Function-Based Authorization Constraints Specification and Enforcement

Wei Zhou
Hasso-Plattner-Institute
University of Potsdam
D-14482 Potsdam, Germany
wei.zhou@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institute
University of Potsdam
D-14482 Potsdam, Germany
meinel@hpi.uni-potsdam.de

Abstract

Constraints are an important aspect of role-based access control (RBAC) and its different extensions. They are often regarded as one of the principal motivation behind these access control models. In this paper, we introduce two novel authorization constraint specification schemes named as prohibition constraint scheme and obligation constraint scheme. Both of them can be used for expressing and enforcing authorization constraints. These schemes strongly bind to authorization entity set functions and authorization entity relation functions, so they can provide the system designers a clear view about which functions should be defined in an authorization constraint system. Based on these functions, different kinds of constraint schemes can be easily defined. The security administrators can use these functions to create constraint schemes for their day-to-day operations. The constraint system can be scalable through defining new functions. This approach goes beyond the well known separation of duty constraints, and considers many aspects of entity relation constraints.

Keywords
Access control, authorization constraints, constraints specification, constraints enforcement

1. Introduction

Constraints are an important aspect of role-based access control and are powerful mechanism for laying out higher-level organizational policy. They have been part of most RBAC models of recent years [1, 2, 3, 4, 5, 6]. Separation of Duty (SoD) is widely considered to be a fundamental principle in computer security. Separation of duty is an important control principle in management whereby sensitive combinations of duties are partitioned between different individuals in order to prevent the violation of business rules. A very simple example of this is that cheques might require two different signatures. Giuri and Iglio [7] defined a formal model for constraints on role-activation. Gligor et al. [8] formalized separation of duty constraints enumerated informally by Simon and Zurko [9]. But with a few exceptions highlighted in related work section they have always been specified using rule-based systems. Unfortunately, rule-based systems, while highly expressive, are harder to visualize and thus to use; thus far they have been avoided by practitioners. A common claim is that such rule-based approaches underlie a higher-level expression, but currently there is still no useful approach to expressing and enforcing constraints. Another important constraint type is activity sequence related constraints. It has been recognized for a long time, but not been received much attention. An example of this is a user can be assigned to role *A* only when he/she is already a member of role *B*. There is still no dedicated work in this area.

In this paper we propose two authorization constraint specification schemes named as prohibition constraint scheme and obligation constraint scheme. Both of them can be used for expressing and enforcing authorization constraints. These schemes strongly bind to authorization entity set functions and authorization entity relation functions. New kinds of constraint schemes can be defined through defining new entity relation functions. On the other hand, this approach goes beyond the well known separation of duty constraints, and considers all the aspects of entity relation constraints including activity sequence related constraints.

The rest of this paper is organized as follows. Section 2 introduces the context for the specification scheme and enforcement model. Section 3 gives the definition of the constraint schemes. Section 4 describes the constraint schemes' evaluation. Section 5 compares our approach with other related work. Finally, Section 6 summarizes the results of this paper.

2. Background

In this section we will introduce the Team and Task-based RBAC (TT-RBAC) access control model that provides the context for the constraint specification schemes and enforcement model.

Role-based access control (RBAC) [4, 6] has emerged as a widely accepted alternative to classical discretionary and mandatory access controls. In RBAC, access rights are associated with roles, and users are assigned appropriate roles thereby acquiring the corresponding permissions. It provides more flexibility to security management over the traditional approach of using user and group identifiers.

We defined the TT-RBAC model through adding sets of two basic data elements called teams and tasks to the RBAC model. This model was first presented in [17]. TT-RBAC model element sets and relations are defined in Figure 1. Besides the relations defined in RBAC, the TT-RBAC model as a whole is fundamentally defined in terms of individual users being assigned to teams and roles, tasks being assigned to teams, and permissions being assigned to tasks. The relations among of them are many-to-many. In addition, the TT-RBAC model includes a set of sessions where each session is a mapping onto an activated subset of roles and an activated subset of teams that are assigned to the user. By virtue of team membership, users get access to team's resources specified by assigned tasks. However, for each user, the exact permissions he/she obtains to a team's resources will determined by his/her role and the current activity of the team.

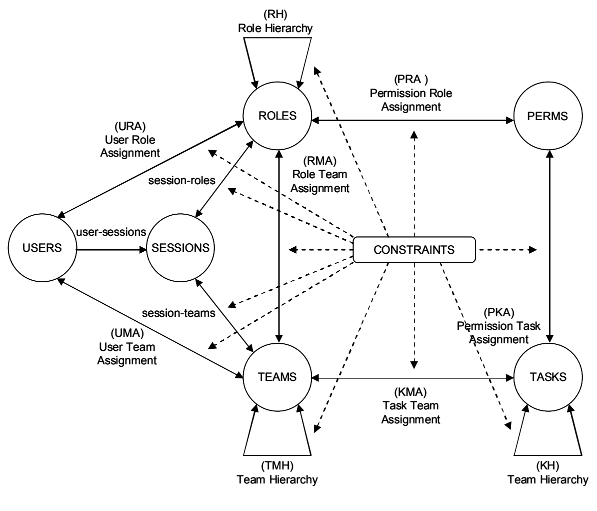

Figure 1. Constraints within TT-RBAC

In TT-RBAC model the added two new entities make the relations among the entities more complicated than in RBAC model. Besides the separation of duty constraint, some other types of constraints need to be considered. For example, at least five users should be assigned to a team, or each team must have one team header and two vice-team headers. We have developed a constraint language that considers all aspects of TT-RBAC constraints, not just those applying to role related constraints. In TT-RBAC the entity relations can happen among any entities. This approach goes beyond separation of duty to include obligation constraints.

3. Constraint schemes

Here we adopt the same classification for types of constraints as used in [13], i.e. the *prohibition constraints* and *obligation constraints*. The prohibition constraints are constraints that forbid the TT-RBAC component from being (or doing) something which it is not allowed to be (or do). Most of SoD constraints are included in this class. Obligation constraints are constraints that force the TT-RBAC component to do (or be) something. In some related literature, the obligation constraint is also called as prerequisite constraint. An example of obligation constraint is a user can be assigned to one role only when he/she is already member of some other roles. We develop two types of constraint schemes that are used to express and enforce prohibition constraints and obligations constraints, respectively.

3.1. Prohibition constraint scheme

The prohibition constraint scheme can be formatted as a triple (*S*, *C*, *T*), where *S* is the scope element that expresses how many entities defined in the scope set are applicable to each entity defined in the constraint set, *C* is the constraint element that expresses what kind of constraints will be applied to each entity defined in the scope set, and *T* is the constraint type and takes one of the following values: *SPC*, *DPC* and *HPC*, which denote *Static Prohibition Constraint*, *Dynamic Prohibition Constraint*, and *Historical Prohibition Constraint*, respectively.

The scope element *S* is further defined as a 4-tuple (*SS*, *SF*, *SO*, *SN*). *SS* is the scope set that needs to be constrained, for example, a user set. *SF* is the scope function that maps a value belonging to the constraint set defined in the constraint element to a set of values that have the same type with the scope set. For example, the function *assigned_role_users* maps a role to a set of users. *SO* is a relational operator that can be ">", ">=", "<", "<=", "=" or "$\neq$". *SN* as right operand of *SO* is a natural number. The (*SO*, *SN*) pair expresses the cardinality constraint to the scope set. If *SF* is not specified, then all the members in the scope set are applicable to the constraint element. An example of constraint scope element is ({u_1, u_2, u_3}, *assigned_role_users*, <, 3). The meaning of this example is that less than three users defined in the scope set can be assigned to a given role.

The constraint element *C* is further defined as a 4-tuple (*CS*, *CF*, *CO*, *CN*). *CS* is a constraint set that expresses the constraint entities applied to the scope set, for example, a role set. *CF* is constraint function that maps a value belonging to the scope set defined in the scope element to a set of values that have the same type with the constraint set. *CO* is a relational operator which can be ">", ">=", "<", "<=", "=" or "≠". *CN* as the right operand of *CO* is a natural number. The (*CO*, *CN*) pair expresses the cardinality constraint to the constraint set. An example of constraint element is ({r_1, r_2, r_3}, *assigned_user_roles*, <, 2). The meaning of this example is that less than two roles defined in the constraint set can be assigned to a given user.

In TT-RBAC, the scope set and constraint set are subsets of one of the following sets: *USERS*, *ROLES*, *PERMS*, *TEAMS*, *TASKS* and *OBJS*. Any constraints can be defined among these entity sets if the corresponding set functions and relation functions are defined. The entity set functions are used to get a set of entities, for example, *get_account_users* gets all the users in account department. The entity relation functions are used to obtain a set of entities associated to a given entity, for example, *assigned_user_roles* gets all the roles assigned to a given user. One prohibition constraint scheme example is shown as:

(({u_1, u_2, u_3}, *assigned_role_users*, <, 3), ({r_1, r_2, r_3}, *assigned_user_roles*, <, 2), *SPC*).

The meaning of this constraint scheme is that no user defined in the scope set {u_1, u_2, u_3} can be assigned to more than one role defined in the constraint set {r_1, r_2, r_3}, and less than three users defined in the scope set {u_1, u_2, u_3} can be assigned to any role defined in the constraint set {r_1, r_2, r_3}.

3.2. Obligation constraint scheme

The obligation constraint scheme can be formatted as a 4-tuple (*S*, *R*, *C*, *T*), where *S* is the scope element that defines the entities needs to be constrained, *R* is the request element that defines the entities requested by the entities defined in the scope element, *C* is the constraint element that expresses what kind of constraints will be applied to each entity defined in the scope set, and *T* is the constraint type and takes one of the following values: *SOC*, *DOC* and *HOC*, which denote *Static Obligation Constraint*, *Dynamic Obligation Constraint* and *Historical Obligation Constraint*, respectively.

The scope element *S* only contains the scope set *SS*, in which the entities need to be constrained, for example, a user set. The request element *R* only contains the request set *RS*, in which the entities that the entities defined in scope set want to be assigned or activate, for example, a role set. The constraint element *C* has the same structure as in the prohibition constrain scheme.

In TT-RBAC, the scope set, request set and constraint set are a subset of one of the following sets: *USERS*, *ROLES*, *PERMS*, *TEAMS*, *TASKS* and *OBJS*. One obligation constraint scheme example is shown as:

({u_1, u_2, u_3}, {r_3, r_4}, ({r_1, r_2}, *assigned_user_roles*, >, 1), *SOC*).

The meaning of this constraint scheme is that any user defined in the scope set {u_1, u_2, u_3} can be assigned to any role defined in the request set {r_3, r_4} if and only if he/she is already be assigned to more than one role defined constraint set {r_1, r_2}.

4. Constraint scheme evaluation

The prohibition constraint schemes and obligation constraint schemes are not only used for expressing authorization constraints, but also used for enforcing authorization constraints. In this section, we investigate how these authorization constraint schemes are evaluated at runtime.

4.1. Functions defined for constraint scheme evaluation

The constraint request can be expressed as: (*s*, *o*, *a*), where *s* is the constraint request subject, *o* is the constraint request object, and *a* is the constraint request action. For example, (u_1, r_1, *RUA*) is a role-user assignment request. The *RUA* is the request action that denotes the action of assigning a role to a user. We also define three functions *CRS*, *CRO* and *CRA* used to get constraint request subject, object and action, respectively.

As a general notational device we have the following convention. For any set valued function *f* defined on set *X*, We understand

$$f(X) = f(x_1) \cup f(x_2) \cup \ldots \cup f(x_n), \text{ where } X = \{x_1, x_2, \ldots, x_n\}$$

For example, suppose we want to get all roles that are assigned to a set of users $U = \{u_1, u_2, u_3\}$. We can express this using the function *assigned_user_roles*(*U*) as equivalent to

$$assigned\_user\_roles(u_1) \cup assigned\_user\_roles(u_2) \cup assigned\_user\_roles(u_3).$$

The functions related to the constraint scheme evaluation are defined as follows.

- *SSF*(*X*) is the scope set function that maps a set value *X* to another set value that have the same type with the scope set. At runtime, the *SSF* is replaced by the constraint scheme scope set function, and the value *X* is replaced by the constraint scheme constraint set. For example, $assigned\_role\_users(\{r_1, r_2\}) = \{u_1, u_2, u_3\}$.
- *CSF*(*x*) is the constraint function that maps a value to a set value that have the same type with the constraint set. At runtime, the *CSF* is replaced by the constraint scheme constraint function, and the value *x* is replaced by the constraint request object. For example, $assigned\_user\_roles(u) = \{r_1, r_2\}$.
- *EN*(*X*) is a function to get the element number of a set value *X*. For example, $EN(\{r_1, r_2, r_3\}) = 3$.

4.2. Prohibition constraint scheme evaluation

The prohibition constraint scheme evaluation comprises two parts. One is the *scope element evaluation* that is composed of scope element applicable check and scope element cardinality check. Another is the *constraint element evaluation* that is composed of constraint element applicable check and constraint element cardinality check. When ((*SS*, *SF*, *SO*, *SN*), (*CS*, *CF*, *CO*, *CN*), *T*) is the prohibition constraint scheme and *q* is the constraint request, the prohibition scope element evaluation can be formulated as:

$$ESE(q) = \begin{cases} CRS(q) \in SS & \text{if } SF = null \\ CRS(q) \in SS \wedge EN((SSF(CS) \cup CRS(q)) \cap SS) & \\ \text{satisfy}(SO, SN) & \text{if } SF \neq null \end{cases}.$$

The *ESE* is the function for the scope element evaluation. There are two cases for this evaluation. In the case of scope function is null, we only need to do the scope element applicable check. The scope element is "Applicable" if and only if the constraint request subject is defined in the scope set; otherwise the scope element is "NotApplicable". In the case of scope function is not null, besides the scope element applicable check, we need to check if it still satisfies the scope set cardinality constraint after the object is assigned to the subject or activated by the subject. If so, *ESE* returns the value of "Permit", otherwise returns the value of "Deny".

The prohibition constraint element cardinality check can be formulated as:

$$ECE(q) = CRO(q) \in CS \wedge EN((CSF(CRS(q)) \cup CRO(q)) \cap CS)\, \text{satisfy}(CO, CN).$$

The *ECE* is the function for the constraint element evaluation. For the constraint element evaluation, we need to do the constraint element applicable check and constraint element cardinality check. The constraint element is "Applicable" if and only if the constraint request object is defined in the constraint set; otherwise the constraint element is "NotApplicable". If the constraint element is applicable, the *ECE* then checks if it still satisfies the constraint set cardinality constraint after the object is assigned to the subject or activated by the subject. If so, *ECE* returns the value of "Permit", otherwise returns the value of "Deny".

A prohibition constraint scheme evaluation result is "Permit" if and only if both scope element cardinality check and constraint element cardinality check return "Permit". The prohibition constraint scheme evaluation can be formulated as:

$$EPC(q) = ESE(q) \wedge ECE(q).$$

The *EPC* is the function for the prohibition constraint scheme evaluation.

4.3. Obligation constraint scheme evaluation

The obligation constraint scheme evaluation comprises three parts. The first is the *scope element evaluation* that is composed of scope element applicable check. The second is the *request element evaluation* that is composed of request element applicable check. The third is the *constraint element evaluation* that is composed of constraint element cardinality check. When (*SS*, *RS*, (*CS*, *CF*, *CR*, *CN*), *T*) is the obligation constraint scheme and *q* is the constraint request, the obligation scope element evaluation can be formulated as:

$$ESE(q) = CRS(q) \in SS.$$

The *ESE* is the function for the scope element evaluation. Here it only needs to do the scope element applicable check. The scope element is "Applicable" if and only if the constraint request subject is defined in the scope set; otherwise the scope element is "NotApplicable".

The obligation request element evaluation can be formulated as:

$$ERE(q) = CRO(q) \in RS \cdot$$

The *ERE* is the function for the request element evaluation. Here it only needs to do the request element applicable check. The request element is "Applicable" if and only if the constraint request object is defined in the request set; otherwise the request element is "NotApplicable".

The obligation constraint element evaluation can be formulated as:

$$ECE(q) = EN((CSF(CRS(q)) \cup CRO(q)) \cap CS) \text{satisfy}(CO, CN) \cdot$$

The *ECE* is a function for the constraint element evaluation. Here it only needs to do the constraint element cardinality check. It checks if the constraint element still satisfies the constraint set cardinality constraint after the object is assigned to the subject or activated by the subject. If so, *ECE* returns the value of "Permit", otherwise returns the value of "Deny".

An obligation constraint scheme evaluation result is "Permit" if and only if scope element applicable check and request element applicable check return "Applicable" and constraint element cardinality check return "Permit". The obligation constraint scheme evaluation can be formulated as:

$$EOC(q) = ESE(q) \wedge ERE(q) \wedge ECE(q)$$

The function *EOC* is the function for the obligation constraint scheme evaluation.

4.4. Constraint schema

The constraint schema is the basic unit of managing constraint schemes and checking constraint requests. Each constraint schema manages a set of constraint schemes. These constraint schemes can be in different types introduced in previous sections. At runtime a constraint request is sent to a constraint schema for evaluation. The schema will check this constraint request against to all its constraint schemes. If any of these constraint schemes is evaluated to "Deny", then the evaluation result of this schema is "Deny".

Now we illustrate how to use prohibition constraint schemes and obligation constraint schemes to construct a constraint schema through an example. The application scenario is: all the users (*U*) with the role "Staff" can be assigned with the roles "President" and "Vice-President", there is only one user can be assigned with the role "President", there are no more than two users can be assigned with the role "Vice-President", and "President" and "Vice-President" are mutually-exclusive roles. In order to implement these constraints, the corresponding constraint schema should include the following schemes:

(*U*, {*President*, *Vice-President*}, ({*Staff*}, *assigned_user_roles*, >, 0), *SOC*);

((*U*, *assigned-role-user*, <, 2), ({*President*}, *assigned-user-roles*, <, 2), *SPC*);

((*U*, *assigned-role-user*, <, 3), ({*Vice-President*}, *assigned-user-roles*, <, 2), *SPC*);

((*U*, , ,), ({*President*, *Vice-President*}, *assigned-user-roles*, <, 2), *SPC*).

5. Related work

Nyanchama and Osborn [10] define a graphical model for role-role relationships that includes a combined view of role inheritance and separation of duty constraints based on roles. Osborn and Guo [11] extended the model to include constraints involving users. However, neither the basic model nor the extended model distinguishes between accidental relationships and explicitly constructed relationships. Thus, these models do not support policies with a historical component.

Ahn and Sandhu [12] propose a limited logical language called RCL 2000 for expressing separation of duty constraints in a RBAC model. RCL 2000 still provides significant expressive power, but remains quite complex. The combination of quantification functions and modelling concept functions makes the constraints expressed in the language difficult to visualize. Thus, this approach is an improvement over a completely general logical language, but it is still too complex.

Jaeger and Tidswell [14] proposed an approach to expressing constraints in graphical. An access control policy is expressed using a graphical model in which the nodes represent sets (e.g., of subjects, objects, etc.) and the edges represent binary relationships on those sets and constraints are expressed using a few, simple set operators on graph nodes. This model has been designed to be applicable in a general access control model, not just in role-based access control models.

Bacon et al. [15] defined two kinds of rules in OASIS, which can be used to express entity sequence related constraints in RBAC. But these rules are not specifically defined for this purpose, they also considers other facts such as environmental predicates.

The OASIS approach also belongs to the rule-based systems.

Crampton [16] proposed a separation of duty constraints specification language for role-based access control systems. The specification model is set-based and has a simpler syntax than existing approaches. Easy to understand is addressed by this proposal. But this specification language cannot specify those constraints that are based on the aggregation of users and permissions with quantification over sets and members of sets. This shortcoming will limit its usages in many cases.

The major difference between our approach and other approaches mentioned previous is that we strongly use entities relation functions to define the entity sets relation in the constraint schemes. The advantages of such an approach are: (1) providing a clear view about the relation between the constrained entities and the constraint entities, (2) giving a clear view about which kinds of entity set functions and entity relation functions should be defined in an authorization constraints enforcement system, (3) based on the basic entity set functions and entity relation functions, different constraint schemes can be easily defined, and (4) new relation functions are defined, then new types of constraints can be expressed and enforced.

6. Conclusion

The major contributions of this paper are to provide two novel specification schemes for expressing and enforcing RBAC and TT-RBAC constraints. To our knowledge ours is the first constraint specification language that is used for both expression and enforcement aims. These constraint schemes strongly bind to some functions, so they can provide the system designers and security administrators a clear view about which functions should be defined in an authorization constraints enforcement system. Based on these functions, different kinds of constraint schemes can be easily defined, and then enforced. We believe that our approach is far simpler to understand, has a much less cumbersome syntax and more near the real world. Moreover, implementation can be based on the functions that already exist in a system, and the authorization constraints system is scalable through adding new functions.

7. References

[1] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems", ACM Trans. Inf. Syst. Sec. (TISSEC) 1, 2 (Feb), 1999.

[2] E. Bertino, S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A unified framework for enforcing multiple access control policies", in Proceedings of ACM SIGMOD Conference on Management of Data, May 1997.

[3] E. C. Lupu, and M. Sloman, "A policy based role object model", in Proceedings of the 1st IEEE Enterprise Distributed Object Computing Workshop, Calif, Oct. 1997.

[4] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models", IEEE Computer, vol. 29, pp. 38-47, Feb. 1996.

[5] R. S. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles", ACM Trans. Inf. Syst. Sec. 1, 2, Feb. 1999.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control", ACM Transactions on Information and System Security, vol. 4, pp. 224-274, Aug. 2001.

[7] L. Giuri, and P. Iglio, "A formal model for role-based access control with constraints", in Proceedings of 9th IEEE Workshop on Computer Security Foundations, pp. 136–145, Kenmare, Ireland, June 1996.

[8] V. D. Gligor, S. Gavrila, and D. Ferraiolo, "On the formal definition of separation of duty policies and their composition", in Proceedings of the 1998 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 172–183, Oakland, CA, May 1998.

[9] R. Simon, and M. E. Zurko, "Separation of duty in role based access control environments", in Proceedings of the 10th IEEE Workshop on Computer Security Foundations, pp. 183–194, Rockport, MA, June 1997.

[10] M. Nyanchama and S. Osborn, "The role graph model and conflict of interest", ACM Trans Inf. Syst. Sec. 2, 1 (Feb.), 1999.

[11] S. Osborn and Y. Guo, "Modelling users in role-based access control", in Proceedings of the 5th ACM Role-Based Access Control Workshop, July 2000.

[12] G. Ahn and R. Sandhu, "Role-based authorization constraint specification", ACM Trans. Inf. Syst. Sec. 3, 4 (Nov.), 2000.

[13] G. Ahn, "The RCL 2000 language for specifying role-based authorization constraints", Ph.D. Dissertation, George Mason Univ., Fairfax, VA, 2000.

[14] T. Jaeger and J. Tidswell, "Practical safety in flexible access control models", ACM Transactions on Information and System Security 4, 2, pp. 158–190, 2001.

[15] J. Bacon, K. Moody and W. Yao, "A Model of OASIS Role-Based Access Control and Its Support for Active Security", ACM Transactions on Information and System Security, Vol. 5, No. 4, Pages 492–540, November 2002.

[16] J. Crampton, "Specifying and enforcing constraints in role-based access control", in Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003), pages 43–50, Como, Italy, June 2003.

[17] W. Zhou, V. H. Raja, C. Meinel, M. Ahmad. A Framework for Cross-Institutional Authentication and Authorisation. In Proceedings of the eChallenges e-2005 Conference(e-2005), P. 1259-1266, Ljubljana, Slovenia, October 2005.