7659, England, UK
Journal of Information and Computing Science
Vol. 2, No. 3, 2007, *pp*. 235-240

Research on RBAC-based Separation of Duty Constraints

Guangqian Kong [+] , Jianshi Li

College of Computer Science and Technology, Guizhou University, Guiyang 550025, China

(*Received 11 April 2007, accepted 12 May 2007*)

Abstract. Separation of duty (SOD) is an important characteristic in the role-based access control (RBAC) system. In view of some issues such as various variations of SOD constraints (SODs), ambiguous relations among constraint states, this paper formally defines several typical SODs and analyzes the transition relations among different SODs states. In combination with a delegation case, it goes an exploration and discussion on the SODs state transition issues, and proposes some corresponding solutions.

Keywords: Role-based access control, Separation of duty, Delegation

1. Introduction

In recent years, as a security mechanism, role-based access control (RBAC) is increasingly receiving widespread concerns. General access control policies, such as discretionary access control (DAC) and mandatory access control (MAC) both directly associate the subjects with the operation permissions while assigning privileges. This authorization mode is easy to use, but when there is a dramatic increase in the number of users and permissions, or there are frequent changes in the user's access permissions, authorization management will become extremely difficult. By introducing an intermediary – role, role-based access control policy divides the whole access control process into two parts, that is, first the permissions are allocated to the roles, then the roles are assigned to the users, so achieving the logical separation of permissions and users. This role-based access control method is largely using the permissions and responsibilities distribution methods in the real world's organizational structure for reference, so it can use a corresponding manner of the organizational structure to achieve system security strategy. It is not only conducive to the realization of the principle of minimal privileges, but also can greatly simplify the authorization management process.

In the actual large-scale commercial applications, thousands of users, roles and permissions make the applications more and more complicated. In order to ensure data accuracy and security, prevent fraud and collusion, RBAC96 model [1] introduces separation of duty constraints to restrict the assignment and use of users, roles and permissions. For example, in a financial management system, if a user was assigned to two roles, buyer and cashier, this could cause fraud. Separation of duty is to ensure that there will be no similar fraud and collusion.

With the increasing complexity in the application of RBAC, various SOD variations have emerged. However, there is less literature to systematically analyze and compare these SOD variations, especially the research results about formal description of various SODs and their state transitions are still limited. In view of the above problems, this paper systematically classifies SOD variations and formally describes seven typical SODs, and on this basis discusses the issues about SODs' state transition. Through a roles delegation case, it analyzes the influence of user's delegation actions on system constraint states, and proposes some corresponding solutions.

2. Brief introduction to RBAC96 model

RBAC96 model is the most well-known role-based access control model, and is also the basis for various role-based access control extension models. In fact, RBAC96 is a model family, which includes four submodels: RBAC0~RBAC3. RBAC0 is a basic RBAC model, which describes some basic elements of RBAC model; RBAC1 adds the role hierarchy based on the basic model; RBAC2 and RBAC3 introduce the SODs. RBAC96 model is shown in Fig. 1, and the model is defined as follows:

[+] Corresponding author. *E-mail address*: gq_kong@hotmail.com

- U, R, P, S which are respectively the sets of users, roles, permissions and sessions.
- PA⊆P×R, which is a many-to-many permission assignment relation assigning permissions to roles.
- UA⊆U×R, which is a many-to-many user assignment relation assigning user to roles.
- user: S→U, which is a function that maps a session to a single user.
- roles: S→2^R, which is a function that maps a session to a set of roles, roles(s_i) ⊆{r| (user(s_i),r)∈UA}.
- RH⊆R×R, is a partial order on R called role hierarchy.

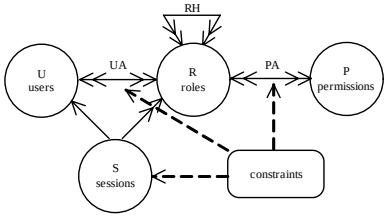

Fig. 1: RBAC96 model.

3. SOD constraints and their relations' description

SOD, whose objective is to reduce the risk of fraud and errors, divides security-sensitive tasks into several parts and distributes the permissions necessary for completing the tasks to many people to achieve the objective of separation of duty. In the role-based system, the separation of duty is achieved by controlling the distribution of members, the assignment of permissions and the activation of roles [2].

3.1. Formal description of separation of duty

To facilitate interpretation, some elements are pre-defined as follows: U: users set; CR: conflicting roles set; ObjSet: objects set; OpSet: operations set; States: system states set; role_members (r): a function that maps a role to its assigned members; current_role_set (s, u): a function that defines a set of roles that are enabled or active for the user u in state s; op(s_i,obj,u, s_j): a function that defines a set of operations for user u on the object obj from state s_i to state s_j; roles: a function that maps a user to its authorization roles set; auth: a function that maps a set of roles to the set of operations in its operation objects; |OpSet|: the number of members in a operations set. Assuming any users only activate one session at any time.

- Static separation of duty (SSOD)

Static separation of duty requires that any two conflicting roles can not be assigned to a given user.

$$\forall r_1, r_2 \in CR, r_1 \neq r_2 \Rightarrow role\_members(r_1) \cap role\_members(r_2) = \phi$$

- Dynamic separation of duty (DSOD)

Conflicting roles can share members, but users can not activate conflicting roles at one time.

$$\forall s \in States, \forall r_1, r_2 \in CR, \forall u \in U,$$

$$r_1 \neq r_2 \wedge r_1 \in current\_role\_set(s, u) \Rightarrow r_2 \notin current\_role\_set(s, u)$$

- Object-based separation of duty (ObjSOD)

Conflicting roles can share members, and users can also activate conflicting roles at one time, but the same object can not be operated more than two times. In [3], the author saw ObjSOD as a variation of dynamic separation of duty. If considering whether roles can operate the same object more than twice, it can also separate an object-based static separation of duty (ObjSSOD). Object-based dynamic separation of duty (ObjDSOD) is described as follows:

$$\forall s_0, s_1 \in States, \forall r \in CR, \forall obj \in ObjSet, \forall u \in U$$

$$current\_role\_set(s_0, u) \cap CR \neq \phi \Rightarrow |op(s_0, obj, u, s_1)| \leq 1$$

The formal description of object-based static separation of duty (ObjSSOD) is:

$$\forall s \in States, \forall u \in U, \forall op_1, op_2 \in OpSet, \forall obj \in ObjSet,$$

$$op_1 \neq op_2 \wedge roles(u) \cap CR \neq \phi \Rightarrow \{op_1, op_2\} \not\subset auth(s, roles(u), obj)$$

- Operation-based separation of duty (OpSOD)

Similar to ObjSOD, it is also divided into two types: static and dynamic.

Operation-based static separation of duty (OpSSOD): the subsets of conflicting roles can have the

common members, but they can not be authorized to complete all the operations in the set of operations, and they have nothing to do with the objects.

$$\forall s \in States, \forall u \in U, \forall obj \in ObjSet,$$

$$|OpSet| \geq 2 \wedge roles(u) \cap CR \neq \phi \Rightarrow OpSet \not\subset auth(s, roles(u), obj)$$

Operation-based dynamic separation of duty (OpDSOD): the set of user's active roles can not implement all the operations necessary for completing tasks, and it has nothing to do with the objects.

- History-based dynamic separation of duty (HDSOD)

Conflicting roles can share members, and the sets of roles can complete all the operations necessary in the whole task, but all the operations can not be completed in the same object.

$$\forall s_0, s_1 \in States, \forall op \in OpSet, \forall obj \in ObjSet, \forall u \in U$$

$$|OpSet| \geq 2 \wedge current\_role\_set(s_1, u) \cap CR \neq \phi \Rightarrow OpSet \not\subset op(s_0, obj, u, s_1) \cup \{op\}$$

3.2. Analysis of SOD constraint relations

To express the transition relations among various SODs states more directly, we convert the above formal description into state transition graph, shown in Fig. 2. In accordance with seven categories of SODs, we define seven SODs states. At any time, system is uniquely and definitely in one of the seven states.

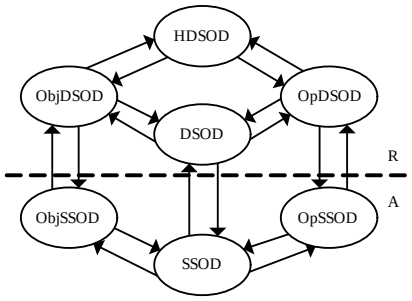

Fig. 2: State transition graph of SODs.

In Fig. 2, the states in running (R) are above the dotted lines, and the states in authorization (A) are under the dotted lines. The system's initial state is in the static separation of duty (SSOD), and when the administrators take the authorization operations, there are four possible states: (1) When the conflicting roles are assigned to the same user, the state of system will change to the dynamic separation of duty (DSOD); (2) When the conflicting roles are assigned to the same user and have not the authorization to implement operations for each object more than twice, the state of system will change to object-based static separation of duty (ObjSSOD); (3) When the conflicting roles are allocated to the same user and have not the authorization to complete all the operations in the set of operations, the state of system will change to operation-based static separation of duty (OpSSOD); (4) In other circumstances, the state of system will not change. In contrast, when the system is in the state of DSOD, ObjSSOD or OpSSOD, if the front authorization is withdrawn, the state of system will return to the SSOD state. After this, when a user starts a session, the system will be in a certain running state. For example, when a user has conflicting roles but not activate them, the system is in the state of dynamic separation of duty (DSOD); if the user activates the conflicting roles and only can operate the same object one time, the system will enter the ObjDSOD state. It can be seen from Fig. 2 that the system state which has the strongest constraint power is in the bottom of the state graph: SSOD state in authorization and DSOD state in running. More upward the arrow is, weaker the constraint power is. For implementing SOD policies in RBAC system, weaker the SOD constraint power is, it would be relatively easier and more flexible to implement; on the contrary, stronger the SOD constraint power is, it would be more difficult to achieve. When implementing system security strategy, it usually can be achieved through the combination of a number of SODs to increase the flexibility of the implementation. For example, HDSOD can be replaced by the combination of ObjDSOD and OpDSOD.

4. SODs state transition under roles delegation context

Delegation is a method for authorizing an active entity in system to another active entity, and the latter can implement the corresponding tasks on behalf of the former [4]. In RBAC96 system, user-role assignment is taken charge by security administrator. For some large, distributed applications, this single authorization

method greatly increases the administrator's work, and makes it very difficult to achieve security management. In such application environments, roles delegation mechanism becomes a necessary method for the decentralized management of missions. Based on the continuity of time, roles delegation can be divided into two types: permanent roles delegation and temporary roles delegation [5]. Permanent roles delegation is that a grantee completely replaces the grantor and the grantor can not cancel this delegation; on the other hand, temporary roles delegation has a time limit, and this kind of delegation will be automatically void if time expired. The below is a roles delegation case based on RBAC, we analyze the influence of user's roles delegation activities on the system constraint states, and propose some corresponding solutions.

4.1. Roles delegation case

The cheque signature system [7] as shown in Fig. 3 is a typical RBAC-based application. The system is divided into three parts: users, roles and permissions. Roles of Managers, accountants and clerks have the permissions of signing cheques, preparing cheques and dispatching cheques respectively, and the three roles are defined as pairwise mutually exclusive. Users of Alice, Bob and Carol are assigned to supervisor, accountant and clerk respectively, and the system's initial state is in the state of static separation of duty (SSOD).

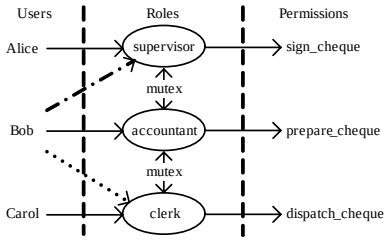

Fig. 3: Cheque signature system.

Now consider the situation after adding roles delegation mechanism based on the original system. Assuming that, DU: a set of delegation users; DR: a set of delegation roles; delegation (u_i, r, u_j, c): a function that user u_i delegates role r to user u_j, c={P, T}, P means permanent roles delegation, T means temporary roles delegation.

- $Carol \in DU \wedge clerk \in DR \wedge delegation(Carol, clerk, Bob, P) \Rightarrow clerk \in roles(Bob)$

Through roles delegation, Carol permanently delegates her authorized role of "clerk" to Bob, then Bob will have two mutually exclusive roles of accountant and clerk. The system state will transfer from static separation of duty (SSOD) to dynamic separation of duty (DSOD), and this state change is permanent until the manager reassigns the roles.

- $Alice \in DU \wedge supervisor \in DR \wedge delegation(Alice, supervisor, Bob, T) \Rightarrow supervisor \in roles(Bob)$

Alice temporarily roles delegates her role of "supervisor" to Bob, then Bob will have three mutually exclusive roles of supervisor, accountant and clerk. These three roles will be able to complete the entire cheque task, so the system state changes to operation-based dynamic separation of duty (OpDSOD). After this, when Bob activates the three exclusive roles at one time, but doesn't complete the entire cheque task for the same cheque, then the system state will change to history-based dynamic separation of duty (HDSOD). Since this roles delegation is temporary, once Alice cancels the delegation, the system will revert to the original state.

Through this case we can see that roles delegation mechanism has enhanced the flexibility of the system management and reduced the manager's work, but also increased the security risk of the system.

4.2. Solutions

In view of the influence of the roles delegation mechanism on the system's running state, in the process of implementing traditional delegation method, the corresponding strategies and mechanisms need to be added to ensure system security. Common methods to achieve delegation is to set up a delegation server [4]. A grantor sends a request to the server, then the server judges it according to delegation information, accepts or refuses the delegation request. The communications among grantor, grantee and server are in accordance with delegation protocol. As previously mentioned, the delegation activities will have a great impact on the system's running states, so in the deployment of delegation servers, in addition to the corresponding delegation rules (such as the set of delegation roles, the set of permissions and the set of users etc.), some

additional guarantee measures must be also added to effectively meet the system's security needs.

1) Introduce into SOD rules

After receiving a delegation request, except for judging it according to delegation rules, the delegation server should also take SOD rules into account. Embedding the SOD rules into the server can effectively assist the delegation server to make a correct judgment.

2) Record delegation history information

It is useful to save delegation history information into delegation server. For each receiving delegation request, all the delegation request information (grantor, delegated role, grantee, delegation time) should be stored in the delegation record table. When the latter requests reach, it can make an appropriate decision in accordance with both the delegation records and the SOD rules.

3) Add delegation audit mechanism

In the implementation of system security policies, the sufficient condition to guarantee the security of a system is that the roles are completely mutually exclusive [6], and all other SODs are not sufficient for the system security. So adding a corresponding audit mechanism is very necessary for the reduction of system errors. As shown in Fig. 4, after adding an audit server into the delegation implementation framework, the delegation server periodically sends the delegation records to the audit server, then it will automatically generate audit reports according to audit rules, and submit to the relevant personnel for analysis and decision-making.

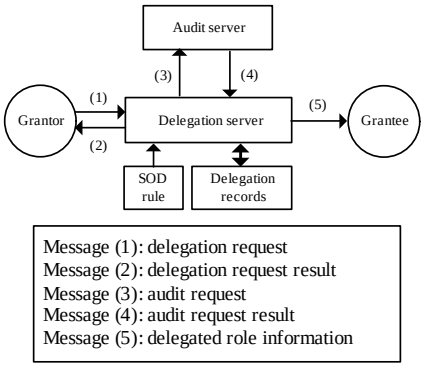

Fig. 4: Implementation framework of roles delegation.

5. Conclusion

Separation of duty is a well-known security rule, and it is also one of the most desired features in the application of RBAC. Along with that various security mechanisms become more and more complicated, there are more and more SOD variations. However, the research on the systematic analysis and description of these SOD variations is still less. This paper has summarized and categorized SODs on the basis of RBAC96 model. It gives the formal description of seven typical SODs, and analyzes their state transitions. In combination of a roles delegation case, it has an exploration and discussion on SODs state transition issues under the delegation context, and proposes some corresponding solutions, which are proved to be effective and feasible in a practical application of Chinese traditional medicine marketing management system.

6. References

[1] R. Sandhu, E. Coyne, H. Feinstein, et al. Role-Based Access Control Models. *IEEE Computer*. 199629(2): 38-47.

[2] D. Ferraiolo, J. Cugini, D. R. Kuhn. Role-Based Access Control (RBAC): Features and Motivations. *Proc. 1995 Computer Security Applications Conference*. 1995, 241-248.

[3] R. Simon, and M. E. Zurko. Separation of Duty in Role-Based Environments. *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, Rockport, MA, 1997, 183-194.

[4] SangYeob Na, SuhHyun Cheon. Role Delegation in Role-Based Access Control. *ACM*, 2000, 39-43.

[5] E. Barka. Framework for Role-Based Delegation Models. *Proc of 16th Annual Computer Security Application Conference*. 2000, 168-176.

[6] D. R. Kuhn. Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access. *Control Systems*. 1997, 23-30.

[7] A. Schaad. Detecting Conflicts in a Role-based Delegation Model. *Proceedings of the 17th Annual Computer Security Applications Conference*. 2001, 117-126.