Proceedings of 3rd ACM Workshop on Role-Based Access Control, Fairfax, Virginia, October 22-23, 1998

An Oracle Implementation of the PRA97 Model for Permission-Role Assignment*

Ravi Sandhu and Venkata Bhamidipati
Laboratory for Information Security Technology and
Information and Software Engineering Department
George Mason University

Abstract

In role-based access control (RBAC) permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. Using RBAC to manage RBAC provides additional administrative convenience. ARBAC97 is an administrative model recently proposed by Sandhu et al [SBC+97]. In this paper we demonstrate the implementation of one of the components of ARBAC97 which deals with permission-role assignment and is called PRA97. Although PRA97 is quite different from that built into the Oracle database management system, we demonstrate how to use Oracle stored procedures to implement it.

1 INTRODUCTION

Role-based access control (RBAC) has recently received considerable attention as a promising alternative to traditional discretionary and mandatory access controls. In RBAC permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Role-role relationships can be established to lay out broad policy objectives.

Using RBAC to manage RBAC provides additional administrative convenience. Sandhu et al [SBC+97] recently introduced a comprehensive model for role-based administration of RBAC called ARBAC97 (administrative RBAC '97). ARBAC97 has three components.

- URA97 for user-role assignment
- PRA97 for permission-role assignment
- RRA97 for role-role assignment

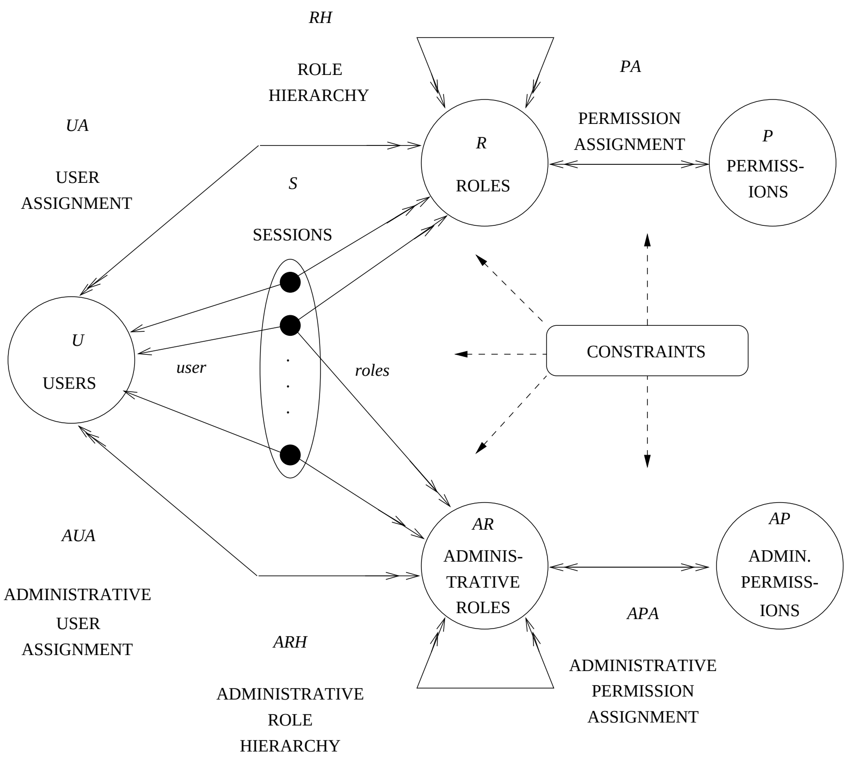
Assigning permissions to roles is typically the province of application administrators. Thus a banking application can be implemented so credit and debit operations are assigned to a teller role, whereas approval of a loan is assigned to a managerial role. Assignment of actual individuals to the teller and managerial roles is a personnel management function. Assigning roles to roles has aspects of user-role assignment and role-permission assignment. Role-role relationships establish broad policy. Control of these relationships would typically be relatively centralized in the hands of a few security administrators.

The main contribution of this paper is to show how PRA97 can be implemented in Oracle. The PRA97 model for permission-role assignment is a dual of the URA97 model for user-role assignment developed by Sandhu and Bhamidipati [SB97]. An Oracle implementation of URA97 is described in [SB98]. Although conceptually PRA97 and URA97 are duals, the implementation of PRA97 is more complicated and challenging because Oracle permissions are stored in multiple internal tables and different permissions have different characteristics.

The rest of this paper is organized as follows. We briefly review the PRA97 model in section 2. In sec-

*This work is partially supported by grant CCR-9503560 from the National Science Foundation at the Laboratory for Information Security Technology at George Mason University.

All correspondence should be addressed to Ravi Sandhu, ISE Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22030, `sandhu@isse.gmu.edu`, `www.list.gmu.edu`.



- U, a set of users
  R and AR, disjoint sets of (regular) roles and administrative roles
  P and AP, disjoint sets of (regular) permissions and administrative permissions
  S, a set of sessions

- $UA \subseteq U \times R$, user to role assignment relation
  $AUA \subseteq U \times AR$, user to administrative role assignment relation

- $PA \subseteq P \times R$, permission to role assignment relation
  $APA \subseteq AP \times AR$, permission to administrative role assignment relation

- $RH \subseteq R \times R$, partially ordered role hierarchy
  $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
  (both hierarchies are written as $\geq$ in infix notation)

- $user : S \rightarrow U$, maps each session to a single user (which does not change)

  $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

  session s_i has the permissions $\cup_{r \in roles(s_i)} \{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$

- there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.

Figure 1: Summary of the RBAC96 Model

tion 3 we describe our implementation of PRA97 in Oracle. Section 4 concludes the paper.

2 THE PRA97 ADMINISTRATIVE MODEL

PRA97 is defined in context of the well-known RBAC96 model [SCFY96]. For convenience, RBAC96 is summarized in figure 1. PRA97 is defined in two steps: granting permissions to a role and revoking permissions from a role. It is a precise dual of the URA97 model for user-role assignment [SB97, SB98]. For convenience, we define PRA97 completely here without reference to URA97.

2.1 PRA97 GRANT MODEL

In the simplest case permission-role assignment can be completely centralized in a single chief security officer. This is readily implemented in existing systems like Oracle. However, our goal is to decentralize the permission-role assignment. The key concept of PRA97 is to impose restrictions on what permissions can be assigned to a role by whom. We achieve this by means of prerequisite conditions.

Definition 1 A *prerequisite condition* is a boolean expression using the usual $\wedge$ and $\vee$ operators on terms of the form x and $\overline{x}$ where x is a regular role (i.e., $x \in R$). A prerequisite condition is evaluated for a permission p by interpreting x to be true if $(\exists x' \geq x)(p, x') \in PA$ and $\overline{x}$ to be true if $(\forall x' \geq x)(p, x') \notin PA$. For a given set of roles R let CR denote all possible prerequisite conditions that can be formed using the roles in R. □

Definition 2 The PRA97 model controls permission-role assignment by means of the relation *can-assignp* $\subseteq AR \times R \times 2^R$. □

The meaning of *can-assignp*(x, y, Z) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a permission whose current membership, or non-membership in regular roles satisfies the prerequisite condition y to regular roles in range Z.

To appreciate the motivation behind the *can-assignp* relation consider the role hierarchy of figure 2 and the administrative role hierarchy of figure 3. Figure 2 shows the regular roles that exist in a engineering department. There is junior-most role E to which all employees in the organization belong. Within the engineering department there is a junior-most role ED and senior-most role DIR. In between there are roles for two projects within the department, project 1 on the left and project 2 on the right. Each project has a senior-most project lead role (PL1 and PL2) and a junior-most engineer role (E1 and E2). In between each project has two incomparable roles, production engineer (PE1 and PE2) and quality engineer (QE1 and QE2).

Administrative Role	Prerequisite Condition	Role Range
DSO	DIR	[PL1, PL1]
DSO	DIR	[PL2, PL2]
PSO1	PL1 $\wedge$ $\overline{\text{QE1}}$	[PE1, PE1]
PSO1	PL1 $\wedge$ $\overline{\text{PE1}}$	[QE1, QE1]
PSO2	PL2 $\wedge$ $\overline{\text{QE2}}$	[PE2, PE2]
PSO2	PL2 $\wedge$ $\overline{\text{PE2}}$	[QE2, QE2]

Table 1: Example of *can-assignp*

Administrative Role	Role Range
DSO	(ED, DIR)
PSO1	[QE1, QE1]
PSO1	[PE1, PE1]
PSO2	[QE2, QE2]
PSO2	[PE2, PE1]

Table 2: Example of can-revokep

Figure 2 suffices for our purpose but this structure can, of course, be extended to dozens and even hundreds of projects within the engineering department. Moreover, each project could have a different structure for its roles. The example can also be extended to multiple departments with different structure and policies applied to each department.

Figure 3 shows the administrative role hierarchy which co-exists with figure 2. The senior-most role is the senior security officer (SSO). Our main interest is in the administrative roles junior to SSO. These consist of two project security officer roles (PSO1 and PSO2) and a department security officer (DSO) role with the relationships illustrated in the figure.

For sake of illustration we define the *can-assignp* relation shown in table 1. The PSO1 role has partial responsibility over project 1 roles. Let Alice be a member of the PSO1 role and BACKUP_ANY_TABLE be a permission assigned to PL1. Alice can assign the permission to either QE1 or PE1 but not to both. If Bob is a member of DSO role he can add a permission to PL1 or PL2 if the permission is already assigned to DIR.

Role ranges are specified in PRA97 by means of the familiar closed and open interval notation.

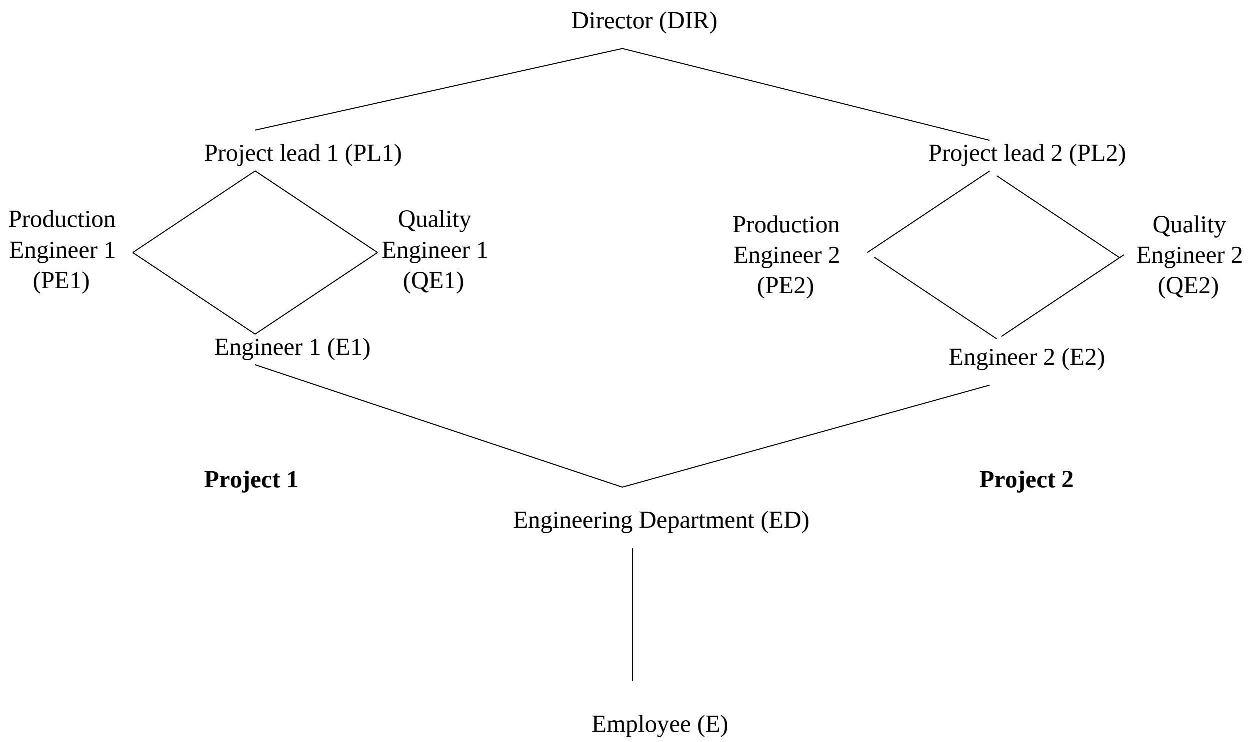


Figure 2: An Example Role Hierarchy

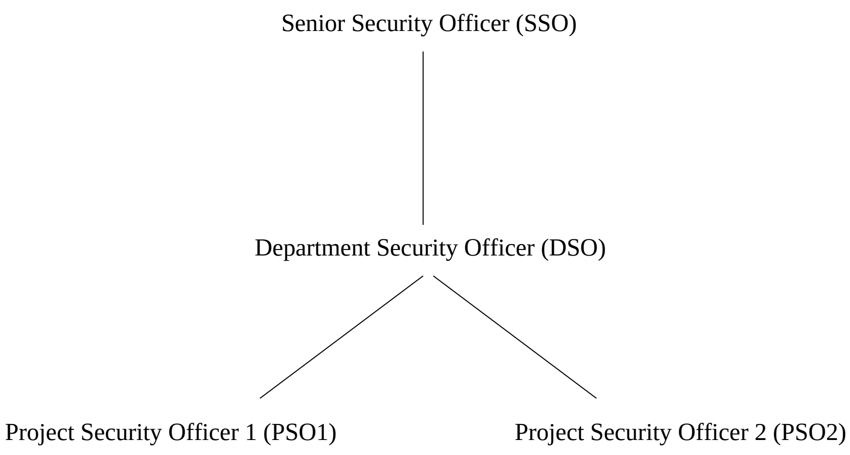


Figure 3: An Example Administrative Role Hierarchy

$$
\begin{aligned}
[x, y] &= \{r \in R \mid x \geq r \wedge r \geq y\} \\
(x, y] &= \{r \in R \mid x > r \wedge r \geq y\} \\
[x, y) &= \{r \in R \mid x \geq r \wedge r > y\} \\
(x, y) &= \{r \in R \mid x > r \wedge r > y\}
\end{aligned}
$$

2.2 PRA97 REVOKE MODEL

We now turn to consideration of the PRA97 revoke model. The objective is to define a revoke model that is consistent with the philosophy of RBAC. The revocation in PRA97 is independent of assignment. If Alice by means of some administrative role can revoke a permission from a role say Bob. The revocation takes place independent of how that permission was assigned to Bob. The granting or revoke of permission is done for functional reasons and application needs and not merely at the discretion of administrators.

Definition 3 The PRA97 model controls permission - role revocation by means of the relation *can-revokep* $\subseteq AR \times 2^R$. □

The meaning of *can-revokep*(x, Y) is that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a permission from any regular role $y \in Y$. We say Y defines the *range of revocation*. For example look at table 2.

Similar to URA97 we have two notions of revocation in PRA97 called *weak* and *strong* .

2.2.1 Weak revocation

Definition 4 Let us say a permission P is an *explicit member* of role x if (P,x)$\in PA$, and that p is an *implicit member* of role x if for some x$'$ <x, (P,x$'$)$\in PA$. □

Note that P can simultaneously be explicitly and implicitly assigned to a role. Weak revocation has an impact only on explicit membership. It has the straightforward meaning stated below.

Definition 5 [Weak Revocation Algorithm]

1. Let Alice have a session with administrative roles $A = \{a_1, a_2, \ldots, a_k\}$, and let Alice try to weakly revoke permission P from role x.

2. If P is not an explicitly granted to x this operation has no effect, otherwise there are two cases.

   (a) There exists a *can-revokep* tuple (b, Y) such that there exists $a_i \in A, a_i \geq b$ and $x \in Y$.

   In this case P's explicit assignment from x is revoked.

   (b) There does not exist a *can-revokep* tuple as identified above.

   In this case the weak revoke operation has no effect.

□

Suppose Alice who is a member of PSO1 role wants to weakly revoke a permission from PE1 role. The revoke will go through if the permission is explicitly assigned to PE1 role.

2.2.2 Strong Revocation

The strong revocation algorithm is expressed in terms of weak revoke by the following all-or-nothing transaction.

1. Let Alice have a session with administrative roles $A = \{a_1, a_2, \ldots, a_k\}$, and let Alice try to strongly revoke P from role x.

2. Find all roles $y \leq x$ and P is a member of y.

3. Weak revoke P from all such y as if Alice did this weak revoke.

4. If any of the weak revokes fail then Alice's strong revoke has no effect otherwise all weak revokes succeed.

Suppose Alice who is a member of PSO1 role wants to weakly revoke a permission from PE1 role. The revoke will go through if the permission is explicitly assigned only to PE1 role. However the revoke will fail if the permission was also explicitly assigned to role E, as the role is not in PSO1's revocation range.

3 IMPLEMENTING PRA97 IN ORACLE

3.1 Oracle Related Features

Permissions in Oracle are of two types: system privileges and object privileges. System privileges are permissions at the database level, for example, create table system privilege authorizes creation of tables. There are over 60 distinct system privileges. Object privileges authorize actions on a specific object of a schema (table, view, procedure, package etc.). Typical examples of object privileges are select rows from a table, delete rows, execute procedures etc.

Who can grant or revoke privileges from roles? The answer depends on various issues such as whether it is a system or an object privilege, and whether the object is owned by the user, etc. In order to grant or

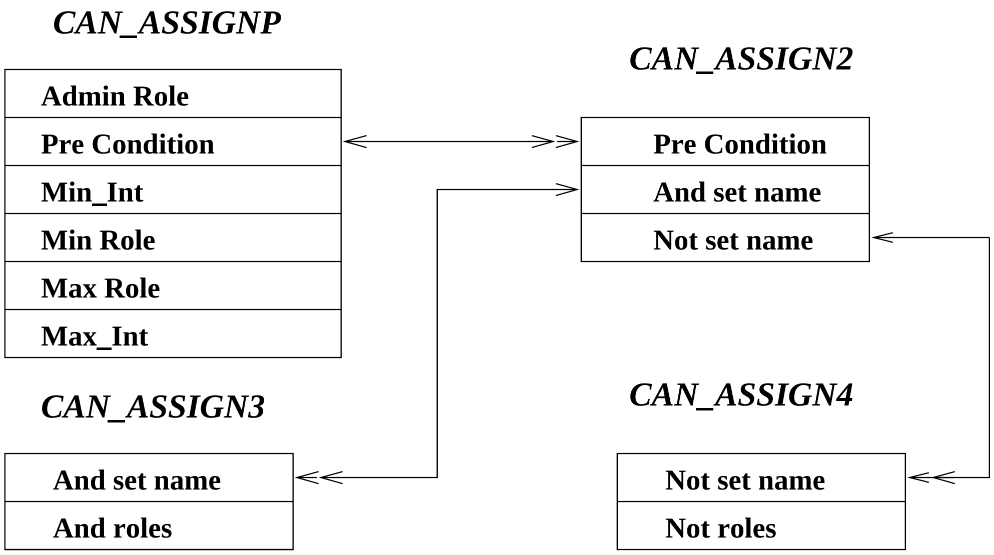


Figure 4: Entity-Relation Diagram for *can-assign* Relation

revoke a system privilege the user should have the admin option on that privilege or the user should have the GRANT_ANY_PRIVILEGE system privilege. In order to grant or revoke an object privilege a user should own that particular object or the user should have grant option on the object if it is owned by someone else.

Oracle provides a programmatic approach to manipulate database information using procedural schema objects called PL/SQL (Procedural Language/SQL) program units. Procedures, functions and packages are different types of PL/SQL objects. PL/SQL extends the capabilities of SQL by providing some programming language features such as conditional statements, loops etc.

A stored program unit runs with the privileges of the user who owns it and not the user who is executing it. This feature gives great flexibility in enforcing security. For example suppose we want a user to perform some operations on a database but we do not want to grant privileges explicitly. Then one can write a procedure embedded with necessary operations, and grant execute privileges on the procedure to the user.[1]

[1]The privileges that are exercised in a procedure should have been explicitly granted to the user who owns the procedure. Privileges obtained by the owner via a role cannot be referenced in a procedure.

3.2 Implementation of PRA97

To implement PRA97 we define Oracle relations which encode the *can-assignp* and *can-revokep* relations of PRA97. The *can-assignp* relation of PRA97 is implemented in Oracle as per the entity-relation diagram of figure 4. We assume that the prerequisite condition is converted into disjunctive normal form using standard techniques. Disjunctive normal form has the following structure.

$$(\ldots \wedge \ldots \wedge \ldots) \vee (\ldots \wedge \ldots \wedge \ldots) \vee \ldots \vee (\ldots \wedge \ldots \wedge \ldots)$$

Each $\ldots$ is a positive literal x or a negated literal $\overline{x}$. Each group $(\ldots \wedge \ldots \wedge \ldots)$ is called a disjunct.

For a given prerequisite condition *can-assign2* has a tuple for each disjunct (indicated by the double-headed arrow from *can-assignp* to *can-assign2*). All positive literals of a single disjunct are in *can-assign3*, while negated literals are in *can-assign4*. In our scheme the *can-assignp* relation shown in table 1 is represented as table 3 (this example only has a single disjunct for each prerequisite condition). For example, consider the first row for PSO1 in table 1. It is represented by the first row for PSO1 in table 3(a) which references the disjunct C2 in table 3(b). C2 in turn references the positive literal set ASET2 in table 3(c) and the negative literal set NSET2 in table 3(d).

The *can-revokep* relation of PRA97 is represented by a single Oracle relation. For example table 2 is represented as shown in table 4.

AR	PC	Min_Int	Min_Role	Max_Role	Max_Int
DSO	C1	[	PL1	PL1	]
DSO	C1	[	PL2	PL2	]
PSO1	C2	[	PE1	PE1	]
PSO1	C3	[	QE1	QE1	]
PSO2	C3	[	PE2	PE2	]
PSO2	C4	[	QE2	QE2	]
...	...	...	...	...	...

(a) *can-assign*

PC	and_set_name	not_set_name
C1	ASET1	null
C2	ASET2	NSET2
C3	ASET2	NSET3
C4	ASET3	NSET4
C5	ASET3	NSET5
...	...	...

(b) *can-assign*2

and_set_name	and_roles
ASET1	DIR
ASET2	PL1
ASET3	PL2
...	...

(c) *can-assign*3

not_set_name	not_roles
NSET2	QE1
NSET3	PE1
NSET2	QE2
NSET3	PE2
...	...

(d) *can-assign*4

Table 3: Oracle *can-assign* Relations for PSO1 from Table 1

AR	Min_Int	Min_Role	Max_Role	Max_Int
DSO	(	ED	DIR	)
PSO1	[	PE1	PE1	]
PSO1	[	QE1	QE1	)
PSO1	[	PE2	PE2	]
PSO1	[	QE2	QE2	)

Table 4: Oracle *can-revoke* Relation

We define a special user called Rolemanager who owns all the relations of tables 3 and 4. Rolemanager decides the content of these relations. He also has the DBA[2] role. The user Rolemanager should also be granted with select privileges on some database internal tables owned by the user SYS.

In addition we have a set of accompanying procedures and a package to perform the grants and revokes. We have two procedures to perform the assign operation, one for assignment of system privileges and other for assignment of object privileges. Similarly we have two procedures for weak revoke, and another two procedures for strong revoke. The package consists of several functions which contain the routines to check whether requirements are met or not. To grant or revoke a permission an administrator has to call a procedure and provide required parameters. The details of the procedure calls are given below.

3.2.1 Granting Permissions to a Role

When a privilege is being assigned to a role the user calls the appropriate procedure depending upon whether it is a system privilege or it is a schema object privilege. The procedure calls are give below.

- grant_syspriv (role, tprivilege, arole)
- grant_objpriv (role, tprivilege, object, schema, arole)

The parameters role and tprivilege specify what privilege should be granted to the role. The parameter arole specifies the administrative role that should be applied.[3] The parameters object and schema in grant_objpriv specify the the object name and schema[4] to which the object belongs.

When the procedure is executed, it checks if the user has the arole turned on. If the arole is not turned on an error message is generated and execution stops. After this initial check is performed the procedure gets the tuples from *can-assignp* which correspond to arole and all its junior roles and checks whether any of them satisfy the range check and prerequisite conditions. If any of the tuple satisfies the condition we perform the grant.

[2]DBA is a predefined role in Oracle and it has all the system privileges assigned to it.

[3]Oracle does not provide a facility for a stored procedure to determine which roles have been turned on in a given session. To circumvent this limitation we explicitly require the parameter arole to specify which admnistrative role should be used for the requested operation. More generally, arole should be a set. Our implementation can be easily extended to do this.

[4]We require that the schema user who owns the objects should grant all the privileges on the object to Rolemanager with grant option.

In order to check whether the user has activated arole we use Oracle's built in function IS_ROLE_ENABLED. To check whether the role is in the specified range for one of the relevant *can-assignp* tuples (in case of revoke it will be from *can-revokep*) we use Oracle CONNECT BY clause in our queries. By using CONNECT BY clause, one can traverse a tree structure corresponding to the role hierarchy in one direction. One can start from any point within the role hierarchy and traverse it towards junior or senior roles. But there is no control on the end point of the traversal. Specific branches or an individual node of the tree can be excluded by hard coding their values. Such hard coding is not appropriate for a general purpose stored procedure. In our implementation we overcome this problem by performing multiple queries and intersecting them to get the exact range. We specifically do not hard code any parameters in our queries. In order to check if the prerequisite condition is satisfied or not we perform queries against Oracle internal tables (objauth$, sysauth$ and obj$ etc.) In order to modularize our implementation we have developed a package which performs the necessary checks involved. All the procedures call this package to do the verification. The package contains several functions. Each one is designed to perform certain tasks, for example we have a function called *is_role_in_porder* to check whether role is in specified range or not. This function returns the results to the calling PL/SQL unit. Similarly there are other functions to perform other necessary checks.

Our implementation is convenient for the DBA since the stored procedures and packages we provide are generic and can be reused by other databases. The DBA only needs to define the roles and administrative roles, and configure the *can-assignp* and *can-revokep* relations. Our implementation is available in the public domain for other researchers and practitioners to experiment with.

3.2.2 Revoking Permissions from a role

The procedures that perform the revoke operation are give below.

- weak_revoke_syspriv (role, tprivilege, arole)
- weak_revoke_objpriv (role, tprivilege, object, schema, arole)
- strong_revoke_syspriv (role, tprivilege, arole)
- strong_revoke_objpriv (role, tprivilege, object, schema, arole)

The parameters are same as that of granting procedures but the operation changes from assignment to revoking from the role. Chaecking the authorization for the operation is similar to that for the grant operation.

In case of strong revoke we get all the roles which are junior to the role specified and see if the privilege being revoked is assigned to them. We follow all or nothing semantics so the revoke will go through only if all the junior roles which have the privilege fall in the range of administrative role.[5]

4 CONCLUSION

In this paper we described the PRA97 model for assigning permissions to roles and revoking permissions from roles and and showed how the model can be implemented in Oracle. PRA97 is a component of ARBAC97 [SBC+97]. It can deployed as an individual component or it can deployed along with other components of ARBAC97. In conjunction with earlier work we currently have implementations of PRA97 and URA97 in Oracle. Given our success in implementing PRA97 and URA97 in Oracle, it should be possible to implement RRA97 which has been recently defined [SM98].

References

[SA98] Ravi Sandhu and Gail-Joon Ahn. Decentralized group hieraches in unix: An experiment and lessons learned. In *Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, Arlington, VA, October 5-8 1998.

[SB97] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.

[SB98] Ravi S. Sandhu and Venkata Bhamidipati. Role-based administration of user-role assignment: The URA97 model and its Oracle implementation. *The Journal Of Computer Security*, 1998. in press.

[SBC+97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, Fairfax, VA, November 6-7 1997. ACM.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SM98] Ravi Sandhu and Qamar Munawer. The RRA97 model for role-based administration of role hierarchies. In *Proceedings of 13th Annual Computer Security Application Conference*, Scotsdale, AZ, December 7-11 1998.

[5] In some papers [SA98] two forms of strong revoke have been identified. These are called *drop* and *continue*. If one of the weak revokes fails the drop option stipulates that strong revocation has no effect, whereas with the continue option those weak revokes that are authorized will be performed.