CHAPTER 22

Database Security Integration Using Role-Based Access Control

Sylvia Osborn

*Department of Computer Science, The University of Western Ontario
London, Ontario, Canada, N6A-5B7*

svlvia@csd.uwo.ca

Abstract: Role-based access control provides very flexible mechanisms for managing access control in complex systems. The role graph model of Nyanchama and Osborn is one example of how role administration can be implemented. In previous research, we have shown how the access control information of existing systems can be mapped to a role graph. In this paper, we extend this research by showing how, when two systems are being integrated, their role graphs can also be integrated.

Keywords: role-based access control, database integration, security integration

1. INTRODUCTION

Database interoperability and federated databases have been the subject of much research [13]. The architecture of federated databases requires each participating database system to provide an export schema for the data it is willing to share. Most research in this area has focused on the subsequent integration of the database schemas into a federated schema. Much less work has been done on integrating the access control information in a meaningful way.

Recently, there has been a great deal of activity concerning role-based access control (RBAC), for general system security as well as database security. Database security, like systems security, falls into two broad categories: discretionary access control (DAC), typically found in commercial relational database packages, and mandatory access control (MAC) typified by a security lattice and mandatory labelling of subjects and

objects [2]. Many systems today require access control, which has properties somewhere between these two extremes. RBAC has been shown to be capable of modelling this broad spectrum of security requirements. To date, no one has examined interoperability of RBAC systems.

Research has been done on interoperability of security systems; some of this work is cited here. Bonatti et al. has looked at merging two MAC lattices [1]. [7,4] and [15] have examined the specification of authorisation at both the local level and the federated level. Tari's DOK system is based on constraint checking, and focuses on providing a secure architecture [14]. Jonscher and Dittrich's Argos system has roles, but the focus of the security mechanism is to interpret individual queries in a consistent manner [6].

In this paper, we provide a first step in the integration of systems whose access control is represented by role graphs [8]. Once two systems' role graphs have been integrated, further design of the access control for the federation can proceed by using RBAC administration algorithms to enhance the initial federated role graph. The role graph model is introduced in the next section. Section 3 contains a discussion of the integration of two role graphs. Section 4 presents the algorithm. Two examples, one discretionary and one mandatory, are presented in Section 5, followed by conclusions.

2. ROLE-BASED ACCESS CONTROL

Role-based access control facilitates the management of permissions in systems with large numbers of users, objects and permissions [12]. Roles correspond to job functions, and have names which reflect this.

2.1 The Role Graph Model

The role model of Nyanchama and Osborn [9] emphasises three important entities in modelling access control in a complex system: users, roles, and privileges. A privilege consists of an object and an access mode on the object. Access modes could be as simple as read or write in traditional security models, or very complex methods on objects. *Roles* are sets of privileges. We use the term *group* to refer to sets of users. In our reference model, we emphasise three planes: the user/group plane, the role plane and the privileges plane.

As well as the three basic entities, there are relationships among the entities which provide a rich way of modelling who is allowed to do what to what. Implications can exist among privileges within the privileges plane: for example, an update privilege might imply a read privilege. Users can be assigned to groups, such as "the Physics Department" or "ProjectAlpha",

and the groups can be related to each other in the user/group plane. Role-role relationships can exist: the set of privileges assigned to the secretary role might be a subset of those assigned to the manager role. As well as these relationships, there are assignments of users/groups to roles, and assignments of privileges to roles. Following through from a user to the roles the user is assigned to, and the privileges assigned to these roles, one can then determine the details of access control. At the same time, by being able to create groups and roles that correspond to the application area, the flexibility of management of access control is greatly enhanced.

In our role graph model [9], we model the role-role relationships by an acyclic directed graph we call the *role graph*. In this graph, the nodes represent roles and the edges represent the *is-junior* relationship. A role is represented by a pair (rname, rpset), giving the role name and privilege set respectively. Role r_1 is-junior to role r_2 iff $r_1.rpset \subset r_2.rpset$. We also say that r_2 is senior to r_1 . By specifying that role r_1 is-junior to r_2 , one makes available all the privileges of r_1 to any user or group authorised to role r_2 . We require that two roles have distinct privilege sets, i.e. that the graph be acyclic, so that each role offers a unique set of privileges. We distinguish between the *direct* privileges of a role, which are privileges not found in any of the role's juniors, and *effective* privileges, which are all the privileges contained in a role either as direct privileges or "inherited" from one of the junior roles. The model does not deal with negative privileges. We also include in each role graph two distinguished roles: MaxRole and MinRole. MaxRole contains all the privileges in the system. MinRole.rpset may be empty. It is not necessary for every role to have users or groups assigned to it.

Role graphs have the following *Role Graph Properties*: (1) There is a single MaxRole. (2) There is a single MinRole. (3) The Role Graph is acyclic. (4) There is a path from MinRole to every role r_i . (5) There is a path from every role r_i to MaxRole. (6) For any two roles r_i and r_j , if $r_i.rpset \subset r_j.rpset$, then there must be a path from r_i to r_j . By convention, we draw the role graphs with MaxRole at the top, MinRole at the bottom, all edges going up the page, and without redundant edges.

We have, in our previous research, introduced algorithms for inserting and deleting roles, edges, and privileges [8,9]. With these algorithms, role management is a dynamic process, since roles can be altered quite easily, and the structure of the role graph (i.e. the interactions between the roles) can be changed quite frequently if required. All the algorithms run in time polynomial in the size of the role graph and the size of the privilege sets [9]. In [10] we describe how to map a role graph onto the system tables for a relational database system so that the same permissions described by the role graph can be maintained by the resulting relational database. We also show

how to take the permission tables from a relational database and form an equivalent role graph. Going in this direction gives roles with system-defined names, which can subsequently be given more meaningful names using the role graph tools. We have also shown how to map Unix permissions onto roles [5]. Oracle currently allows roles to be used to help specify access control [3], although their mechanisms are not as rich as our role-graph algorithms. It has even been shown that mandatory access control can be modelled by roles [11]. So, it is reasonable to assume that if two database systems or two other complex systems are to be integrated, their access control portion can be represented by a role graph.

2.2 Some Algorithms

In [9], algorithms are given for a number of role graph operations. Of interest in this paper are the two role insertion algorithms. The first, which we will call Insert1, takes a role graph, a role name, a set of what are to be the direct privileges, the proposed immediate senior and junior roles. The algorithm creates edges from these proposed juniors to the new role, edges from the new role to the proposed seniors, computes the effective privileges of the new role from the proposed direct privileges and the juniors' effective privileges, and then reestablishes the role graph properties. The algorithm aborts if a cycle would be created (which would create redundant roles), leaving the graph unchanged.

The second role insertion algorithm, which we will call Insert2, takes a role graph, a role name, a set of what are to be the effective privileges, and inserts the role into the graph. It generates all edges to juniors and seniors, as determined by comparing effective privileges, and reestablishes the role graph properties. The algorithm aborts if the new role duplicates an existing role.

Both of these algorithms are polynomial in the size of the role graph and the number of privileges in the privilege sets [9].

3. INTEGRATING ACCESS CONTROL

Suppose we have two database systems each of whose access control is described by a role graph. According to our reference model, we have three different kinds of information: the users, the roles, and the privileges. Both users and roles are represented by names. For the users, we assume that either user names are universal or that a one-to-one mapping can be constructed by the security administrator in a deterministic way. We ignore user groups here, and just concentrate on individual user-role assignments.

The role names involve the use of natural language with all its ambiguities. For this reason, we propose that if two systems have matching role names, they will be presented to a human being, whom we shall call the security administrator or SA, to verify that they are intended to be the same. As well, as described below, we will present roles which seem to have similar privileges but different names to the SA to verify whether they should be considered to be the same role, in which case they can be merged. This decision will always be made by the SA.

Privileges are composed of an object name and an operator name. In integrating two systems, there may be underlying data objects which should be regarded as "the same". We assume that database integration techniques have been used to decide which objects from systems 1 and 2 should be regarded as the same. We make a similar assumption concerning operator names. If the operations are applications or packages, we assume that database integration techniques can be used to decide, for example, the "hire" application in one system is the same as the "create-new-employee" application in the other. In the worst case, every program and application can be broken down into a set of reads and writes on individual database objects. So if the integration of the database objects has been done, and the code for the applications analysed, complex operations can be compared to see if they represent the same privilege.

For the sake of the following discussion, if systems 1 and 2 are to be integrated, then for any two database objects or operations deemed to be the same, the object name or operation name in system 2 has been mapped onto that of system 1. The following notation will be used: system 1, $S_1(U_1, R_1, P_1)$ and system 2, $S_2(U_2, R_2, P_2)$ consist of three sets: users (U_1 and U_2), roles (R_1 and R_2), and privileges (P_1 and P_2). As well as these, there are two role graphs: $RG_1(R_1, \rightarrow_1)$ and $RG_2(R_2, \rightarrow_2)$. In turn, each P_i consists of object-operation pairs. There are 8 cases to consider:

1. U_1 and U_2 are disjoint; R_1 and R_2 are disjoint; P_1 and P_2 are disjoint
2. U_1 and U_2 are disjoint; R_1 and R_2 are disjoint; P_1 and P_2 not disjoint
3. U_1 and U_2 are disjoint; R_1 and R_2 not disjoint; P_1 and P_2 are disjoint
4. U_1 and U_2 are disjoint; R_1 and R_2 not disjoint; P_1 and P_2 not disjoint
5. U_1 and U_2 not disjoint; R_1 and R_2 are disjoint; P_1 and P_2 are disjoint
6. U_1 and U_2 not disjoint; R_1 and R_2 are disjoint; P_1 and P_2 not disjoint
7. U_1 and U_2 not disjoint; R_1 and R_2 not disjoint; P_1 and P_2 are disjoint
8. U_1 and U_2 not disjoint; R_1 and R_2 not disjoint; P_1 and P_2 not disjoint

We will first discuss isolated cases, and then the whole process.

Consider first cases 1 and 5. Since both the role sets and the privilege sets are disjoint, the role graphs can be combined as shown in Figure 1.

Once the new role graph is created, users in U_1 remain assigned to all the roles they were assigned to in S_1 , and similarly, users in U_2 can remain assigned to all the roles they were assigned to in S_2 . This user assignment is the same whether or not U_1 and U_2 are disjoint. After the two role graphs are merged, the role graph algorithms can be used to fine-tune the result. For example, MaxRole and MinRole from the original graphs can be deleted if they have no users assigned to them.

Figure 1. Disjoint Roles and Privilege Sets

Now consider cases 2 and 6. In both cases the privilege sets are not disjoint (they have non-empty intersection). First, consider the result graph to be RG_1 . For each role r_i in R_2 , insert r_i with its rname and effective privileges into RG_1 using Insert2. Insert2 may report that a duplicate role exists. Since duplicates are not allowed in our role graphs, this insertion will be rejected. I.e., the two roles have equal effective privilege sets, and we can treat this as merging the two roles. Role r_i should be mapped to the role that it is found to be equivalent to. The system must keep track of these mappings. After the role graph merging is complete, the usual role graph algorithms can be used to rename roles, etc.

Once the role graphs have been merged, the user assignments have to be looked at. If U_1 and U_2 are disjoint, then the users from U_1 are assigned to the roles they were assigned to in RG_1 , and the users from U_2 are assigned to the roles they were assigned to in RG_2 or whatever they got mapped into. If U_1 and U_2 are not disjoint, then we begin by assigning the users from U_1 to their original roles. For each user in U_2 , a situation can arise as shown in Figure 2. Suppose a user A in U_1 was assigned to role r_3 in S_1 and to r_8 in S_2 . Further suppose that roles r_2 and r_9 are merged. According to S_2 , this user should not be assigned to the privileges in r_9 . However, according to S_1 , this user was given these privileges. The merging makes it appear from S_2 's point of view that user A would receive new privileges (even though the

integration has equated whatever privileges make up r2 and r9). In any case, the SA should be notified so that a decision can be made regarding this user's role assignments in the integrated system.

Figure 2. Merging two roles

Now consider cases 3 and 7: some duplicate role names, but no duplicate privileges. In particular, consider one role name which corresponds to a role in both graphs, but whose privilege sets are disjoint. This situation has to be handled by the SA. There are two possibilities: (1) one of the (duplicate) role names should be changed. If this is done, we are back to cases 1 and 5. (2) the SA may want to keep this common role. An example of this is an accounting role in two systems, say payroll and purchasing, which have no duplicate objects and therefore no duplicate privileges. Let the two duplicate roles be denoted by s from S_1 and r from S_2 and let the common name be n . To keep a role with name n , insert a new role with n as its name, s and r as its immediate juniors. The new role has no direct privileges. The immediate seniors of the new role will be established by the algorithm. Roles s and r need to be renamed (to some s' and r'). The new role, then, has the union of the privileges of the roles, which had the same name in the two original systems. The new role needs to be flagged for user assignment - users assigned to the original r and s may or may not be assigned to the new role instead of to s or r - this decision should be made by the SA.

Figure 3. Common names with (a) disjoint privileges, (b) common privileges

Finally, cases 4 and 8: (some) duplicate role names, and (some) duplicate privilege names. Two roles with distinct names and duplicate privileges were handled by cases 2 and 6. Two roles with duplicate names and completely disjoint privileges were handled by cases 3 and 7. Completely duplicate privilege sets with duplicate role names can just be merged. The interesting case is if we have duplicate role names and some overlap of the privileges. Suppose we have two such roles, s from $\mathbf{S}_1$, and r from $\mathbf{S}_2$. What we suggest here is to create a new role with the common name n , which contains all the privileges of the intersection of the privileges of s and r , rename s and r (to some s' and r'), insert the new role with s' and r' as its immediate seniors. This situation is shown in Figure 3. Each such role should be assigned to all users who were assigned to s in $\mathbf{S}_1$ and those assigned to r in $\mathbf{S}_2$. The new role will be implicitly assigned to all such users if the users assigned to s (r) in the original role graph are assigned to s' (r') in the merged graph.

The overall algorithm proceeds, then, by taking one of the input role graphs and inserting the roles from the second graph one at a time into it, taking the above cases into account.

4. THE ALGORITHM

In the algorithm, we merge all the cases above by starting with one of the role graphs, and inserting the roles from the second graph into it. One problem that arises is that in the cases shown in Figure 3(a) and (b), when the new role is inserted, not all of the seniors or juniors of the original role from $\mathbf{RG}_2$ may have been inserted yet. This is handled by noting that the graph itself has redundant information - the effective privileges can be deduced from the edges, and vice versa. For Insert 1, the privileges given with the new role are what are believed to be the direct privileges, but if they are found to be redundant (present in an immediate junior), the role insertion algorithm will clean this up. So, if, when we use Insert 1, we give it the effective privileges as what we think will be the direct privileges, when all the roles have been inserted, and the graph edges and paths cleaned up, the required juniors or seniors will be connected as expected. The algorithm is given next.

The output of the algorithm is a merged role graph that retains all of the information contained in the original two role graphs. Specifically all of the original roles are present, possibly with new names given by the mapping m , and each of these roles retains its original effective privilege set. Furthermore, any roles originally junior or senior to any role are still in this relationship in the new graph.

Algorithm GraphMerge

Inputs: $S_1(U_1, R_1, P_1)$, $S_2(U_2, R_2, P_2)$, //the two systems,
 $RG_1(R_1, \rightarrow_1)$, $RG_2(R_2, \rightarrow_2)$, //the two role graphs,
 URA_1 , URA_2 // the user-role assignments.

Output: $NewRG(R, \rightarrow)$ // a single role graph
 $m(R_1)$, $m(R_2)$ // mappings of original roles onto R .

Method:

```

NewRG =  $RG_1$ ;
for each role  $r$  in  $R_2$ 
  if name( $r$ ) any role name in NewRG (cases 1 and 5)
  then Insert2(NewRG, name( $r$ ), effective( $r$ ));
    if duplicates, then  $m(r) = \text{name}(\text{duplicate})$ ;
    //(cases 2 and 6)
  else (role name equals to a role  $s$  in NewRG)
    if effective( $r$ ) effective( $s$ ) //(cases 3 and 7)
    ask SA if  $r$  should be renamed;
    if yes then  $m(r) = \text{newname}$ ;
    Insert2(NewRG,  $m(r)$ , effective( $r$ ))
    else  $n = \text{name}(r)$ ;
     $m(r) = r'$ ;
     $m(s) = s''$ ;
    Insert2(NewRG, name( $r'$ ), effective( $r$ ));
    Insert1(NewRG,  $n$ , effective( $r$ )  $\cup$  effective( $s$ ),  $\phi$ ,
      { $s'' \cup r'$ });
    else //(cases 4 and 8)
    if effective( $r$ ) = effective( $s$ )
    then  $m(r) = s$  //merge the two roles)
    else  $n = \text{name}(r)$ ;
     $m(r) = r'$ ;
     $m(s) = s''$ ;
    Insert2(NewRG, name( $r'$ ), effective( $r$ ));
    Insert1(NewRG,  $n$ , effective( $r$ )  $\cap$  effective( $s$ ),
      { $s'' \cup r'$ },  $\phi$ );
for each  $u$  in  $U_1$  //(now look at user-role assignment)
  for each role  $r$  that  $u$  is assigned to in  $URA_1$ 
    assign  $u$  to  $m(r)$ ;
for each  $u$  in  $U_2$ 
  for each role  $r$  that  $u$  is assigned to in  $URA_2$ 
    if  $u$  is assigned to role  $s$  in NewRG senior to
       $m(r)$  or junior to  $m(r)$ 
    then notify the SA for intervention
    else assign  $u$  to  $m(r)$ ;
for each new role created by the algorithm
  notify the SA to specify user assignments;

```

Figure 4. Role Graph Integration Algorithm

5. AN EXAMPLE

We will illustrate the process by presenting an example showing two traditional relational databases using standard (discretionary) access control. System 1 is an Oracle database with relations Staff and Accounts. It has an Admin role with rights: read Staff, update Accounts and read Accounts. A Clerk role has rights: read Accounts and insert Accounts. User U1 has been granted the right to update Staff, and to perform role Admin. User U2 has been granted role Admin and users U3 and U4 have been granted role Clerk.

System 2 is a relational database whose access control has been specified in a traditional relational database way, without roles. The method given in [10] has been used to derive a role graph from the permission tables. As a result, the roles have generic names.

Figure 5(a,b). Input role graphs

The two role graphs are shown in Figure 5(a) and (b). We have shown the users in brackets beside the role names, and the effective privileges (using rA for read Accounts, iS for insert to Staff, etc.) below the role names. Both systems have a database administrator assigned to MaxRole.

When MaxRole from **RG**₂ is inserted into NewRG, the SA will be asked if one of them should be renamed. If the answer is yes, the role graph shown in Figure 6 results. If the answer is no, the MaxRole from **RG**₂ is inserted, and then the algorithm will try to insert a role with the union of the privileges of the two MaxRoles. This equals the privileges of MaxRole2, so this second insert aborts. The only difference is the ultimate role names (which

can be changed later). Note that the Admin role from **RG₁** and role R4 from **RG₂** are merged.

Figure 6 shows the default user assignments. If U1 happens to be Sue from System2, the SA will be notified and can decide whether or not she should be assigned to Role1 or to R4/Admin in the integrated system.

Figure 6. Merged role graph

6. SUMMARY

We have shown how, using our role graph algorithms, we can devise an algorithm for merging two role graphs. This algorithm is particularly useful in integrating the security systems of two databases, which are to be integrated. Because of the wide applicability of role-based access control, this technique can be used in a wide variety of integration activities.

References

[1] P.A. Bonatti, M.L. Sapino, and V.S Subrahmanian. Merging heterogeneous security orderings. In Martella Bertino, Kurth and Montolivo, editors, *Computer Security - ESORICS96, LNCS1146*, pages 183-197. Springer-Verlag, 1996.

[2] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1994.

[3] Oracle Corporation. *Oracle Version 7, Chapters 6 & 7*. http://www.oracle.com/security/html/chap_6.html. 1995.

- [4] S. di Vimercati and P. Samarati. An authorization model for federated systems. In Martella Bertino, Kurth and Montolivo, editors, *ESORICS96, LNCS 1146*, pages 99-117. Springer-Verlag, 1996.
- [5] L. Hua and S. Osborn. Modeling unix access control with a role graph. In *Proc. International Conference on Computers and Information*, June 1998.
- [6] D. Jonscher and K.R. Dittrich. Argos - a configurable access control system for interoperable environments. In *Database Security IX, Status and Prospects*.
- [7] D. Jonscher and K.R. Dittrich. An approach for building secure database federations. In *Proceedings of 20th VLDB Conference*, pages 24-35, 1994.
- [8] M. Nyanchama and S. L. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgenstem, and C. E. Landwehr, editors, *Database Security, VIII, Status and Prospects WG11.3 Working Conference on Database Security*, pages 37-56. North-Holland, 1994.
- [9] M. Nyanchama and S. L. Osborn. The role graph model and conflict of interest. *ACM TISSEC*, 2(1):3-33, 1999.
- [10] S.L. Osborn, L.K. Reid, and G.J. Wesson. On the interaction between role based access control and relational databases. In *Database Security X, Status and Prospects*.
- [11] R. Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Computer Security - ESORICS 96, LNCS1146*, pages 65-79. Springer Verlag, 1996.
- [12] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29:38-47, Feb. 1996.
- [13] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3): 183-236, Sept. 1990.
- [14] Z. Tari. Designing security agents for the dok federated system. In Lin and Qian, editors, *Database Security XI*, pages 35-59. Chapman & Hall, 1997.
- [15] M. Templeton, E. Lund, and P. Ward. Pragmatics of access control in mermaid. *IEEE-CS TCData Engineering*, pages 33-38, Sept. 1987.