An Adaptive Authorization Model Based on RBAC

Radu Constantinescu, Lucian Corlan
AES, Business Informatics Dept, ROMANIA
UTI Group, R&D Dept, ROMANIA
radu.constantinescu@ie.ase.ro,lcorlan@gmail.com

Abstract: In the article we present a data model and a possible implementation suited to allow proper access control in a system. In order to do that, we started from the extended RBAC model which is focused on roles which are associated to different functions existing in the system. The access control is implemented not just for some application's functionalities but also for granulated data details, like data attributes.

Keywords: authorization, access control models, roles, security, RBAC

1. Introduction

In the area of access control mechanisms, there are two well known classical models: Mandatory Access Control – MAC and Discretionary Access Control –DAC. DAC is defined in TCSEC as "a means of restricting access to objects based on the identity of the subjects and groups to which they belong". [12] The model is called discretionary because it is possible for a subject to pass a subset of his permissions, even if not directly, to any other subject. On the other hand MAC is defined in TCSEC as "a means of restricting access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity". [12] The sensitivity of the information is represented by labels and the formal authorization is also known as clearance. Those two models were for a long period of time considered to be the only patterns for access control models for till Role Based Access Control – RBAC was proposed and documented. Studies have demonstrated that RBAC is neither MAC nor DAC even if there are similar aspects that the model shares with both of them.
The RBAC model is focused on roles which are associated to different functions existing in systems or in organizations. The roles are associated to users depending on their responsibilities and/or qualification. Roles are also associated with permissions. The association of users and roles tends to change faster than the association between roles and permissions as the organizational frameworks tends to modify slower than the way in which users are allocated to tasks.

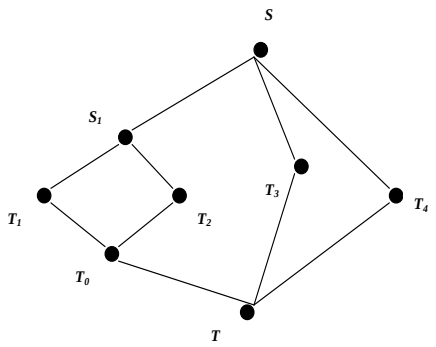
RBAC is also used as a foundation for complex access control mechanisms. RBAC is appropriate to be used in organizations that focus more on integrity than on confidentiality, so it is suitable for economical applications. It implements role hierarchies and constraints. The conceptual model is layered. The first layer is $RBAC_0$ which is the base model. On top of it are two independent layers $RBAC_1$ and $RBAC_2$ which are concerned with role hierarchies and respectively constrains. $RBAC_3$ consolidates both $RBAC_1$ and $RBAC_2$. On top of $RBAC_3$, security architects can define and implement customized levels for specific activities. [1]

2. Model Description

ISSN 2067 – 4074

The main components of RBAC model are: users (U), roles (R), permissions (P) and sessions (S). A user can be defined as a human being but in automated systems it also can represent an entity. Roles are job functions related to the organizations' or systems' particularities. A role implies, as we already mentioned, competency, responsibility and authority. Permissions are related to the specific functions inside organizations. A role can be assigned to one-to-many persons and also one person can have one-to-many roles. This is also valid for role-permissions relationship. A session is a mapping between a user and an activated subset of roles that are assigned to him. [2] In the basic model the relations defined are:
Permission assignment between permissions and roles: $PA \subseteq P \times R$
User assignment between users and roles: $UA \subseteq U \times R$
Session-user mapping: $user: S \rightarrow U$
Session-roles mapping: $roles: S \rightarrow 2^R$, $roles(s_i) \subseteq \{r | (user(s_i), r) \in UA\}$ which can change in time and session s_i has permissions $U_{r \in roles(s_i)} \{p | (p,r) \in PA\}$.
Regarding permissions, we distinguish between standard permissions and administrative permissions. Standard permissions refer to data and resource objects. Administrative permissions are required to modify the sets of users, roles, permissions and the relations between them. The administrative permissions are managed in a dual RBAC system. Sessions are in control of individual users. A user can create sessions and can choose to activate a subset of possible roles and then he can modify the active roles and even close the session.
$RBAC_1$ model introduces the role hierarchy concept. [7] This is used to structure the roles in order to reflect organizations' patterns. The most important roles stay in the top of the hierarchy inheriting permissions from the less important roles. This relation is a partially ordered relationship which means that it is reflexive, transitive and anti-symmetrical. In consequence, a role inherits its own permissions and the permissions of lower roles connected directly or indirectly with it and the bidirectional inheritance is denied in order to exclude redundancy. Therefore, $RBAC_1$ introduces a new relation: $RH \subseteq R \times R$, and the function roles is modified accordingly: $roles(s_i) \subseteq \{r | (\exists r' \geq r)[(user(s_i), r') \in UA\}$ and session s_i has the permissions $U_{r \in roles(s_i)} \{p | (\exists r'' \leq r) (p, r'') \in PA\}$.

There are many common cases in which is desired that some permission should remain available only to specific roles and to avoid inheritance to superior levels. This issue can be solved by defining private roles that are not inherited. This is represented in Figure 1. In the first graph is represented a role hierarchy that contains the roles T, T_0, T_1, T_2, T_3, T_4, S_1 and S. The role T is the base role for this example and is inherited by all the other roles. Role S is the top role. S_1 is a subproject role. In case there are permissions for S_1 that shouldn't be inherited by S, then the whole subproject area of the graph is replicated using private roles as in the second picture of Figure 1. [13]



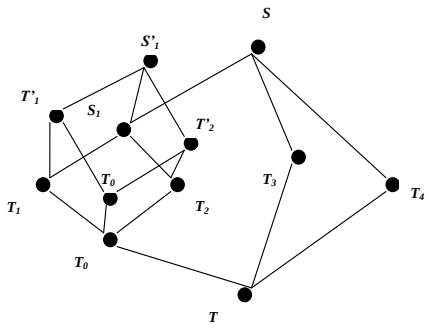


Fig. 1. Standard role hierarchy and private role hierarchy

$RBAC_2$ model introduces the concept of constrains which are mandatory for any organization. In the category of constrains we can mention the mutual exclusion of roles, cardinality restrictions or the prerequisite roles constraint. The restrictions should be simple in order to be easily implemented in an efficient way. Restrictions can be related to sessions and to the *roles()* and *user()* functions. Even though the role hierarchy is also a constraint, given its importance, it is discussed in the context of a different layer – $RBAC_1$. [3]
The consolidated model combines $RBAC_1$ and $RBAC_2$. There are some problems that arise from this association. The constraints in $RBAC_2$ could limit the number of superior or inferior roles that a role can have. There are also situations in which the inheritance could violate restrictions like mutual exclusion between roles. This issue can be addressed using private roles, as in Figure 1.

3. Formal Model

Given the variables of the model that were already discussed in the article, we will present the formal description of the basic properties of RBAC model. The properties are known in literature as: consistent subject, role assignment, role authorization, privilege authorization and role hierarchy. [4]
The consistent subject property states that for any subject *s* associated with user *u*, the authorized role set *R[s]* includes role *r* if and only if *u* is authorized for *r*:

$$(\forall s)(\forall u)(\forall r) \mid U[s] = u, u \in M[r] \Leftrightarrow r \in R[s] \quad (1)$$

Where $M[r]$ represents the set of users authorized for role *r* and $R[s]$ is the set of roles for which subject s is authorized.
The role assignment property states that a subject can execute a privilege only if the subject is assigned a role that is active at that moment:

$$(\forall s)(\forall p) \mid X[s, p] \Rightarrow AR[s] \neq \phi \quad (2)$$

Where $X[s, p]$ is the true if and only if subject *s* is able to execute permission *p* and $AR[s]$ is the set of active roles for *s*.
The role authorization property states that a subject's active role must be in the set of authorized roles for the subject:

$$(\forall s) \mid r \in AR[s] \Rightarrow r \in R[s] \quad (3)$$

ISSN 2067 – 4074

The privilege authorization property states that a subject can execute a certain permission only if the permission is assigned to the active role of the subject:

$$(\forall s)(\forall p)(\exists r) \mid X[s, p] \Rightarrow r \in AR[s] \wedge p \in PA[r] \quad (4)$$

Where $PA[r]$ is the set of permissions associated with role *r*.
The role hierarchy property states that a authorized role includes the permissions of the roles it inherits.

$$(\forall r, q)(\forall s) \mid (r \in AR[s] \wedge r \geq q \Rightarrow q \in AR[s]) \wedge (r \in R[s] \wedge r \geq q \Rightarrow q \in R[s]) \quad (5)$$

4. Solution design

We will use the extended RBAC model, developed from the formal model rules presented in section 4, in order to implement access control in a real application. We'll present a mechanism that offers the means to define, manage and verify the access rights of a user in the system. This means that users will be able to access only the application's resources they are authorized to.
The designed mechanism can be used for any type of system which needs authentication and authorization. More important, the modular characteristic of the solution allows for a suitable package of the solution to be delivered to a particular system, to meet performance requirements. As such, we can identify three types of systems:
systems which needs general access control to authorize the usage of certain functionalities;
systems which require data access control (besides general access control), in which the access is restricted for some information from the system according to some properties of the entities and users involved. An example of such a system would be an application which handles structured documents, with sensitive information, across some workflows.
systems which involves business rules in execution of their procedures and functionalities, and depending on the context of access control, those business rules are evaluated and the access is granted according to the result of this evaluation.
We can combine the requirements from these three types of systems to produce any desired combination. For example, an ERP system will require functionality authorization, data authorization and business rules driven authorization, in any combination, depending on the user request to access system resources in some particular context.
Two important principles will be applied:
proactive verification – when an user is authenticated in system, the interface will show only those resources that he can access according to his roles (area of competence principle);
reactive verification – when an user accesses a functionality of the application, he will access only the data he has the right to (necessity to know principle).
From a conceptual point of view, access control takes place at two levels: the f**unctionalities level** and the **data level**. By **functionalities level** we understand the domain specific actions that comprise the application's business logic. By **data level** we refer the data set which a user can access using a functionality of the system in concordance to his rights, and also to how fine-grained are the rights defined for an entity. [9], [10], [11]
Data level access control is required because applications use very sensitive data. This is why proper mechanisms to allow a flexible and fine-grained access must be in place.
In order to design the access control model we will use the following concepts**:**
Object – is an entity which copes with the business logic of an application and for which is necessary to give access rights.

Attribute – is a property of an entity, for which the access can be restricted in the context of a given functionality. In this stage, we distinguish the following levels of access: editable, readable and hidden. Those levels apply in the context of a user who is executing some operations that implies the entity. **Editable** means that the value of that attribute can be modified, if this is the case, according to the business logic of the functionality. **Readable** means that the value of the attribute is read-only, although it could be modified if the required rights exist. **Hidden** it means that the attribute will not be displayed at all.
Instance – represent a concrete representation of an entity at run-time. In special cases, special rights can be given for an instance, prevailing over other existing rights.
Constraint – is a business logic rule which specifies restrictions over the normal way of executing a given functionality. In other words, it defines the conditions which must be fulfilled in order to do that action. Constraints are defined using the entities attributes.
Operation – represents an abstract action in a particular context.
Functionality – represents the know-how required to execute an operation on an entity. It can be seen as a black box, which for some inputs, makes the necessary processing and returns an output.
We can represent this model based on a black box model, as in Figure 2.

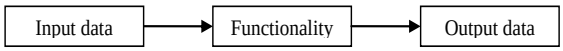


Fig. 2. Black box model

The access control will be implemented at 3 levels:
1. ***Input data level***: verification of right to execute that functionality for the specific input data. In this case, it is verified that the constraints are fulfilled for the entered values.
2. ***Functionality level***: verification of right to execute certain functionality.
3. ***Output data level***: verification of right to access data from the output set of data obtained after functionality execution. Here will be applied constraints for the output data.

Those constraints involve attributes like classification, location or document type. Permission represents the access rights in the system, which can be defined using the above concepts. The role (profile) – represents a set of permissions which comprise the necessary rights to execute a certain business logic activity.

5. Data model

We propose the following data model, as depicted in figure 3, which satisfies the requirements defined above. The relevant attributes for access control are detailed for each business logic entity. The attributes will be used to define constraints and to specify fine-grained access rights. Access at instance level can be granted at run-time for sensitive entities. The constraints for functionalities can be defined also using those attributes. Certain functionality is defined at designed time using operations and entities.

ISSN 2067 – 4074

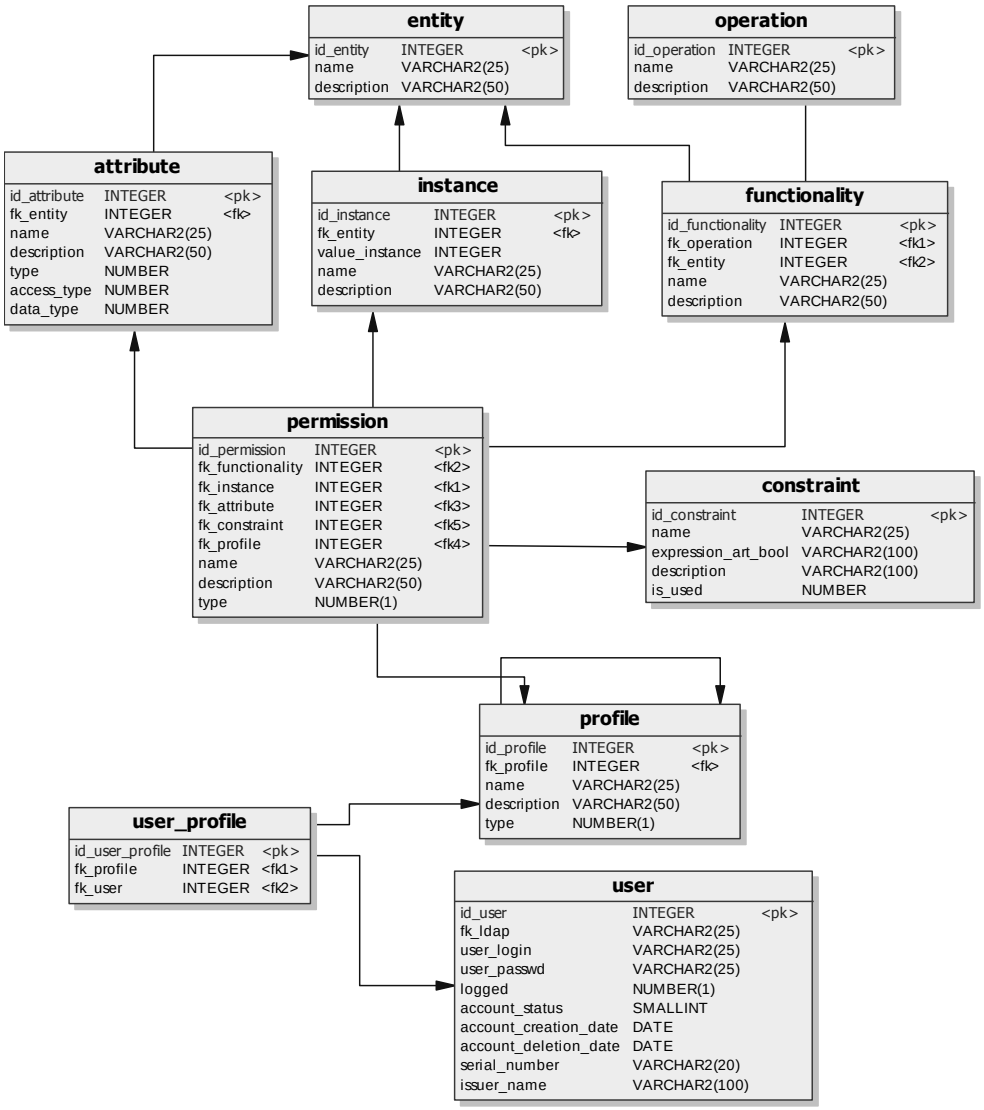


Fig. 3. Data model diagram

The use of operations is only to ease the administrative task of granting rights. A primary right is defined as a permission, and a set of permissions can be aggregated into a profile (role). The profile describes an area of competence

6. Implementation

The implementation of the authorization module based on the conceptual model implies two important aspects:

- Access validation to specific functionality in terms of *Yes* or *No*;
- Access validation to data used by the functionality (input and output).

Access validation for input data determines the execution of certain functionality in specific conditions on the basis of input data. Access validation for output data means filtering the data set obtained in the execution of a program's functionality. This implies the removal of data for which the access is denied.
To obtain a better behavior in terms of time and performance we suppose that:

1. Database information that refers to authorization module will be properly processed and cached.
2. Some tables will be populated at design time.

The steps in integration and usage of authorization module in the application are:

1. Establishing the methods for which is desired to restrict the access. For each method is assigned certain functionality. The methods will be accessed using application's GUI – menu, buttons or other controls.
2. The functionalities are included in the appropriate table – "functionality" table.
3. The association between functionalities and methods is saved in a file, eventually xml file. In other words, we mark that some methods represent certain functionalities even if not all the methods will have assigned one but will be used by other methods which represents functionalities.
4. For each method call it is verified if it has a functionality associated. In the affirmative case, the access right to that functionality is validated.
5. The authorization issues will be disjoined by the core of the functionality. Therefore, besides the Business Logic method, we'll introduce two more methods, one for the input data constraints and second for the output data constraints. The first method will be a standard one and implies loading and interpreting the input data constraints. It can be implemented using a rule engine or classical if-then statements. If the conditions are obeyed, the access to the functionality will be granted. The second method will be more flexible because it implies some processing work on SQL to inject some extra clauses. The purpose of this method is to filter the sensitive data from the returned set of data and by this the access to date will be restricted. The programmer will develop methods considering the possible use-case scenarios. The authorization module API will deliver only the constraints on output data for the specific functionality.
6. The authorization layer for the method call will identify the method as functionality and will validate if it is associated with access methods for input/output data. If the method uses input data the authorization layer will establish if the access is granted. If the method implies output data, then the authorization layer will create an interrogation based on constraints and then the method will be executed.

The overhead introduced by this module will be minimal, because the usage of the caching and optimization mechanism will ensure that no access to databases or file systems is required, and all the processing will be in the internal memory.

7. Conclusions

The purpose of this article is to describe relevant aspects of a possible implementation of RBAC in an application which works with sensitive information and needs both access to functionalities and granular access to data. The filtering is applied both at input data level and output data level. The optimization of application's performance is made also by caching data in tuned data structures. The security architecture is based on RBAC data model which was also briefed in the article.

A preliminary draft of this paper was presented the SECITC 2008 International Conference.

References

[1] D.F. Ferraiolo and D.R. Kuhn (1992) *"Role Based Access Control" 15th National Computer Security Conference*, Oct, 1992

[2] R. Sandhu, D.F. Ferraiolo, D, R. Kuhn "*The NIST Model for Role Based Access Control: Towards a Unified Standard*", NIST, 2000

ISSN 2067 – 4074

[3] D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli, "*Role Based Access Control*" (book), Artech House, 2003, 2nd Edition, 2007
[4] D.R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems" *Second ACM Workshop on Role-Based Access Control*, 1997
[5] D.F. Ferraiolo, J. Barkley, D.R. Kuhn, "A Role Based Access Control Model and Reference Implementation within a Corporate Intranet", *ACM Transactions on Information Systems Security*, Volume 1, Number 2, February 1999.
[6] Beznosov, Deng, Blakley, Burt, Barkley, "A Resource Access Decision Service for CORBA-based Distributed Systems", *ACSAC (Annual Computer Security Applications Conference)*, 1999
[7] R. Sandhu, D. Ferraiolo, R. Kuhn, "The NIST Model for Role Based Access Control: Towards a Unified Standard," *Proceedings, 5th ACM Workshop on Role Based Access Control*, July 26-27, 2000.
[8] R.Chandramouli, "Specification and Validation of Enterprise Access Control Data for Conformance to Model and Policy Constraints", *7th World Multi-conference on Systemics*, Cybernetics and Informatics, 2003
[9] R. Constantinescu, A. Barbulescu, "Systems Security through Capability Models", *"Competitiviness and European Integration" International Conference*, Cluj, Romania, Oct, 2007
[10] R. Constantinescu, F. Nastase, "Process Models for Security Architectures", *Informatics in Economy Journal*, no. 4, 2006
[11] R. Constantinescu, I. Ilie-Nemedi, "eBusiness Security" poster session, *12th Intel EMEA Academic Forum*, Budapest, 12-14 June 2007
[12] Department of Defense Standard, "*Trusted Computer System Evaluation Criteria*", 1985
[13] R. Sandhu, E. Coyne, H. Feinstein, "Role Based Acces Control Models", *IEEE Computer*, 1995