Modeling UNIX Access Control with a Role Graph

Lingling Hua
The University of Western Ontario
lhua@ca.ibm.com

Sylvia Osborn*
The University of Western Ontario
sylvia@csd.uwo.ca
http://www.csd.uwo.ca/faculty/sylvia/

Abstract

In this paper, we show how to model UNIX file access using a role-based approach. A role-based access control model is presented, and its use in reflecting the existing permissions in a UNIX environment is described.

1 Introduction

Role-based access control can be used as an aid to managing access control in a complex environment. Role graphs, one manifestation or role-based access control, provide a way of visualizing the permissions granted in a complex system. For example, in [ORW96], it has been shown how an interface between a role graph system and the relational database system DB2 can be built. This paper describes the beginnings of an interface between role-based access control and UNIX. Ultimately we would like to be able to go from a role-based description of permissions to the UNIX version, and vice versa. This paper describes an experiment in going in the second direction, i.e. in modeling the existing permissions in a UNIX system using roles.

We begin by describing role-based access control and a role graph model in Section 2. In Section 3, we briefly review access control in a UNIX system. The description of the application of the role **graph** model to UNIX access control and the description of the experiment are found in Section 4. Section 5 contains conclusions.

2 The Role Graph Model

The role graph model is one manifestation of role-based access control (RBAC). Roles in RBAC are used to group together permissions to perform certain actions in a way that makes sense to the enterprise or environment. Individual users or groups of people can then be assigned to the roles as required.

Roles provide a very natural and powerful way for an enterprise administrator or security officer to describe the privileges of various job functions. In previous work, Nyanchama and Osborn have introduced a role graph model, which provides a way of visualizing the interactions among roles and their seniors and juniors [NO94]. They have also introduced algorithms for manipulating these role graphs. It has also been shown how roles can be used to model mandatory access control [NO95, Osb97, San96].

Roles have been studied in a variety of contexts and environments; we summarize some of them here. An early reference to roles is found in [LW88], where roles are defined and arranged in a generalization hierarchy, and agents representing people are assigned to roles as necessary. Early work by T.C. Ting [Tin88] describes the use of roles to develop application-dependent security controls. Ting's work was also incorporated into a software design system [TDH, HDT94]. Thomsen's work talks about roles, subroles and the mandatory enforcement of policies in a role-based environment [Tho91]. The Named Protection Domains of Baldwin [Bal90] are very similar to our roles. One feature of Baldwin's model is that only one Named Protection Domain can be active at one time. Mohammed and Dilts [MD94] discuss the design of a role-based model for a specific application in an event-dependent, dynamic environment. von Solms and van der Merwe give a 4-level model where roles form a layer between users on the one hand and transactions and projects on the other [vSvdM94].

The role graph model is based on a very general notion of privilege. A *privilege* is a pair (x, m) where x refers to an object, and m is a non-empty set of access modes for object x. The object referred to by x can be an object in an object oriented environment, a database granule in a database environment, or any system resource whose access needs to be controlled. The access modes m, can be any valid operations on x. In systems with simple access modes such as read, write, execute, etc., m is a subset of these access modes. Where x is an object in an object-oriented environment, m would be the execute mode of one or more methods. In transactional systems, m would be a list of transactions that

*This research was funded by a grant from the Natural Sciences and Engineering Research Council of Canada.

facilitate access to x. The details of x and m depend on the application environment and the associated security policy [NO93].

A *role* is a named set of privileges. It can be represented by a pair $(rname, rpset)$, where $rname$ is the name of the role, and $rpset$ represents the set of privileges of the role. Given a role r, we will use $r.rname$ and $r.rpset$ to refer to the role's name and privilege set, respectively. Of interest in analyzing roles are role-role relationships. We say that role r_i *is-junior to* r_j, if $r_i.rpset \subset r_j.rpset$. We also say that r_2 *is senior to* r_1. By specifying that role r_i is-junior to r_j, one makes available all the privileges of r_i to role r_j.

The total management of the authorization of privileges to users can be modeled on three planes (see Figure **??**). On the plane closest to the objects are the privileges themselves, without any groupings. Relationships, or perhaps more precisely implications, can exist among privileges. One cause of these implications results from knowledge of the semantics or the procedural structure of the operation being authorized. Such implications are found in the authorization type lattice of [RBKW91]. For example, the privilege which allows a user to update an object might imply that any such user should also be able to read the object. A complex method can contain calls to other methods. Depending on the security policy, any authorization to such a complex method may or may not imply separate or direct authorization to these called methods. An example in Unix is when a program is executed from a file which has the set-group-id or set-user-id bits set. These bits indicate that the program should be executed with the permissions of the group or user who owns the file; i.e. the permission to execute such a program implies that these extra permissions hold at least temporarily.

Another cause of implications on the privileges plane is object containment. Authorization to read a whole object can imply authorization to read any of its individual parts. Authorization to read a set of objects can imply authorization to read the members of the set. These implications result from the kinds of information represented in the authorization *object* lattice of [RBKW91], which deals with the granularity at which a privilege is specified.

Implications on the privileges plane can also result from inheritance in an object oriented system. For example if one can raise the salaries of all *Employees*, and *Professors* is a subclass of *Employees*, then one should also be able to raise the salaries of professors.

The middle plane is where roles are considered. In the role plane, the nodes correspond to roles; i.e. they represent named sets of privileges. Here, role-role relationships can be declared and managed, and tools to help manage role-based security can be modeled. This is the plane on which the role graphs are described.

The third plane is where user and user group relationships can be modeled. User groups would be created to help manage the assignment of users to roles. For example, all the users in a given project might be put in one group. This group can then be assigned to roles relevant to the project, whereas individual users might be assigned to other roles one at a time. In this plane, implications also exist, which are again the result of different modeling activities. The information about which groups a user is assigned to, and which groups are or are not contained in other groups, and other user-user relationships, is represented here.

If we are using the role model to *prescribe* access control, there are two places where assignments take place which ultimately determine who is authorized to do what. One is in the assignment of users/groups to roles, which is called the User-Role Authorization, and the other is in the assignment of privileges to roles, which is called the Role-Privilege Authorization (see Figure **??**). Role-role relationships, which are defined within the role plane, are another way that privileges may be assigned to roles. Thus given a user-privilege pair, the decision about whether the user is authorized to the privilege is the result of the implications in the user/group plane, the assignment of users/groups to roles, the role-role relationships, the assignment of privileges to roles, and the implications in the privileges plane.

In this paper we use a role graph to *reflect* what has been assigned through UNIX authorization mechanisms. We analyze what has been specified for a UNIX environment and present it as a role graph. The role graph models the roles on the role plane with an acyclic, directed graph, in which the nodes represent the roles in a system, and the edges represent the $is-junior$ relationship. In addition to the user-defined roles, every role graph has a MaxRole and a MinRole. MaxRole represents the union of all the privileges of the roles in the role graph. MaxRole does not need to have any users authorized to it. It is in the role graph to have a place to summarize all of the privileges in the system, and where appropriate, to represent the role of a superuser. MinRole represents the minimum set of privileges available to all roles. MinRole.$rpset$ can be empty. Role graphs have the following *Role Graph Properties*:

- There is a single MaxRole.
- There is a single MinRole.
- The Role Graph is acyclic.
- 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. This implies that there is a path from MinRole to every r_i, and there is a path from every r_i to MaxRole.

The role graphs are drawn without redundant edges, i.e. the graph is represented by its transitive reduction [AGU72]. The nodes are arranged on the page so that all $is-junior$ edges go up the page. In addition, for every role, one can distinguish between its effective privileges, and its direct privileges. The *direct privileges* of role r are those which are not contained in the $rpset$ of any of r's immediate juniors. The *effective privileges* of role r are the union of its direct

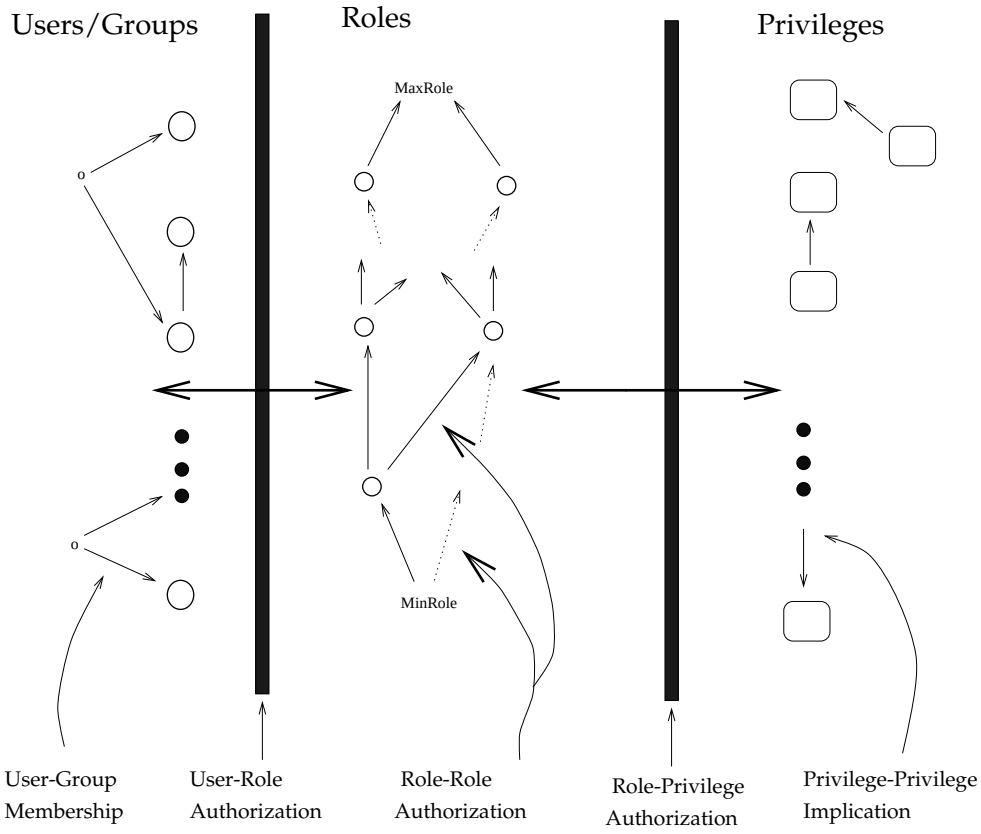

Figure 1: Three Kinds of Authorization

privileges and the effective privileges of all its juniors. Note that $r.rpset$ corresponds to the effective privileges of r. Seeing the graph edges that result from privilege assignment and the specification of role-role relationships, as well as seeing direct and effective privileges, helps the person analyzing a situation to understand the implications of access control assignments.

An example role graph is shown in Figure **??**. The graph shown contains no redundant edges, and the privileges shown are the direct privileges. Table **??** gives both the direct and effective privileges for each role. Note that the $is-junior$ information (i.e. the edges) can be deduced from the effective privileges, and vice versa.

Role Name	Direct Privileges	Effective Privileges
MaxRole	ϕ	{1,2,3,4,5,6,7,8,9,10,11}
VP1	{9,10}	{1,2,3,4,5,6,7,8,9,10}
VP2	{11}	{1,2,3,4,5,6,7,8,11}
L1	{3,4}	{1,3,4}
L2	{4,5}	{1,2,4,5}
L3	{5,6}	{1,2,5,6}
L4	{7,8}	{2,7,8}
S1	{1}	{1}
S2	{2}	{2}
MinRole	ϕ	ϕ

Table 1: Roles and their effective privileges

3 UNIX Permissions

The UNIX operating system should be familiar to most readers, so we will summarize briefly the relevant aspects of its access control properties. Everything executable or readable in a UNIX system resides in a file. Thus controlling access to files effectively controls access to software and data on the system.

In the terminology of the previous section, the objects of importance in the UNIX system are files, and the access modes are read, write and execute. There are three sets of three bits labeling each file description in UNIX: three bits describe the file owner's privileges, three the group's privileges, and three the privileges assigned to "world" or others, which is all users of the system. Within each three bit set, one bit says whether or not the read privilege is granted, one says whether or not write is granted, and the third says whether or not execute is granted. In some displays of this information, it is represented by the character string:

```
rwxrwxrwx
```

in which the first rwx refers to the owner's privileges, the second to the group's and the third to the world's. If the privilege is not granted, then a "-" appears, e.g.

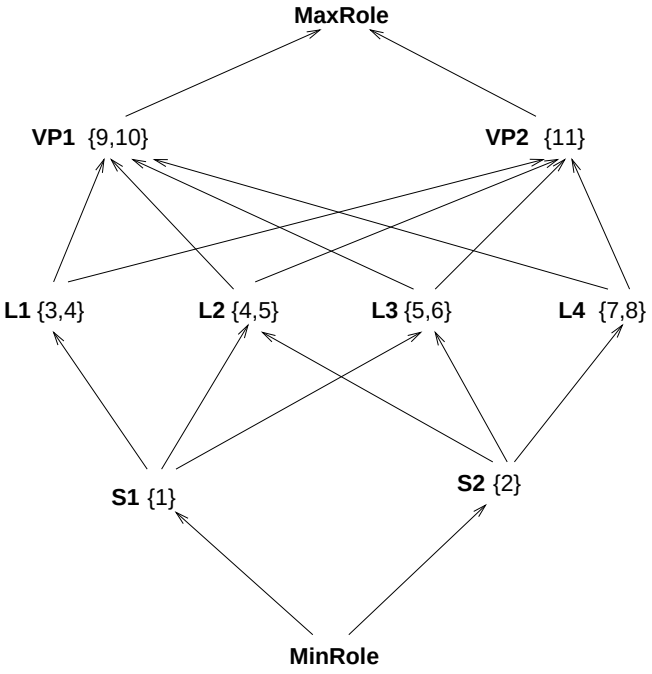

Figure 2: A Sample Role Graph

```
rw-------
```

These permissions can be maintained and changed by the UNIX commands `chgrp`, `chmod`, `chown` and `umask`.

In addition, there is a fourth set of three bits which indicate special features associated with a file. They are the set-user-id, set-group-id and sticky bits. In a directory listing, set-user-id is indicated by an s or S replacing the x in the owner's permissions. Its setting or clearance is done by issuing the command `chmod u+s` or `chmod u-s`. The set-group-id bit is specified by an s or S in the x position of the group permissions. Its setting or clearance is done by using `chmod g+s` or `chmod g-s`. The sticky bit is indicated by a t or T replacing the x in the others permissions. The letter t is only meaningful for the owner of the file. The setting and clearance of the sticky bit are carried out by `chmod u+t` and `chmod u-t`. For these three access permissions, if a small letter is specified, it represents that both the x permission and the permission this small letter indicates are turned on. Otherwise, only the permission the (capital) letter specifies is turned on. These access permission bits have different meaning when applied to files and directories. For files, their meaning is summarized in Table **??**; for directories, in Table **??**[AL96, Cur92].

UNIX always checks permissions for the smallest category that applies. For instance, a user in the owner's group (other than the owner) is always given the group permissions even if the permissions for "others" are broader[AL96].

The home directory is the working directory when a user first logs in to a UNIX system. The location of the home directory is specified in the /etc/passwd file or in the NIS database passwd. The pathname of this directory is stored in the HOME shell variable.

All the user's files are under the user's home directory, including the startup files and possibly other directories. The user can grant other users access to his or her files by modifying the group permissions or the world permissions. Suppose that such a file is directly under the user's home directory. As discussed above, other users can actually read, write or execute it only if they are also granted execute permission to the user's home directory. If the file is located under one or more sub-directories of the home directory, these users need to have not only execute permission for this sub-directory, but also execute permission for all directories on the path from the home directory in order to be able to read, write or execute this file.

Therefore, the home directory is the ultimate outer defense against any access to the files of a user. As pointed out previously, the account could be easily broken into or taken over when the security of the home directory is poor. On the other hand, if the home directory is protected properly, even if the files it contains seem to have permissions for others, the access would not be granted. Without x permission on your home directory, even if other users are allowed to write to your startup files, they can not access them. In what follows, then, we will concentrate on the access granted to home directories, rather than modeling the access to all the files under them.

4 Modeling a UNIX System with a Role Graph

A UNIX system could have hundreds of users with hundreds of files. In order to model what is essential about file access in a given environment, we need to focus on those permissions which can create security problems. In the previous section, we have explained that one economy we will take, where user's files are concerned, is to only model access to home directories. There are other directories in a UNIX system which contain the code for system commands, compilers and other utilities. We did not include these files in this initial prototype. They usually have permission bits set as `r-xr-xr-x`. These files are installed and maintained by system administrators. Including these files in a role graph will be the subject of future work. Therefore, a privilege for the role-graph we will build is defined as a home directory and a set of access permissions on it.

Users always have the right to read, write and execute their own files, and to change the permissions on their own files. Such permissions which allow a user to create and edit his or her own files are not the permissions which are going to create compromising situations. In the role graph model, the concept of "user" on the user plane in Figure **??** is the smallest grouping of subjects which can be assigned to a role. For this study, we will use the UNIX group as the access unit to the home directories, instead of a single user. The reasons for this are:

read	If set, the file may be read.
write	If set, the file may be written (modified).
execute	If set, the program contained in the file may be executed (run). Executing compiled programs requires only execute permission on the file, while executing shell scripts requires both read and execute permission since the shell must be able to read commands from the file.
set-user-id	Set-user-id status means that when a program is executed, it executes with the permissions of the user who owns the program, in addition to the permissions of the user executing it. The effective user id of the process becomes the id of the owner of the executable file. The real user id of the process remains that of the user who initiated the process. This bit is meaningless on non-executable files.
set-group-id	It behaves in exactly the same way as the set-user-id bit, except that the program operates with the permissions of the group associated with the file. When a process is executed with set-group-id bit turned on, the effective group id of the process becomes the group id of the owner of the executable file and the program thus executes with permissions of that group. The real group id of the process remains that of the user who initiated the process. This bit is meaningless on non-executable files.
sticky	If set on an executable binary file, the 'sticky' bit tells the operating system to maintain the image of the executing process in the swap area, even when execution is terminated.

Table 2: Access Permissions for Files

- every file, including a user's home directory, belongs to a group.
- UNIX always checks permissions for the smallest category that applies, i.e. the group permissions are checked before the world permissions. Since we will not worry about owner permissions, the group permissions are the smallest category we need to worry about.
- every user belongs to at least one group, and may belong to many groups.

The process of extracting the relevant information proceeds as follows:

1. for each group of interest in the system
   for each user which is a member of this group
   extract the group and other permission bits from this user's home directory and represent each permission as a privilege
   create a role for this group and the resulting set of privileges
2. for each pair of roles thus created
   if the privilege sets are equal
   then merge the roles
3. compute graph edges
4. for each role
   compute direct privileges
5. create MaxRole and MinRole if necessary
6. display the graph

After Step 1, there may be roles with identical privilege sets, which would make the resulting role graph have a cycle. Step 2 merges such roles into a single role. At this point, the privilege set represents all privileges granted to the users who belong to the group(s) which have been used to create these roles. In the terminology of role graphs, these are the effective privileges. Step 3 uses the containment of one set of effective privileges in another to create the edges in the role graph. Step 4 calculates the direct privileges by computing set differences between the effective privilege sets. After these steps, there may be a single role which has no seniors, in which case this role is renamed MaxRole. Similarly, if there is a single role with no juniors, this role is renamed MinRole. If either of these roles does not exist, then a MaxRole (or MinRole) is created with no users, and all the roles with no immediate seniors (juniors) are connected as its immediate juniors (seniors). The role graph display is done by a tool written in Tcl [Ous94].

One problem with creating roles in this way is that there is not any way for the program to figure out a suitable role name. When we go in the other direction, using the role graph tool to create roles and then map these onto the underlying system of interest, then some semantics of the situation can be built into the role name chosen. Here, we just arbitrarily number the roles starting with 1.

A prototype of these ideas to examine the protection of home directories in the research network in our department was built. The environment consists of a number of UNIX machines running several versions of Sun OS. Altogether there are over 200 users on the system. Users' home directories are organized into six sub-directories on this system, one each for faculty, staff, graduates, undergraduates, project and people. The project group is used for some special research projects. The people directory is for visitors. The prototype lets the user chose some subset of the six directories mentioned above, and then constructs the role graph. When all of the directories are asked for, the role graph generated is shown in Figure **??**. A display generated from the View menu in the role graph tool for role 4 is shown in Figure **??**. We can see that someone called `murphy` in the group `undergrad`

read	Read permission allows a user to see the contents (file names) of the directory, but is insufficient for accessing the files whose names appear in it. For instance, you can not read the contents of a file in a directory if you only have r access to the directory.
write	Write permission to a directory implies the ability to create, delete, and rename files in this directory. The w permission for a directory is required in order to add files to it or delete files from it. However, w permission for a directory is not required in order to modify a file listed in the directory or delete its contents.
execute	In the case of a directory, execute permission implies access the files contained therein. It allows you to operate on the names in that directory if you know them, but does not let you find out what they are if you do not. Normally r is granted whenever x is; you can get some strange effects if a directory has x but not r. For instance, if a directory has x turned on but not r, you cannot list its contents-but if you already know its contents, you might be able to read its files if you had that permission.
set-user-id	This bit is meaningless on directories.
set-group-id	On some systems this bit has a special meaning when set on directories. For example, in SunOS 4.x, the group id of a file is set to the group id of the directory in which it is created if the set-group-id is set on the directory. Otherwise, the group id of a file is set to the primary group id that the owner belongs to.
sticky	If a directory has its sticky bit set, users may not delete or rename files in this directory that are owned by other users. The sticky bit is usually set on world-writable directories.

Table 3: Access Permissions for Directories

has write permission to their home directory turned on.

5 Conclusions

The modeling of permissions granted on home directories in a UNIX system as a role graph was successfully carried out. From the example shown, it can be seen that such a model, even with just numbers for role names, can be useful in spotting anomalies in the permissions granted, or just for finding out who the users are and what permissions are set on their home directories, without searching through numerous directories.

There are two essential extensions required to completely model existing permissions in a UNIX environment. The system file permissions must be modeled, and links between files must be modeled. We will consider each of these separately.

Files and directories containing system commands and utilities are installed in our department by very knowledgeable systems staff. This might not always be the case in every UNIX installation. Therefore, it would be useful to model these permissions also. This would involve actually traversing the directories and extracting the non-owner permissions for individual files. In the terminology of the role graph, the privileges would now record access modes on individual files. The role graph would have a lot more privileges to record, but not necessarily a lot more roles unless there were a lot of different patterns of permissions granted.

Another way users gain access to their own or other's files is through links in the UNIX file system. Again, we are not interested in what a user does to files he or she owns, or how these might be linked within a single user's directories. However, if there is a file below one user's home directory, say user A, which has a link to it from a second user, user B's directory, then user B can access this file even if user A's home directory is not readable. In order for user B to actually be able to do anything with this file, the file's group or other permissions must be turned on. To analyze these situations, all user files would have to be traversed looking for files which are links, and checking the owner of the file accessed. If this owner is different from the owner of the directory from which it is accessable via a link, then the permissions of this file must also be extracted as privileges to be dealt with in the role-based analysis. In this case the privilege is only available to users who have such links in their directory, so the set of users modeled in the role graph (which is currently just groups) will have to be extended to include such individual users as well. Other members of user B's group would also have these privileges if all of user B's directories on the path from B's home directory to the directory containing the link have the appropriate permissions granted to "group". These two extensions are feasible, but both involve a much more complex traversal of the file system in an UNIX system than was carried out for the prototype presented here.

Acknowledgements We would like to thank David Wiseman for helping us to understand the UNIX environment in our department.

References

[AGU72] 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, June 1972.

[AL96] P. W. Abrahams and B. R. Larson. *Unix for the Impatient, second edition*. Addison-Wesley

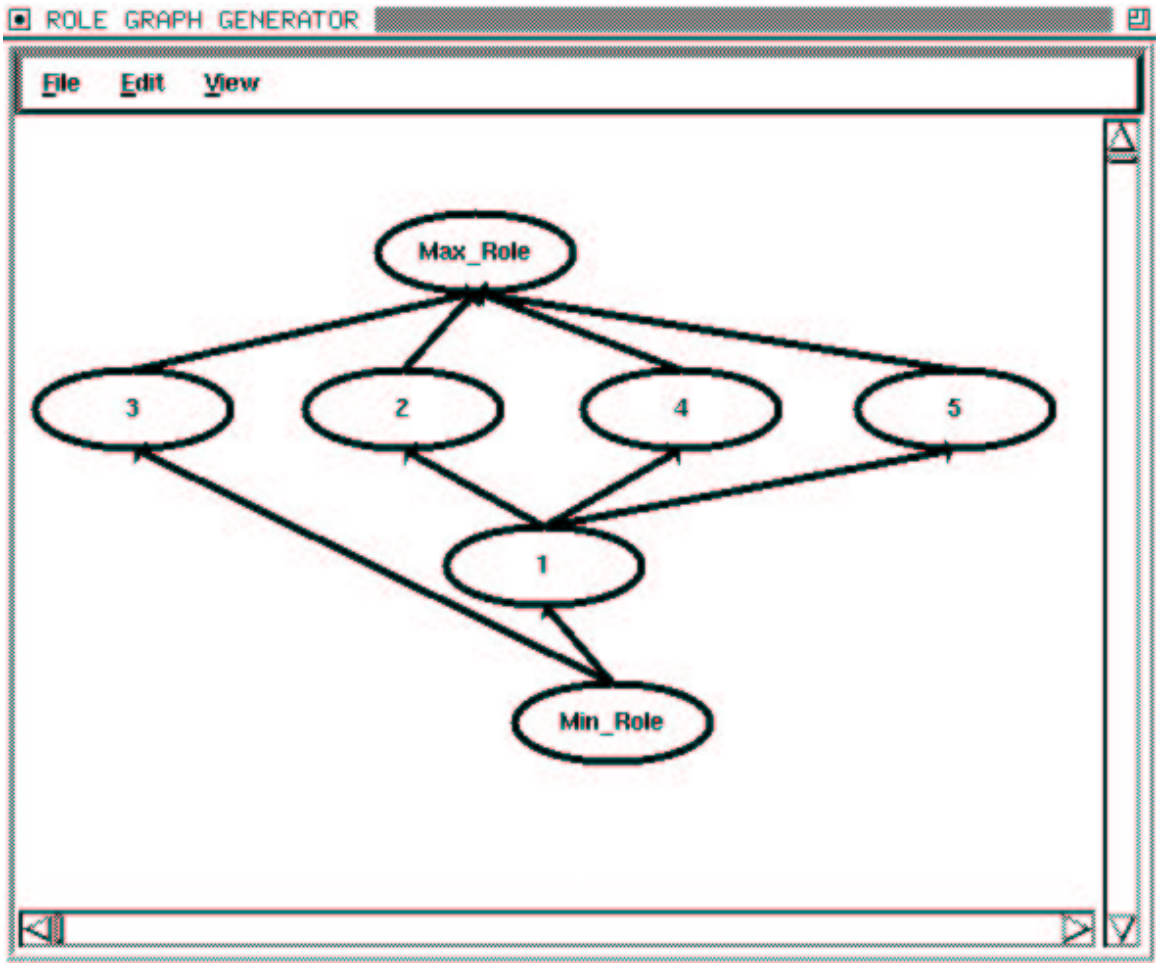

Figure 3: Role graph generated for all users

Publishing Company, 1996.

[Bal90] R.W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proc. 1990 IEEE Symposium on Research in Security and Privacy*, pages 116–132. IEEE Computer Society Press, 1990.

[Cur92] D.A. Curry. *UNIX System Security: a Guide for Users and System Administrators*. Addison-Wesley Publishing Company, 1992.

[HDT94] M.-Y. Hu, S. A. Demurjian, and T. C. Ting. Unifying structural and security modeling and analyses in the ADAM object-oriented design environment. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security, VIII, Status and Prospects WG11.3 Working Conference on Database Security*. North-Holland, 1994.

[LW88] F.H. Lochovsky and C.C. Woo. Role-based security in database management systems. In C. Landwehr, editor, *Database Security: Status and Prospects*. North-Holland, 1988.

[MD94] I. Mohammed and D.M. Dilts. Design for dynamic user-role-based security. *Computers and Security*, 13:661–671, 1994.

[NO93] M. Nyanchama and S. L. Osborn. Role-based security, object-oriented databases and separation of duty. *SIGMOD Record*, 22(4), Dec. 1993.

[NO94] 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*. North-Holland, 1994.

[NO95] M. Nyanchama and S. L. Osborn. Modeling mandatory access control in role-based security systems. In D.L. Spooner, S.A. Demurjian, and J.E. Dobson, editors, *Proceedings of the IFIP*

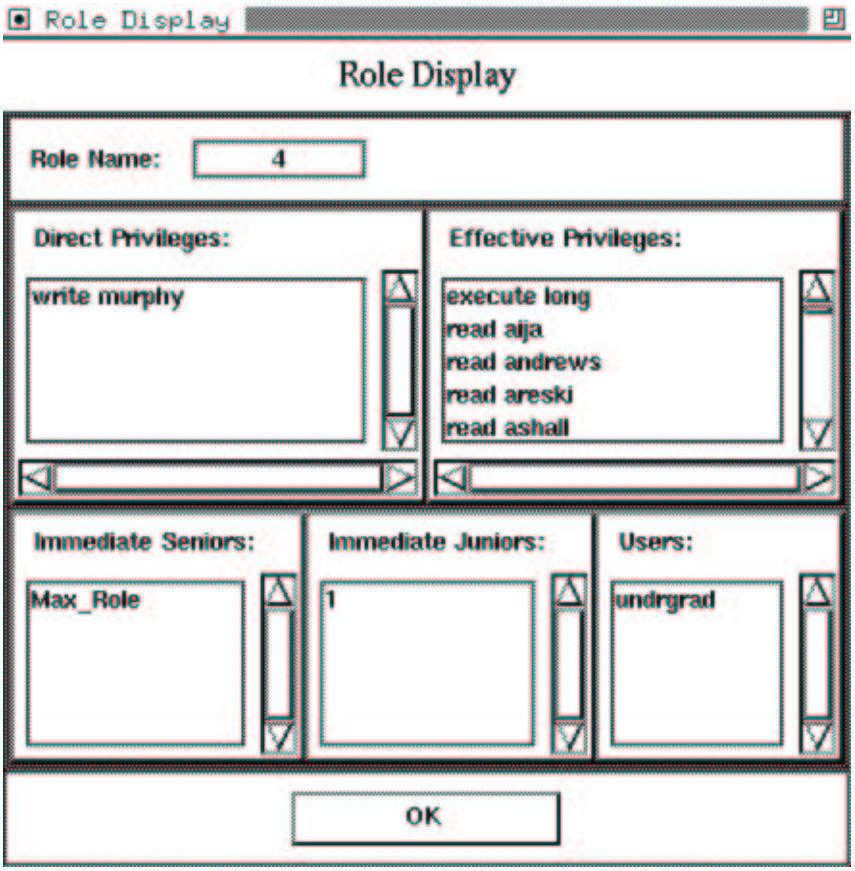

Figure 4: Display of role 4

WG 11.3 Ninth Annual Working Conference on Database Security. Chapman & Hall, 1995.

[ORW96] S.L. Osborn, L.K. Reid, and G.J. 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.

[Osb97] S.L. Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 31–40, Nov. 1997.

[Ous94] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[RBKW91] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans Database Syst*, 16(1):88–131, 1991.

[San96] R.S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Computer Security - ESORICS 96*, pages 65–79. Springer Verlag, 1996. Lecture Notes 1146.

[TDH] T.C. Ting, S.A. Demurjian, and M.-Y. Hu. Requirements, capabilities and functionalities of user-role based security for an object-oriented design model. In C.E. Landwehr and S. Jajodia, editors, *Database Security V, Status and Prospects*. North-Holland.

[Tho91] D.J. Thomsen. Role-based application design and enforcement. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV, Status and Prospects*, pages 151–168. North-Holland, 1991.

[Tin88] T.C. Ting. A user-role based data security approach. In C.E. Landwehr, editor, *Database Security: Status and Prospects*, pages 187 – 208. North-Holland, 1988.

[vSvdM94] S.H. von Solms and I. van der Merwe. The management of computer security profiles using a role-oriented approach. *Computers & Security*, 13:673–680, 1994.