A Methodology for Managing Roles in Legacy Systems

Sylvia L. Osborn
Dept. of Computer Science
The Univ. of Western Ontario
London, Ontario, Canada
N6A-5B7
sylvia@csd.uwo.ca

Yan Han
Dept. of Computer Science
The Univ. of Western Ontario
London, Ontario, Canada
N6A-5B7
yhan@csd.uwo.ca

Jun Liu
Dept. of Computer Science
The Univ. of Western Ontario
London, Ontario, Canada
N6A-5B7
jliu@csd.uwo.ca

ABSTRACT

Role-based access control (RBAC) is well accepted as a good technology for managing and designing access control in systems with many users and many objects. Much of the research on RBAC has been done in an environment isolated from real systems which need to be managed. In this paper, we propose a methodology for using an RBAC design tool we have developed, to manage and effect changes to an underlying relational database. We also discuss how to simulate the role graph model on a Unix system, and extend the methodology just described for relational databases to managing a Unix system when changes are made to the role graph.

Categories and Subject Descriptors

D.4.6 [**Software**]: Security and Protection—*Access controls*; K.6.5 [**Computing Millieux**]: Security and Protection

General Terms

Algorithms, Security

Keywords

role-based access control, relational databases

1. INTRODUCTION

Role-based access control models have been discussed for a number of years [6, 8, 1, 13]. It is well accepted that designing security for situations with many users and many objects is greatly facilitated by using a role-based design. Many of these designs are off-line, similar to the way in which one would use an Entity Relationship model to design a database before approaching the commercial package on which one implements the design. The problem with this approach is that, with systems which have no built-in role-based facility, one still has to install individual (user, permission) pairs to correspond to the assigned roles of a user, and even more worrying, one has to delete all of these pairs when a user is removed from such a role.

We have developed a role graph model [9]. One could interface it with every piece of legacy software, as we did some years ago with DB2 [11]. However we have greatly improved the tool since then. Recently, the tool has been enhanced with the option of saving the graphs in an XML format rather than the ad hoc file formats which were being used. We envisage the tool being used to initially design the role graph and group memberships. Then, from time to time, the security administrator will interact with the tool to add and delete users, their group memberships, their role assignments, alter the role graph and possibly the role-permission assignments. When such a session is finished, a new version of all of this information can be saved. What we describe below is how this information can be used to alter the permissions recorded in some legacy system when the session with the role graph tool is completed. This change can be run against a database as a transaction, for example.

Some relational packages (Oracle and Sybase, for example) do have roles, and roles are part of the SQL99 standard. However, one is not forced to use roles, and they can be mixed in with individual user-permission assignments. The relational database packages also do not have tools to deal with role hierarchies. Thus, even if one is using a database package that has roles, it is better to have a more sophisticated tool to use in designing the access control for a complex environment. It can also be argued that it is, in some circumstances, better to have all access control managed by a tool such as the role graph tool, rather than allowing individual user-permission assignments to take place. The proposals in this paper assume that this is the case.

We do not deal here with decentralized administration of roles. In the database environment, we assume that the database administrator owns all the tables, so that individual users to not have the ability to circumvent the permissions being managed by the role graph methodology. Similarly, in the Unix environment, for centralized control, we assume that the superuser (or root) owns all the files, and we use another mechanism to achieve a rich blending of permissions for users as can be achieved by the RBAC model.

The paper is organized as follows: Section 2 describes the Role Graph Tool. Section 3 contains a suggested methodology for interfacing the tool with any relational database system, In Section 4 we describe an experiment we did to interface the role graph model with a Unix system, and then discuss how the methodology in Section 3 can be adapted for a Unix environment. Section 5 concludes the paper.

SACMAT'03, June 2–3, 2003, Como, Italy.

2. THE ROLE GRAPH TOOL

The role graph model, originally proposed by Nyanchama [7], has had a group graph model added to it [2, 10]. Roles consist of a role name and a set of privileges, each of which is an (object, access mode) pair. If the role graph were governing a relational database application, the table privileges would be (relation name, access mode) where the access modes are Insert, Select, Update, Delete etc. If it is governing a Unix system, the objects are files, and the access modes are just read, write and execute.

Roles are arranged in a role graph, with two distinguished roles: MaxRole and MinRole. MaxRole represents all the privileges in the role graph and need not be assigned to any user or group. MinRole represents the least privileges assigned to anyone in the system. We distinguish between direct privileges which are those directly assigned to a role, and effective privileges which consist of the direct privileges and those inherited from junior roles. The effective privileges for role r are also denoted by $r.rpset$. Role graphs have the following properties:

- there is a single MaxRole,
- there is a single MinRole,
- the graphs are 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,
- by convention we draw the graphs with MaxRole at the top, MinRole at the bottom, and junior roles lower on the page than their seniors. We also remove transitive edges from the display to make the graph less cluttered.

The operations available in the role graph panel are outlined in [9].

The Group Graph model allows one to create sets of users, say to represent committees or people assigned to a project, who may not have the same job title. To simplify the model, each individual user is regarded as a group of cardinality 1. The edges in the group graph are determined by the subset relationship between two groups. By convention we draw the group graph with the Base group which contains all users at the bottom, and the individual user groups at the top.

In Section 4, we will be talking about Unix groups. Unix groups are also defined to be a set of users. When confusion is possible, we will say RBAC groups or Unix groups to distinguish between the two.

Together these models have been incorporated into a tool written in Java. An image of the tool is shown in Figure 1. With the tool, one can develop a role graph in the right-hand panel, a group graph in the left hand panel, and make user/group to role assignments (these would be connections between nodes in the two panels; they are not explicitly drawn in the display in the current version of the software). Each user is shown as a group of cardinality 1 at the top of the group graph. The privilege information is available through the roles. When one clicks the mouse over a role, say VP2, we see the image in Figure 2, i.e. that Sally has been assigned to this role, and that it has 2 direct privileges and several effective privileges.

When one clicks the mouse over a group, one sees a display as in Figure 3, i.e. that the members of the Office5 Group have been assigned to role L4, and therefore are indirectly assigned to MinRole.

3. INTERFACING WITH A RELATIONAL DATABASE

The methodology we propose is this: rather than managing permissions in a relational database using the ad-hoc role facilities provided in the database package, or just using straight user-permission assignments, the security administrator would manage access control using the role graph tool. When changes are required to the underlying database system, the security administrator would use our tool to modify the group and role graphs, and when done, would cause these changes to be reflected in the underlying database. The rest of this section gives some details of our implementation of this methodology.

The role graph tool was recently enhanced with the ability to save both graphs and the group-role assignments as an XML file rather than the ad-hoc formats used previously. An example of the XML file for the sample role and group graphs is in Appendix A. This XML conforms to an XML Schema which has been defined for role and group graphs [3]. In addition to that, we have added a menu choice so that when the user saves the new group and role graphs, they have a choice to also effect any changes in an underlying relational database. An algorithm is run in which the group and role graphs before the session are compared with the group and role graphs after the session. The algorithm works from the XML version of the information, so the graphs do not have to be reconstructed. The alternative to comparing the two XML files would be to issue the changes as the role graph is being modified. The current choice gives a much more straightforward algorithm, and probably a more efficient one (e.g. the security administrator might make many changes and then change their mind). With the current implementation, the comparison is made when the role and group graph changes are completed and a "commit point" has been reached.

In any relational database, permissions are granted to users and revoked from users using GRANT and REVOKE statements. The algorithm creates as output a sequence of these GRANT and REVOKE statements so that any additional users or permissions, or any additional pairs resulting from enhancing the group graph or the role graph are generated as GRANTs. Also, if any users are deleted or any privileges are no longer available to a user because of changes to one of the graphs, this will be the subject of a REVOKE statement generated by the algorithm. Once this sequence of GRANTS and REVOKES is built, it can be executed in the database as a single transaction, so that its results are seen by other transactions/users all at once.

The algorithm is given in Figure 4. It has run time polynomial in the number of users, the number of roles and the number of permissions.

4. INTERFACING WITH UNIX

In [12], Sandhu and Ahn describe a way of using the group mechanism in Unix for access control. They focus on the de-

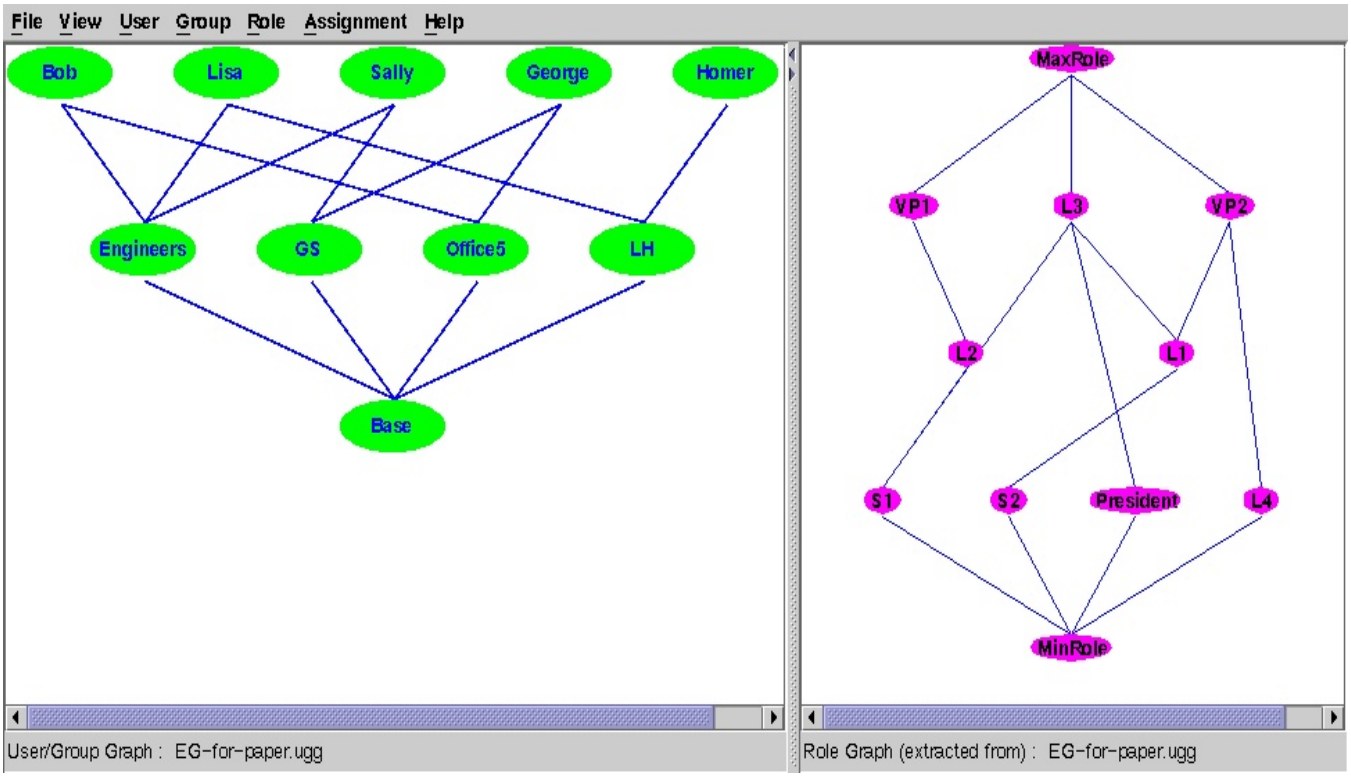

Figure 1: Screen Image of the Role Graph Tool

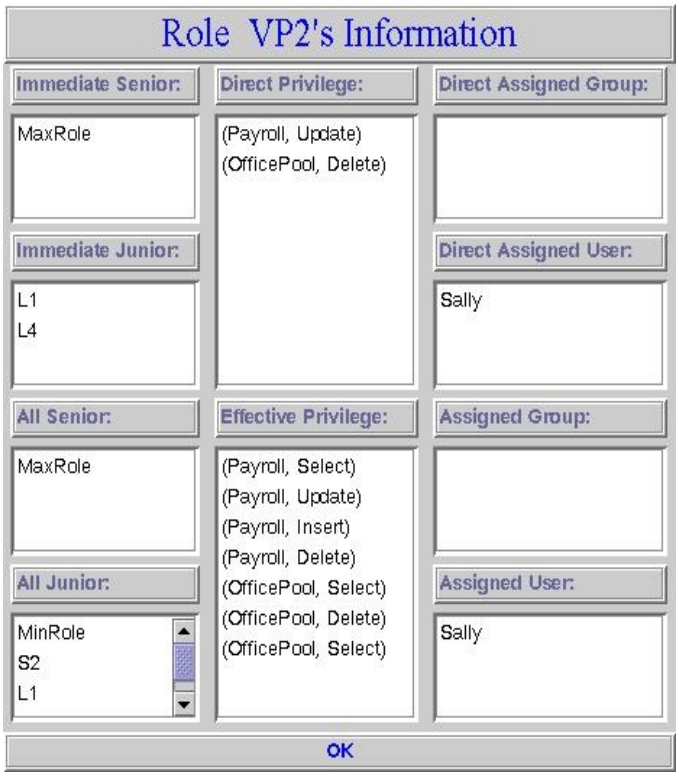

Figure 2: Image of Role VP2 display

Figure 3: Image of Office5 group's display

centralized administration of access control which is not the topic of this paper. They also do not deal with the privileges, but seem to assume that the information associated with each project is a set of files totally under the control of that role.

In considering the privileges available in a Unix environment, the objects one can talk about are simply Unix files, and the operations available on these files are read, write and execute. The basic Unix mechanism for controlling access involves the familiar 9 bits, of which 3 give read, write and execute permissions for the owner of the file, 3 give the permissions for the group the file belongs to, and the final 3 give the permissions for all other users. The owner of a file cannot request, as in a relational database, that they want to give read permission to a specific user by name, nor can they give permissions to a group by its (Unix) group name.

Other than the 9 bits, Unix groups are the only other mechanism provided for dealing with access control in original Unix systems. Most modern implementations of Unix now have access control lists (ACLs). With an access control list, one can specify an arbitrary number of (groupname, access mode) or (username, access mode) pairs for each file.

In a recent experiment [5], we looked at simulating role graphs in Unix.[1], i.e. we looked for a way to take an arbitrary role graph and map these permissions onto a Unix system so that the privileges assigned to a user in the role graph would be exactly those available in the Unix system, assuming that by privileges we mean read/write/execute on Unix files.

We found that Unix groups by themselves could not simulate the rich combinations of privileges that a truly arbitrary role graph demands. Consider the very simple role graph in Figure 5. If two different users u1 and u2 own File1 and File2 respectively, then for other users to be in either role R1 or role R2, we must associate these users with different Unix groups, say G1 and G2. But with the split of the read and write permissions of File2, this means that File2 would have to belong to 2 groups, and files in Unix have one group. If (without loss of generality) we assume that Unix group G1 corresponds to role R1, then the read permission for File2 cannot be represented by the bits for the owner, nor for the group because this will not make the read permission available to role R2. If we use the "all other users" bits to make the read permission available, then users who are not assigned to role R2 would also get this privilege. The situation is the same if both files are owned by the superuser but belong to different groups.

The solution proposed in [5] is to use both groups and access control lists. Given that we have groups and access control lists, we now associate each role in a role graph with a Unix group (which can have the same name as the role), and for each privilege in the effective privileges of that role, we add the pair (Group name, access mode) to the ACL for the file. This solution assumes a centralized administration of the role graph, and assumes that some superuser is the owner of all the files being managed by the role graph RBAC system.

Going back to the methodology described in the previous section, when the security designer works on the role and (RBAC) group graphs with the tool, the changes made from the previous version need to be installed in the Unix system which is being managed by the role graph. Once again, we can compare the previous version of the graph with the new version, by comparing the XML files output by the RBAC tool. There are two ways we can proceed. The first way is faithful to the Unix experiment in [5]; the second is simpler and accomplishes the same access control.

[1]Previously we had looked at taking the file permissions existing in a Unix system and modeling them as a role graph [4].

Algorithm: Role-Graph Comparison(R-GG1, R-GG2)

```
Input: R-GG1 /* the original Role and Group Graphs */
   R-GG2 /* the new Role and Group Graphs */
Output: A String representation of the differences of the two graphs,
in the form of SQL Grant and Revoke statements.
Method:
   userSet-org = R-GG1.BaseGroup.UserSet;
   userSet-new = R-GG2.BaseGroup.UserSet;
   for all users u in userSet-org do
      for all roles r assigned to u in R-GG1 do
         for all effective privileges p assigned to role r in R-GG1 do
            add p to u.privs-org
   for all users u in userSet-new do
      for all roles r assigned to u in R-GG2 do
         for all effective privileges p assigned to role r in R-GG2 do
            add p to u.privs-new
   /* Case 1: users in the original graph, but NOT in the new graph */
   for all users u1 in userSet-org do
      /* a user in userSet-org, but not in userSet-new, REVOKE ALL privileges from this user */
      if (u1 NOT in userSet-new)
         for all p in u1.privs-org do
            append REVOKE p from u1 instruction to output
   /* CASE 2: users in the new graph, NOT in the original graph */
   for all users u2 in userSet-new
      if (u2 NOT in userSet-org)
         for all p in u2.privs-new do
            append GRANT p to u2 instruction to output
   /* CASE 3: users in both Graphs, compare the privileges */
   for all users u1 in userSet-org
      /* u1′ is the corresponding user in the new graph */
      find the same user in userSet-new – call it u1′
         if ( u1′ exists )
            for all privileges in u1.privs-org - u1′ .privs-new
               append REVOKE p from u1 instruction to output
            for all privileges in u1′ .privs-new - u1.privs-old
               append GRANT p to u1′ instruction to output
```

Figure 4: Role Graph Comparison Algorithm

In the first case, we need to make several changes to the algorithm. First, if roles have been added to or deleted from the role graph, we need to create or delete a Unix group corresponding to the new/deleted role. This is done with the Unix groupadd and groupdel commands. This code would be inserted before "Case 1" in the algorithm. We would also have to modify the algorithm to add/delete users from groups.

In the prototype developed in [5], the access control lists are built up in terms of groups. Therefore, if role R1 were going to acquire read and write permissions on file F1, the following Unix command would accomplish this:

```
setfacl -m g:R1:w F1
```

The second proposed solution, again using the algorithm in Section 3, is simpler. Access control ultimately concerns whether or not individual users can access individual files. This can be accomplished by issuing individual user permission additions or deletions. We would not have to create the Unix groups at all. Instead of issuing the relational database Grant and Revoke statements relating to a particular user, we change the output from the algorithm in Figure 4 to issue

```
setfacl -m u:username:--- filename
```

to add a permission for user *username* to read, write or execute (replacing "`---`" with, appropriately, r`--`, -w- or `--`x) for file *filename*. To revoke a permission, the following command should be substituted for the Revoke statement in the algorithm:

```
setfacl -d u:username:--- filename
```

The changes to the algorithm discussed above to add Unix groups are not even necessary as the algorithm in Figure 4 will give the appropriate permissions to users assigned to the roles. Thus, the only change to the algorithm is to replace the Grant instructions with setfacl -m commands and replace the Revoke instructions with setfacl -d commands. Once the algorithm has generated these commands, they must then be run in the Unix environment to reflect the changes made in the RBAC system.

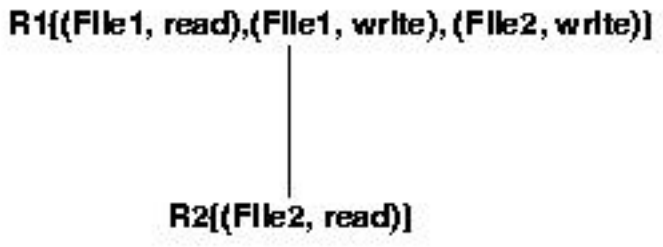

Figure 5: Simple Role Graph with 2 Roles

The second solution does leave as much information about the organization of the access control in the Unix system, as Unix groups are not created. It is, however, a simpler solution.

5. CONCLUSIONS

The paper began by describing a tool we have been developing to manage role and group graphs for access control for a complex system.

One of the problems with adopting RBAC methodology in existing installations has been that legacy systems have to be retrofitted with the new access control features. What we have proposed in this paper is a methodology in which a tool to manage role and group graphs is used to design the access control for a complex system, after which the mechanisms already available in the legacy system can be used to execute a sequence of security-related statements to make the corresponding privileges available to users precisely as designed in the role graph system. We showed how this can be done for a standard relational database package. We also discussed how the original Unix security provisions make truly arbitrary role graphs impossible to simulate, but that with a Unix system with Access Control Lists, one can use the same methodology to manage access control as proposed for databases, i.e. designing the access control using the RBAC tool, and then executing a sequence of setfacl instructions to make the privileges available to users as designed in the RBAC tool.

6. ACKNOWLEDGEMENTS

The following students have worked on the current version of the software for the role graph and group graph tool: Rizwan Qureshi, He Peng, Kefeng Yan and Yan Shi. Many thanks go to David Wiseman in the UWO Computer Science Department for answering questions about Unix. Sincere thanks also go to Dave Martin of the UWO Computer Science Department for his help with the final paper. The financial support of the Natural Sciences and Engineering Research Council of Canada is greatfully acknowledged.

7. REFERENCES

[1] D. Ferraiolo, J. Cugini, and D. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings 11th Annual Computer Security Applications Conference*, 1995.

[2] Y. Guo. User/group administration for RBAC. Master's thesis, Dept. of Computer Science, The University of Western Ontario, 1999.

[3] Y. Han. An XML model for RBAC for interaction with relational databases. Master's thesis, The University of Western Ontario, 2003.

[4] L. Hua and S. Osborn. Modeling UNIX access control with a role graph. In *Proceedings of International Conference on Computers and Information*, June 1998.

[5] J. Liu. Mapping the role graph model to UNIX. Master's thesis, The University of Western Ontario, 2002.

[6] F. Lochovsky and C. Woo. Role-based security in database management systems. In C. Landwehr, editor, *Database Security: Status and Prospects*. North-Holland, 1988.

[7] M. Nyanchama. *Commercial Integrity, Roles and Object Orientation*. PhD thesis, Department of Computer Science, The University of Western Ontario, London, Canada, Sept. 1994.

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

[11] S. Osborn, L. Reid, and G. Wesson. On the interaction between role based access control and relational databases. In P. Samarati and R. Sandhu, editors, *Proceedings of the Tenth Annual IFIP WG 11.3 Working Conference on Database Security*. Chapman & Hall, Aug. 1996.

[12] R. Sandhu and G.-J. Ahn. Decentralized group hierarchies in UNIX: An experiment and lessons learned. In *National Information Systems Security Conference*, 1998.

[13] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29:38–47, Feb. 1996.

APPENDIX

A. A SAMPLE XML FILE

The following is the XML file corresponding to the graphs in Figure 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<RBAC xmlns="http://www.csd.uwo.ca/rolegraph"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.csd.uwo.ca/rolegraph. ./RoleGraph_1106.xsd">
    <GroupGraph>
        <Base>
            <UserSet>Bob Lisa Sally George Homer </UserSet>
            <SubGroupSet>Office5 LH GS Engineers </SubGroupSet>
        </Base>
        <Group>
            <GName>Engineers</GName>
            <UserSet>Bob Lisa Sally</UserSet>
        </Group>
        <Group>
            <GName>GS</GName>
            <UserSet>George Sally</UserSet>
        </Group>
        <Group>
            <GName>Office5</GName>
            <UserSet>George Bob</UserSet>
            <AssignedRole>L4</AssignedRole>
        </Group>
        <Group>
            <GName>LH</GName>
            <UserSet>Homer Lisa</UserSet>
        </Group>
    </GroupGraph>
    <RoleGraph>
        <Privilege>
            <PName>SELECT_Payroll</PName>
            <PObject>Payroll</PObject>
            <PAccess>SELECT</PAccess>
        </Privilege>
        <Privilege>
            <PName>SELECT_Employee</PName>
            <PObject>Employee</PObject>
            <PAccess>SELECT</PAccess>
        </Privilege>
        <Privilege>
            <PName>INSERT_Employee</PName>
            <PObject>Employee</PObject>
            <PAccess>INSERT</PAccess>
        </Privilege>
        <Privilege>
            <PName>UPDATE_Employee</PName>
            <PObject>Employee</PObject>
            <PAccess>UPDATE</PAccess>
        </Privilege>
        <Privilege>
            <PName>INSERT_Payroll</PName>
            <PObject>Payroll</PObject>
            <PAccess>INSERT</PAccess>
        </Privilege>
        <Privilege>
            <PName>UPDATE_Payroll</PName>
            <PObject>Payroll</PObject>
            <PAccess>UPDATE</PAccess>
        </Privilege>
        <Privilege>
            <PName>DELETE_Payroll</PName>
            <PObject>Payroll</PObject>
            <PAccess>DELETE</PAccess>
        </Privilege>
        <Privilege>
            <PName>DELETE_Employee</PName>
            <PObject>Employee</PObject>
            <PAccess>DELETE</PAccess>
        </Privilege>
```

```
        <Privilege>
            <PName>SELECT_OfficePool</PName>
            <PObject>OfficePool</PObject>
            <PAccess>SELECT</PAccess>
        </Privilege>
        <Privilege>
            <PName>DELETE_OfficePool</PName>
            <PObject>OfficePool</PObject>
            <PAccess>DELETE</PAccess>
        </Privilege>
        <MaxRole>
            <ImmJunior>L3 VP1 VP2</ImmJunior>
        </MaxRole>
        <MinRole>
            <ImmSenior>S1 S2 President L4</ImmSenior>
            <AssignedGroup>Office5</AssignedGroup>
        </MinRole>
        <Role>
            <RName>S1</RName>
            <DirPrivilege>SELECT_Employee INSERT_Employee</DirPrivilege>
            <ImmSenior>L2 L3</ImmSenior>
        </Role>
        <Role>
            <RName>S2</RName>
            <DirPrivilege>SELECT_Payroll INSERT_Payroll</DirPrivilege>
            <ImmSenior>L1</ImmSenior>
        </Role>
        <Role>
            <RName>President</RName>
            <DirPrivilege>SELECT_Payroll SELECT_Employee</DirPrivilege>
            <ImmSenior>L3</ImmSenior>
            <AssignedGroup>Lisa</AssignedGroup>
        </Role>
        <Role>
            <RName>L1</RName>
            <DirPrivilege>Delete_Payroll </DirPrivilege>
            <ImmSenior>L3 VP2</ImmSenior>
            <AssignedGroup>Bob</AssignedGroup>
        </Role>
        <Role>
            <RName>L2</RName>
            <DirPrivilege>UPDATE_Employee </DirPrivilege>
            <ImmSenior>VP1</ImmSenior>
        </Role>
        <Role>
            <RName>L3</RName>
            <ImmSenior>MaxRole</ImmSenior>
        </Role>
        <Role>
            <RName>L4</RName>
            <DirPrivilege>SELECT_OfficePool </DirPrivilege>
            <ImmSenior>VP2</ImmSenior>
            <AssignedGroup>Office5</AssignedGroup>
        </Role>
        <Role>
             <RName>VP1</RName>
             <DirPrivilege>DELETE_Employee </DirPrivilege>
             <ImmSenior>MaxRole</ImmSenior>
             <AssignedGroup>George</AssignedGroup>
        </Role>
        <Role>
             <RName>VP2</RName>
             <DirPrivilege>UPDATE_Payroll
                 DELETE_OfficePool</DirPrivilege>
             <ImmSenior>MaxRole</ImmSenior>
             <AssignedGroup>Sally</AssignedGroup>
        </Role>
    </RoleGraph>
</RBAC>
```