Enhanced Role-Based Access Control Administration Tool

Burin Yenmunkong, and Chanboon Sathitwiriyawong

Faculty of Information Technology, and
Research Center for Communications and Information Technology (ReCCIT),
King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
Email: bom_burin@hotmail.com, chanboon@it.kmitl.ac.th

Abstract: This paper propose an extended model for role-permission assignment based on locations called "Enhanced Role-Based Access Control (ERBAC03)". The proposed model is built upon the well-known RBAC model. Assigning permissions to role is considered too complex activity to accomplish directly. Instead we advocate breaking down this process into a number of steps. The concept of jobs and tasks is specifically introduced to facilitate role-permission assignment into a series of smaller steps. This model is suitable for any large organization that has many branches. Each branch consists of many users who work in difference roles. An administration tool has been developed to assist administrators with the administration of separation of duty requirements. It demonstrates how the specification of static requirements can be done based on "conflicting entities" paradigm. Static separation of duty requirements must be enforced in the administration environment. Finally, we illustrate how the ERBAC03 prototype is used to administer the separation of duty requirements.

Keywords: Role-Based Access Control, Access Control Administration, Distributed System, Database Security, Information Security

1. INTRODUCTION

With organization's increased awareness to protect the confidentiality and integrity of applications and their data, system administrators are continuing to implement access control mechanisms. Historically, user access has been granted by adding the necessary permissions to each individual application. Administering access to many users for several different applications quickly becomes tedious and error prone, this is particularly true when the user changes positions and requires a different set of accesses.

The well-known RBAC model's permission assignment provides the efficiency of allowing the administrator to assign users to roles rather than directly assigns a role to permissions. In large enterprises, a number of roles are created for various job functions and able to double to hundreds or thousands. The management of these roles, permissions, users, and their interrelationships, is a formidable task. It is often centralized in a small team of security administrator. Each enterprise has many branches, so we must improve efficiency of RBAC model for enterprises in the real world. The RBAC model was viewed as a distributed model, not a centralized model.

An enhanced role-based access control (ERBAC03) is developed to support any organization that has many branches. Each branch consists of many users who work in difference roles. This model prevents the security violation of users who receive the roles at difference locations. The management of a centralized system can be viewed as a distributed system that is appropriate for a small team of security administrator in large organizations. The static separation of duty concept can be applied to the ERBAC03 model to prevent fraud of users.

Even with the introduction of the location, job and task abstraction, the administration of separation of duty requirements remains a huge task. In a large organization, there may be thousands of objects that require protection. The organization may have thousands of users, filling hundreds of different positions in the organization.

The ERBAC03 prototype is introduced to assist security administrators with the specification of access control requirements according to RBAC and ERBAC03 principles. More specifically, the ERBAC03 prototype is intended to assist the administration of separation of duty requirements. The "conflicting entities" administration paradigm as used within the ERBAC03 prototype is demonstrated.

Epstein [2] constructed a new model that extends the permission assignment of RBAC model between the roles and permissions. His goal was to define a layered model that served as a basis for detailing an effective methodology to assign permissions to roles.

Mavridis [3] defined access control mechanisms for privacy protection in distributed medical database that is defined on the basis of RBAC components and supports both mandatory and discretionary features.

Botha [4] and Perelson [5] proposed a mathematical model based on the concept of "conflicting entities" to express static separation of duty requirements. ERBAC03 model [6] also applies the concept of conflicting entities for higher efficiency of access control technologies.

Designing and developing for ERBAC03 administrator tool can be accomplished by the previous research [6][7]. First, a brief review of role-based access control principles is provided. Thereafter, the additional concept of location, job, and task is introduced. This is followed by a discussion on the use of the "conflicting entities" paradigm to specify separation of duty requirements. Finally, we illustrate how the ERBAC03 prototype is used to administer separation of duty requirements.

2. BASIC CONCEPTS

This section provides the necessary background to explain the principle of separation of duty within both RBAC and ERBAC03 models.

2.1 Role-based access control model (RBAC)

The RBAC model consists of the following components:

U - a set of users
R - a set of roles
P - a set of permissions
UA - user-role assignment relation, $UA \subseteq U \times R$
PA - permission-role assignment relation, $PA \subseteq P \times R$
RH - role hierarchy, $RH \subseteq R \times R$

For the purpose of this paper we slightly modify the notion of sets U, R and P to introduce uniform naming. These sets will define all possibly existing users, roles, and permissions.

Access control policy defines permissions granted to a particular user. So, only UA and PA relations determine access control policy. Users, roles, and permissions that are not used in UA or PA have no impact on the access control policy. The only changes of RBAC components that affect the access control policy are the insertion and removal of elements to and from UA and PA.

2.2 Enhanced role-based access control model (ERBAC03)

In the previous research [6], we extended the RBAC model by defining three additional entities as shown in Fig. 2. The concept of the new model was influenced by the ideas in [2].

The ERBAC03 model requires the modification of the end-points (i.e., roles and permissions) by introducing three additional entities which consist of locations, jobs, and tasks, as can be seen in Fig. 1.

We consider that a location consists of many roles that reflect the model management in the distributed aspect. A role may perform more than one type of work and is responsible for all activities that are required to perform the work. We define each type of work as a job. The jobs do not need to be in any sequence and we show that the tasks requiring access to application will be mapped to the permissions granted the desired access.

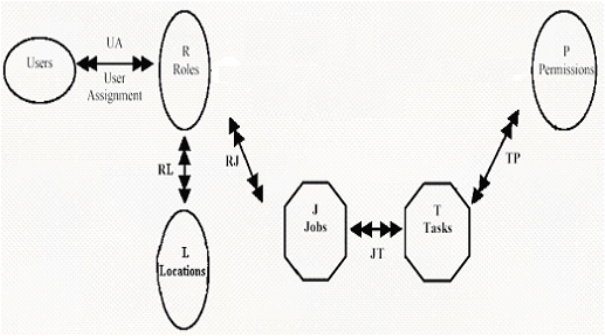

Fig. 1 ERBAC03 model

Locations may be related through a partial order. A location inherits roles assigned to the locations that are junior to it in the partial order. For example, the “Bangkok” location may be considered senior to the “Bangkapi” location. The “Bangkok” location will, therefore, inherit the roles assigned to the “Bangkapi” location. Fig. 2 shows how the ERBAC03 prototype manages the associations between locations. In the ERBAC03 model, locations are related to other locations within disjoint, named location networks.

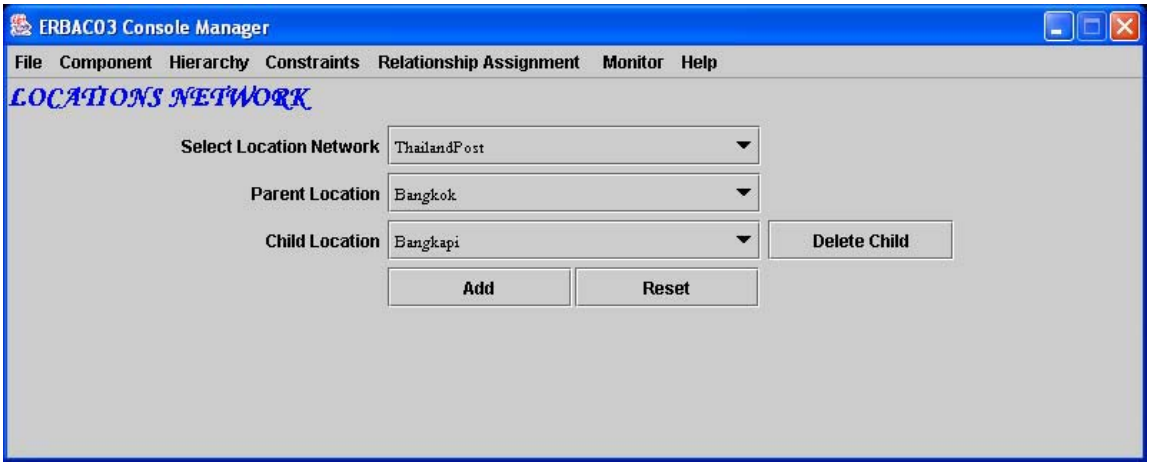

Fig. 2 Location hierarchy management

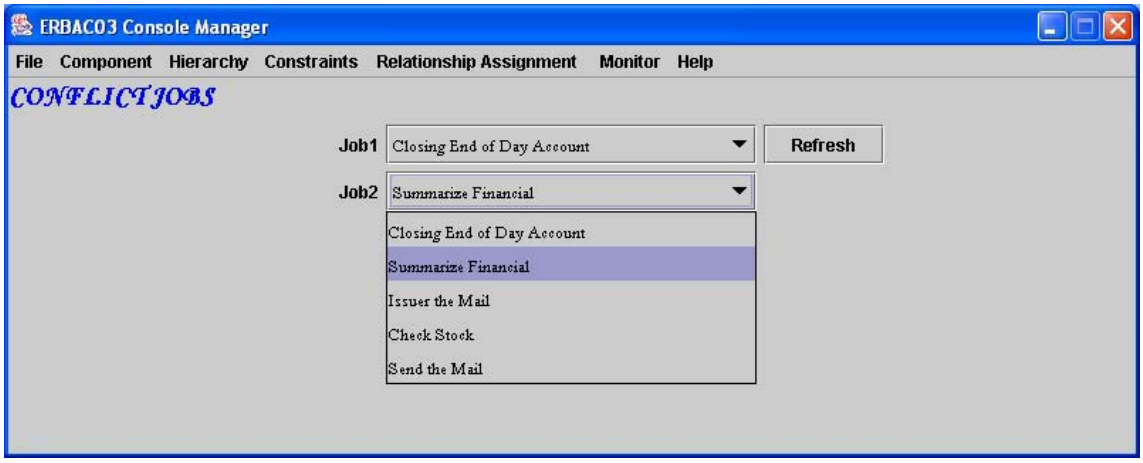

Fig. 3 Defining conflicting jobs

Consider a short formal summary of the relevant components in the extension of RBAC model, based on the work of Ahn & Sandhu [8].

Definition 2.2.1: For ERBAC03 entities
U = a set of users $\{u_1, u_2, \ldots, u_n\}$
R = a set of roles $\{r_1, r_2, \ldots, r_n\}$
P = a set of permissions $\{p_1, p_2, \ldots, p_n\}$
L = a set of locations $\{l_1, l_2, \ldots, l_n\}$
J = a set of jobs $\{j_1, j_2, \ldots, j_n\}$
T = a set of tasks $\{t_1, t_2, \ldots, t_n\}$

Definition 2.2.2: For ERBAC03 associations
$UA \subseteq U \times R$, a many-to-many user-to-role assignment relation.
$RH \subseteq R \times R$, a partial order on R called the role hierarchy.
$RL \subseteq L \times R$, a many-to-many location-to-role assignment relation.
$RJ \subseteq J \times R$, a many-to-many role-to-job assignment relation.
$JT \subseteq T \times J$, a many-to-many job-to-task assignment relation.
$TP \subseteq P \times T$, a many-to-many task-to-permission assignment relation.
$LH \subseteq L \times L$, a partial order on L called the location hierarchy.

Definition 2.2.3: For roles function
$roles(u_i) = \{r \in R \mid (u_i, r) \in UA\}$
$roles(u_i)^* = \{r \in R \mid (\exists r \leq r')[(u_i, r') \in UA]\}$
$roles: U \cup L \cup J \cup P \rightarrow 2^R$, a function mapping the sets U and P to a set of roles.
$roles^*: U \cup L \cup J \cup P \rightarrow 2^R$, extends roles in the presence of a role hierarchy.
$roles: R \rightarrow 2^L$, a function mapping the sets R to a set of locations.
$roles(l_i) = \{r \in R \mid (l_i, r) \in RL\}$
$roles(j_i) = \{r \in R \mid (j_i, r) \in RJ\}$
$roles(l_i)^* = \{r \in R \mid (\exists r \leq r')[(l_i, r') \in RL]\}$
$roles(j_i)^* = \{r \in R \mid (\exists r \leq r')[(j_i, r') \in RJ]\}$

Definition 2.2.4: For locations function
$location(r_i) = \{l \in L \mid (l, r_i) \in RL\}$
$location(u_i) = \{l \in L \mid (\exists r \in roles(u_i))\ [(l, r) \in RL]\}$
$location(j_i) = \{l \in L \mid (\exists r \in roles(j_i))\ [(l, r) \in RL]\}$
$location(p_i) = \{l \in L \mid (\exists r \in roles(jobs(tasks((p_i))))$
$[(l, r) \in RL]\}$
$location(r_i)^* = \{l \in L \mid (\exists r' \leq r)[(l, r') \in RL]\}$
$location(u_i)^* = \{l \in L \mid (\exists r \in roles^*(u_i))$
$[(l, r) \in RL]\}$
$location(j_i)^* = \{l \in L \mid (\exists r \in roles^*(j_i))$
$[(l, r) \in RL]\}$
$location(p_i)^* = \{l \in L \mid (\exists r \in roles^*(jobs^*(tasks^*((p_i))))$
$[(l, r) \in RL]\}$
$locations: U \cup R \cup J \rightarrow 2^L$, a function mapping users, roles and permissions to a set of locations.
$locations^*: U \cup R \cup J \rightarrow 2^L$, extends locations in the presence of a location hierarchy.

Definition 2.2.5: For jobs function
$jobs: J \rightarrow 2^T$, a function mapping the sets J to a set of tasks.
$jobs^*: J \rightarrow 2^T$, extends jobs in the presence of a role hierarchy or location hierarchy.
$jobs(t_i) = \{j \in J \mid (t_i, j) \in JT\}$
$jobs(t_i)^* = \{j \in J \mid (\exists j \leq j')[(t_i, j') \in JT]\}$

Definition 2.2.6: For tasks function
$tasks: T \rightarrow 2^P$, a function mapping the set T to a set of permissions.
$tasks^*: T \rightarrow 2^P$, extends tasks in the presence of a role hierarchy or location hierarchy.
$tasks(p_i) = \{t \in T \mid (p_i, t) \in TP\}$
$tasks(p_i)^* = \{t \in T \mid (\exists t \leq t')[(p_i, t') \in TP]\}$

Definition 2.2.7: For permissions function
$perm(u_i) = \{p \in P \mid (\exists r \in tasks(jobs(roles(u_i))))\ [(p, t) \in TP]\}$
$perm(u_i)^* = \{p \in P \mid (\exists r \in tasks^*(jobs^*(roles^*(u_i))))$
$[(p, t) \in TP]\}$
$perm(l_i) = \{p \in P \mid (\exists r \in tasks(jobs(roles(l_i))))\ [(p, t) \in TP]\}$
$perm(j_i) = \{p \in P \mid (\exists t \in tasks(j_i))\ [(p, t) \in TP]\}$
$perm(l_i)^* = \{p \in P \mid (\exists r \in tasks(jobs(roles^*(l_i))))$
$[(p, t) \in TP]\}$
$perm(j_i)^* = \{p \in P \mid (\exists t \in tasks^*(j_i))\ [(p, t) \in TP]\}$
$perm: U \cup R \cup L \cup J \cup T \rightarrow 2^P$, a function mapping users, roles, locations, jobs and tasks to a set of permissions.
$perm^*: U \cup R \cup L \cup J \cup T \rightarrow 2^P$, extends permissions in the presence of a role hierarchy or location hierarchy.

Static separation of duty (SSoD) constraint is used in ERBAC03 model [7]. It concerned with the prevention of fraud to ensure that a single user does not have too much authority. The authority is set as a permission based on locations, therefore the essence of our paradigm depends on the conflicting permissions and locations.

Conflicting users are users who are likely to conspire.
Conflicting roles are roles that together have the ability to conspire, i.e. they are assigned some (but not all) conflicting permissions
Conflicting locations are locations requiring conflicting roles to complete.
Conflicting jobs are jobs that together have the ability to fraud.
Conflicting tasks are tasks requiring conflicting permissions to complete.
Conflicting permissions are permissions that can result in unnecessary power if bestowed on the same person.

Fig. 3 shows how conflicting jobs are identified within the ERBA03 prototype. The other conflicts are specified in a similar manner. The next section discusses how this is implemented in the ERBAC03 prototype.

3. STATIC SEPARATION OF DUTY ENFORCEMENT IN ERBAC03

To enforce the static separation of duty, the ERBAC03 prototype can ensure that the integrity of the associations between entities is maintained. If an action cannot be performed, remedial actions are suggested. For example, if conflicting jobs are assigned to non-conflicting roles, the user is given the option of making the roles conflicting.

To illustrate how the ERBAC03 prototype maintains the associations, this section will review different static separation of duty implementations of the requirement: “A person who issues the money order may never approve an account”. Five approaches to enforcing this separation of duty requirement in a static fashion are proposed. This is done by rephrasing the separation of duty requirement in the following ways:

(1) An accountant and a cashier may not perform the same jobs.
(2) An accountant and a cashier may not be assigned to the same location.
(3) The "Issue Money Order" job and the "Approve an Account" job may not be assigned to the same user.
(4) The "Checking the Mail Address" task may not be performed by someone who performs the "Checking the Old Account" task.
(5) The "Read Account Record" permission and the "Read the Transaction Record" permission may not be assigned to the same task.

The separation of duty constraints will be implemented as conflicting locations, conflicting roles, conflicting jobs, conflicting tasks, and conflicting permissions. Conflicting users can be used in these combinations.

If two users are conflicting, the chance of colluding with each other is very high. In essence, they should, therefore, be treated as if they were one user. For example, if two jobs may not be performed by the same user, two conflicting users may not perform them either as the chance of a conspiracy is high. We shall now consider how each of the approaches can, in turn, be handled in the prototype.

3.1 Conflicting roles

First consider (1) - An accountant and a cashier may not perform the same jobs.

Since the accountant approves an account, and the cashier issues a money order, the "Accountant" role in the parent role network and the "Cashier" role in the child role network may be set to conflict (see Fig. 4). Due to the inheritance property of role networks, conflicting roles cannot exist in the same role network. If conflicting roles are allowed in one role network, the senior's most roles in that role network would inherit the permissions of both conflicting roles. This clearly defeats the purpose. The role may conflict with more than one role in another network. Conflicts are, however, inherited up the partial order and setting more than one conflict, as such, may not be necessary. The ERBAC03 prototype will remove any unnecessary conflict.

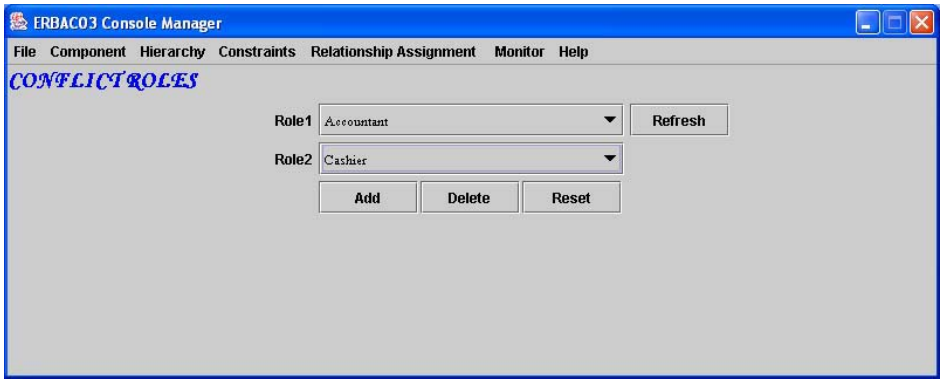

Fig. 4 Conflicting roles

3.2 Conflicting locations

Now consider (2) - An accountant and a cashier may not be assigned to the same location.

The "Accountant" role may not be assigned to somewhere where may have the "Cashier" role. If conflicting locations are assigned to roles, these roles are automatically made conflicting. This approach can be considered to be the reverse of that. Two roles are defined to be conflicting. Subsequently, the roles that must be assigned to the user are conflicting. Locations may be related through a partial order. A location inherits roles assigned to the locations that are junior to it in the partial order. For example, "Bangkna" location and "Bangkapi" location are conflicting. If the "Accountant" role is assigned to "Bangna" location, then the "Cashier" role may be assigned to "Bangkapi" location.

3.3 Conflicting jobs

Now consider (3) - The "Issue Money Order" job and the "Approve an Account" job may not be assigned to the same user.

Conflicting jobs may only be assigned to conflicting roles. If this is not enforced, conflicting jobs could be assigned to conflicting users. These conflicting users belong to non-conflicting roles, which have conflicting jobs that were incorrectly assigned to the non-conflicting roles. The ERBAC03 prototype, therefore, only allows conflicting roles to receive conflicting jobs.

If the roles are not conflicting, they are made conflicting, subject to additional integrity checking. Roles cannot be made conflicting if conflicting users are assigned to the said roles. It can, therefore, be seen that even if the "Accountant" and "Cashier" roles were not initially identified to be conflicting, they will be made conflicting when the two conflicting jobs are assigned to these two roles.

3.4 Conflicting tasks

Now consider (4) - The "Checking the Mail Address" task may not be performed by someone who performs the "Checking the Old Account" task.

Conflicting tasks may only be assigned to conflicting jobs. If conflicting tasks were assigned to jobs, these jobs were automatically made conflicting. This approach can be considered to be the reverse of that. Two tasks are defined to be conflicting. Subsequently, the jobs that must be assigned to the role must be conflicting. If two non-conflicting jobs are assigned, the jobs are made conflicting, subject to a series of integrity checks being performed. Fig. 5 depicts the "Issue Money Order" job as being assigned to the "Checking the Mail Address" task. If two tasks are initially not indicated to be conflicting, but they are assigned to conflicting jobs, the tasks are made conflicting tasks.

3.5 Conflicting permissions

Now consider (5) - The "Read Account Record" permission and the "Read the Transaction Record" permission may not be assigned to the same task.

Conflicting permissions may only be assigned to conflicting tasks. If this is not enforced, conflicting permissions could be assigned to conflicting users. These conflicting users belong to non-conflicting roles, which have conflicting permissions that were incorrectly assigned to the non-conflicting roles. This clearly opens the door for a conspiracy, The ERBAC03 prototype, therefore, only allows conflicting tasks to receive conflicting permissions. The results of the conflicting role, job and task approaches are thus identical. The conflicting permission approach can, however, be considered stricter. Conflicting permissions must be performed by conflicting tasks. However, conflicting tasks do not only have conflicting permissions.

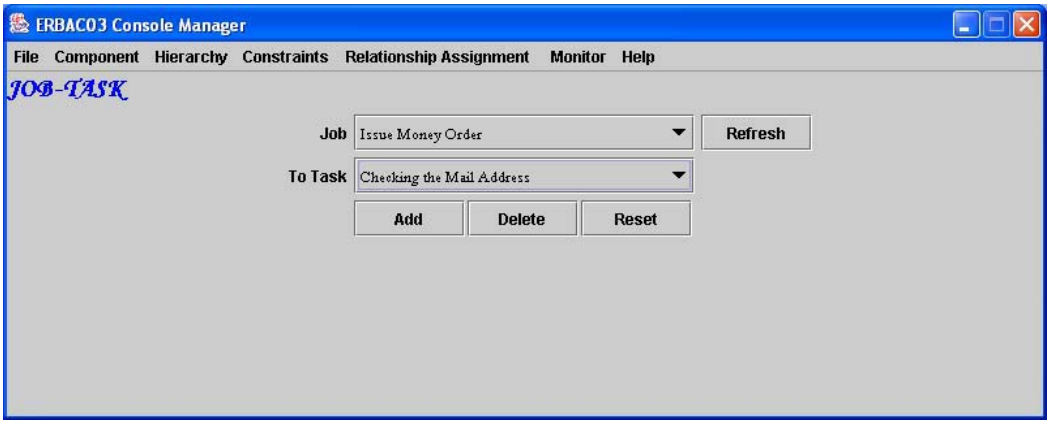

Fig. 5 Assignment of conflicting tasks to conflicting jobs

4. CONCLUSIONS

In this paper, we propose an enhanced model of RBAC for large enterprises using location hierarchy approach and call ERBAC03 model. The ERBAC03 model requires the modification of the end-points by introducing three additional entities which consist of locations, jobs, and tasks and are able to make the permissions to roles more clear, also to apply to large enterprises effectively. The definition of entity location is related to roles and reflects large enterprises which have various branches. Each branch consists of many similar roles that reflect the model management in the distributed aspect, not the centralized aspect.

The constraint features used in this model are referred as the static separation of duty, and can deal with the ERBAC03 components. In particular, it demonstrates how static separation of duty requirements specified through the use of conflicting permissions, users, roles, jobs, tasks, and locations.

An administration tool was developed to assist the administration of separation of duty requirements. We demonstrated the "conflicting entities" paradigm as a way of specifying separation of duty requirements. It shows that both static requirements can be formulated according to the "conflicting entities" paradigm in the ERBAC03 prototype. It was, furthermore, shown that the ERBAC03 prototype enforces static separation of duty requirements.

It results in the higher efficiency of data security and data integrity, prevention of fraud, the higher efficiency of work in enterprises and easier management for RBAC administrator.

REFERENCES

[1] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role Based Access Control Models," *IEEE Computer*, *Vol. 2*, pp. 38-47, 1996.

[2] P. Epstien and R. Sandhu, "Engineering of Role/Permission Assignments," *Proceedings of the 17th Annual Computer Security Applications Conference*, December 2001.

[3] I. Mavridis, G. Pangalos, M. Khair, and L. Bozios, "Defining Access Control Mechanisms for Privacy Protection in Distributed Medical Databases," *Proceedings of IFIP Working Conference on User Identification and Privacy Protection*, Stockholm, Sweden, June 1999.

[4] R. A. Botha and J. H. P. Eloff, "Separation of Duties for Access Control Enforcement in Workflow Environments," *IBM Systems Journal, Vol. 40, No. 3*, 2001.

[5] S. Perelson and R. A. Botha, "Conflict Analysis as a Means of Enforcing Static Separation of Duty Requirements in Workflow Environments," *South African Computer Journal*, *Vol. 26*, pp. 212-216, 2000.

[6] C. Sathitwiriyawong and B. Yenmunkong, "An Enhanced Role-Based Access Control for Large Enterprises," *Proceedings of the 3rd International Symposium on Communications and Information Technologies*, pp. 279-283, 2003.

[7] B. Yenmunkong and C. Sathitwiriyawong, "An Enhanced Role-Based Access Control Model using Constraint Features," *Proceedings of the 7th National Computer Science and Engineering Conference*, pp. 103-108, 2003.

[8] G. J. Ahn and R. Sandhu, "The RSL99 Language for Role-based Separation of Duty Constraints," *Proceedings of the 4th ACM Workshop on Role-based Access Control*, Fairfax, Virginia, pp. 43-53, 1999.