Session | user_of_session(s) = u $\bullet$ roles_of_session(s) $\subseteq$ roles_of_user(u)

Enforcement of Separation of Duty Enforcement

In order to preserve the integrity of information systems, every integrity-related application should not be executed by a single user. Such an application should be divided into sessions and each of those sessions should be assigned to different user not to compromise the integrity.

Provision of Higher Level of Abstraction

In the previous works of RBAC model, applications as regarded as a collection of objects, operations on them, and other constraints such as execution sequence, etc[9]. As a consequence, some SOD properties are described in terms of primitive objects and operations. But, session-oriented application design approach where objects and operations are encapsulated in session provides higher level of abstraction and management of SOD properties to application designers and security administrators.

3.1. Application in Session-Oriented Approach

As we defined session in formal method, application consists of well-defined sessions, constrains such as SOD properties, execution order, temporal limitations, its unique name, etc. Applications can be regarded as not just a service providing entity but as a dynamic integrity enforcement unit providing session activation, session-to-user mapping according to their constraints.

3.2. Session-Based Separation of Duty

The definition of operational SOD requires that all the operations associated with a particular business function, no single user can be allowed to perform all of those operations[7,9]. But there exists integrity hole even if applications satisfy operational SOD.

It may be possible all sessions of an application perform integrity-related activities. But in most cases, if some sessions are assigned to a single user, integrity of information system may be severely compromised. In other words, compliance with operational SOD does not always guarantee the integrity properties. For example, suppose a simple item-purchasing application that consists of 3 sequential sessions:

s_1: making an item-purchasing order
s_2: checking /recording numbers of the arrived items
s_3: authorizing the payment for items

Suppose user, u_1, is assigned to session s_1 and s_2, and u_2 executes s_3. Although this application satisfies operational SOD, but if u_1 intentionally checks on the items in wrong way, the integrity of organization's information system is seriously compromised. Therefore, we propose a new *session-based SOD* to prevent such possible security problems.

As SOD is application-dependent, application designer can specify which sessions should not be assigned to a single user as in mutually exclusive roles. Mutually exclusive sessions are formally expressed as follows:

MUTEX_SESSIONS ==
$\mathbb{P}(\text{Session} \times \text{Session}) \setminus \{ s_i, s_j : \text{Session} \mid s_i = s_j \bullet (s_i, s_j) \}$

SessionSOD

app?: Application
mutex_sessions: MUTEX_SESSIONS

$\forall s_i, s_j : \text{Session} \mid s_i \neq s_j,$
$s_i \in \text{app?.sessions}, s_j \in \text{app?.sessions} \bullet$
$(s_i, s_j) \in \text{mutex\_sessions} \lor (s_j, s_i) \in \text{mutex\_sessions}$
$\Rightarrow \text{user\_of\_session}(s_i) \neq \text{user\_of\_session}(s_j)$

3.3. Session-User, Session-Role Assignment

In RBAC model, both UA and PA procedures are performed by security administrator. But there is no rule or research effort on how and when session-to-user, session-to-role(s) mapping should be done. As there are a lot of different configurations to assigning roles and user to a session, *Session-User Assignment*(SUA) and *Session-Role Assignment*(SRA) policy may be an important factor for deciding the flexibility of integrity enforcement tasks of the system. If application designer designates both user and roles for a specific session in application design phase, performance of system may be enhanced but the flexibility should be diminished. But constructing a session and assigning to a session in runtime greatly may increase the flexibility of system, but the performance should be degraded.

In order to provide moderate flexibility and performance of system, we propose the policy in which SRA is performed in application design phase and SUA is carried out in application operation phase. Considering the relationship between roles and permissions is rather stable, whereas the changes in users and roles relationship is relatively frequent, proposed SRA and SUA policy is reasonable.

3.4. Feasibility Checking of Application

Feasibility checking in application design phase only checks the existence of the instance in which the designed application can be executed as specified. Due to dynamic, operational SOD and constraints of RBAC components such as number of users, maximum number of session permitted per user, maximum number of session that system can afford, some applications may not be performed to the end even if they are passed by feasibility algorithm.

Before we present feasibility checking schema, we describe a new variant of SOD to enhance the flexibility of integrity enforcement process and some application specification constructs in the first place.

Feasibility checking in application design phase is whether there exists a case in which session-based SOD is satisfied. In order to define schema for checking application's feasibility, we introduce schema named 'CandidateUserSet'.

CandidateUserSet

session?: Session
candidate_users!: $\mathbb{P}$ ***USERS***

candidate_users! = { $\forall$ u : ***USERS***; $\forall$ r : Role |
r $\in$ roles_of_session(session?) $\bullet$
u $\in$ users_of_role(r) }

It returns a set of users whose authorized roles contain roles of a given session, and each of them is capable of performing given session. Formal definition for checking the feasibility of a given application based on session-based SOD is as follows:

FeasibilityCheck
SessionSOD

$\forall s_i, s_j : \text{Session} \mid (s_i, s_j) \in \text{mutex\_sessions} \bullet$
$\text{CandidateUserSet}(s_i) \neq \varnothing \land$
$\text{CandidateUserSet}(s_j) \neq \varnothing \land$
$\text{CandidateUserSet}(s_i) \setminus \text{CandidateUserSet}(s_j) \neq \varnothing \land$
$\text{CandidateUserSet}(s_j) \setminus \text{CandidateUserSet}(s_i) \neq \varnothing$

3.5. Session-Oriented Application Specification

Session-oriented application specification constructs are useful not only for building applications but as a frame of reference in application operation phase. Several application specification constructs are showed in table 1.

In table 1, session-list may be any session composed by specification constructs. A special construct for termination, **abort**, is used to quit the application if specific condition holds. In addition, temporal constraints may be added to enhance the integrity properties of information systems. Temporal constraints may include the limitation of the session start or termination time, the maximum time interval between a pair of sessions, etc.

Table 1: Application Specification Construct Examples

construct	notation
sequential	s_1**;** s_2
parallel	s_1 ‖ s_2
conditional	**if** <condition> **then** s_1 **else** s_2
loop	**while** <condition> **do** <session-list> **with max_loop** = <integer>
grouping	**(**<session-list>**)**
termination	**abort**

Figure 2 depicts the overall structure of session-oriented integrity enforcement application design and

operation framework. Session layer provides reusable, functional unit to applications and runtime module provides session activation, session-to-user mapping according to the constraints of given application.

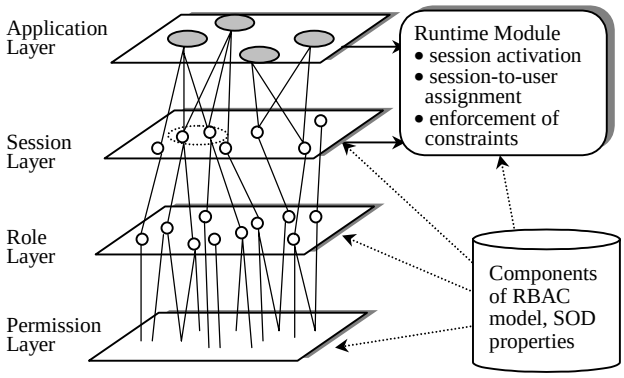

Figure 2: Integrity Enforcement Application Design and Operation Framework

In RBAC administration phase, security administrator decides which SOD policies adopted, adds or deletes RBAC components based on SOD policies. For example, before adding a user to member of role or roles, compliance with static SOD policy should be checked.

Application designer analyzes the characteristics of business function in the first place and designs integrity-preserving application by constructing constituent sessions of application. Deciding which roles are grouped together into a session, and which user is assigned to each session, checking the feasibility of designed application should be compliant with various SOD policies.

In application operation, both dynamic and operational SOD are referred to ensure the integrity of information system(Table 2).

Table 2: Phases and Activities of Session-Oriented Security Systems

Phase	Activities	Role
security administration	- management of components of RBAC model, SOD policies	security administra-tor
application design	- session-to-role assignment - specification of application's constraints	application designer
application operation	- session-to-user assignment - enforcement of application's constraints	application user

4. Conclusion and Future Work

In this paper, we present a basic but important framework for session-oriented integrity enforcement application design and operation in RBAC systems. The refined and dynamic features needed to architect the model for integrity-preserving application design and operation are described. We conclude session is a dynamic component of application design and integrity enforcement process and will play an important role in automated information systems such as distributed and workflow management systems. On the other hand further work is necessary on the description method for session-oriented application specification, constraints on sessions, and exception handling.

References

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

[2] John Barkley, "Comparing Simple Role Based Access Control Models and Access Control Lists," August 1997.

[3] Silvana Castano et al., *Database Security*, Addison-Wesley, 1994, pp18-34.

[4] D. D. Clark, D. R. Wilson, "A Comparison of a Model for Computer Integrity," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987, pp.184-194.

[5] U.S. Department of Defense, *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, National Computer Security Center, December 1985.

[6] David Ferraiolo, Richard Kuhn, "Role-Based Access Controls," *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, October 1992, pp.554-563.

[7] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn, "Role-Based Access Control(RBAC): Features and Motivations," *Proceedings of the 11th Annual Computer Security Applications Conferences*, December 1995, pp. 241-248.

[8] Warwick Ford, *Computer Communications Security*, Prentice Hall, 1994

[9] Virgil D. Gligor, Serban I. Gavrila, David Ferraiolo, "On the Formal Definition of Separation-of-Duty Policies and their Composition," *Proceedings of the IEEE Symposium on Security and Privacy*, May 1998, pp. 172-183.

[10] K. R. Poland, M. J. Nash, "Some Conundrums Concerning Separation of Duty," *IEEE Symposium on Computer Security and Privacy*, 1990.

[11] Ravi S. Sandhu, Edward J. Coyne, "Role-Based Access Control Models," *IEEE Computer*, February 1996, pp.38-47.

[12] Ravi S. Sandhu, Pierangela Samarati, "Access Control: Principle and Practice," *IEEE Computer*, September 1994, pp.40-48.

[13] Ravi S. Sandhu, "Role Hierarchies and Constraints for Lattice-Based Access Controls," *Proceedings of the 4th European Symposium on Research in Computer Security*, September 1996.

[14] Ravi S. Sandhu, "Access Control: The Neglected Frontier," *Proceedings of the 1st Australian Conference on Information Security and Privacy*, June 1996.

[15] J. M. Spivey, *Understanding Z*, Cambridge University Press, 1988.