International Journal of Control and Automation
Vol.2, No.2, June 2009

Hierarchical Role Graph Model for UNIX Access Control

Abderrahim Ghadi[1, 2], Driss Mammass[1], Maurice Mignotte[2], and Alain Sartout[2]

[1] *Irf-Sic–Fsa, Ibn Zohr University, Morocco*
[2] *Irma, University of Strasbourg, France*
{ghadi, mignotte, sartout}@math.u-strasbg.fr, mammass@univ-ibnzohr.ac.ma

Abstract

The access control system is a very important step in the implementation of the security policy of an information system. Access control checks what a user can do directly, as well as what programs executing on behalf of the users are allowed to do. In this way the access control seeks to prevent the activities which will be able to endanger the safety of the system.

The aim of this paper is to try to model the access control system in the operating systems of the type UNIX. The modeling will be based on a combining of the UNIX access control system, namely Super-User model, and RBAC[1] model [1, 11, 12]. In order to get a model nearest possible at reality, we will use the notion of roles and the privileges graph to build our graph. The properties of graph theory well are used in order to evaluate the stability and the robustness of our model.

Keywords: *Privilege, Role, Graph, Hierarchy, Access Control, SuperUser, DAC, MAC, RBAC*

1. Introduction

Safety, and more particularly access control [8, 9], are current problems in data processing [3, 4, 7]. Indeed, it becomes today important to be able to control the floods of information in the networks and the information systems. It is advisable to develop within the computing systems of the mechanisms making it possible to filter the accesses in order to let pass only those authorized. It is a question for that of laying down a security policy, i.e. the characterization of the allowed accesses. Access control is the center of gravity of computer security. Its function is to control which subjects (users, processes, machines, etc.) have access to which resources in the system, which files they can read, which programs they can execute, how they share data with other subjects, and so on.

The aim of this paper is to combine the access control model of UNIX (model of the type DAC[2] [13], based on the access modes and the concept of user-group-other) and RBAC model based on the roles [2]. A frequently asked question is what is the difference between roles and groups? A major difference between most implementations of groups and the concept of roles is that groups are typically treated as a collection of users and not as a collection of permissions. A role is both a collection of users on one side and a collection of permissions on the other one. The role serves as an intermediary to bring these two collections together. The resulting model will be presented in the form of graph [5, 6] which one will release from the mathematical results. The modeling of access control system give a

[1] RBAC: Role Based Access Control
[2] DAC: Discretionary Access Control

clearer vision of security system and consequently limits the maximum the intrusions usually based on transfers of privileges.

2. SuperUser model: UNIX Access Control

We summarize briefly the aspects of UNIX access control. Everything in a UNIX system resides in a file. Thus controlling access to files effectively controls access to software and data on the system. In conventional UNIX systems, the root user (also referred to as super user) is all powerful, with the ability to read and write to any file, run all programs, and send kill signals to any process. For this reason, the standard model of UNIX access control is called SuperUser model.

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 (rwx rwx rwx): three bits describe the file owner's privileges, three the group's privileges, and three the privileges assigned to 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. If the privilege is not granted, then a "-" appears, e.g:

- rw- r- - r- - 1 ghadi ghadi 28190 2009-03-01 12:41 UNIX-Admin.pdf

These permissions can be maintained and changed by the UNIX commands chgrp, chmod, chown and umask. In addition to the three modes (read, write, execute) we have three additional access modes: the Setuserid, setgroupid and sticky bits. In a directory listing, setuserid is indicated by an s or S replacing the x in the owner's permissions. The sticky bit is indicated by a t or T replacing the x in the others permissions.

Setuserid 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 or on directories.

Setgroupid It behaves in exactly the same way as the setuserid bit, except that the program operates with the permissions of the group associated with the file. When a process is executed with setgroupid 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.

On some systems this bit has a special meaning when set on directories. For example, in SunOS, the group id of a file is set to the group id of the directory in which it is created if the setgroupid 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 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. 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.

All the user's files are under the user's home directory /home/$USER, including the startup files and possibly other directories. Therefore, the home directory is the ultimate outer defense against any access to the files of a user. The user account could be easily pirated if the access rights of the home directory are badly managed.

3. Role Based Access Control: RBAC Model

A security policy is a set of rules which specify how to manage, protect or distribute information or resources of a system. In the case of the access control, a security policy is the definition of the authorized accesses. This definition will depend on the concept of entities, of the information of safety available as well as characterization of the access.

Hereafter, the keywords used in the field of the modeling of the information systems:

Definition 1

- Subject: A person or automated agent,
- Object: Any system resource, such as a file, terminal, printer, database record, etc,
- Role: Job function or title which defines an authority level,
- Permission: The ability or the right to perform some action on some resource,
- Operation: A level permission that a resource manager uses to identify security procedures,
- Session: A mapping involving subject, role and/or permission.

One generally distinguishes two big classes from security policies: discretionary policies (DAC) and mandatory policies MAC[3] [13].

The principal difference between a DAC and an MAC is the way in which the information of safety of the objects is modified and created. In the DAC model, each object is under the responsibility of one or more subjects. Change of informations of object security is thus carried out at the discretion of one or more persons in charge. Thus, a subject can potentially have access to any object, provided the persons in charge of this last gives him the permission of this object. Conversely, in the MAC model, Each subject has a set of fixed permissions, which it cannot change.

A typical example of DAC model is the security policy employed in the operating system UNIX (SuperUser Model). The Bell-LaPadula [14] model of access control uses mandatory access control.

The analysis of the security policies and the existing models makes it possible to conclude that the control systems of existing accesses are insufficient. Indeed, discretionary access control present of serious disadvantages with respect to the escapes of information and the Trojan horses, while the obligatory access control very rigid and is badly adapted to the systems really distributed.

In computer systems security, RBAC is an approach to restricting system access to authorized users. It is a newer alternative approach to DAC and MAC. RBAC is a policy neutral and flexible access control technology sufficiently powerful to simulate DAC and

[3] MAC: Mandatory Access Control)

MAC. Conversely, MAC can simulate RBAC when the role hierarchy is restricted to a tree rather than a partial order.

RBAC is a access control system to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as create, delete or modify a file. Roles are defined according to job competency, authority, and responsibility within the enterprise. Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles.

In RBAC model:

- Subject can have multiple roles,
- Role can have multiple subjects,
- Role can have much permission,
- Permission can be assigned to many roles.

4. Privileges graph

The privileges graph [15, 16] is a graph whose nodes are roles which represent a set of privileges on a set of objects. So, we propose to model the access control system in the form of this type of graph. Thus, we can improve the security policies by the application of a process with the three different phases:

1. Build the privileges graph. The arc of graph is a transfer method of privileges, in other words, the arc
2. is a mean to acquire privileges,
3. Check if the arcs of the graph are licit,
4. Compare the obtained graph with the hoped security policy.

The nodes and the arcs of the graph are created by the application of the rules which compose the authorization scheme. First we start by inspecting the files */etc/passwd* and /etc/group to check off the list of the users and the groups of the system (Node/Role). Then, several programs and scripts allow us to identify all the existing arcs in our system.

The authorization scheme is considered sure if from an initial protection sure state, we cannot reach a unsure state by the application of the rules of this scheme.

The figure 1 represents a sample example of graph of privileges.

In this example, we have 8 roles which correspond to the privileges of the 8 groups of a system. We have two administration roles: R_admin_1 (administration role whose members are R_5 and R_6) and R_admin_2 (administration role with only one member R_5). An analysis of the file system has to detect transfers of the following privileges: R_1 and R_2 are in the file *$HOME/.shosts* of R_8, what is presented in the graph by arc 1. Likewise, R_3 and R_7 are present in *$HOME/.shosts* of R_4. The inspection of the configuration files special to each user reveals that R_3 uses *.xinitrc* file of R_2 (*.xinitrc* is the file which cements all the process of starting of X). This transfer of privileges is presented here by arc 3. The arc 4 reflected the following situation: «the privilege of a role is a subset of privileges of another role".

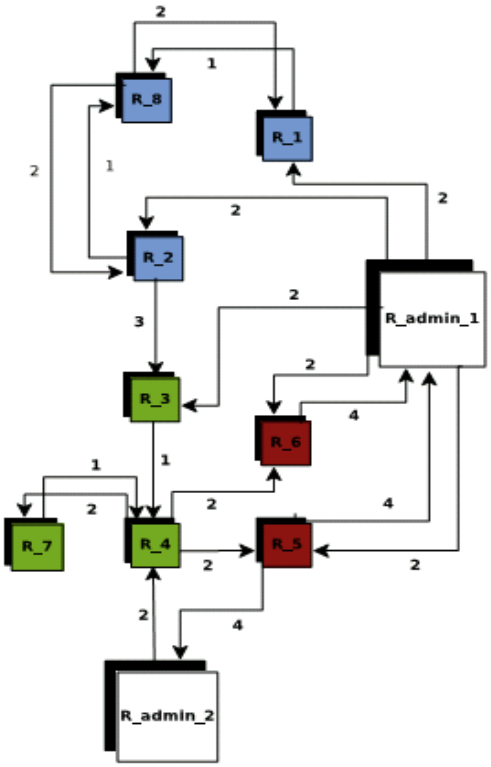

Figure 1. Example of privileges graph

5. Role graph model

Definitions 2

A$\Re$: The set of all access rights of a system;

Privilege: A privilege is a pair p = (*Obj*, A) where *Obj* refers to an object, and A is a non-empty set of access rights for *Obj* (A $\subseteq$ A$\Re$);

Role: A role r is a collection of privileges. r can be represented by a pair (r_{name}, r_{pset}), where r_{name} is the name of r , and r_{pset} represents the set of privileges of r;

$\Re$: Set of roles of a system;

Pv: Set of privileges of a system;

PL: Set of system profiles;

$\Re\Re$: Set of Redundant Roles;

F$\Re$: Father Role;

S$\Re$: Son Role;

UID : Set of identifiers of system users;

GID: Set of identifiers of system groups;

ID = UID $\cup$ GID

G = {g such as g is UNIX group}

$\mathbf{U_g}$ = {u such as u $\in$ g}

Arc One of problems encountered in the system administration of UNIX is that there is no hierarchy between the groups. Consequently, there will be a redundancy of privileges. For this reason, we added the concept of hierarchy of group via the

hierarchy of the roles. In this vision, we regard the arc of the roles graph as methods of transfer of privileges. In other word, an arc between two roles will be a hierarchy relationship. Let r_1 and r_2 two roles, if there is an arc between r_1 and r_2 ($r_1 \curvearrowright r_2$): r_1, we said that r_1 is a son-role of r_2. In this case all the privileges of r_1 will be transmitted to r_2.

5.1. Build of graph

The aim of this section is the build of the elements of the roles graph.

5.1.1 Build of roles

The build of the roles depends on the security policy of the system, and consequently there are more ways of proceeding in this stage. One can give, for example, one of those methods of build of role starting from the groups:

$\forall g \in G\{$
$\forall u \in U_g\{$
extract the privileges of the user u
create a role for this group
$\}$
$\}$

In addition to the roles created via the groups or of the users, there are two special roles to define: R_{root} and C_{ommon}:

R_{root}: Union of all privileges:

$$R_{root} = \bigcup_{p \in P_V} p$$

R_{common} Set of common privileges for all users. For example, the read access to the file */etc/passwd* in order to have the possibility of changing the password.

5.1.2 Build of edges

Definitions 3

1. Let ζ a function enumerating the privileges of a given role:

$$\zeta : \Re \longrightarrow P_V$$
$$r \longmapsto \zeta(r) = r.r_{pset}$$

$r.r_{pset}$ present the privileges of the role r

2. Default privileges (DP): Privileges assigned to a given role during its creation. In other term, are the initial privileges without counting the privileges acquired by hierarchy?

2. Hierarchical privileges (HP): Privileges obtained thanks to the hierarchy of roles [11],

3. Effective privileges (EP): Union of the default privileges of immediate son-roles. Effective privileges of a role r are: $(EP(r) = DP(r) \cup HP(r) = \zeta(r) = r.r_{pset})$

$\forall r_i, r_j \in \Re$. if $\zeta(r_i) \subseteq \zeta(r_j)$ Then an arc between r_i and r_j well be created: $r_i \curvearrowright r_j$

$\forall r_i, r_j \in \Re$, r_j is authorized to accede to the privileges of r_i if and only if $\zeta(r_i) \subseteq \zeta(r_j)$ we say, in this case, that r_i is son-role of r_j.

5.2. Graph properties

Our graph is a graph whose vertexes are roles and edges are the hierarchical relationship between these roles; methods of transfer of privileges between father-role and son-role. The aim of this paragraph is to quote some properties of this graph.

Definitions 4

We say that there exists a path, between two r_i role and r_j and noted $r_i \rightsquigarrow r_j$ if there exists $r_{k_1}, r_{k_2}, \cdots, r_{k_n} \in \Re$ such as $r_i \curvearrowright r_{k_1} \curvearrowright r_{k_2} \curvearrowright \cdots, \curvearrowright r_{k_n} \curvearrowright r_j$ We note the set of all paths in the graph Path, and thus our graph is noted: $(\Re, P_V, P_{ath}, \curvearrowright)$.

Property 1: Reflexivity

$\forall r_i \in \Re \; r_i \curvearrowright r_i$ (A role can be a son-role with himself since $(\zeta(r_i) \subseteq \zeta(r_i))$)

Property 2: Antisymmetry

$\forall r_i, r_j \in \Re$ *if* $r_i \curvearrowright r_j$ *and* $r_j \curvearrowright r_i$ *then* $r_i \equiv r_j$
$(\zeta(r_i) \subseteq \zeta(r_j)$ *and* $\zeta(r_j) \subseteq \zeta(r_i) \Rightarrow \zeta(r_i) = \zeta(r_j))$

This property ensures the no-redundancy of roles.

Property 3: Transitivity

$\forall r_i, r_j, r_k, \in \Re$
if $r_i \curvearrowright r_j$ *and* $r_j \curvearrowright r_k$ *then* $r_i \curvearrowright r_k$
$(\zeta(r_i) \subseteq \zeta(r_j)$ *and* $\zeta(r_j) \subseteq \zeta(r_k) \Rightarrow \zeta(r_i) \Rightarrow \zeta(r_k))$

Figure-2 presents an example of graph of role with a presentation in the form of table (table-1).

Now we can say that our graph. $(\Re, P_V, P_{ath}, \curvearrowright)$ is a partially ordered set (also called a poset).

In the graph of roles, we can have two roles without relation between them; therefore we conclude that our graph is partially ordered set (also called a poset).

- 1-4 it means 1,2,3,4
- Let $\{1-25\} \in P_V$
- The direct privileges of a role are those which are not obtained via transfer of privileges (privileges by defect affected by the administrator),
- The indirect privileges of a role are those which are obtained via transfer of privileges,
- The effective privileges of role are the union of its direct and indirect privileges,
- The path p is such as :

$$R_{commun} \curvearrowright R_2 \curvearrowright R_7 \curvearrowright R_1 1 \curvearrowright R_1 4 \curvearrowright R_{root}$$
$$\Leftrightarrow$$
$$\{1-4\} \curvearrowright \{1-4,7\} \curvearrowright \{1-4,7,8,15\} \curvearrowright \{1-4,7,8,15,21\} \curvearrowright \{1-4,7,8,15,21,26,27\}$$

After this discussion, we can say that our graph is a Lattice. In Mathematics, a Lattice is a poset in which sets of any two elements have a unique supremum (the elements' least upper bound; called their join) an infimum (greatest lower bound; called their meet).

Our graph is too a directed acyclic graph. It means a graph with no directed cycles; that is, for any vertex r, there is no nonempty directed path that starts and ends on r.

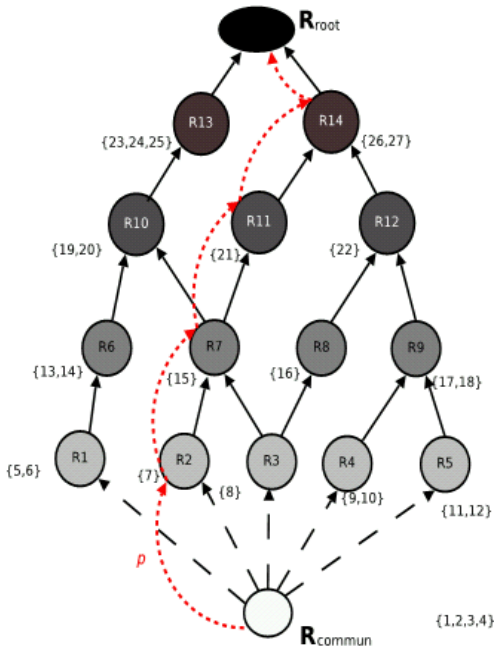

Figure 2. Example of privileges graph

Roles	Direct Privileges	Indirect Privileges	Effective Privileges
R_{common}	1-4	0	1-4
R1	5,6	1-4	1-6
R2	7	1-4	1-4,7
R3	8	1,2,3,4	1-4,8
R4	9,10	1-4	1-4,9,10
R5	11,12	1-4	1-4,11,12
R6	13,14	1-6	1-6,13,14
R7	15	1-4,7,8	1-4,7,8,15
R8	16	1-4,8	1-4,8,16
R9	17,18	1-4,9-12	1-4,9-12,17,18
R10	19,20	1-8,13-15	1-8,13-15,19,20
R11	21	1-4,7,8	1-4,7,8,21
R12	22	1-4,8-12,16-18	1-4,8-12,16-18,22
R13	23,24,25	1-8,13-15	1-8,13-15,23,24,25
R14	26,27	1-8,13-15,19,20	1-8,13-15,19,20,26,27
R_{root}	$P_V - \{1-25\}$	1-25	P_V

Table 1. Transfer of privileges for role-graph of figure 2.

5.1. Role Graph Algorithms

We have developed algorithms to:

- add a role giving its direct privileges, expected juniors and seniors
- add a role giving effective privileges
- add/delete a privilege to/from a role
- add/delete an edge

Creation of the roles starting from the profiles

$\forall p_l \in P_L$
Create role r_{p_l}
assigned permissions to r_{p_l}
add r_{p_l} to $\Re$

Addition of the redundant roles in $\mathfrak{RR}$

$\forall r_1 \in \Re$
{
  $\forall r_2 \in \Re$
    {
      if $(\zeta(r_1) = \zeta(r_2))$ { Add r_1 et r_2 into $\Re$ $\Re$}
      if $(\zeta(r_1) \subset \zeta(r_2))$ { Add r_1 into $CR(r_2)$ }

    }
}

Creation of the heritage relation $\mathfrak{RR}$

$\forall r \in \Re$ {
  $\forall r_s i \in CR(r)$ {
    Add the heritage relation between r and $r_s i$
  }
  Delete the redundant permissions
}

8. Conclusion

In terms of access control, the question which often arises is : What are the users who have permission to access to a given service ?. Modeling the access control system in the form of oriented and hierarchical graph allows visualizing the impacts of the modifications of permissions of a role. The search for all possible ways in the graph makes it possible to filter the transfers of privileges. This research is perfectly feasible by using the algorithms of the theory of graphs.

The model is not of course complete, there is still again a lot of work in order to improve it. There are two essential extensions required to supplement SuperUser model in a UNIX environment: 1. the system file permissions must be modeled, 2. the links between files must be modeled too.

References

[1] D.F. Ferraiolo, R. Kuhn, R. Sandhu (2007), "RBAC Standard Rationale: comments on a Critique of the ANSI Standard on Role Based Access Control', IEEE Security & Privacy, vol. 5, no. 6 (Nov/Dec 2007), pp. 51-53.
[2] J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Formal Foundations for Hybrid Role Hierarchy," ACM Transactions in Information and Systems Security, in print for Nov. 2007.
[3] ISO/IEC 27001:2005, Requirements for Information security management systems, 2005.
[4] ISO/IEC 27002:2005, Code of practice for information security management, 2005.
[5] O. M. Sheyner. Scenario graphs and attack graphs, Thesis of School of Computer Science, Computer Science department, Carnegie Mellon University, Pittsburgh, PA, 2004.
[6] S. Jha, O. Sheyner and J. Wing, Two formal analyses of attack graphs, Computer Security Foundation Workshop, 2002.
[7] G. Vache, Towars Information System Security Metrics, European Dependable Computing Conference 7, Proceedings Supplemental Volume, pp 41-44, Kaunas, 7-9 May 2008.
[8] A..Ghadi, D. Mammass, M. Mignotte and A.Sartout, «Formalism of the access control model based on the Marked Petri Nets". International Journal of u- and e- Service, Science and Technology Vol.1, No.2, Mars, 2009.

[9] M Jaume and C Morisset. “A formal approach to implement access control”, .Journal of Information Assurance and Security, 2:137–148, 2006.
[10] Feng Xiaoning;Wang Zhuo; Yin Guisheng; “HierarchicalObject-Oriented Petri NetModelingMethod Based on Ontology” Internet Computing in Science and Engineering, 2008. ICICSE ’08. International Conference on 28-29 Jan . 2008 Page(s):553-556.
[11] S. Gavrila, J. Barkley, "Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management" (1998), Third ACM Workshop on Role-Based Access Control.
[12] J. Barkley, "Implementing Role Based Access Control Using Object Technology", First ACM Workshop on Role-Based Access Control (1995).
[13] P. Samarati1 and S. de Capitani di Vimercati, “Access Control: Policies, Models, and Mechanisms“, Foundations of Security Analysis and Design, Springer Berlin / Heidelberg, 2001 Page(s):137-196.
[14] D. Bell and L. LaPadula, ldquoSecure Computer Systems: unified Exposition and Multics Interpretation, rdquo Tech. Rep. MTR-2997,MITRE Co., July 1975.
[15] Paul Ammann, Duminda Wijesekera, and Saket Kaushik, “Scalable, Graph-Based Network Vulnerability Analysis,” Proceedings of the 9th ACMConference on Computer and Communications Security, 2002, pp. 217–224.
[16] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. Handbook of Graph Grammars and Computing by Graph Transformations. Vol. II: Applications, Languages, and Tools. World Scientific, 1999. 236,242.