Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1782-1796
submitted: 15/8/08, accepted: 25/4/09, appeared: 1/5/09 © *J.UCS*

Authorization Algorithms for Permission-Role Assignments

Lili Sun, Hua Wang, Jianming Yong
(University of Southern Queensland, Queensland, Australia
{sun, wang, yong}@usq.edu.au)

Abstract: Permission-role assignments (PRA) is one important process in Role-based access control (RBAC) which has been proven to be a flexible and useful access model for information sharing in distributed collaborative environments. However, problems may arise during the procedures of PRA. Conflicting permissions may assign to one role, and as a result, the role with the permissions can derive unexpected access capabilities.
This paper aims to analyze the problems during the procedures of permission-role assignments in distributed collaborative environments and to develop authorization allocation algorithms to address the problems within permission-role assignments. The algorithms are extended to the case of PRA with the mobility of permission-role relationship. Finally, comparisons with other related work are discussed to demonstrate the effective work of the paper.

Keywords: access control, conflicts, authorization
Categories: H.1.1, H.2.8, I.1.2

1 Introduction

Information security currently has vast impact to information industry, but also to individuals and our daily life [Bella, 2008, Chen et al. 2008]. Access control, as a significant method to secure the information in distributed environments, has been studied by many researchers [Harrer et al. 2008, Wang et al. TKDE]. The National Institute of Standards and Technology developed the role-based access control (RBAC) prototype [Feinstein H. L. 1995] and published a formal model [Ferraiolo and Kuhn 1992]. RBAC has been widely used in database system management and distributed environments since it enables managing and enforcing security in large-scale and enterprise-wide systems. RBAC involves individual users being associated with roles as well as roles being associated with permissions (Each permission is a pair of objects and operations). As such, a role is used to associate users and permissions. A user in this model is a human being. A role is a job function or job title within the organization associated with authority and responsibility.

Permission is an approval of a particular operation to be performed on one or more objects. As shown in Figure 1, the relationships between users and roles, and between roles and permissions are many-to-many (i.e. a permission can be associated with one or more roles, and a role can be associated with one or more permissions).

Recently, role based access control (RBAC) has been widely used in database system management and operating system products since its management advantages [Wang et al. 2003ADC]. In 1993, the National Institute of Standards and Technology (NIST) developed prototype implementations, sponsored external research [Feinstein

H.L. 1995], and published formal RBAC models [Ferraiolo and Kuhn 1992]. Many organizations prefer to centrally control and maintain access rights, not so much at the system administrator's personal discretion but more in accordance with the organization's protection guidelines [David et al. 1993]. RBAC is being considered as part of the emerging SQL3 standard for database management systems, based on their implementation in Oracle 7 [Sandhu R. 1997]. Many RBAC practical applications have been implemented [Barkley et al. 1999, Ferraiolo et al. 1999, Sandhu R. 1998, Wang et al. ACSC04, Wang et al. TSMC].

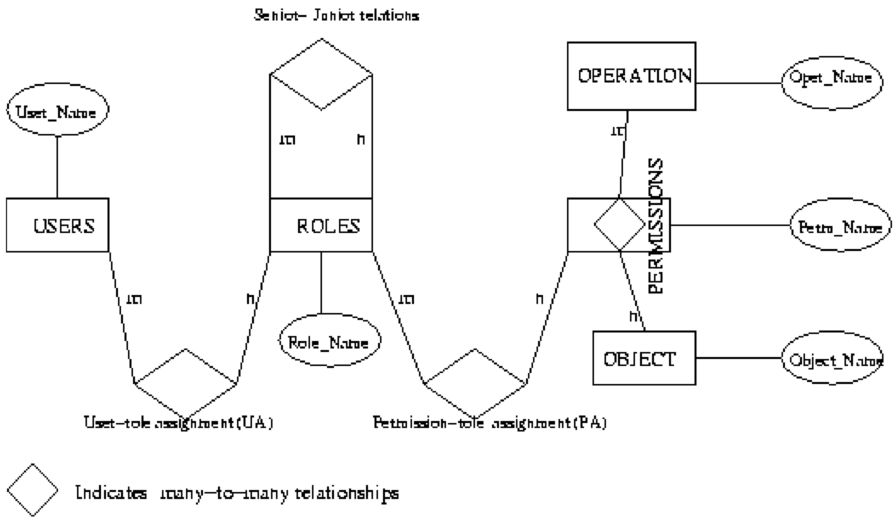

Figure 1: RBAC relationship

However, there is a consistency problem when using RBAC management. For instance, if there are hundreds of permissions and thousands of roles in a system, it is very difficult to maintain consistency because it may change the authorization level, or imply high-level confidential information to be derived when more than one permission is requested and granted.

The permissions assigned to a role by administrators may conflict. For example, the permission for approving a loan in a bank is conflicting with the permission of funding a loan. These two permissions cannot be assigned to a role; however, because of role hierarchies, a role may still have these permissions even if they have been revoked from the role. In the latter case, a user with this role is able to access objects in the permission and has operations on the objects. There are evident problems with the processes of assigning and revocation.

Authorization granting problem -- How to check whether a permission is in conflict with the permissions of a role?

Authorization revocation problem -- How to find whether permissions of a role have been revoked from the role or not?

For example, Figure 2 shows a system administrative role (BankSO) in a bank to manage regular roles such as AUDITOR, TELLER, ACCOUNT_REP and MANAGER. Role MANAGER inherits AUDITOR and TELLER. ACCOUNT_REP has a SSD relationship with AUDITOR as well as DSD relationship with TELLER [Wang et al. TKDE].

The administrative role BankSO can assign audit permission or cash operation permission to a role but not both, otherwise it compromises the security of a bank system. Our aim is to provide relational algebra algorithms to solve the problems and then automatically check conflicts when assigning and revoking.

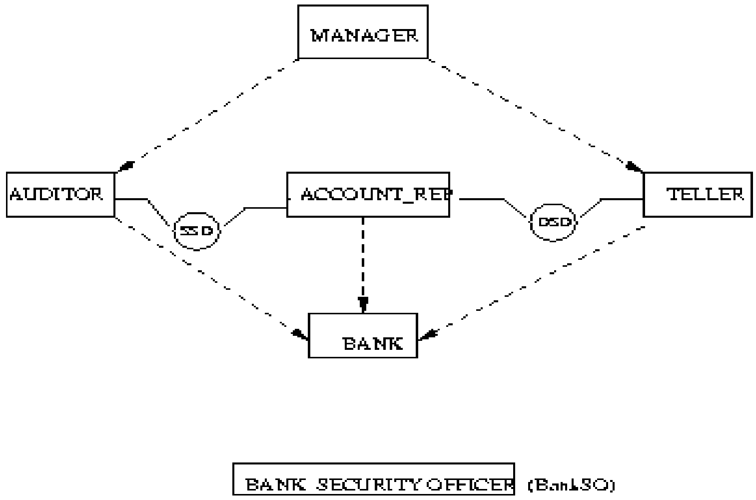

Figure 2: Administrative role and role Relationships in a bank

Based on the database and its tables such as ROLES, SEN-JUN in the paper [Wang et al. 2003ADC], this paper is going to develop formal approaches to check the conflicts and thereby help allocate the permissions without compromising the security. The formal approaches are based on relational structure and relational algebra operations. To my knowledge, this is the first attempt in this area to develop formal approaches for permission allocation and conflict detection.

RoleName	MANAC	AUDC	ACCOUNT_REPC	TELLERC
MANAGER	0	0	0	0
AUDITOR	-1	0	-1	-1
ACCOUNT_REP	-1	-1	0	-1
TELLER	-1	-1	-1	0

Table 1: The relation ROLES in Figure 2

The ROLES relation in Figure 2 is in Table 1. The attribute TELLERC shows whether the role TELLER is conflicting with the RoleName in the relation or not. For instance, in the third tuple, a user with role TELLER has conflicts with the role AUDITOR.

SEN-JUN - This is a relation of roles in a system. Senior is the senior of the two roles. Table 2 expresses the SEN-JUN relationship in Figure 2.

Senior	Junior
MANAGER	AUDITOR
MANAGER	TELLER
TELLER	BANK
AUDITOR	BANK

Table 2: SEN-JUN in Figure 2

PermName	Oper	Object	ConfPerm
Approval	approve	Cash/check	Funding
Funding	invest	cash	Approval
Audit	audit	record	Teller
Teller	transfer	cash	Audit

Table 3: An example of the relation PERM

The new tables like PERM and ROLE_PERM are needed. PERM - This is a relation of {PermName, Oper, Object, ConfPer}:

PermName is the primary key for the table, and is the name of the permission in the system.

Oper is the name of the operation granted. It has information about the object that the operation is granted on.

Object is the database item that can be accessed by the operation. It can be a database, a table, a view, an index or a database package.

ConfPer is a set of permissions that conflicts with the PermName in the relation. For example, a staff member in a bank cannot have both permissions of approval and funding as well as both permissions of audit and teller. The relation of PERM can be expressed as Table 3.

ROLE-PERM - is a relationship between the ROLES and the PERM, listing what permissions are granted to what roles. It has two attributes:

RoleName is a foreign key RoleName from the table ROLES.

PermName is a foreign key PermName from the table PERM which is assigned to the role.

Suppose the permission Approval is assigned to role TELLER and the permission Funding to role MANAGER, Table 4 expresses the permission-role relationship.

RoleName	PermName
MANAGER	Funding
TELLER	Approval

Table 4: ROLE-PERM

Admin.role	Role Range
BankSO	[Bank, MANAGER)

Table 6: Example of Can-revokep

Based on these relations, we describe the Authorization granting algorithm and revocation algorithms in this paper.

The paper is organized as follows. We recall the relational algebra-based authorization granting and revocation algorithms developed in our previous work. The extensions of the algorithms are described in section 3. Comparisons to related work are discussed in section 4 and the conclusions are in section 5.

2 Authorization granting and revocation algorithms for PRA

We recall granting and revocation algorithms for PRA based on relational algebra in this section. Details can be found from [Wang et al. 2003ADC].

The notion of a P*rerequisite conditionp, Can-assignp* and *Can-revokep* mentioned in the paper is a key part in the processes of permission_role assignment. The *Prerequisite conditionp* is used to test whether or not permission can be assigned to roles while the C*an-assignp* is used to verify what role range's permissions an administrator can assign.

For a given set of roles *R* let *CR* denote all possible prerequisite conditions that can be formed using the roles in *R*. Not every administrator can assign permission to a role. The relation of *Can-assignp* $\subseteq$ AR ×CR × 2^R provides what permissions can be assigned by administrators with prerequisite conditions, where *AR* is a set of administrative roles.

Permission-role assignment (PA) is authorized by *Can-assignp* relation. Table 5 shows the *Can-assignp* relations with the prerequisite conditions in the example.

Admin.role	Prereq.ConditionP	Role Range
BankSO	$BANK \wedge \overline{TELLER} \wedge \overline{AUDITOR}$	[ACCOUNT_REP, ACCOUNT_REP]
BankSO	$BANK \wedge \overline{TELLER} \wedge \overline{ACCOUNT\_REP}$	[AUDITOR, AUDITOR]
BankSO	$BANK \wedge \overline{AUDITOR} \wedge \overline{ACCOUNT\_REP}$	[TELLER,TELLER]

Table 5: Can-assignp relation in Figure 2

Supposing an administrator role *ADrole* wants to assign a permission p_j to a role *r* with a set of permissions *P*. P^* is an extension of *P*,
$P^* = \{p \mid p \in P\} \cup \{p \mid \text{exist } r' \in R, r'<r, (p, r') \in PA\}$. There are two major steps in the following permission granting algorithm. The first step is to check whether the *ADrole* can assign the permission p_j to *r* or not. The set of Prerequisite conditionp associated with *ADrole* can be obtained from the table *Can-assignp* while the set of roles associated with permission p_j is obtained from the table *ROLE-PERM*. The ADrole can build the membership of permission p_j and role *r* only if there is a role in the both sets. This means permission p_j satisfies prerequisite condition. The second step is to determine whether the permission p_j conflicts with the permissions of *r* or not, in other words, whether p_j conflicts with permission set P^* or not. The set of conflicting permissions of p_j can be retrieved from table PERM. Permission p_j is conflicting with role *r* if the intersection of the set and P^* is not empty.

There are related subtleties that arise in RBAC concerning the interaction between granting and revocation of permission-role membership. A relation *Can-revokep* $\subseteq$ AR × 2^R provides which permissions in what role range can be revoked. Table 6 gives an example of the *Can-revokep* relation. We have two revocation algorithms, one is a weak revocation algorithm that is for explicit member of a role only, the other one is a strong revocation algorithm that is used to delete explicit memberships between permissions and roles as well as implicit memberships.

The meaning of *Can-revokep(BankSO, [Bank, MANAGER))* in Table 6 is that a member of the administrative role BankSO can revoke the membership of a permission from any role in [Bank, MANAGER).

Two algorithms were developed for revocation of a permission p_j from a role *r* by an administrative role ADrole. In the weak revocation approach, only whether or not permission p_j is an explicit member of role *r* needs to be determined. Operation of the weak revocation has no effect when the permission p_j is not an explicit member of the role *r*. The role set associated with p_j is gained from the relation of *ROLE-PERM* while the role set of role range with ADrole is obtained from the relation *Can-revokep*. The permission p_j can be revoked if the intersection of these two role sets is not empty.

A role still owns a permission of a system, which has been weakly revoked, if the role is senior to another role associated with the permission. To solve the authorization revocation problem, we need strong revocation, which requires revocation of both explicit and implicit membership. Strong revocation of a permission's membership in role *r* requires that the permission be removed not only from explicit membership in *r*, but also from explicit and implicit membership in all roles junior to *r*. Strong revocation therefore has a cascading effect up-wards in the role hierarchy. The first step in the strong revocation algorithm is to test whether p_j is in P^* or not. If the test is negative, that means p_j is neither an explicit member nor an implicit member of the role *r*. When this case occurs, the strong revocation has no effect for the role. Otherwise, p_j is either an explicit member or an implicit member of *r*. In this step, the membership of p_j is revoked from *r* if $p_j \in P$; then the role set of roles that are junior to role *r* can be retrieved from the relation *SEN-JUN*. For all roles in both the set and *P*, the relationships between these roles and permission p_j are revoked.

3 Extensions of the algorithms with mobility of permissions

Similar to the mobility of user-role relationship, permissions can also be assigned to roles as mobile and immobile members [Wang et al. ACSC2005]. There are four kinds of permission-role membership for a given role *x* [Sandhu and Munnwer 1999, Wang et al. Special].

1: Explicit Mobile Member EMPx: EMPx = $\{p, (p, Mx) \in PA\}$
2: Explicit Immobile Member EIMPx: EIMPx = $\{p, (p, IMx) \in PA\}$
3: Implicit Mobile Member ImMPx: ImMPx = $\{p, \exists x'<x, (p, Mx') \in PA\}$
4: Implicit Immobile Member ImIMPx: ImIMPx = $\{p, \exists x'<x, (p, IMx') \in PA\}$

A *prerequisite conditionPM* is evaluated for a permission p by interpreting role *x* to be true if

$$p \in EMx \vee (p \in ImMx \wedge p \notin EIMx)$$

and $\bar{x}$ to be true if

$$p \notin EMx \wedge p \notin EIMx \wedge p \notin ImMx \wedge p \notin ImIMx$$

In other words x denotes mobile membership (explicit or implicit) and $\overline{x}$ denotes absence of any kind of membership.

For a given set of roles R let CR denote all possible prerequisite conditions with mobility of permission-role relationship that can be formed using the roles in R. Not every administrator can assign a role to a user. The following relations provide what permissions an administrator can assign as mobile members or immobile members with prerequisite conditions.

Can-assignp-M is a relation of $AR \times CR \times 2^R$, which is used for permission-role assignments with mobile members; where AR is a set of administrative roles. Permission-role assignments with immobile members are authorized by the relation *Can-assignp-IM* $\subseteq AR \times CR \times 2^R$.

Permission-role assignment (PA) is authorized by *Can-assignp-M* and *Can-assignp-IM* relations. Table 7 and Table 8 shows the *Can-assignp-M* and *Can-assignp-IM* relations with the prerequisite conditions in the bank example.

Admin.role	Prereq.ConditionPM	Role Range
BankSO	BANK	[BANK, BANK]
BankSO	$\text{BANK} \wedge \overline{\text{TELLER}}$	[AUDITOR, AUDITOR]
BankSO	$\text{BANK} \wedge \overline{\text{AUDITOR}}$	[TELLER, TELLER]

Table 7: Can-assignp-M in the example

Admin.role	Prereq.ConditionPM	Role Range
BankSO	BANK	[BANK, BANK]
BankSO	$\text{BANK} \wedge \overline{\text{TELLER}}$	[AUDITOR, AUDITOR]
BankSO	$\text{BANK} \wedge \overline{\text{AUDITOR}}$	[TELLER, TELLER]

Table 8: Can-assignp-IM in the example

The meaning of *Can-assignp-M(BankSO, BANK, TELLER, AUDITOR)* is that a member of the administrative role *BankSO* can assign a permission whose current membership satisfies the prerequisite condition *BANK* to be a mobile member of roles *TELLER* and *AUDITOR*..

Supposing an administrator role *ADrole* wants to assign a permission p_j to role r with a set of permissions P which has mobile and immobile memberships with r. The p_j has mobile or immobile membership with r if *ADrole* can assign without conflicts. The following algorithm applies to both of mobile and immobile members. P^* is an extension of P, $P^* = \{p \mid p \in P\} \cup \{p \mid \exists\, r' \in R, r' < r, (p, r') \in PA\}$.

Authorization granting algorithm

GrantMP(ADrole, P, p_j)

Input: ADrole, role r and a permission p_j.

Output: true if ADrole can assign the permission p_j to r with no conflicts; false otherwise.

Begin:
Step 1. /* Whether the ADrole can assign the permission p_j to r as mobile or immobile member or not} */

Suppose $S_{M1} = S_M \cap R$ and $S_{IM1} = S_{IM} \cap R$ where

$S_M = \Pi_{\text{Prereq.ConditionPM}} (\sigma_{\text{admin.role = ADrole}} (\textit{Can-assignp-M}))$

$S_{IM} = \Pi_{\text{Prereq.ConditionPM}} (\sigma_{\text{admin.role = ADrole}} (\textit{Can-assignp-IM}))$

$R = \Pi_{\text{RoleName}}(\sigma_{\text{PermName} = p_j} (\text{ROLE-PERM}))$

if p_j is an mobile member of r and $S_{M1} \neq \phi$,
then there exists role \$ $r_1 \in S_{M1}$, such that
$r_1 \in \Pi_{\text{Role Range}} (\sigma_{\{\text{ADrole, r}\}}(\text{Can-assign-M}))$, and ($p_j$, r ∈ PA)
/* p_j is in the range to be assigned as a mobile member by ADrole in Can-assignp-M */

If p_j is an immobile member of r and $S_{IM1} \neq \phi$,
then there exists role $r_i \in S_{IM1}$, such that
$r_i \in \Pi_{\text{Role Range}} (\sigma_{\{\text{ADrole, r}\}}(\text{Can-assign-IM}))$, and ($p_j$, r ∈ PA)
go to step 2
/* p_j is in the range to be assigned as an immobile member by ADrole in *Can-assign-IM* */

else
return false and stop.
/* the admini.role has no right to assign the role r_i as a mobile or immobile member to *R* */

Step 2. /* whether the permission p_j is conflicting with permissions of r or not */

Let
ConfPermS = $\Pi_{\text{ConfPerm}} (\sigma_{\text{PermName} = p_j} (\text{PERM}))$
/* It is the conflicting permission set of the permission p_j */

if ConfPermS $\cap P^* \neq \phi$,
then
return false; /* p_j is a conflicting permission with role r */

else
return true. /* p_j is not a conflicting permission with r */ ◊

This algorithm provides a way to decide whether a permission can be assigned to a role as mobile or immobile member. For mobile member, S_{M1} cannot be empty, and for immobile member, S_{IM1} cannot be empty.

Theorem 1: The authorization granting algorithm can prevent conflicts when assigning a permission to a role with mobile and immobile memberships.

Proof Assuming an administrator role ADrole wants to assign a permission p_j as a mobile member to a role which associates with a permission set *P*. Step 1 in the algorithm has checked whether the ADrole can assign p_j as a mobile member to the role or not, and the second step has decided whether the permission p_j conflicts with permissions in P^* or not. Indeed, p_j can be assigned to the role if for all $p_i \in P^*$, p_i is not in the conflicting permission set of p_j. Otherwise p_j is a conflicting permission with P^*. ◊

Similar to the time complexity analysis in the last section, we have the following corollary.

Corollary 1: The authorization granting algorithm has time complexity $O(n^2)$ for the case of *n* roles in a system.

Now we consider revocation of permission-role membership. Similar to *Can-assignp-M* and *Can-assignp-IM* relations in granting a permission to a role, there are *Can-revokep-M* and *Can-revokep-IM* relations.

Relations *Can-revokep-M* $\subseteq AR \times CR \times 2^R$ and *Can-revokep-IM* $\subseteq AR \times CR \times 2^R$ show which role range of mobile membership and immobile membership administrative roles can revoke respectively, where *AR* is a set of administrative roles.

The meaning of *Can-revokep-M* (ShopSO, SHOP, [SHOP, MANAGER)) in Table 9 is that a member of the administrative role ShopSO can revoke mobile membership of a permission from any role in [SHOP, MANAGER) subject to the prerequisite condition SHOP. *Can-revokep-IM* is similar with respect to immobile membership.

Admin.role	Prereq.ConditionPRM	Role Range
ShopSO	SHOP	[SHOP, MANAGER)

Table 9: Can-revokep-M

The evaluation of a prerequisite condition for the revoke model is different from the grant model. In the revoke model a *prerequisite conditionPRM* is evaluated for a permission *p* by interpreting role *x* to be true if

$$p \in EMx \vee p \in EIMx \vee p \in ImMx \vee p \in ImIMx$$

and $\bar{x}$ to be true if

$$p \notin EMx \wedge p \notin EIMx \wedge p \notin ImMx \wedge p \notin ImIMx$$

Due to role hierarchy, a role *x'* has all permissions of role *x* when *x'>x*. A user with two roles *{x', x}* still has the permissions of *x* if only to revoke *x* from the user. To solve the authorization revocation problem along with mobility of permission, we need to revoke the explicit member of a permission first if a role is an explicit member, then revoke the implicit member.

Following are two algorithms for revocation of a permission p_j as mobile or immobile members from a set of permission *P* by an administrative role ADrole, where *P* is a set of permissions which are assigned to a role *r*. The first one is the weak revocation algorithm and the second is the strong revocation algorithm. The weak revocation only revokes explicit mobile and immobile memberships from *r* and does not revoke implicit mobile and immobile memberships but the strong revocation revokes both explicit and implicit mobile and immobile members. The structure of the weak revocation algorithm is shown in Figure 3.

Weak revocation Algorithm

Weak_revokeMP(ADrole, r, p_j)

Input: ADrole, a roles *r* and a permission p_j.

Output: true if ADrole can weakly revoke role p_j from *r*; false otherwise.

Begin:

If $p_j \notin P = \{p \mid (p,r) \in PA\}$,

return false; /* there is no effect with the operation of the weak revocation since the permission p_j is not an explicit member of the role *r* */

else /* p_j is an explicit member of *r, either an mobile member or immobile member* */

Case1: p_j is an mobile member of *r*,

Roleswithp$_j$ = $\Pi_{\text{RoleName}}(\sigma_{\text{PermName} = p_j}(\text{ROLE-PERM}))$

/* Roles with permission p_j */

PreM = $\Pi_{\text{Prereq.ConditionPRM}}(\sigma_{\{\text{admin.role = ADrole}\}}(\text{Can-revokep-M}))$

/* Prerequisite condition with ADRole */

if RP= Roleswithp$_j$ $\cup$ PreM $\neq \phi$

RevokeRangeM = $\Pi_{\text{Role Range}}(\sigma_{\text{admin.role = ADrole}}(\text{Can-revokep-M}))$

if RR = Roleswithp$_j$ $\cup$ RevokeRangeM $\neq \phi$,

return, true.

/* the mobile member p_j is revoked */

else

return false;

/* the mobile member p_j cannot be revoked since the role *r* is not in the role range to be revoked */

else

return false and stop.

/* The p_j does not satisfy the prerequisite conditions */

Case 2: if p_j is an immobile member of *r*

PreIM = $\Pi_{\text{Prereq.ConditionPRM}}(\sigma_{\{\text{admin.role = ADrole}\}}(\text{Can-revokep-IM}))$

/* Prerequisite condition with ADRole */

if RPI= Roleswithp$_j$ $\cap$ PreIM $\neq \phi$

RevokeRangeIM = $\Pi_{\text{Role Range}}(\sigma_{\text{admin.role = ADrole}}(\text{Can-revokep-IM}))$

if RRI= Roleswithp$_j$ $\cap$ RevokeRangeIM $\neq \phi$,

return true,

/* the immobile member p_j is revoked */

else
return false ;
/* the immobile member p_j cannot be revoked */

else
return false and stop.
/* The p_j does not satisfy the perrequisite conditions */ ◊

The weak revocation algorithm can be used to check whether an administrator can weakly revoke mobile and immobile memberships from roles or not. We have the following result with the weak revocation algorithm.

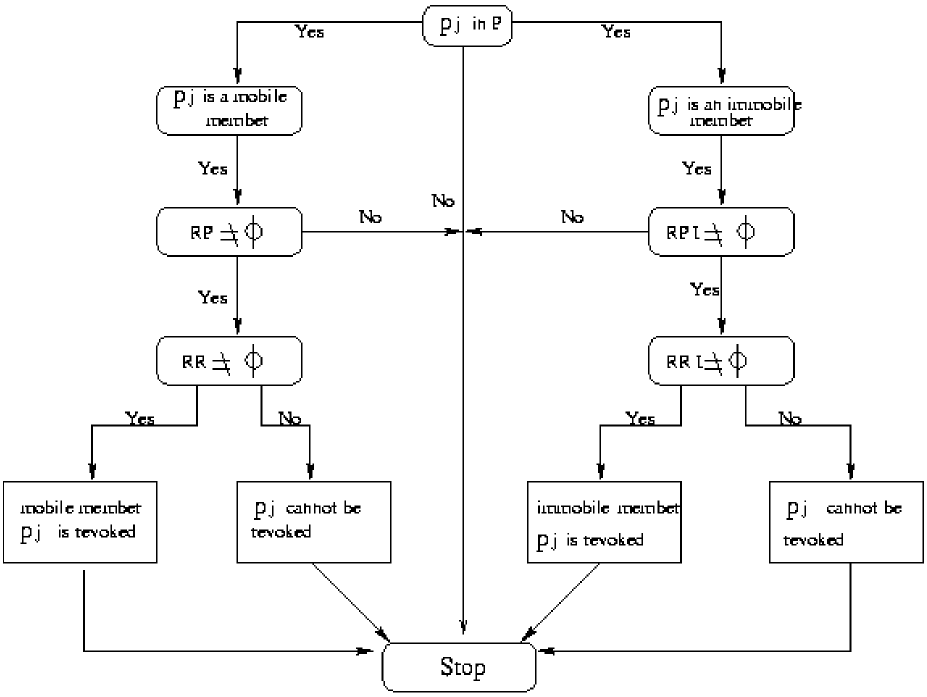

Figure 3: Structure of weak revocation algorithm for permission

Theorem 2: A permission p_j as mobile or immobile member is revoked by the weak revocation algorithm *Weak_revoke(ADrole, r, p_j)* if the permission is an explicit member of role *r* and the ADrole has the right to revoke p_j from the *Can-revoke-M* and *Can-revoke-IM* relations. ◊

A role still owns a permission of a system, which has been weakly revoked, if the role is senior to another role associated with the permission. To solve the authorization revocation problem, we need strong revocation, which requires revocation of both explicit-implicit membership and mobile-immobile memberships. Strong revocation of a permission's membership in role *r* requires that the permission be removed not only from explicit mobile and immobile membership in *r*, but also from explicit, and implicit mobile and immobile membership in all roles junior to *r*.

Strong revocation algorithm

Strong_revoke(ADrole, r, p_j)
Input: ADrole, a role *r* and a permission p_j.
Output: true, if it can strong revoke the permission p_j from *r*; false otherwise.
Begin:
If $p_j \notin P^*$,
return false;

/* there is no effect of the strong revocation since the permission is not an explicit and implicit member of the role *r* */

else,
Case 1. *if* $p_j \in$ P, do *Weak_revoke(ADrole, r, p_j)*;
/* p_j is weakly revoked from *r* as mobile or immobile */;

Case 2. Suppose
Jun = $\Pi_{\text{Junior}}(\sigma_{\{\text{Senior}=r\}}(\text{SEN-JUN}))$,
for all y ∈ Jun ∩ P $,
if $p_j \in R$ is a mobile member of the role *y*, do
Weak_revoke(ADrole, y, p_j) as $p_j \in$ EMy;
if $p_j \in R$ is a immobile member of the role *y*, do
Weak_revoke(ADrole, y, p_j) as $p_j \in$ EIMy;
/* the permission p_j is weakly revoked from all such y ∈ Jun ∩ P */.

If all the weak revocation s are successful,
return true;
otherwise,
return false.
/* if one weak revocation cannot finish*/ ◊

We have the following consequence.

Theorem 3: The explicit mobile and immobile and implicit mobile and immobile members of role p_j are revoked from a role by the Strong revocation algorithm *Strong_revoke(ADrole, r, p_j)* if the ADrole has the right to revoke p_j from the *Can-revokep-M* and *Can-revokep-IM* relations.

Corollary 2: The authorization revocation problem is solved by the Weak revocation algorithm and Strong revocation algorithm.

4 Related work

There are several other related works on the security of relational databases [Chu and Tseng, 2008, Yang et al. 2008, Osborn et al. 1996].

The interaction between RBAC and relational databases are presented in [Osborn et al. 1996]. Two experiments are described. One is a role-based front end to a relational database with discretionary access control. The other is a role graph to show the roles in a standard relational database. Some relational concepts like roles, users and permissions are provided. Our model also supports such concepts even though it has a large variety. However, the main difference between our algorithms and the scheme in [Osborn et al. 1996] is that we focus on the solutions of the conflicts of roles and permissions, and the latter focuses on the correlation of RBAC with discretionary access controls. Their work discusses the relationship between roles and discretionary access controls, they do not address the allocation of permissions to roles without conflicts. In our work, we developed detailed algorithms for allocating roles and permissions and checking their conflicts.

An architecture for the integration of tutoring approaches and process scaffolds into existing collaborative applications have been developed by Harrer et al. [2008]. The architecture allows to combine existing research on explicit representations of collaborative learning processes with the availability of existing and tested collaborative learning environments, but also to control the learning environments and thus enables adaptation of the tools to the current state of the learning process. Their work is different from the work in this paper. We focus on the possible conflicts in the permission assignments with role-based access model while they developed an architecture for adoption existing tools.

An oracle implementation for permission-role assignment has been proposed in [Sandhu and Bhamidipati1998]. In [Sandhu and Bhamidipati1998], the difference between permission-role assignment and Oracle database management system was analyzed. Furthermore, through prerequisite conditions, the paper has demonstrated how to use Oracle stored procedures for implementation. However, the work in this chapter substantially differs from that proposal. Differences are due to the consistency problem that arises in [Sandhu and Bhamidipati1998]:

It is very difficult to keep the consistency by reflecting security requirements between global network objects and local network objects if there are hundreds of roles and thousands of users in a system.

This problem is completely overcome in our algorithms because the algorithms focus on the conflicts between roles and permissions. The authorization granting algorithms are used to find conflicts and prevent some secret information from being derived while the strong revocation algorithms are used to check whether a role still has permissions of another role.

5 Conclusions

This paper has provided new authorization allocation algorithms for mobility of permission-role assignments that are based on relational algebra operations. They are the authorization granting algorithm, weak revocation algorithm, and strong revocation algorithm. The algorithms can automatically check conflicts when granting more than one permission to a role in a system. They can prevent users associated with roles from accessing unauthorized use of facilities when the permissions of the roles are changed within the organization and demand the modification of security rights. The permissions can be allocated without compromising the security in RBAC and provide secure management for systems. For better understanding, a bank example is adopted in the paper to explain the ideas of mobile members, immobile members, roles and users. Finally, we have discussed the related work in this area.

The future work will be the development of a system management with XML which involves the role-based group delegation subsystem.

Acknowledgements

The authors would like to thank the reviewers for their constructive suggestions and comments on the paper.

References

[Barkley et al. 1999] Barkley, J. F., Beznosov, K., Uppal J.: "Supporting relationships in access control using role based access control", Proc. third ACM Workshop on Role-Based Access Control, 55--65, 1999.

[Bella, 2008] Bella, G.: "What is Correctness of Security Protocols?", Journal of Universal Computer Science, Vol. 14, num. 12, 2083--2107, 2008.

[Chen et al. 2008] Chen, L. et al: "Cryptography in Computer System Security", Journal of Universal Computer Science, Vol. 14, num. 3, 314--315, 2008.

[Chu and Tseng, 2008] Chu, C. K. and Tzeng, W. G.: "Efficient k-out-of-n Oblivious Transfer Schemes", Journal of Universal Computer Science, Vol. 14, num. 3, 397--415, 2008.

[David et al. 1993] David, F. F., Dennis, M. G. and Nickilyn, L.: "An examination of federal and commercial access control policy needs", Proc. NIST NCSC National Computer Security Conference, Baltimore, MD, 107--116, 1993.

[Feinstein H. L. 1995] Feinstein, H. L.: "Final report: Nist small business innovative research (sbir) grant: role based access control: phase 1", Technical report, SETA Corp. 1995.

[Ferraiolo and Kuhn 1992] Ferraiolo, D. F. and Kuhn, D. R.: "Role based access control", Proc. 15th National Computer Security Conference, 554--563, 1992

[Ferraiolo et al. 1999] Ferraiolo, D. F., Barkley, J. F. and Kuhn, D. R.: "Role-based access control model and reference implementation within a corporate intranet", TISSEC, Vol. 2, 34--64, 1999.

[Goldschlag et al. 1999] Goldschlag, D., Reed, M., and Syverson, P.: "Onion routing for anonymous and private Internet connections", Communications of the ACM, 24(2), 39--41, 1999.

[Harrer et al. 2008] Harrer, A., Malzahn, N. and Wichmann, A.: "The Remote Control Approach - An Architecture for Adaptive Scripting across Collaborative Learning Environments", Journal of Universal Computer Science, Vol. 14(1), 148--173, 2008.

[Osborn et al. 1996] Osborn, S. L., Reid, L. K. and Wesson, G. J.: “On the Interaction Between Role-Based Access Control and Relational Databases”, Proc. IFIP WG11.3 Tenth Annual Working Conference on Database Security, 139--151, 1996.

[Sandhu R. 1997] Sandhu, R.: “Rational for the RBAC96 family of access control models”, Proc. of the 1st ACM Workshop on Role-based Access Control, ACM Press, 64--72, 1997.

[Sandhu R. 1998] Sandhu, R.: “Role activation hierarchies”, Proc. third ACM Workshop on Role-Based Access Control, ACM Press, 33--40, 1998.

[Sandhu and Bhamidipati 1998] Sandhu, R. and Bhamidipati, V.: “An oracle implementation of the pra97 model for permission-role assignment”, Proc. ACM Workshop on Role-Based Access Control, 13--21, 1998.

[Sandhu and Munawer 1999] Sandhu, R. and Munawer, Q.: “The arbac99 model for administration of roles”, Proc. the Annual Computer Security Applications Conference, ACM Press, 229--238, 1999.

[Wang et al. 2003ADC] Wang, H., Cao, J. and Zhang, Y.: “Formal authorization allocation approaches for permission-role assignments using relational algebra operations”, Proc. the 14th Australian Database Conference, Adelaide, Australia. Pages: 125--134, 2003.

[Wang et al. 2005TKDE] Wang, H., Cao, J., Zhang, Y.: “A flexible payment scheme and its role based access control”, IEEE Transactions on Knowledge and Data Engineering, 17(3), 425--436, 2005.

[Wang et al. ACSC04] Wang, H., Sun L., Cao, J., and Zhang, Y.: “Anonymous access scheme for electronic-services”, Proc.the Twenty-Seventh Australasian Computer Science Conference, Dunedin, New Zealand, 296--305, 2004.

[Wang et al. ACSC05] Wang, H., Sun, L., Zhang, Y., and Cao, J.: “Authorization Algorithms for the Mobility of User-Role Relationship”, Proc.the 28th Australasian Computer Science Conference, Newcastle, Australia, 167--176, 2005.

[Wang et al. 2003Special] Wang, H., Zhang, Y., Cao, J. and Kambayahsi, Y.: “A global ticket-based access scheme for mobile users”, Special Issue on Object-Oriented Client/Server Internet Envir., Information Systems Frontiers, 6(1),35--46, 2003.

[Wang et al. 2003TSMC] Wang, H., Zhang, Y., Cao, J. and Varadharajan, V.: “Achieving secure and flexible m-services through tickets”, IEEE Transactions on Systems, Man, and Cybernetics, Part A, Special issue on M-Services, 697--708, 2003.

[Yao et al. 2001] Yao, W., Moody, K., and Bacon, J.: “A model of oasis role-based access control and its support for active security”, Proc. of ACM Symposium on Access Control Models and Technologies, Chantilly, VA, 171--181, 2001.

[Yang et al. 2008] Yang, G., Wong, D.S. and Deng, X.: “Formal Security Definition and Efficient Construction for Roaming with a Privacy-Preserving Extension”, Journal of Universal Computer Science, Vol. 14, num. 3, 441--462, 2008.