Chapter 2

PRIVILEGE ADMINISTRATION FOR THE ROLE GRAPH MODEL

Cecilia Ionita and Sylvia Osborn

Abstract The role graph model for role-based access control has been introduced in a number of previous papers. In these previous presentations of the role graph model, it is assumed that when privilege p is present in a role, all privileges that might be implied by p are also present in the role. This paper gives revised algorithms to ensure that this is done, using a model for implication of permissions originally developed for object-oriented databases.

Keywords: Role-based access control, privilege administration, propagation of permissions

1. Introduction

The role graph model for role-based access control was first introduced in [2]. The role graph model considers users/groups, the role hierarchy and privileges to be three separate spheres of discussion, which have been referred to as three planes. On the user/group plane, we look at set of users (which we call groups), and the relationships between them [4]. On the role plane, we consider the role hierarchy (which we call the role graph) and its algorithms which permit various operations on the role graph [3]. On the privileges plane, we consider privileges, which are (object, operation) or (object, access mode) pairs. In these previous papers, it has always been assumed that when a privilege p is added to a role, any privileges implied by p are also added to the role. In this paper, we will present an algorithm to ensure that this is done, using a model for implications of permissions originally published in the context of object-oriented databases (OODB) [5]. Note that we use the words permission and privilege interchangeably.

In the next section we review the model of Rabitti et al. for object-oriented databases. Then, in Section 3, we briefly review the role graph model. Section 4 presents revised algorithms for privilege addition and role insertion. Section 5 contains some conclusions.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35697-6_26

2. The OODB Work

The model presented by Rabitti et al. [5] is a discretionary access control model for OODBs. It models an authorization as a triple:

$f : S \times O \times A \rightarrow$(True, False)

where S represents the set of subjects, O the set of objects and A the possible authorizations in a system. For each of S, O and A, they provide a lattice. The lattice of subjects is just a role graph, which they call a role lattice. We should note that the authors never justify why they require lattices for these – directed acyclic graphs are all that is really required since the presence of a unique least upper bound for each of these hierarchies or graphs is never justified. We shall call these graphs, but the correspondence to what are called lattices in [5] should be clear.

The role graph presented in [5] is very similar to the role graphs of the role graph model, having a super-user role which corresponds to MaxRole in the role graph model, and a minimum role which corresponds to our MinRole. They even include the comment that the super-user role does not always need to be assigned, because it makes available to any user so assigned all the privileges available in the system.

The other two graphs will form the basis for our discussion in the rest of this paper. They are graphs which model relationships in A and O, and are, here, called the authorization type graph and the authorization object graph respectively.

The authorization type graph (ATG) allows one to model implications among authorizations or operations in a system. For example, one might have a system in which the write permission on an object o always implies the read permission on o, when these operations make up the A component of an authorization. In this case, the edge write $\rightarrow$ read would be in the authorization type graph. We will call this relationship *implies.* Another example of an *implies* relationship in an object-oriented environment is that the permission to read an object would *imply* the permission to read the class definition for its class, so that one would know how to interpret the object. We will assume that for the system to be protected by a role graph security model, the security designer will provide the authorization type graph. If there are no implications between authorizations, then this graph will consist only of the nodes, which each represent an operation (this would be true if modeling Unix, for example, in which the three permissions on a file, read, write and execute, are usually considered to be completely independent).

Figure 1 shows an example authorization type graph for a relational database and its operations. Note that this *implies* relationship means that for the same object, e.g. the same relation, grant-update implies update, etc.

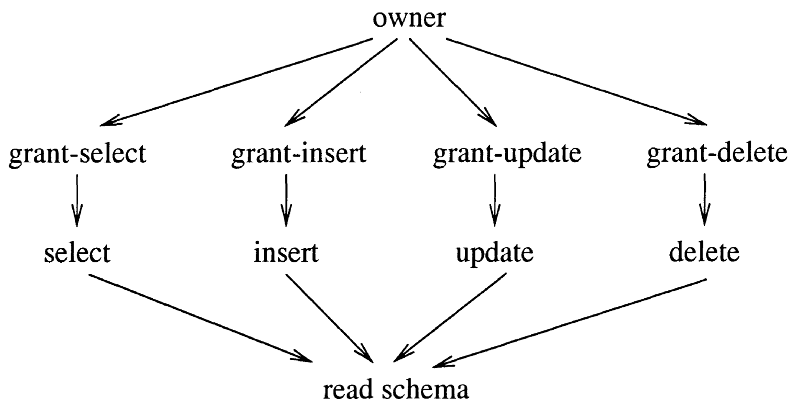

Figure 1. An example of an authorization type graph for a relational database.

The authorization object graph (AOG) in [5] is based on the authorization object schema. Since the authors are discussing a situation for an OODB, the OODB has a schema or class hierarchy, which defines the structure of the objects. As well as the schema, an OODB has a way of representing object containers or collections of objects, which varies from one system to another: some systems provide one container per class to hold all the instances of a class, whereas others provide the ability to create arbitrary named sets of objects. For a Student class, for example, the former system would put all Student instances in a single container, whereas the latter would allow one to build different collections, say MyClass, AllUndergraduates, etc., which could of course overlap. The reason this is necessary is that an OODB is a *database*, and one expresses queries in a database. The containers provide the targets for queries. Since our discussion in this paper is attempting to be for a general security model, not one specifically for an OODB, we will ignore the authorization object schema, and simply build the authorization object graph directly. The AOG contains individual objects or containers. The nodes in this graph refer to any granularity of object on which one can carry out operations, and are thus the granularities about which one can express authorizations.

The authorization object graph models implications which arise because of object containment. For example, in a relational database, the permission to read a relation implies the ability to read the tuples in the relation. It would also imply the permission to read any indexes built on the relation. Thus, the authorization object graph for a relational database might look like Figure 1, where F1, S1, etc. represent individual tuples.

We will refer to the relationship defined by an edge in an authorization object graph as *contains*.

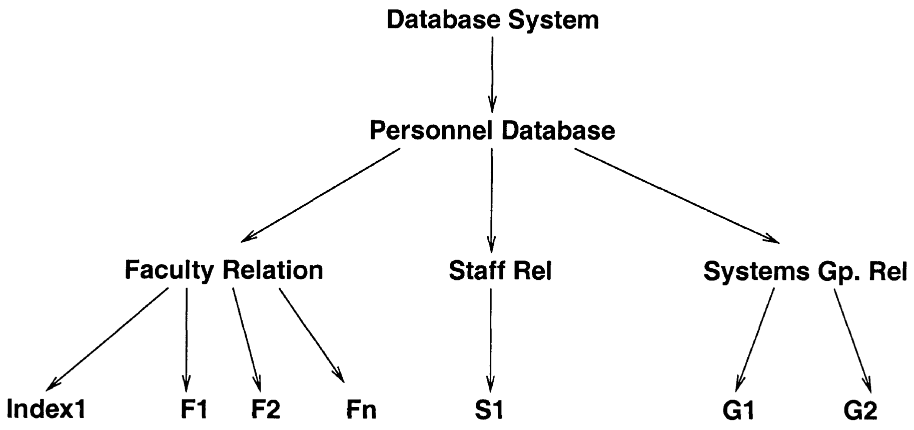

Figure 2. An authorization object graph for a relational database.

As well as the three graphs, the model of [5] includes a way of distinguishing how authorizations propagate in the AOG. In our relational example, select (which would be called read in most systems) would propagate down, so that having select permission on a relation implies having read permission on all the tuples. Similarly select on a database would imply select on all the relations. Some permissions do not propagate, such as grant-select, which might be issued on a specific relation but should not propagate anywhere. Something like read schema, would propagate up, so that permission to read the schema of relation Faculty would imply permission to read the schema of the Personnel Database. The authorization types are thus partitioned into three sets: *a.up, a.down* and *a.nil* depending on how they propagate in the **authorization object graph**. This must also be specified by the security designer as part of the specification of such a system.

It is quite common in relational packages to have very different permission sets for relations than for columns or indexes – i.e. some authorizations do not make sense on some object types. In our example, read schema makes no sense on individual tuples. Insert makes no sense on the Database System granule. The model of [5] also includes an authorization association matrix (AAM) which indicates which (object type, authorization type) pairs can be specified. Propagation would stop if a disallowed pair is encountered.

Finally, the Rabitti et al. model has both negative and positive permission, and strong and weak permissions. Our model does not have negative permissions, as we prefer to deal with these situations using constraints. We also do not have strong and weak permissions.

3. The Role Graph Model

The role graph model [2, 3] models the role-role relationship with an acyclic, directed graph in which the nodes represent roles, and the directed edges represent the *is-junior* relationship. Roles are represented by a pair (rname, pset) where rname is a unique role name, and pset is the set of privileges assigned to the role. Role r_i *is-junior* to role r_j iff r_i's pset $\subset$ r_j's pset. Conversely, we say that r_j is *senior* to r_i. We also distinguish between the *direct* and *effective* privilege sets of a role. The direct privileges are those assigned directly to the role, and not inherited from any junior roles. The effective privileges consist of the direct privileges unioned with the effective privileges of all junior roles. The is-junior relationship is based on the subset relationship between the effective privilege sets of two roles. In order to ensure that the graph remains acyclic, no two roles are allowed to have equal privilege sets.

In [3], a number of algorithms are given to manipulate role graphs. There are two algorithms for role insertion, one to delete a role, an algorithm to add a privilege to a role, one to delete a privilege, one to add an edge and one to delete an edge. Each of the algorithms restores the graph to a canonical form, which is given by the following *role graph properties*:

- There is a single MaxRole.
- There is a single MinRole.
- The Role Graph is acyclic.
- There is a path from MinRole to every role r_i .
- There is a path from every role r_i to MaxRole.
- 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.

We have already discussed the importance of acyclicity. All of the algorithms abort if a cycle would be created, leaving the graph unchanged. The last three role graph properties ensure that the graph has enough edges. When displaying the graph, we remove redundant edges, put MaxRole at the top, MinRole at the bottom and show the is-junior edges going up the page. The algorithms all have run time which is polynomial in the number of roles, number of edges and size of the privilege sets.

The two role insertion algorithms differ in their inputs. In the first, the new role, its proposed direct privileges and proposed immediate junior and senior roles are given. The algorithm inserts the new role, adds the appropriate edges, and then adjusts the direct and effective privileges of the new role and all seniors on a path from the new role to MaxRole. Some privileges which were given as direct may now be just in the effective privilege set of a senior role. Similarly some privileges which were thought to be direct in the new role may be inherited from a junior role and thus get moved to the effective set. There may be other juniors and seniors defined by the r_i's pset $\subset$ r_j's pset relationship which were not given, for which edges or paths will be added when the role graph properties are restored.

In the second role insertion algorithm, the new role and its proposed effective privileges are given. The algorithm determines its junior and senior roles, and direct privileges.

These algorithms assumed that privileges implied by other privileges somehow appeared in the appropriate roles. We will now show the revised affected algorithms in which permission/privilege implications are explicitly handled. The algorithms which are affected by this are the privilege insertion and role insertion algorithms. We will discuss these changes in the next section.

4. Revised Algorithms

The main task in revising the algorithms is to improve the privilege addition algorithm. After that, we will look at changes to the two role addition algorithms. The original privilege addition algorithm from [3] is given in Figure 3. The algorithms in [3] also considered various forms of conflict of interest. In this paper we just consider conflicts in a very general way, assuming that prohibited privilege-privilege or role-privilege conflicts are represented by a set C.

In the old algorithms, the statement "Reestablish role graph properties" meant that necessary edges would be added. This is done by comparing every pair of effective privilege sets, adding an edge $r_i \rightarrow r_j$ whenever effective(r_i) $\subset$ effective(r_j). Then redundant edges are removed using an algorithm from [1]. After that, the step "Adjust Direct and Effective of affected roles" would use any of these new edges to propagate new privileges to the effective set of all roles senior in the graph, and also possibly remove privileges from a direct set.

We now look at the revised algorithm which will take into account all of the ATG *implies* relationships, the *contains* relationships of the AOG, a.up and a.down (authorizations in a.nil do not propagate) as well as the AAM. To simplify the algorithm, we assume that ATG and AOG are given as their transitive closure. The role graph is represented by RG $=\langle \mathcal{R}, \rightarrow \rangle$ where $\mathcal{R}$ is the set of roles and $\rightarrow$ represents the is-junior relationship (i.e. the edges in the role graph). In the new algorithm, we assume there is information about constraints, which we just refer to as C.

The revised algorithm is given in Figure 4. This algorithm has a structure which repeatedly checks for new implications, since a new privilege (a, o) may

```
Algorithm: PrivilegeAddition(RG, $n$, $p$, P-Conflicts)

Input: RG =$\langle \mathcal{R}, \rightarrow \rangle$ (the role graph),
    $n$, /* the role to which a privilege is to be added */
    $p$, /* the privilege to be added to role $r$ */
    P-Conflicts. /* set of pairs of conflicting privileges */
Output: The role graph with privilege $p$ added to role $r$,
and the role graph properties intact,
        or RG unchanged if an error was detected.
Method:
    Var $r$: role;
    Begin
        If $p \in$ Effective($n$)              /* $p$ is already in $n$ – do nothing */
            Then return;
        Direct($n$) := Direct($n$) $\cup$ $p$;      /* add $p$ to Direct privileges of $n$ */
        Effective($n$) := Effective($n$) $\cup$ $p$/* and to Effective of $n$ */
        /* Reestablish role graph properties */
        /* Adjust Direct and Effective of affected roles */
        If RG has any cycles              /* Detect cycles */
            then abort (message: Graph is not acyclic);

/* Conflict of Interest Detection */
        For every role $r \in \{n\} \cup \mathcal{R}$ - MaxRole Do
            If Effective($r$) contains a pair of privileges which is in P-Conflicts
                Then  abort (message: Privilege addition creates a conflict);
    end.
```

Figure 3. Privilege insertion algorithm from [3].

create new implications either from the ATG or from the AOG. New privileges are added to a temporary set SetToAdd until there are no more changes. Then all of these privileges are propagated to senior roles in the role graph, at which time we check to see if any cycles would be created. Finally we clean up the direct and effective privileges of r and any other roles affected, and correct the edges to reflect the new effective privilege sets of all the roles.

In [3], the role insertion algorithms also ignored possible implications among privileges. In order to modify these algorithms to take into account the improvements, we really just have to arrange for the privilege insertion algorithm to be called when a new role is created.

Recall that the first role insertion algorithm takes a role with its proposed direct privileges, proposed juniors and seniors, and adds this role to the graph if no cycles would be created. The change we need to make is to put the new role in the graph connected to its proposed immediate juniors and seniors, and then use the privilege insertion algorithm to insert the privileges given one at a time, so that all the implied privileges also get included. Since the privilege

```
Algorithm: RevisedPrivAdd(RG, $r$, $p$, ATG, AOG, a.up, a.down, AAM, C)

Input: RG =$\langle\mathcal{R}, \rightarrow\rangle$ (the role graph),
    $r$, /* the role to which a privilege is to be added */
    $p$, /* the privilege also denoted by $(a, o)$ to be added to role $r$ */
    ATG, /* the authorization type graph */
    AOG, /* the authorization object graph */
    a.up and a.down /* two disjoint sets of authorization types */
    AAM, /* the authorization association matrix */
    C /* the constraints on role-permission assignment */
Output: The role graph with privilege $p$ and all privileges implied by it
        added to role $r$, and the role graph properties intact,
        or RG unchanged if an error was detected.
Method:
    begin
        if $p \in$ Effective($r$)    /* $p$ is already in $r$ – do nothing */
            then return;
        SetToAdd $\leftarrow \{(a, o)\}$
        while changes are made to SetToAdd do
            $(a_k, o_k) \leftarrow$ a privilege from SetToAdd
            for all $a_i \in \{a_k\} \cup \{a_j | a_k$ implies $a_j\}$    /* using the ATG */
                if $(a_i, o_k) \notin$ SetToAdd and $(a_i, o_k)$ is allowed by AAM and
                        adding $(a_i, o_k)$ to $r$ is not prevented by C
                    then SetToAdd $\leftarrow$ SetToAdd $\cup \{(a_i, o_k)\}$
            if $a_k \in$ a.down          /* using the AOG */
                then for all $o_i \in \{o_k\} \cup \{o_j | o_k$ contains $o_j\}$
                    if $(a_k, o_i) \notin$ SetToAdd and $(a_k, o_i)$ is allowed by AAM and
                            adding $(a_k, o_i)$ to $r$ is not prevented by C
                        then SetToAdd $\leftarrow$ SetToAdd $\cup \{(a_k, o_i)\}$
            if $a_k \in$ a.up
                then for all $o_i \in \{o_k\} \cup \{o_j | o_j$ contains $o_k\}$
                    if $(a_k, o_i) \notin$ SetToAdd and $(a_k, o_i)$ is allowed by AAM and
                            adding $(a_k, o_i)$ to $r$ is not prevented by C
                        then SetToAdd $\leftarrow$ SetToAdd $\cup \{(a_k, o_i)\}$
        end while
        Direct($r$) $\leftarrow$ Direct($r$) $\cup$ SetToAdd
        Effective($r$) $\leftarrow$ Effective($r$) $\cup$ SetToAdd
        for all $r_i$ in $\mathcal{R}$ such that $r$ is-junior $r_i$
            Effective($r_i$) $\leftarrow$ Effective($r_i$) $\cup$ SetToAdd
        if for any $r_i, r_j$, Effective($r_i$) = Effective($r_j$)
            abort /* duplicate roles create a cycle */
        Reestablish the role graph properties
    end.
```

Figure 4. Revised privilege insertion algorithm.

insertion algorithm reestablishes the role graph properties, we do not need to repeat this step in the new version of role insert 1. The revised algorithm is given in Figure 5.

```
Algorithm: RevisedRoleAddition1(RG, $n$, Seniors, Juniors, P-Conflicts)

Input: RG =$\langle \mathcal{R}, \rightarrow \rangle$ /* the role graph */
  $n$, /*the new role to be added (role name along with its proposed direct privilege
    set) */
  Seniors, /* proposed immediate Seniors for $n$ */
  Juniors, /* proposed immediate Juniors for $n$ */
  C /* role-permission and role-role constraints */
Output: The role graph with $n$ added and role graph properties intact,
    or RG unchanged if an error was detected.
Method:
  begin
    $\mathcal{R} \leftarrow \mathcal{R} \cup$ {role $n$ with empty privilege set}
    for all $r_s \in$ Seniors add the edge $n \rightarrow r_s$
    for all $r_j \in$ Juniors add the edge $r_j \rightarrow n$
    if RG has any cycles
      then abort
    for all $r_j \in$ immediate Juniors of $n$
      Effective($n$) $\leftarrow$ Effective($n$) $\cup$ Effective($r_j$)
    for all $p \in$ proposed Direct($n$)
      RevisedPrivAdd(RG, $n$, $p$, ATG, AOG, a.up, a.down, AAM, C)
  end.
```

Figure 5. Revised Insert 1 algorithm.

The second role insertion algorithm takes as parameters just the new role and its proposed effective privileges; the algorithm determines how the new role is connected to juniors and seniors in the graph. In the revision, this is done first, using the given effective privileges and the relationship between the new role's effective set with other roles' effective sets to insert edges in the graph. Then the starting value for the new role's direct privilege set is initialized to the given effective set minus any privileges in any junior roles. Each of these direct privileges is then explicitly added to the new role using the new privilege insertion algorithm, which in turn will add any implied privileges to the new role and all of its seniors. By calling the new privilege insertion algorithm at the end of this role insertion algorithm, we also know that the role graph properties will be restored. The revised second role addition algorithm is given in Figure 6.

The running time of the new algorithms is increased over that of the original algorithms, mainly because of the **while** loop in the revised Privilege Insertion Algorithm in Figure 4. This **while** loop adds one new permission for every combination of (a, o) that can be derived by the two acyclic graphs ATG and

```
Algorithm: RevisedRoleAddition2(RG, $n$, C)

Input: RG =$\langle \mathcal{R}, \rightarrow \rangle$ /* the role graph */
    $n$, /*the new role to be added (role name with proposed effective privilege
        set) */
    C /* role-permission and role-role constraints */
Output: The role graph with $n$ added and role graph properties intact,
        or RG unchanged if an error was detected.
Method:
    begin
        for all $r \in \mathcal{R}$
            if Effective($n$) = Effective($r$)
                then abort     /* this role is already in the graph*/
        $\mathcal{R} := \mathcal{R} \cup \{n\}$
        for all $r \in \mathcal{R}$                /* Create edges to seniors */
            if Effective($n$) $\subset$ Effective($r$)
                then add the edge $n \rightarrow r$
        for all $r \in \mathcal{R}$                /* Create edges from juniors */
            if Effective($r$) $\subset$ Effective($n$)
                then add the edge $r \rightarrow n$
        Direct($n$) $\leftarrow$ Effective($n$)     /*correct Direct and Effective of $n$*/
        for all $r_j$ junior to $n$
            Direct($n$) $\leftarrow$Direct($n$) - Effective($r_j$)
        for all $p \in$ Direct($n$)
            RevisedPrivAdd(RG, $n$, $p$, ATG, AOG, a.up, a.down, AAM, C)
    end.
```

Figure 6. Revised Insert 2 algorithm.

AOG. In general, the original a and o will not be connected in their respective graphs in a position that makes all the other nodes reachable. However, the upper bound on the number of privileges derivable from a given (a, o) is (number of nodes in ATG) * (number of nodes in AOG), which are, respectively, the number of authorizations and the number of objects in the system.

5. Conclusions

In this paper we have presented a new privilege insertion algorithm for the role graph model, which takes into account any implications given among authorization types or object granules that the security designer can specify. We looked only at the privilege insertion algorithm and the two versions of role insertion. The role graph model also includes role and permission deletion, as well as edge insertion and deletion algorithms. It is fairly clear that edge insertion and deletion can work simply with the effective privileges already determined and stored in their respective roles. One might want to "unwind" some of the privilege implications when privilege deletion is performed. This could be done, but perhaps one of the implied privileges was separately granted to the role. In order to really do this, the tradeoff is the cost of keeping track, for each privilege, of whether it was deduced by the algorithms or directly given by the security designer. At the moment, when a role is deleted, the designer is given the choice of deleting the direct privileges with the role, or propagating them to the immediate seniors of the role being deleted. Coupled with this would be the determination of whether the privileges are original ones or derived ones.

We also saw that the number of implied privileges can be proportional to the size of the ATG, times the size of the AOG. In practical applications, the ATG is unlikely to be very large, but the AOG can be extremely large. The ATG is unlikely to have very many edges, and in many situations the AOG might be fairly flat: an application based on a relational database, for example, will probably have just the depth of granules shown in Figure 1. In managing operating systems security, the ATG likely has no edges, and the AOG also probably treats files as distinct objects, just as UNIX does, and thus would not have any edges either.

This addition to the role graph model provides a succinct way of describing implications among privileges, using the only two dimensions of a privilege, namely the object and the authorization which together make up a privilege.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] A. V. Aho, M. R. Garey and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal of Computing*, 1(2):131–137, 1972.

[2] M. Nyanchama and S. L. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgenstern, and C. E. Landwehr (editors), *Database Security, VIII: Status and Prospects*, pages 37–56. North-Holland, Amsterdam, The Netherlands, 1994.

[3] M. Nyanchama and S. L. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, 1999.

[4] S. Osborn and Y. Guo. Modeling users in role-based access control. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control*, pages 31–38, Berlin, Germany, 2000.

[5] F. Rabitti, E. Bertino, W. Kim and D. Woelk. A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, 16(1):88–131, 1991.