Reasoning about Multiple Access Control Configurations

Supakit Dangprasert and Yongyuth Permpoontanalarp
Logic and Security Laboratory
Department of Computer Engineering, Faculty of Engineering,
King Mongkut's University of Technology Thonburi,
91 Pracha-Uthit Road, Bangkok 10140, Thailand
Tel. +66-2-470-9087, Fax.: +66-2-872-5050

e-mail : supakitd@yahoo.com, yongyuth@cpe.eng.kmutt.ac.th

Abstract: At present, many applications independently provide access control for their own resources, for example Web, Databases and Operating Systems, etc. Such independent access control systems result in multiple access control configurations each of which deals with the access control in its own application context. Since those multiple configurations are *operated in isolation*, and *maintained by possibly different administrators*, they are likely to be *incoherent*. In this paper, we propose a logical specification to reason about multiple access control configurations. Our specification can be used to detect the incoherence in multiple configurations. Furthermore, it offers many kinds of policies for multiple configurations that can capture several kinds of requirements for multiple access control systems.

1. Introduction

Access control is concerned with the restriction of accesses to resources in a computer system. In the past, the access control was available only at Operating Systems (OS) level. Such access control aimed mainly to restrict accesses to *file objects* which are the main resource of OS. And the access control configurations at those days were simple to manage.

Nowadays, computer systems become more complex due to the new development of Internet and Database technologies. Thus, there are many new kinds of resources available in a compute system due to the various new kinds of applications, eg. Database and Internet.

Each of these applications independently provides access control for their own resources, for example Access control for Web, Access control for Databases, Access control for OS, etc. Such independent access control systems result in multiple access control configurations each of which deals with the access control in its own application context. Since those multiple configurations are *operated in isolation*, and *maintained by possibly different administrators*, they are likely to be *incoherent*.

In this paper, we study multiple *incoherent* configurations for Role-based Access Control (RBAC) [1]. RBAC is an access control which considers users' job functionality, ie. roles, in order to make the decision to grant an access. In general, an RBAC model contains sets of users, roles and permitted operations. Moreover, such an RBAC model includes the assignment of a user to a role, the assignment of a permitted operation to a role and role hierarchy. The role hierarchy expresses the supervision relationship between roles in an organization, for example an engineer is supervised by a manager of engineering division. Many applications are adopting RBAC as their access control systems.

Such incoherently access control configurations could create two main problems, namely, the *inconsistency* and the *non-coordination* of multiple configurations. In the context of RBAC, there are three main sources of the inconsistency, namely, *the assignment of a user to a role, the assignment of a permitted operation to a role*, and *the role hierarchy*.

The assignment of a user to a role in those configurations is inconsistent if a user is assigned to different roles in different configurations. For example, in Database configuration *John* is assigned to *Staff* role, but in OS configuration *John* is assigned to *Anonymous* role. As a result, *John* who should be able to access a database is denied to gain such access, since *Anonymous* role is not allow to log on a machine.

Similarly, *the assignment of a permitted operation to a role* in multiple configurations is inconsistent if a permitted operation is not assigned to the same role which appears in different configurations. For example, in OS (client-server) configuration *Lecturer* role is allowed to access a shared folder, but in Intranet (or Internet) configuration the *Lecturer* role is not allowed to access the folder.

Role hierarchy in those configurations is defined inconsistently if the order of any two roles is different in the configurations.

As a result of such inconsistently defined configurations, a user may not be able to access a resource that the user should, or a user may be able to access a resource that the user should not.

In order to allow a user to access a resource in a system with multiple applications, it requires that configurations for those applications are defined in a *coordinated* way. For example, suppose that *Manager* role is allowed to access *salary* table. Thus, it must be the case that *Manager* role is given *select* access to the table and is allowed to *logon* at any machines in the system. As a result of the non-coordinated configurations, a user may not be able to access a resource that the user should.

In order to deal with these problems, we propose a logical specification to reason about multiple access control configurations. Our specification can be used to detect both the inconsistency and the non-coordination of multiple configurations. Furthermore, it offers many kinds of policies for multiple configurations that can capture several kinds of requirements for multiple access control systems.

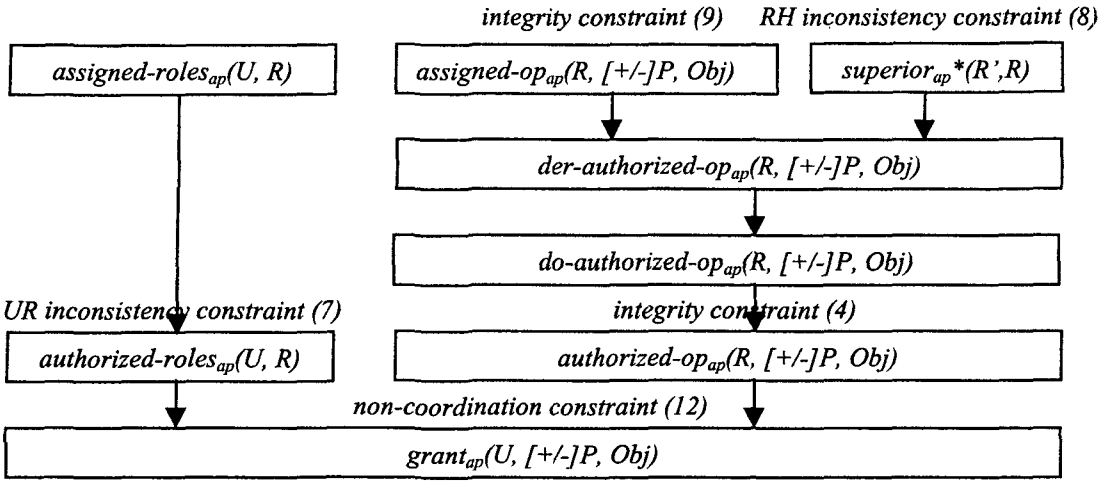

Figure 1. The relationship between predicates and the use of integrity constraints for each predicate

2. Our specification

Our specification extends the logical specification for general access control [4], and that for RBAC [2] to deal with reasoning about multiple configurations. Our specification consists of many sub-specifications where each of the sub-specifications is defined for an application.

In each sub-specification, there are many rules where each rule aims to derive a predicate. For application *ap*, *assigned-roles*$_{ap}$*(U, R)*, *assigned-op*$_{ap}$*(R,+P, O)*, *superior*$_{ap}$ *(R',R)*, *der-authorized-op*$_{ap}$*(R, +P, O)*, *do-authorized-op*$_{ap}$ *(R, +P, O)*, *authorized-op*$_{ap}$*(R, +P, O)*, *authorized-roles*$_{ap}$ *(U, R)* and *grant*$_{ap}$*(U, P, O)* are all predicates used in each sub-specification. Figure 1 illustrates the relationship between predicates and the use of integrity constraints for each predicate. In particular, an arrow from a predicate to another predicate indicates that the former predicate is used to derive the latter predicate. And an integrity constraint written above a predicate means that the constraint is used for the predicate.

In each sub-specification, predicates *assigned-roles*$_{ap}$*(U, R)*, *assigned-op*$_{ap}$*(R,+P, O)* and *superior*$_{ap}$*(R',R)* stand for the assignment of user *U* to role *R*, the assignment of permitted operation *P* on object *O* to role *R* and, and the superiority of role *R'* over *R* in the role hierarchy for application *ap*, respectively. In other words, *superior*$_{ap}$ *(R',R)* means that *R'* supervises *R*. In fact, these predicates represent an *abstract configuration* defined for an application.

Predicate *der-authorized-op*$_{ap}$*(R, +P, Obj)* in (1) shows the propagation of authorizations to roles in the role hierarchy. It is used in the following rule. Note that as a matter of notation, predicate symbols begin with lower case letter whereas variable symbols begin with upper case letter and they are implicitly universally quantified.

$$\begin{aligned} &\textit{der-authorized-op}_{ap}(R, +P, Obj) \leftrightarrow \\ &\qquad \exists R' \in R_{ap} \; [\; \textit{assigned-op}_{ap}(R', +P, Obj) \wedge \\ &\qquad \textit{superior}_{ap}{}^{*}(R, R') \quad] \\ &\textit{der-authorized-op}_{ap}(R', -P, Obj) \leftrightarrow \\ &\qquad \exists R' \in R_{ap} \; [\; \textit{assigned-op}_{ap}(R, -P, Obj) \wedge \\ &\qquad \textit{superior}_{ap}{}^{*}(R, R') \quad] \end{aligned} \tag{1}$$

where R_{ap} stands for a set of roles for application ap.

The propagation of *positive* authorization for role-permission is *upward* whereas the propagation of *negative* authorization for role-permission is *downward.*

Predicate *do-authorized-op*$_{ap}$*(R, +P, O)* deals with the conflict resolution between positive and negative authorizations. For simplicity, we can use an existing conflict resolution rule defined in [4]. Note that in this paper, we focus on multiple configurations rather than the conflict resolution. However, it requires that the following constraint holds for the predicate.

$$\begin{aligned} &(\textit{do-authorized-op}_{ap}(R, +P, O) \wedge \\ &\quad \textit{do-authorized-op}_{ap}(R, -P, O)) \rightarrow \quad \textit{false} \end{aligned} \tag{2}$$

Predicate *authorized-op*$_{ap}$*(R, +P, O)* is used for the propagation of authorizations across applications. Indeed, we use rules (3), (4) and (5) defined for the predicate to deal with the inconsistency of the assignment of operations to roles.

Such rules state three kinds of policies : open, close and neutral policies. We argue that these three kinds of policies are useful for expressing requirements for multiple configurations.

Rule 3 captures the *open* policy which states intuitively that if an operation is allowed for a role in *at least* an application, then the operation is allowed for the role in all other applications in which the operation is not disallowed explicitly.

$$\begin{aligned} &\textit{authorized-op}_{ap'}(R, +P, O) \leftrightarrow \exists ap \\ &\qquad (\textit{do-authorized-op}_{ap}(R, +P, O) \wedge \\ &\qquad O \in Obj_{ap'} \wedge R \in R_{ap'} \wedge P \in OP_{ap'} \wedge \\ &\qquad \neg \textit{do-authorized-op}_{ap'}(R, -P, O)) \\ &\textit{authorized-op}_{ap}(R, -P, O) \leftrightarrow \\ &\qquad \textit{do-authorized-op}_{ap}(R, -P, O) \end{aligned} \tag{3}$$

where *ap* and *ap'* stand for any applications, and $Obj_{ap'}$ and $OP_{ap'}$ stand for sets of objects and operations, respectively, in application *ap'*.

Rule 4 which captures the *close* policy states intuitively that if an operation is not allowed for a role in *at least* an application, then the operation is not allowed for the role in all other applications in which the operation is allowed explicitly.

$$\begin{aligned}&\textit{authorized-op}_{ap'}(R, -P, O) \leftrightarrow \exists ap \\ &\qquad (\textit{do-authorized-op}_{ap}(R, -P, O) \wedge \\ &\qquad O \in \textit{Obj}_{ap'} \wedge R \in R_{ap'} \wedge P \in \textit{OP}_{ap'} \wedge \\ &\qquad \neg\textit{do-authorized-op}_{ap'}(R, +P, O)) \\ &\textit{authorized-op}_{ap}(R, +P, O) \leftrightarrow \\ &\qquad \textit{do-authorized-op}_{ap}(R, +P, O) \qquad (4)\end{aligned}$$

Rule 5 expresses the *neutral* policy which states that if an operation is neither allowed nor disallowed explicitly for a role in an application, then the operation is either allowed or disallowed for the role in the application. This policy is neutral in that a configuration for an application does not have any effect on another configuration for an application.

$$\begin{aligned}&(\neg\textit{do-authorized-op}_{ap}(R, +P, O) \wedge \\ &\neg\textit{do-authorized-op}_{ap}(R, -P, O)) \rightarrow \\ &\qquad (\textit{authorized-op}_{ap}(R, +P, O) \vee \\ &\qquad \textit{authorized-op}_{ap}(R, -P, O)) \\ &\textit{do-authorized-op}_{ap}(R, +P, O) \rightarrow \textit{authorized-op}_{ap}(R, +P, O) \\ &\textit{do-authorized-op}_{ap}(R, -P, O) \rightarrow \textit{authorized-op}_{ap}(R, -P, O) \\ &\qquad (5)\end{aligned}$$

Note that the open and close policy for multiple applications are similar to those for single application in that they either maximize or minimize the accessibility in the system.

Predicate $\textit{authorized-roles}_{ap}(U,R)$ expresses the propagation of authorized roles for a user according to the role hierarchy. The following shows a rule for this predicate, and it is similar to that in [2].

$$\begin{aligned}&\textit{authorized-roles}_{ap}(U,R) \leftrightarrow \\ &\qquad \exists R' \in R_{ap} [\textit{assigned-roles}_{ap}(U,R') \wedge \\ &\qquad \textit{superior}_{ap}{}^*(R',R) \quad] \qquad (6)\end{aligned}$$

where $\textit{superior}_{ap}{}^*(R',R)$ represents the reflexive and transitive closure of $\textit{superior}_{ap}$.

We use the constraint (7) on predicate *authorized-roles*$_{ap}$ to deal with the inconsistency of the assignment of users to roles. In particular, the following constraint states that if user U is authorized for role R in application *ap1* and U and R are recognized in application *ap2*, then U must be authorized for R in *ap2* also.

$$\begin{aligned}&\forall U \in U_{ap1} \ \forall R \in R_{ap1} \\ &(\textit{authorized-roles}_{ap1}(U, R) \wedge U \in U_{ap2} \wedge R \in R_{ap2} \rightarrow \\ &\qquad \textit{authorized-roles}_{ap2}(U, R)) \\ &\qquad (7)\end{aligned}$$

where U_{ap1} and U_{ap2} stands for sets of users in applications *ap1* and *ap2*, respectively.

We use the following constraint to deal with the inconsistency of the role hierarchy.

$$\begin{aligned}&\forall RH_{ap1}, RH_{ap2} \ \forall R,R' \\ &(R \in R_{ap1} \wedge \textit{superior}_{ap1}{}^*(R, R') \wedge R,R' \in R_{ap2} \rightarrow \\ &\qquad \textit{superior}_{ap2}{}^*(R, R')) \\ &\qquad (8)\end{aligned}$$

where RH_{ap1}, RH_{ap2} stands for role hierarchy in application *ap1* and application *ap2*, respectively.

The following is a general constraint on the configuration.

$$\begin{aligned}&(\textit{assigned-op}_{ap}(R, +P, Obj) \wedge \\ &\textit{assigned-op}_{ap}(R, -P, Obj)) \rightarrow \textit{false} \qquad (9)\end{aligned}$$

where *Obj* stands for set of objects.

Predicate $\textit{grant}_{ap1}(U, P, O)$ indicates that user U is granted operation P on object O in application *ap1*. The rule for the predicate *grant* for a certain application is defined as follows.

$$\begin{aligned}&\forall U \in U_{ap} \ \forall P \in Op_{ap} \ \forall Obj \in Obj_{ap} \\ &[\textit{grant}_{ap}(U, [+/-]P, Obj) \leftrightarrow \\ &\qquad \exists R \in R_{ap} (\textit{authorized-roles}_{ap}(U, R) \wedge \\ &\qquad \textit{authorized-op}_{ap}(R, [+/-]P, Obj)) \ ] \qquad (10)\end{aligned}$$

where U_{ap}, Op_{ap}, Obj_{ap} stands for set of users, operations and objects in application *ap*, respectively

The following shows a rule for predicate grant which is independently of any application.

For applications *ap1*, ..., *apn*

$$\begin{aligned}&\textit{grant}(U, P, Obj) \leftrightarrow \\ &\qquad [\textit{grant}_{ap1}(U, P, Obj) \vee \textit{grant}_{ap2}(U, P, Obj) \vee \ldots \vee \\ &\qquad \textit{grant}_{apn}(U, P, Obj) \ ] \qquad (11)\end{aligned}$$

where *Obj* stands for set of objects

To deal with the non-coordination problem, we employ the concept of *application-level-based supported authorizations*. In the concept, an application in a lower level provides necessary supported authorizations to another application in an upper level. In other words, the concept shows *supported* authorizations that are required by an application and are provided by other *supported* applications. Consider the non-coordination example discussed above. Database will be located on top of OS. The supported authorization, provided by OS, which allows *Manager* role to access a table is to allow *Manager* to log on to a machine in the system.

The concept is formalized and is represented in its simple form by constraint (12). Intuitively, constraint (12) expresses that in order to allow user U to perform operation P on object O in application *ap*, it requires supported authorizations, eg. $Q'$and Q'' from supported applications $ap_1 \ldots ap_n$.

$$\begin{aligned}&\textit{grant}_{ap}(U, P, O) \rightarrow \\ &\qquad (\textit{grant}_{ap1}(U, Q', O') \wedge \ldots \wedge \textit{grant}_{apn}(U, Q'', O'')) \\ &\qquad (12)\end{aligned}$$

Recall the non-coordination example discussed in the introduction. The constraint for the non-coordination example there is as follows.

$grant_{db}(U, +select, Db\text{-}table) \rightarrow$
$\exists M \in Machine\ grant_{os}(U, +logon\text{-}locally, M)$ (12')

where *Machine* stands for a set of machine from which an authorized user can access *Db-table*.

We argue that the use of such supported authorizations concept provides an intuitive and simple way to construct constraints to deal with the non-coordination.

In the following, we show how our specification can be used to deal with the inconsistency and the non-coordination.

Example 1 The inconsistent assignment of user to role

Recall the example of the inconsistent assignment of users to role discussed in the introduction. Suppose that $\{Staff, Anonymous\} \subseteq R_{ap1}, R_{ap2}$, and $John \in U_{ap1}, U_{ap2}$. Given the following configuration

$assigned\text{-}roles_{os}(john, anonymous)$
$assigned\text{-}roles_{db}(john, staff)$
$\neg assigned\text{-}roles_{db}(john, anonymous)$
$\neg assigned\text{-}roles_{os}(john, staff)$

$authorized\text{-}roles_{os}(john, anonymous)$ and $\neg authorized\text{-}roles_{db}(john, anonymous)$ are derived by rule (6), and thus constraint (7) does not hold. So, the logical inconsistency is obtained.

Example 2 The non-coordination

Recall the non-coordination example discussed in the introduction and recall the constraint 12'. If $authorized\text{-}op_{db}(manager, +select, salary\text{-}table)$ and $\neg\exists M \in Machine\ authorized\text{-}op_{os}(manager, +logon\text{-}locally, M)$ can be derived from any configurations, then the constraint 12' does not hold. Thus, the logical inconsistency can be obtained.

3. Related Work

Existing specifications [1][2][3][4] for access control offers reasoning about a *single* configuration. Thus, they are *inadequate* to deal with the problems of multiple access control configurations.

[5] proposed a framework which reasons about *general* composition of *local* policies and their effects as a *global* policy. Local policies are policies that are defined for a local unit in an organization whereas the global policy is the policy for the whole organization. The framework [5] neither addresses how to compose multiple local policies in order to satisfy a specific global policy, nor deals with the inconsistent local policies. Note that the kind of composition of multiple local policies for satisfying a specific global policy needs to deal with the coordination of multiple local policies. Thus, the composition framework [5] does not deal with the inconsistency and the non-coordination whereas our approach does.

4. Conclusion

In this paper, we have identified two problems of multiple configurations which are the inconsistency and the non-coordination. We have proposed a logical specification which can be used to detect the inconsistency and the non-coordination of multiple configurations. Our specification can be seen as an extension of existing specifications [2][4] for dealing with multiple configurations. In [6], some case studies were examined for our specification. Currently, we are implementing a software prototype which will be applied to such case studies.

Acknowledgement

The second author would like to acknowledge support from National Research Council of Thailand.

References

[1] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, Role-Based Access Control Models, *IEEE Computer*, 29(2), Febuary 1996.

[2] S.I. Gavrila and J.F. Barkley, Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management, *In Proceedings of ACM Workshop on Role-Based Access Control*, ACM press, 1998.

[3] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo, A Logic-Based Approach for Enforcing Access Control, *Journal of Computer Security*, 8(2&3), 2000.

[4] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian, Flexible Support for Multiple Access Control Policies, in *ACM Transactions on Database Systems*, vol. 26, n. 2, June 2001, pp. 214-260.

[5] P. Bonatti and P. Samarati, An Algebra for Composing Access Control Policies, in *ACM Transactions on Information and System Security*, 2002.

[6] S. Dangprasert, Reasoning about Access Control Configurations, Master Thesis, Department of Computer Engineering, King Mongkut's University of Technology Thonburi, Bangkok Thailand, 2001.