An Enhanced Role-Based Access Control Model using Static Separation of Duty Concept

Burin Yenmunkong, and Chanboon Sathitwiriyawong

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

Abstract: This paper proposes a simple but practically useful model for preventing fraud of users called "ERBAC03". The new model consists of qualified mandatory and discretionary features for roles and locations, including the assignment of permissions for the appropriate roles and the assignment of roles for the appropriate locations. Moreover, a static separation of duty (SSoD) principle is applied to the new model for integrity requirements of security systems. The paper also explores some extensions of ERBAC03 including the new model using the SSoD concept from some experiments. The experimental results prove the efficiency improvement of the proposed model that can make benefits for large enterprises.

Keywords: Role-Based Access Control, Conflict of Interest, Static Separation of Duty, Data Integrity

1. INTRODUCTION

In large enterprises, a number of roles are created for various job functions and can be doubled to hundreds or thousands. Each enterprise has many branches (such as a post office and a bank) and each branch consists of many similar roles, so the efficiency of the existing RBAC model [1] must be improved to suit enterprises in the real world.

This paper focuses on a model for the constraint features referred to as the static separation of duty (SSoD) [2] that users are not fully authorized. The objectives of this research work are to improve the existing RBAC model for the efficiency of data integrity, to prevent fraud, and to increase the efficiency of system administration in large enterprises.

Perelson and Botha [3] proposed a mathematical model that is based on the concept of "conflicting entities" to express static separation of duty requirements. The proposed ERBAC03 model has adapted the concept of conflicting entities based on locations for data integrity.

Epstein [4] constructed a new model that extends the permission assignment of RBAC model between roles and permissions. His goal was to define a layered model that can be served as a basis for detailing an effective methodology to assign permission to roles.

Ahn and Sandhu [5] defined the RSL99 language to specify separation of duty constraints. Their SoD requirements are based on the concepts of conflicting users, conflicting roles and conflicting permissions. New static separation of duty properties are discovered through the application of their RSL99 language that is very useful to define the proposed ERBAC03 model.

2. ROLE-BASED ACCESS CONTROL MODEL

Sandhu [1] describes four different conceptual RBAC models. The first is $RBAC_0$, which is the base model that specifies the minimum set of requirements for any system that implements RBAC fully. The second is $RBAC_1$, which adds the concept of role hierarchies that allow roles to inherit permissions from other roles. The third is $RBAC_2$, which adds the concept of constraints that imposes restrictions upon the configuration of various components within RBAC. The final model is $RBAC_3$, which includes $RBAC_1$, $RBAC_2$ and $RBAC_0$. This family of models is better known as RBAC.

The RBAC model consists of four main entities: users, roles, permissions, and sessions. These entities and their relationships with another can be seen in Fig. 1

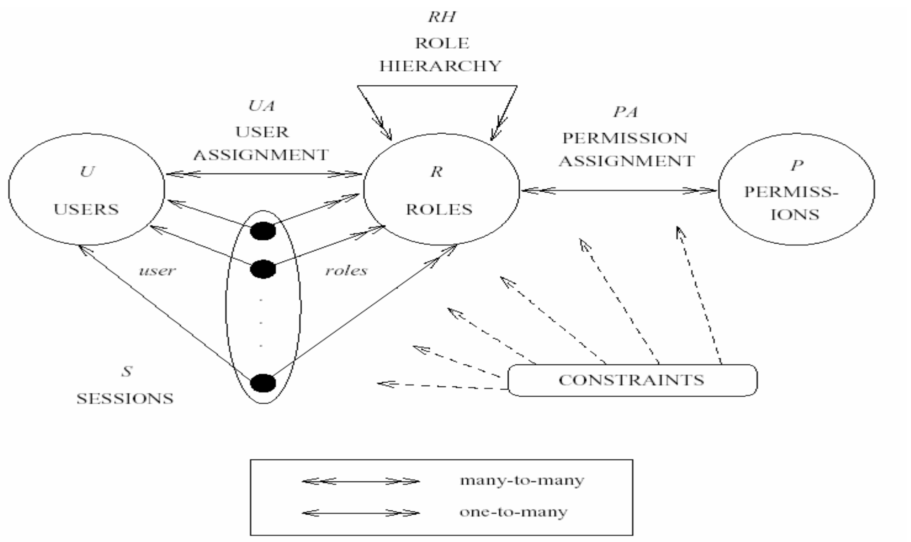


Fig. 1 RBAC model

A user denotes a human being but can also denote any automated process. To keep the model simple, it will be assumed that it is a human being. As shown in Fig.1, users are part of a set called U.

A role is generally a job function or title within an organization. A user that is assigned to a particular role would generally be working within the job function that the role denotes. Roles are part of a set called R as shown in Fig. 1.

A permission is a privilege to access objects in the organization in a certain way. The permission is always positive and will allow the user to access referenced objects in the way defined by the permission. The objects that are referred to could be either data objects or resource objects. Permissions are part of a set called P as shown in Fig. 1.

A session is a mapping of a user to one or more roles. This means that a session is unique to a particular user. The user also receives the union of all the permissions of the role. This means that a user that belongs to a few sessions could be associated with many roles. Each role may, in turn, associate with many permissions to the user and as such care must be taken to ensure that the user does not have the ability to commit fraud. Sessions are part of a set called S as shown in Fig. 1

3. ENHANCED ROLE-BASED ACCESS CONTROL (ERBAC03) MODEL

The RBAC model is extended by defining three additional entities as shown in Fig. 2. The concept of the new model was influenced by Epsteinŝ idea.

The ERBAC03 model requires the modification of the end-points (i.e., roles and permissions) by introducing three additional entities that consist of locations, jobs, and tasks. We concentrate on the creation of the elements between the end-points such as jobs and tasks in order to make permissions to roles more clear, also to apply it to large enterprises effectively.

A location consists of many roles and reflects large enterprises that have various branches. Each branch consists of many roles that reflect the model management in the distributed aspect. A role may perform more than one type of work. A role is responsible for all the activities that are required to perform the work. We define each type of work as a job. Jobs are not necessary in any sequence. We also show that the tasks requiring access to an application will be mapped to the permissions granted to the desired access.

The static separation of duty (SSoD) constraint is used in the proposed ERBAC03 model. It concerns 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.

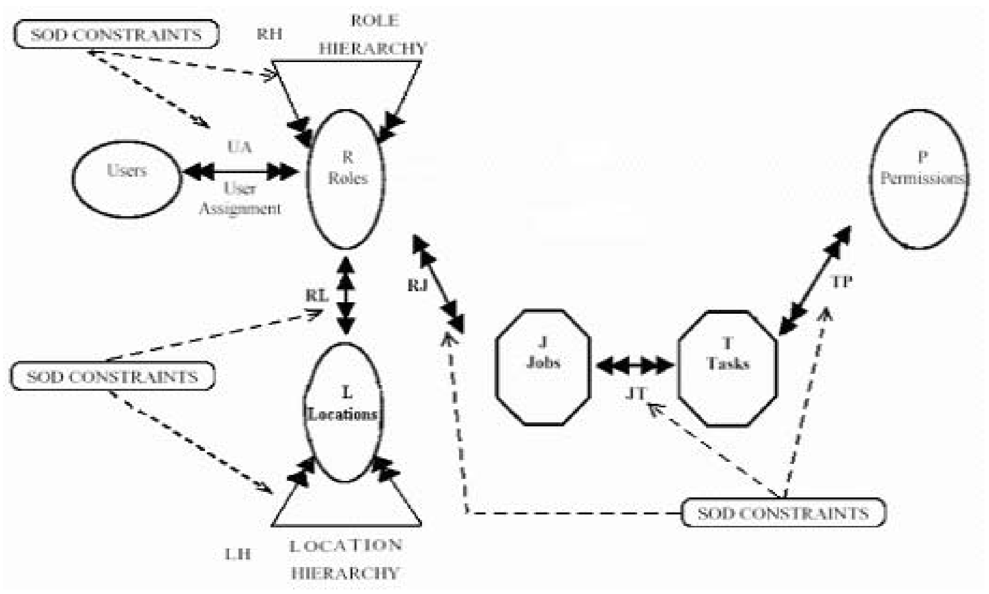


Fig. 2 ERBAC03 model

The term conflicting permissions (CP) indicates two permission entities that are in conflict with one another. If two conflicting permissions are associated with a common user, he will be able to commit fraud.

The term conflicting users (CU) is misleading. The users are in fact not conflicting at all. Rather, it is the usersŝ friendship that can result in an alliance to commit fraud. With this description it becomes relevant to consider that single users are in conflict with themselves.

The term conflicting roles (CR) indicates two role entities that are in conflict with one another. It is through conflicting roles that the associations between all entities can be controlled.

The term conflicting locations (CL) indicates two location entities that are in conflict with one another. If two conflicting locations are associated with a common role, the user would be able to commit fraud.

The term conflicting jobs (CJ) indicates two job entities that are in conflict with one another. It is through conflicting jobs that the associations between all entities can be controlled.

The term conflicting tasks (CT) are workflow process tasks that may not be performed by conflicting jobs. If conflicting jobs are allowed to perform these tasks, there exists the possibility of fraud being committed.

4. ALGORITHMS FOR ENTITY CONFLICTS

It is important to be able to correctly maintain the conflicts between entities in order for an administration tool to control data integrity effectively. We adapted some algorithm from Perelsonš idea [3].

4.1 Creating conflicting entities

Conflict assignments follow the same rules for all entities. The same general algorithm can be used to control all conflict assignments. These are depicted mathematically as $(u_i, u_j) \in CU$ for conflicting users, $(r_i, r_j) \in CR$ for conflicting roles, $(li, lj) \in CL$ for conflicting locations, $(j_i, j_j) \in CJ$ for conflicting jobs, $(t_i, t_j) \in CT$ for conflicting tasks and $(p_i, p_j) \in CP$ for conflicting permissions. This algorithm is depicted in Fig. 3

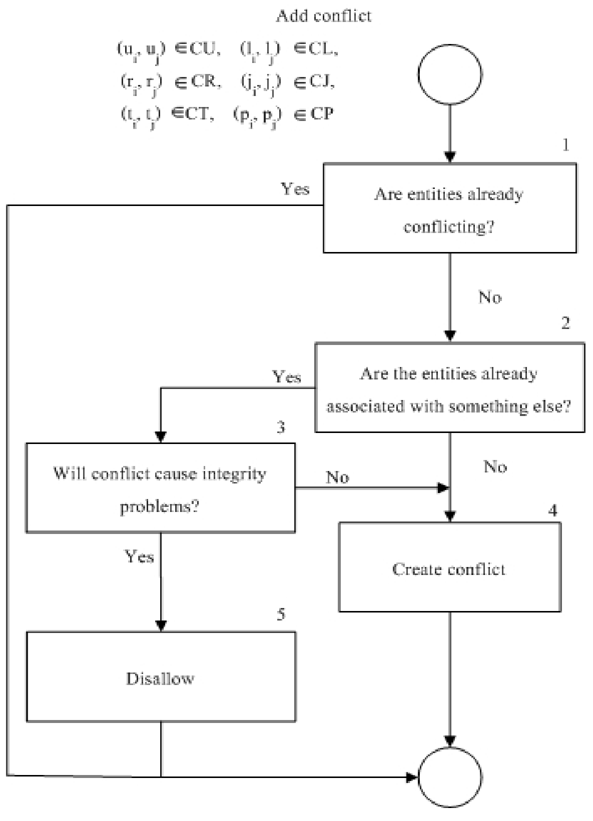

Add conflict
$(u_i, u_j) \in CU$, $(l_i, l_j) \in CL$,
$(r_i, r_j) \in CR$, $(j_i, j_j) \in CJ$,
$(t_i, t_j) \in CT$, $(p_i, p_j) \in CP$
1
Are entities already conflicting?
Yes
No
2
Are the entities already associated with something else?
Yes
No
3
Will conflict cause integrity problems?
No
Yes
4
Create conflict
5
Disallow

Fig. 3 Adding conflicting entities

Step 1: Check whether the entities are already conflicting with each other. This is done by checking the records in the relevant database table for one that matches the current entities. If a record is found, this conflict assignment has already been done and should not be done again. If the existing conflict is not found, it will be proceeded to step 2.

Step 2: Check whether the entities form associations. This is done by iterating through all the applicable database tables, searching for any prior associations for either of the entities. If an association is found, step 3 must be performed. Otherwise, the conflict may be created.

Step 3: An association may become invalid if the conflict assignment goes ahead. It is therefore necessary to disallow any conflict assignment that will cause integrity problems. If no problems are caused by the conflict, the conflict can be created. Otherwise, the conflict must be disallowed.

Step 4: Disallow the addition. This is accomplished in the form of an on-screen dialog that warns the administrator.

Step 5: Create the conflict assignment by writing a record into the appropriate database table.

4.2 Deleting conflicting entities

It is important to know what integrity requirements must be checked and maintained for each conflict type that is to be deleted.

The conflict of users, roles, locations, jobs, tasks, and permissions may be safely removed without affecting the system integrity. The deletion of entity conflicts requires proper integrity checks to ensure that the existing associations remain valid. This algorithm is shown in Fig. 4

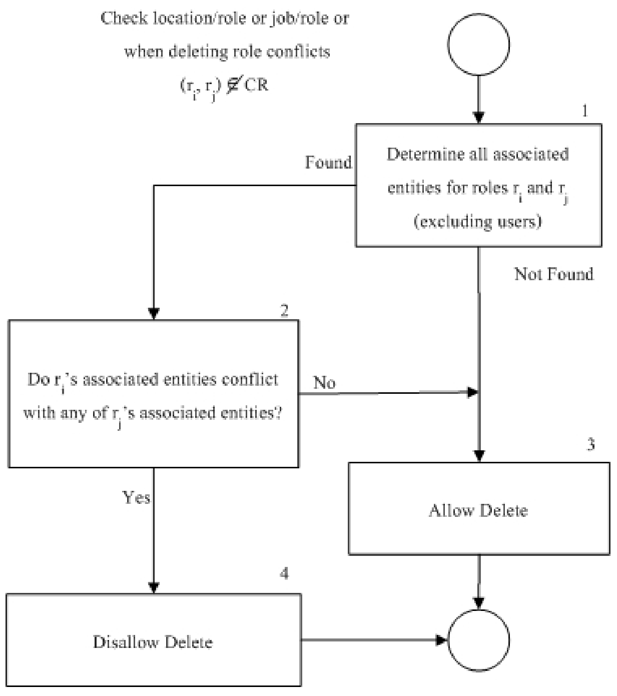

Check location/role or job/role or when deleting role conflicts
$(r_i, r_j) \notin CR$
1
Determine all associated entities for roles r_i and r_j (excluding users)
Found
Not Found
2
Do r_i's associated entities conflict with any of r_j's associated entities?
No
Yes
3
Allow Delete
4
Disallow Delete

Fig. 4 Checking entity associations for role conflict deletion

Step 1: We find all associated locations and jobs for both roles r_i and r_j. If associated entities are found for both roles, it will be proceeded to step 2. Otherwise, the conflict is allowed to be deleted.

Step 2: Check whether any of the jobs that are associated with the role r_i are conflicting with any of the jobs that are associated with the role r_j are conflicting with any of the locations that are associated with the role r_j. This is done by checking for the existence of the entity pairing in the correct conflicting entity database table. If any job or location is conflicting, the conflict is not allowed to be deleted.

Step 3: Do not delete the conflict, as it will cause an existing association to become illegal.

Step 4: Delete the conflict by removing the record from the appropriate database table.

5. THE EXPERIMENTATION

We define the hypothesis of the experiment, with metric and expected result, for the proposed ERBAC03 model as shown in table 1.

Table 1 An example of the hypothesis of the experiment

Hypothesis	Metric	Expected Result
ERBAC03 increases data integrity by constraint features	Elimination of incorrect data	Percentage of incorrect data that can be eliminated by ERBAC03

5.1 Experimental environment and limitation

- One computer (1 CPU, 450 MHz, RAM 128 MB, HDD 8 GB)
- One security administrator
- All experimental data are owned by the Thailand Post Company
- Oracle database management system 8i

5.2 Experimental design

Using database triggers to enforce the conflict paradigm constraints was the primary focus of this experiment. In designing the triggers, it was realized that they would have to examine the contents of their own changing database tables. This forced them to be designed in such a way to prevent mutating table errors. This problem can be demonstrated practically with the following example. It is possible to begin with any of the associations or conflict assignments, but in order to keep this explanation simple, the scenario will begin with the creation of conflicts.

We assume that only two roles are conflicting: the “roaprd” role (r_1) and the “glint” role (r_2). If these roles are not conflicting, they will be able to exist in the same role network. Since permissions are inherited from the role network, one of the roles would have the permissions that would permit fraud. In order to set the conflict, which is denoted as $(r_1, r_2) \in CR$, a SQL statement will insert a record into the CR database table. For these roles, it can be coded as:

INSERT INTO CR VALUES(1, 2);

The trigger will check that this is possible by first checking that they are not already conflicting. The primary key constraint already present for the database table will capture this error sometimes, but the roles may be reversed for the current record and so this check is necessary. It will then check whether the two roles form part of the same role network. If they are, then the following error is raised:

ORA-20208: Cannot insert role conflict - both roles are already part of a role network.
ORA-06512: at "ERBAC03.CRSTATAFTER_TRIG", line 24
ORA-04088: error during execution of trigger 'ERBAC03.CRSTATAFTER_TRIG'

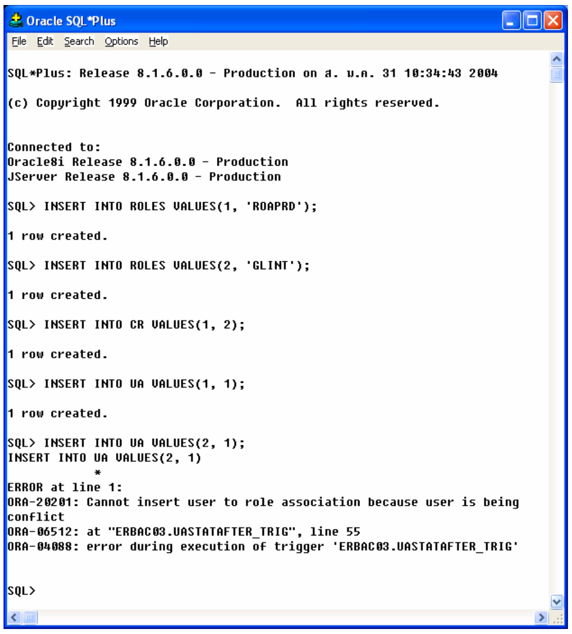
The trigger will proceed to check whether any of the users that are associated with the first role are conflicting with any of the users that are associated with the second role. If there are any, then it will raise the following error:

ORA-20209: Cannot insert role conflict - both roles are already associated to conflicting users.
ORA-06512: at "ERBAC03.CRSTATAFTER_TRIG", line 54
ORA-04088: error during execution of trigger 'ERBAC03.CRSTATAFTER_TRIG'

A trigger will fire when user received the same roles or conflicting roles. Both user and roles are inserted into the user-to-role association database table. When we define both roles for the same user, then the following error is raised:

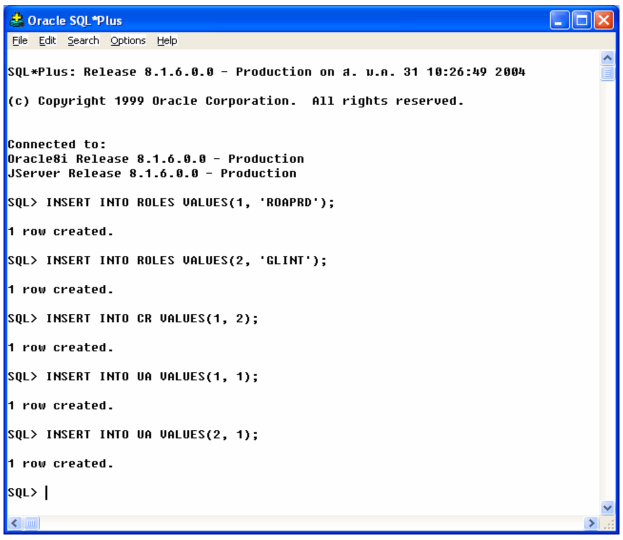
ORA-20201: Cannot insert user to role association because user is being conflict
ORA-06512: at "ERBAC03.UASTATAFTER_TRIG", line 55
ORA-04088: error during execution of trigger 'ERBAC03.UASTATAFTER_TRIG'

From the experimental result of the ERBAC03 model, as depicted in Fig. 5, any conflicting role is not allowed to add its association as a record to the appropriate database table. Therefore, all of the data in the user-to-role table are always correct. In contrast, the experimental result of the RBAC model shows that the resulting association is often incorrect since it allows the association to be added into the database table as shown in Fig. 6.



```
SQL*Plus: Release 8.1.6.0.0 - Production on a. u.n. 31 10:34:43 2004

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to:
Oracle8i Release 8.1.6.0.0 - Production
JServer Release 8.1.6.0.0 - Production

SQL> INSERT INTO ROLES VALUES(1, 'ROAPRD');

1 row created.

SQL> INSERT INTO ROLES VALUES(2, 'GLINT');

1 row created.

SQL> INSERT INTO CR VALUES(1, 2);

1 row created.

SQL> INSERT INTO UA VALUES(1, 1);

1 row created.

SQL> INSERT INTO UA VALUES(2, 1);
INSERT INTO UA VALUES(2, 1)
            *
ERROR at line 1:
ORA-20201: Cannot insert user to role association because user is being
conflict
ORA-06512: at "ERBAC03.UASTATAFTER_TRIG", line 55
ORA-04088: error during execution of trigger 'ERBAC03.UASTATAFTER_TRIG'



SQL>
```

Fig. 5 The result of ERBAC03 model



```
SQL*Plus: Release 8.1.6.0.0 - Production on a. u.n. 31 10:26:49 2004

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to:
Oracle8i Release 8.1.6.0.0 - Production
JServer Release 8.1.6.0.0 - Production

SQL> INSERT INTO ROLES VALUES(1, 'ROAPRD');

1 row created.

SQL> INSERT INTO ROLES VALUES(2, 'GLINT');

1 row created.

SQL> INSERT INTO CR VALUES(1, 2);

1 row created.

SQL> INSERT INTO UA VALUES(1, 1);

1 row created.

SQL> INSERT INTO UA VALUES(2, 1);

1 row created.

SQL> |
```

Fig. 6 The result of RBAC model

6. CONCLUSIONS

In this paper, we propose an enhanced model of RBAC for large enterprises using location hierarchy approach and call ERBAC03 model. It requires the modification of the end-points by introducing three additional entities which consist of locations, jobs, and tasks to make permissions to roles effectively, especially for large enterprises. The definition of entity location is related to roles and reflects large enterprises that have a number of branches. Each branch consists of many similar roles that reflect the model management in distributed aspect. The constraint features used in this model are referred to as the static separation of duty, and able to deal with the ERBAC03 components. In the ERBAC03 model, the association of any conflicting role is not allowed to be added to the target database table. Therefore, all of the data in the user-to-role table are always correct. In contrast, the role association of the RBAC model is often added into the database table incorrectly. In particular, the experimental results for both models can describe the efficiency of data correctness which the proposed ERBAC03 model is more efficiency than the existing RBAC model.

Future work would be undertaken to study how to evolve the conflicting entities administration paradigm to include the dynamic separation of duty concept, including the development of administration tools that should assist system administrators in ensuring that the access control requirements are met. In addition, the delegation problems in the distributed environment will be studied.

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] 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.

[3] 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.

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

[5] Ahn G. J., and Sandhu R., "The RSL99 Language for Role-Based Separation of Duty Constraints," *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pp. 28-29, 1999.