Journal of Universal Computer Science, vol. 18, no. 17 (2012), 2432-2451
submitted: 16/1/12, accepted: 29/8/12, appeared: 1/9/12 © J.UCS

A Formal Approach for Risk Assessment in RBAC Systems

Ji Ma
(The Christian Doppler Laboratory for Client-Centric Cloud Computing
Softwarepark 21, 4232 Hagenberg, Austria
j.ma@cdcc.faw.jku.at)

Abstract: Risk assessment and access control are important issues in cloud computing. In this paper, we propose a formal approach to risk assessment for RBAC Systems, in which access control decisions are taken after consideration of risk assessment. The risk assessment method considers partial orderings on objects and actions, which allow us to effectively capture the notions of importance of objects and criticality of actions and then to determine the risk of assigning a specific role to a specific user. We in particular consider the cases of permission assignment and delegation assignment.
Key Words: Risk assessment, access control, RBAC, poset, security classification
Category: H.1.0

1 Introduction

In Role Based Access Control (RBAC) systems, users hold certain roles, and may or may not be allowed to access the objects requested and take actions on these objects. For any access request, apart from identifying the user and its role, the access control system must know: which object is requested and what action may be applied by the user on the object. [Denning 1976], [Biba 1977], [Bell and LaPadula 1996], and [Sandhu 1993] proposed latticed-based access control models to deal with information flows in computer system. In latticed-based access control models, a subject is only allowed to access an object if the security level of the subject is higher than or equal to the security level of the object.

Risk assessment is important for RBAC systems. There are many different approaches of risk management for RBAC system. [Celikel et al. 2007] discussed managing risks in distributed RBAC systems. [Aziz et al. 2006] discussed reconfiguring role based access control policies using risk semantics. [Kondo et al. 2008] discussed extending the RBAC model for large enterprises and quantitative risk evaluation. [Nissanke and Khayat 2004] discussed risk based security analysis of permissions in RBAC systems.

Trust is an important issue for access control systems, and it changes dynamically. However, there are only few papers that discuss the dynamics of trust [Ma and Orgun 2006]. [Jonker and Treur 1999] proposed two functions, *trust evolution function* and *trust update function*. [Dimmock et al. 2004] discussed how to extend existing access control architectures to incorporate trust-based evaluation and reasoning. [Bhargava and Lilien 2004] proposed an approach enhancing role-based access control with trust ratings. [Chakraborty and Ray 2006] proposed an approach integrating trust relationships into the RBAC model. [Asnar et al. 2007] proposed an approach to assess risk on the basis of trust relations among actors.

Access control models need to handle many different scenarios in many different systems, where security requirements, contexts and environments can be highly dynamic. Therefore, systems that rely on large amounts of assumed knowledge or pre-configuration are unnecessary and inflexible. Generic methods and techniques for handling various access control scenarios are highly desirable. In our previous work [Ma et al. 2010a, Ma et al. 2010b, Ma et al. 2010c], we have discussed risk and trust issues for RBAC systems, and proposed a risk assessment method based on partially ordered set, where access permission is defined as a pair of an action and an object; a role is then defined as a set of permissions. From these posets, we automatically extract an ordering relation on permission sets (corresponding to roles). The risk of role assignment is then evaluated. We also investigate the risk of permission assignment and delegation assignment. In this paper, we further discuss this method by giving its formal representation, and then propose a formal approach of risk assessment for RBAC systems, in which access control decisions are taken after consideration of risk assessment. The risk assessment method considers partial orderings on objects and actions, such that we can effectively capture the notions of importance of objects and criticality of actions and then determine the risk for decision making.

The rest of this paper is organized as follows: Section 2 presents a poset-based security classification system. Section 3 introduces role based access control systems. Section 4 discusses risk assessment of RBAC systems based on the poset-based security classification system. Section 5 discusses the implementation issues of risk assessment for RBAC systems. Section 6 concludes this paper and discusses further works.

2 Security Classification

For a given access control system, there is a set of objects: $O = \{o_1, ..., o_i\}$. Objects are entities that can undergo actions, the operations of objects include:

- Let $o_i, o_j \in O$, $o_i + o_j$ is a new object in O, that is, $o_i + o_j = o_k$, $o_k \in O$.
- Let $o \in O$, if o^+ is a new object generated by adding something to the object o, that is, $o \sqsubset o^+$. then o^+ is a new object in O.
- Let $o \in O$, if o^- is a new object generated by removing something from the object o, that is, $o^- \sqsubset o$. then o^- is a new object in O.

For a given access control system, there is a set of users: $U = \{u_1, ..., u_i\}$. users are active agents that can perform actions.

Definition 1 *$(L, \leq)$ is a* security classification system*, where L is a set of security levels, and $\leq$ is a partial ordering relation defined on L. For any $l_i, l_j \in L$, $l_i \leq l_j$ means that l_j is a higher security level than l_i. If $l_i \leq l_j$ but $l_i \neq l_j$, that is $l_i < l_j$, in this case, l_i is strictly dominated by l_j.*

$l_i \leq l_j$ means that $(l_i, l_j) \in \leq$. $(l_i, l_j) \notin \leq$ means that $l_i \not\leq l_j$. For all $l \in \mathcal{L}$:

(1) $l_i \leq l_i$. (Reflexivity)
(2) $l_i \leq l_j$ and $l_j \leq l_i$, then $l_i = l_j$. (Antisymmetry)
(3) $l_i \leq l_j$ and $l_j \leq l_k$, then $l_i \leq l_k$. (Transitivity)

A security classification system $(L, \leq)$ can be considered as a finite lattice. As a lattice, The security classification system has two basic operators which are derived from the relation $\leq$:

$$l_i \vee l_j = l.u.b.\{l_i, l_j\}.$$
$$l_i \wedge l_j = g.l.b.\{l_i, l_j\}.$$

$l.u.b.\{l_i, l_j\}$ is called the least upper bound (join) of the set $\{l_i, l_j\}$, that is, if $l.u.b.\{l_i, l_j\} = l_k$, then $l_i \leq l_k$, $l_j \leq l_k$. And for any l, if $l_i \leq l$ and $l_j \leq l$, then $l_k \leq l$.

$g.l.b.\{l_i, l_j\}$ is called the greatest lower bound (meet) of the set $\{l_i, l_j\}$. that is, if $g.l.b.\{l_i, l_j\} = l_k$, then $l_k \leq l_i$, $l_k \leq l_j$. And for any l, if $l \leq l_i$ and $l \leq l_j$, then $l \leq l_k$.

The security level of an object represents the level of protection of the object. The security level of an user represents the level of clearance of the user. For any object o, it has exactly one security level. Similarly, for any user u, it has exactly one security level. $O(l_i)$ is defined as the set consisting of all objects whose security level is l_i, $U(l_j)$ is defined as the set consisting of all users whose security level is l_j,

We define the following principles for security level assignment (for any $o \in O$, and any $u \in U$):

(1) If object o is created (owned) by user u, then $l(o) \leq l(u)$.

(2) $l(o) \leq l(o^+)$.

(3) $l(o^-) \leq l(o)$.

(4) $l(o_1), l(o_2) \leq l((o_1) + (o_2))$.

For all $l \in L$, there exists a unique element, l_n, $l_n \leq l$. Symmetrically, for all $l \in L$, there exists a unique element, l_m, $l \leq l_m$. Thus, l_n and l_m are called the *minimum security level* and *maximum security level* of this lattice respectively.

Let $(A = \{a_i \mid i = 0, 1 \ldots, j\}, \leq)$ and $(O = \{o_i \mid i = 0, 1 \ldots, k\}, \leq)$ be partially ordered sets of actions and objects, respectively. If $a' \leq a$ and $a' \neq a$, then we have $a' < a$. The relation $a' < a$ means action a' is *less critical* than action a. Similarly, $o' < o$ means object o' is *less important* than object o.

For example, Figure 1 presents a partial ordering of a system containing the set of actions $\{a_1, a_2, a_3, a_4\}$ and the set of objects $\{o_1, o_2, o_3, o_4\}$. in Figure 1(a), the action a_4 is considered to be more critical than actions a_2 and a_3, and the action a_1 to be less critical than actions a_2 and a_3. However, actions a_2 and a_3 have no such relation. These facts are described by the following relations $a_2 \leq a_4$, $a_3 \leq a_4$, $a_1 \leq a_2$, and $a_1 \leq a_3$, but a_2 and a_3 are not comparable. In this case, $a_2 \vee a_3 = a_4$, and $a_2 \wedge a_3 = a_1$. Any path from the *minimum security level* to the *maximum security level* forms a linear hierarchy classification system. In a linear hierarchy classification system, there are no incomparable security levels. For all $l_i, l_j \in L$, if $i < j$, then $l_i < l_j$, and $l_i \vee l_j = l_j$, and $l_i \wedge l_j = l_i$.

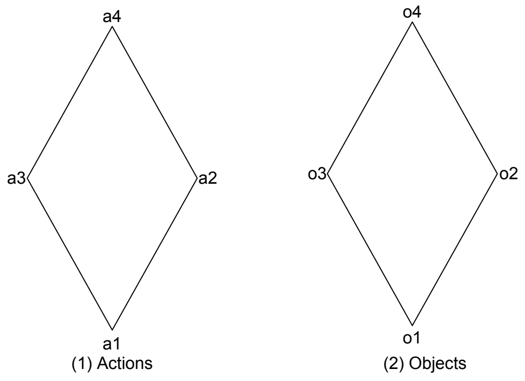

Figure 1: Actions and objects

3 Role Based Access Control Model

Mandatory access control (MAC) and discretionary access control (DAC) are traditional techniques for restricting system access to authorized users. Nowadays role based access control (RBAC), is widely applied to computer security.

Definition 2 (Role Based Access Control Model) *A role based access control (RBAC) model, denoted by M_{rbac}, has the following components:*

- *U: The set of users.*
- *O: The set of objects.*
- *A: The set of actions.*
- *R: The set of roles.*
- *C: The set of contexts.*
- *P: The set of permissions, $P \subseteq A \times O$.*
- *SE: The set of sessions, a mapping involving U, R and/or P.*
  session_user: $SE \to U$, mapping each session to a user.
  session_roles: $SE \to 2^R$, mapping each session to a set of roles.
  session_permissions: $SE \to 2^P$, mapping each session to a set of permissions.
- *RA: role assignment, $RA \subseteq U \times R$.*
- *PA: permission assignment, $PA \subseteq R \times P \times C$.*
- *RH: partially ordered role hierarchy, $RH \subseteq R \times R$.*
- *DA: delegation assignment, $DA \subseteq U \times U \times P \times C$.*

As it is usually defined, RBAC does not involve contexts or delegation. The standard formalism for RBAC is not adequate to express realistic dynamic systems. Our approach can handle these aspects.

Definition 3 (Access State) *An access state for a given RBAC system is a particular assignment of the model.*

For a given system, we assume that, at the initial state, the formal assignment of the model is as follows:

$U = \{u_1, u_2, u_3, u_4\}$,
$A = \{a_1, a_2, a_3, a_4\}$,
$O = \{o_1, o_2, o_3, o_4\}$,
$R = \{r_1, r_2, r_3, r_4\}$,
$C = \{c_1, c_2, c_3, c_4\}$,
$P = \{(a_1, o_1), (a_2, o_2), (a_3, o_3), (a_4, o_4)\}$,

$$RA = \left\{ \begin{array}{l} (u_1, r_1), \\ (u_2, r_2), \\ (u_3, r_3), \\ (u_4, r_4). \end{array} \right\}$$

$$PA = \left\{ \begin{array}{l} (r_1, a_1, o_1, c_1), \\ (r_2, a_2, o_2, c_2), \\ (r_3, a_3, o_3, c_3), \\ (r_4, a_4, o_4, c_4), \\ (r_4, a_2, o_2, c_2), \\ (r_4, a_1, o_1, c_1). \end{array} \right\}$$

$$RH = \left\{ \begin{array}{l} (r_4 \ \geq \ r_3), \\ (r_4 \ \geq \ r_2), \\ (r_4 \ \geq \ r_1), \\ (r_3 \ \geq \ r_1), \\ (r_2 \ \geq \ r_1). \end{array} \right\}$$

$$DA = \left\{ \begin{array}{l} (u_4, u_3, (a_2, o_2), c_2), \\ (u_4, u_2, (a_1, o_1), c_1). \end{array} \right\}$$

A context $c \in C$ is a mapping from a set of attributes of the system to a set of values. $s \vdash c$ means that context c is satisfied within state s.

In order to specify the access control policies, we define the following basic predicates:

– $is\_user(X)$: X is a user.

– $holds(U, R)$: User U holds role R.

– $permit(R, A, O, C)$: Role R is permitted to perform action A on object O within context C.

– $user\_permit(U, A, O, C)$: User U is permitted to perform action A on object O within context C.

– $inherits(R_i, R_j)$: Role R_i inherits the permissions of Role R_j.

– $can\_delegate(U_i, U_j, (A, O), C)$: User U_i can delegate user U_j to perform action A on object O within context C.

The predicates *holds*, *permit*, *inherits*, and *can_delegate* can be formally defined as:

- $holds(U, R)$, iff $(U, R) \in RA$.

- $permit(R, A, O, C)$, iff $(R, A, O, C) \in PA$.
- $inherits(R_i, R_j)$, iff $(R_i \geq R_j) \in RH$.
- $can\_delegate(U_i, U_j, (A, O), C)$, iff $(U_i, U_j, (A, O), C) \in DA$.

For the given system, we define the following access control rules:

R1. Role assignment.
If conditions Con_1 through Con_n hold, then role R is assigned to user U. The role assignment rule can be formalised as:

$$Con_1 \wedge \ldots \wedge Con_n \rightarrow holds(U, R).$$

R2. Permission assignment.
A user can be granted a permission, if he holds an appropriate role. The permission assignment rule can be formalised as:

$$holds(U, R) \ \wedge permit(R, A, O, C) \ \rightarrow user\_permit(U, A, O, C).$$

R3. Delegation assignment
A user may delegate a permission to another user. The delegation assignment rule can be formalised as:

$$can\_delegate(U_i, U_j, (A, O), C) \ \rightarrow user\_permit(U_j, A, O, C).$$

The access control rules provide a basis for reasoning about the properties of systems. We give a proof example as follows:

Proof Example 1 (User permission)

(1) $holds(u_1, r_1)$. *(Assumption)*
(2) $permit(r_1, a_1, o_1, c_1)$. *(Assumption)*
(3) $holds(u_1, r_1) \ \wedge permit(r_1, a_1, o_1, c_1)$. *(from (1) & (2), by* $\wedge$*_introduction)*
(4) $holds(U, R) \ \wedge permit(R, A, O, C) \rightarrow \ user\_permit(U, A, O, C)$ *(R2)*
(5) $holds(u_1, r_1) \ \wedge permit(r_1, a_1, o_1, c_1) \rightarrow user\_permit(u_1, a_1, o_1, c_1)$
(from (4), by variable instantiation)
(6) $user\_permit(u_1, a_1, o_1, c_1)$. *(from (3) & (5), by* $\rightarrow$*_elimination)*
□

4 A Formal Approach for Risk Assessment

Role assignment is a major component of a RBAC system and risk analysis is a main consideration of role assignment. The basic idea regarding risk assessment for role assignment is as follows:

- For each user U, we assign a security level, denoted by $l(U)$.
- For each role R, we calculate a security level for the role, denoted by $l(R)$.
- Then the risk value $rv(U, R)$ of role assignment can be calculated by the following formula:

$$rv(U, R) = \begin{cases} 0, & \text{if } l(U) \geq l(R) \\ 1 - \frac{l(U)}{l(R)}, & \text{otherwise} \end{cases}$$

Based on the above formula, in order to obtain the risk value $rv(U, R)$, the key is to calculate the security level $l(R)$ for each role R.

In our method, permission is defined as a pair consisting of an action and an object. $A \times O$ is the set of all permissions, based on the posets $(A, \leq)$ and $(O, \leq)$, we define permission ordering relation as follows:

$$(a', o') \leq (a, o) \overset{\text{def}}{=} a' \leq a \wedge o' \leq o$$

where $(a', o') \leq (a, o)$ means permission (a', o') is less critical than permission (a, o). As shown in Figure 2, role is a subset of permission $(A \times O)$.

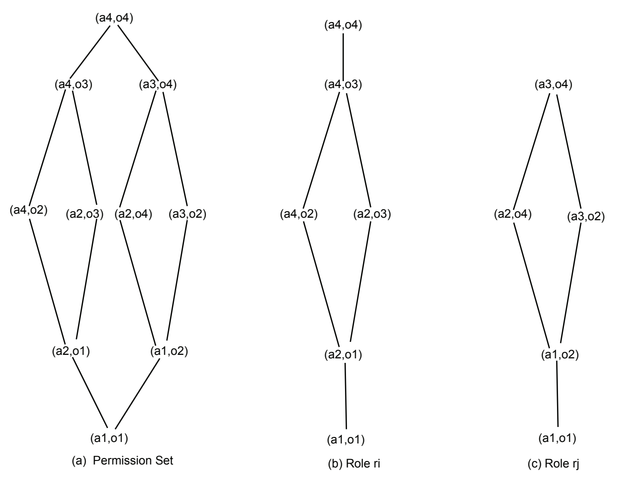

Figure 2: Permissions and Roles

In order to calculate the security level of a role, we introduce first the following definitions:

- A *chain* Ch of a role R is a subset of R having a total ordering, i.e. for all $(a, o), (a', o') \in Ch$, $(a, o) \leq (a', o')$ or $(a', o') \leq (a, o)$. So, all nodes in a chain Ch are comparable. Thus, a chain in a role can be represented by a path with directed edges between adjacent nodes.

- The length of a chain is the number of directed edges connecting two nodes in the chain. Let $length(Ch)$ be the length of chain Ch, and $node(Ch)$ be the number of nodes of chain Ch, then $length(Ch) = node(Ch) - 1$. For example, in Figure 2(b), (a1, o1) is a shortest chain that contains only one node and its length is 0; (a1,o1)–(a2,o1) is a chain that contains two nodes, its length is 1; and (a1,o1)–(a2,o1)–(a4,o2) is a chain that contains three nodes, and its length is 2.
- Let Ch_R be the set of all chains in a role $(R, \leq)$, we define the security level $l(R)$ to be the length of the longest chain in R, i.e.:

  $$l(R) = max\{length(Ch) | Ch \in Ch_R\}$$

  In Figure 2, we have $l(r_i) = 4$, and $l(r_j) = 3$.

This formula is based on the assumption that all direct edges connecting two adjacent nodes have an equal weight of 1, however in some applications it may be necessary to assign different weights to different edges based on importance considerations.

Delegation risk is also a main issue in risk analysis. Here, we present a generic formula for computing delegation risks. Let $del\_rv(U_i, U_j)$ be the risk value of user U_i delegating to user U_j one of his permissions. The delegation risk depends on the security levels of the delegator and the delegatee:

$$del\_rv(U_i, U_j) = \begin{cases} 0, \text{ if } l(U_j) \geq l(U_i) \\ 1 - \frac{l(U_j)}{l(U_i)}, \text{ otherwise} \end{cases}$$

Risk is not explicitly included in the usual RBAC model, so in the permission and delegation rules presented in Section 3 we did not consider the effects of risk. In this section, we add a risk parameter in these rules, and we define the following predicates:

- $permit\_with\_risk(u, a, o, c, v)$: User u is permitted to perform action a on object o in context c with a risk value v.
- $risk\_threshold(a, o, c)$: This predicate specifies a risk threshold value. The value is used to determine whether the risk is acceptable for the Access Control Decision Point (ACDP). Note that we cannot give a general definition for this predicate. Its definition is rather related to specific access control applications (banking, hospital, etc.). For instance, in a banking application, the threshold may be higher if the requested loan (parameters a and o) is not high, and the economic indicators (parameter c) are good.

R4. Permission assignment with risk assessment.

$$\frac{\begin{array}{c} permit(r, a', o', c') \in PA(s), s \vdash c', holds(u, r), \\ rv(u, r) \leq risk\_threshold(a, o, c), a \leq a', o \leq o', c \leq c' \end{array}}{permit\_with\_risk(u, a, o, c, v)}$$

where s is *access state*, and $PA(s)$ is the set of permission assignments in the state s. The permission rule states that: if permission $permit(r, a', o', c')$ is in $PA(s)$, context c' is satisfied in state s, and user u holds role r, then for any subaction a of a' ($a \leq a'$), any subobject o of o' ($o \leq o'$), and any subcontext c of c' ($c \leq c'$, for example, c = [1pm, 5pm], and c' =[9am, 5pm]). If the risk value v given by the function $rv(u, r) \leq$ the risk threshold given by the function $risk\_threshold(a, o, c)$, then user u is permitted to perform action a on object o in context c with the risk value v.

Base on the rule, we can reason about permission assignment with risk assessment.

Proof Example 2 (Permission assignment with risk assessment)

(1) $permit(r_4, a_2, o_2, c_2) \in PA(s)$ *(assumption)*
(2) $s \vdash c_2$. *(assumption)*
(3) $holds(u_4, r_4)$. *(assumption)*
(4) $l(r_4) = 8$. *(assumption)*
(5) $l(u_4) = 10$. *(assumption)*
(6) $a_1 \leq a_2$. *(assumption)*
(7) $o_1 \leq o_2$. *(assumption)*
(8) $c_1 \leq c_2$. *(assumption)*
(9) $risk\_threshold(a_1, o_1, c_1) = 0.1$. *(assumption)*
(10)$rv(u_4, r_4)$ *= 0.* *(by (4) and (5))*
(11)$rv(u_4, r_4) \leq risk\_threshold(a_1, o_1, c_1)$. *(by (9) and (10))*
(12)$permit\_with\_risk(u_4, a_1, o_1, c_1, 0)$.
(by assumptions, (11), and rule R4)
□

The last formula means that user u_4 is permitted to perform action a_1 on object o_1 in context c_1 with a risk value 0.

R5. Delegation assignment with risk assessment.

$$\frac{\begin{array}{c} delegate(u_i, u_j, a', o', c') \in DA(s),\ s \vdash c', \\ permit\_with\_risk(u_i, a, o, c, v), \\ v + del\_rv(u_i, u_j) \leq risk\_threshold(a, o, c), a \leq a',\ o \leq o', c \leq c' \end{array}}{permit\_with\_risk(u_j, a, o, c, v + del\_rv)}$$

where s is *access state*, and $DA(s)$ is the set of delegation assignments in the state s, and $u_i, u_j \in U$, u_i is the delegator, u_j is the delegatee. The delegation rule states that: If a delegation $delegate(u_i, u_j, a', o', c')$ is in $DA(s)$, context c' is satisfied in state s, for any subaction a of a' ($a \leq a'$), any subobject o of o' ($o \leq o'$), and any subcontext c of c' ($c \leq c'$). If the risk $\leq$ the risk threshold given by the function $risk\_threshold(a, o, c)$, then user u_i can delegate to user u_j to conduct action a on object o in context c,

This rule reveals an accumulation of risk due to the delegation. Suppose that a user u_i can perform an action a on an object o in context c with a risk value v, and that user u_i can delegate this permission to another user u_j with a delegation risk value del_rv, then the risk value for user u_j to perform action a on object o in context c is $(v + del\_rv)$.

Base on the rule, we can reason about delegation assignment with risk assessment.

Proof Example 3 (Delegation assignment with risk assessment)

(1) $can\_delegate(u_4, u_3, (a_2, o_2), c_2) \in DA(s)$ *(assumption)*
(2) $s \vdash c_2$. *(assumption)*
(3) $permit\_with\_risk(u_4, a_1, o_1, c_1, 0)$. *(by Proof Example 2, step (12))*
(4) $a_1 \leq a_2$. *(assumption)*
(5) $o_1 \leq o_2$. *(assumption)*
(6) $c_1 \leq c_2$. *(assumption)*
(7) $l(u_4) = 10$ *(assumption)*
(8) $l(u_3) = 9$. *(assumption)*
(9) $risk\_threshold(a_1, o_1, c_1) = 0.15$. *(assumption)*
(10)$v = 0$. *(by (3))*
(11)$del\_rv(u_4, u_3) = 0.1$. *(by (7) and (8))*
(12)$v + del\_rv(u_1, u_2) = 0.1$. *(by (10) and (11))*
(13)$v + del\_rv(u_1, u_2) \leq risk\_threshold(a_1, o_1, c_1)$. *(by (9) and (12))*
(14)$permit\_with\_risk(u_3, a_1, o_1, c_1, 0.1)$.
(by assumptions, (13), and rule R5)
$\square$

The last formula means that user u_3 is permitted to perform action a_1 on object o_1 in context c_1 with a risk value 0.1.

5 Implementation

In this section, we discuss the implementation issues of risk assessment for RBAC systems.

5.1 An generic access control architecture

Access control systems need to handle many different scenarios, where security requirements and environments can be highly dynamic. There are a number of papers [Chen et al. 2006, Eyers et al. 2008] that discuss dynamic access control systems.

Figure 3 presents a generic access control architecture, which includes the following components:

- $User$: in access control systems, a user needs a permission to access a resource.
- $Object$: an accessible resource.
- $Access\ control\ enforcement\ point$ ($ACEP$): controls access based on access control decisions, contains a set of enforcement functions (EF).
- $Access\ control\ decision\ point$ ($ACDP$): makes access control decisions, contains a set of decision functions (DF).
- $Access\ state(S)$: contains data for authentication and authorisation.

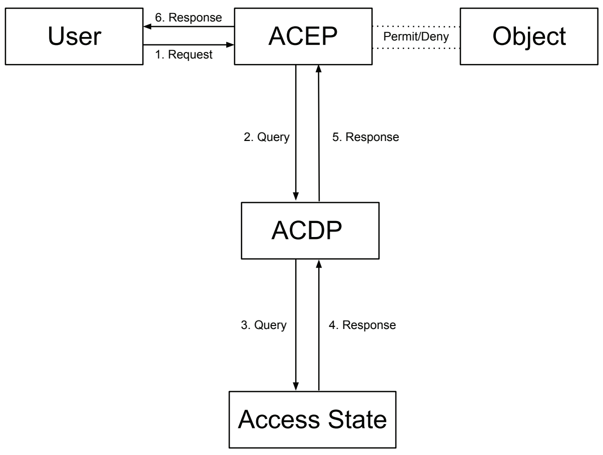

Figure 3: Generic access control architecture

The generic access control architecture, is a 5-tuple,

$$\langle U, O, ACEP, ACDP, S\rangle$$

It is a dynamic framework, since access state changes dynamically. Within the framework, we address the following aspects:

(1) A user $u \in U$ could be a single user or a group of users acting as a single user.

(2) For any object $o \in O$, there is a set of services $Serv_o = \{Serv_1^o, \ldots, Serv_n^o\}$ that it provides. For example, an ATM machine provides a set of services: $serv_{ATM} = \{balance\_checking, deposit, withdraw\}$.

(3) A request is in the form: $(u, serv_i^o)$, which means user u requires to access service $serv_i^o$.

(4) For any request $(u, serv_i^o)$, access state contains related data for decision making.

(5) For any request $(u, serv_i^o)$, there is a decision function $df \in DF$ that related to service $serv_i^o$.

(6) For any request $(u, serv_i^o)$, there is an enforcement function $ef \in EF$ that related to service $serv_i^o$.

Definition 4 (Access control Decision Function) *An access control decision function df, is a mapping of the form:*

$$df: \quad U \times Serv_i^o \xrightarrow{S} \mathcal{D},$$

where $U \times Serv_i^o$ is the request set, any pair $(u, serv_i^o) \in U \times Serv_i^o$ is a request. S is the access state. $\mathcal{D} = \{1, 0\}$ is called the decision set.

The access control decision function (DF), which is implemented in the ACDP, is defined as: If conditions hold, the request $(u, serv_i^o)$ is *permitted*, otherwise is *denied*.

$$df(u, serv_i^o)(s) = \begin{cases} 1: & c_1 \wedge ... \wedge c_n = \top. \\ 0: & c_1 \wedge ... \wedge c_n = \bot. \end{cases}$$

Definition 5 (Access control Enforcement Function) *An access control enforcement function ef, is a mapping of the form:*

$$ef: \quad U \times Serv_i^o \xrightarrow{DF} \mathcal{E},$$

where DF is the set of decision functions, and $\mathcal{E}$ is called the execution set, which consists of all actions that the system may take.

After the decision is taken, an execution e will start. The execution e consists of a sequence of activities that the system will take, i.e., $e = \{act_1, \ldots, act_i\}$, $(1 \leq i \leq n)$.

The access control enforcement function (EF), which is implemented in the ACEP, is defined as:

$$ef(u, serv_i^o)(df) = \begin{cases} e_p : & df = 1. \\ e_d : & df = 0. \end{cases}$$

Here e_p is the execution set corresponding to the permit decision, such as:

$$\{permit\_notification, execution, record, termination\}.$$

and e_d is the execution set corresponding to the deny decision, such as:

$$\{deny\_notification, ...\}.$$

Suppose that for a loan approval, the decision function considers three factors: the applicant's identity, the applicant's reputation, and the amount he wants to borrow. For an example, if Alice wants to borrow $\$10,000$, her identity is verified, her reputation is satisfied, but the amount is exceeded.

In this case, the decision function returns false:

$$df(alice, loan\_\$10,000) = 0.$$

Then the enforcement function starts the sequence of activities corresponding to the deny decision:

$$ef(alice, loan\_\$10,000) = \{deny\_notification, record, termination\}.$$

5.2 System Implementation in Prolog

For the Prolog implementation, we define a unified predicate, `lessEq(X,Y)`, to represent the relations $X \sqsubseteq_a Y$ and $X \sqsubseteq_o Y$. Note that, when using the predicate, X and Y must be the same type of variables or constants, i.e., in any instance of the predicate, both X and Y are actions or both are objects. Thus, in the Prolog program, we may have the following facts:

```
lessEq(a1,a1).
lessEq(a1,a2).
lessEq(a2,a2).
lessEq(o1,o1).
lessEq(o1,o2).
lessEq(o2,o2).
lessEq(c1,c1).
lessEq(c1,c2).
lessEq(c2,c2).
```

In Prolog, the permission assignment rule can be expressed as the following:

```
user_permit(U,A1,O1,C1) :- permit(R,A2,O2,C2), exists(C2), holds(U,R),
                           lessEq(A1,A2), lessEq(O1,O2), lessEq(C1,C2).
```

This rule says, user U is permitted to perform action $A1$ on object $O1$ within context $C1$, iff the conditions $permit(R, A2, O2, C2)$, $exists(C2)$, $holds(U, R)$, $lessEq(A1, A2)$, $lessEq(O1, O2), lessEq(C1, C2)$ are all true.

In first-order logic, the permission assignment rule can be expressed as the following:

$$\forall u \forall a \forall o \forall c (user\_permit(u, a, o, c) \rightarrow \exists R \exists A \exists O \exists C (permit(R, A, O, C) \wedge exists(C) \wedge holds(u, R) \wedge lessEq(a, A) \wedge lessEq(o, O) \wedge lessEq(c, C))).$$

where A, O and C are variables, and a, o and c are specific subject, object and context, respectively.

In order to translate FOL formulas to Prolog rules, we need to define some predicates that can be specifically used for this purpose. For example, for this risk function:

$$rv(U, R) = \begin{cases} 0, \text{ if } l(U) \geq l(R) \\ 1 - \frac{l(U)}{l(R)}, \text{ otherwise} \end{cases}$$

We introduce the following predicates:

- $rv(U, R, V)$: The risk value of assigning role R to user U is V.
- $l(N, D)$: The security level of N is D, where N is a user or a role.
- $is\_user(N)$: N is a user.
- $is\_role(N)$: N is a role.

Then, the risk function $rv(U, R)$ can be translated into two Prolog rules:

```
(1) rv(U,R,V):- holds(U,R), is_user(U), is_role(R), l(U,X), l(R,Y),
                X >= Y, V = 0.

(2) rv(U,R,V):- holds(U,R), is_user(U), is_role(R), l(U,X), l(R,Y),
                Y > X, V is 1 - X/Y.
```

When $l(U) \geq l(R)$, the first rule is applied, otherwise, the second one is applied.

For obtaining appropriate Prolog rules for a practical application, we need to consider the access state, S, where each context has a certain truth value. For example, if in a state s we have $permit(r, a', o', c') \in PA(s)$, and $s \vdash c'$, then at the state s we have that both $permit(r, a', o', c')$ and $exists(c')$ are true. For example, assume that r, a', o', c' represents “trainee”, “write”, “records”, and “guidance” respectively, and $exists(c')$ represents “guidance is available”. Therefore, we defined the permission assignment with risk assessment rule as the following:

$$\frac{permit(r,a',o',c') \in PA(s), s \vdash c', holds(u,r), \quad rv(u,r) \leq risk\_threshold(a,o,c), a \leqslant a', o \leqslant o', c \leqslant c'}{permit\_with\_risk(u,a,o,c,v)}$$

In order to translate FOL formulas to Prolog rules, for risk threshold, we introduce the following predicate.

- $risk\_threshold(A, O, C, M)$: The risk threshold for performing action A on object O in context C is M.

In Prolog, the permission assignment with risk assessment rule can be expressed as the following:

```
permit_with_risk(U,A1,O1,C1,V) :- permit(R,A2,O2,C2), exists(C2),
                                  holds(U,R), lessEq(A1,A2),
                                  lessEq(O1,O2), lessEq(C1,C2),
                                  rv(U,R,V),
                                  risk_threshold(A1,O1,C1,M), V =< M.
```

After the system implementation, the correctness of the implementation must be checked. Model checking is a technique for verifying finite state systems. There are a number of model checking tools, such as SPIN [Holzmann and Peled 1996], SMV [Zhang and Jia 2006], and PRISM [Hinton et al. 1999]. In this paper, we choose SMV for verifying the correctness of the Prolog implementation.

A SMV program has two main components:

- The abstraction of a given system.
- The set of security assertions of the system.

As shown in Figure 4, SMV supports the integration of the implementation and validation of systems. The SMV program verifies the system by checking:

- Whether the policies are correctly implemented.
- Whether the desired security assertions are satisfied.

In a Prolog program, if a fact is in the program, then it is true; if a fact is absent in the program, then it is false. For example, if only the following facts are in a program.

```
holds(u_1, r_1).
holds(u_2, r_2).
holds(u_3, r_3).
holds(u_4, r_4).
```

then, we have

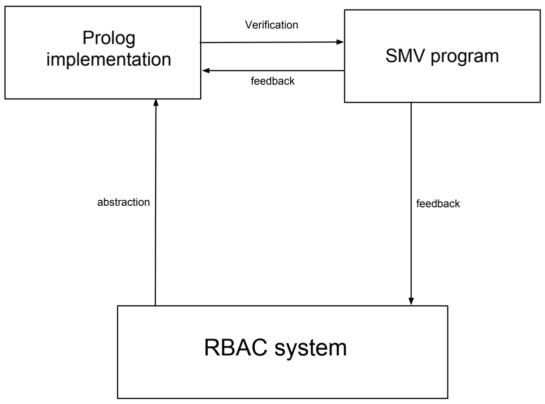

Figure 4: SMV Model Checking

```
holds(u_1, r_1) = 1 (true).
holds(u_2, r_2) = 1 (true).
holds(u_3, r_3) = 1 (true).
holds(u_4, r_4) = 1 (true).
holds(u_1, r_2) = 0 (false).
holds(u_4, r_3) = 0 (false).
...
```

That is, for any user u and any role r, the Prolog program defines the truth values of $holds(u, r)$. Therefore, in the corresponding SMV program we have:

```
$forall\ (u\ in\ USERS)$
  $forall\ (r\ in\ ROLES\ )$
    $holds[u][r] := \{0, 1\};$
```

This means that, for any user u and any role r, we assign $holds(u, r) = 1$ or 0. Model checking will check all possible cases. Therefore, all possible assignments are considered. Here, assume $USERS = \{u_1, u_2\}$, $ROLES = \{r_1, r_2\}$, then there will be 16 role assignments.

If we have a Kripke model M $= \langle S, R, I\rangle$ for a given system and a logical formula φ expressing a property of the system, we need to identify the set of all states in S that satisfy φ: $\{s \in S, |M, s \models \varphi\}$.

In first-order logic, the permission assignment with risk assessment rule can be expressed as the following:

$$\begin{aligned}&\forall u\forall a\forall o\forall c\exists V(permit\_with\_risk(u, a, o, c, V) \rightarrow\\&\quad\exists R\exists A\exists O\exists C\exists M(permit(R, A, O, C)\ \wedge exists(C) \wedge holds(u, R)\\&\qquad\wedge lessEq(a, A) \wedge lessEq(o, O) \wedge lessEq(c, C) \wedge rv(u, R, V)\\&\qquad\quad\wedge risk\_threshold(A, O, C, M) \wedge\ (V \leq M))).\end{aligned}$$

In the SMV program, permission assignment with risk assessment rule in the Prolog implementation can be abstractly translated by the following SMV statement:

```
$forall\ (u\ in\ USERS)$
  $forall\ (a\ in\ ACTIONS)$
    $forall\ (o\ in\ RESOURCES)$
      $forall\ (c\ in\ CONTEXTS)$
        $permit\_with\_risk[u][a][o][c][E\_V] :=$
          $permit[E\_R][E\_A][E\_O][E\_C]$
            $\&\ holds[u][E\_R]$
              $\&\ exists[E\_C]$
                $\&\ lessEq[a][E\_A]$
                  $\&\ lessEq[o][E\_O]$
                    $\&\ lessEq[c][E\_C]$
                      $\&\ rv[u][E\_R][E\_V]$
                        $\&\ risk\_threshold[a][o][c][E\_M]$
                          $\&\ (E\_V <= E\_M);$
```

where $[E\_R], [E\_A], [E\_O], [E\_C], [E\_V], [E\_M]$ represent $\exists R, \exists A, \exists C, \exists C, \exists V, \exists M$ in the logic formula, respectively.

SMV has two basic temporal operators, **F** and **G**, **G** corresponds to $\Box$, and **F** corresponds to $\Diamond$. Formula $\mathbf{F}\phi$ is true at a given time, if ϕ is true at some later time. Formula $\mathbf{G}\phi$ is true, if ϕ is true at all times. The following assertion is used for verifying whether the Prolog program correctly implements the permission assignment with risk assessment rule:

```
$\mathbf{G}\ permit\_with\_risk[u][a][o][c][E\_V] \rightarrow$
    $permit[E\_R][E\_A][E\_O][E\_C]$
      $\&\ holds[u][E\_R]$
        $\&\ exists[E\_C]$
          $\&\ lessEq[a][E\_A]$
            $\&\ lessEq[o][E\_O]$
              $\&\ lessEq[c][E\_C]$
                $\&\ rv[u][E\_R][E\_V]$
                  $\&\ risk\_threshold[a][o][c][E\_M]$
                    $\&\ (E\_V <= E\_M);$
```

The SMV model checking program checks whether the security assertions are satisfied by this implemented model. If all assertions are proved to be true. That is, the model satisfies the security requirements of the given system, then the model is a successful model. Otherwise, it is a failing model.

6 Conclusion

Risk analysis is an important issue for access control systems. In this paper, we propose a formal risk assessment approach for RBAC Systems, which enables systems to evaluate the risks associated with access requests. This approach can be used for the designing and implementation of access control systems. There are no existing general and systematic risk management techniques or tools for access control systems. Therefore the approach proposed in this paper have potential to be applicable in many diverse applications, such as could computing, E-commerce, and services computing.

Risk value is based on many factors, such as trust, assurance, cost, etc. Therefore, for risk management, the following issues should be investigated: how to evaluate risk values, and how to determine the risk thresholds. Different trust thresholds may lead to different policy implementations. Future work will also include refining the partial orders to include more sophisticated risk measures.

Dynamic aspects of RBAC will also be taken into consideration, an interesting topic is policy revision [Gabbay and Rodrigues 1997, Gabbay et al. 2003, Mazzieri and Dragoni 2007]. The base revision approach and controlled revision approach [Gabbay and Rodrigues 1997, Gabbay et al. 2003] are particularly useful for practical applications.

Acknowledgement

This research has been supported in part by the Christian Doppler Society. Ji Ma would like to thank Prof. Kamel Adi and Prof. Luigi Logrippo for many fruitful discussions while he was at the Université du Québec en Outaouais, Canada.

References

[Ahn et al. 2000] Ahn, G., Sandhu, R., Kang, M., Park, J.: "Injecting RBAC to secure a web-based workflow system"; Proc. ACM Workshop on Role-Based Access Control, (2000), 1-10.

[Asnar et al. 2007] Asnar, Y., Giorgini, P. , Massacci, F., Zannone, N.: "From trust to dependability through risk analysis"; Proc. ARES, (2007), 19–26.

[Aziz et al. 2006] Aziz, B., Foley, S., Herbert, J., Swart, G.: "Reconfiguring role based access control policies using risk semantics"; J. High Speed Networks, 15,3 (2006), 261-273.

[Bell and LaPadula 1996] Bell, D., LaPadula, L.: "Secure computer systems: A mathematical model"; Volume II:. Journal of Computer Security, 4, 2/3 (1996), 229-263.

[Bhargava and Lilien 2004] Bhargava, B., Lilien, L.: "Vulnerabilities and threats in distributed systems"; Proc. ICDCIT, (2004), 146-157.

[Biba 1977] Biba, K.: "Integrity considerations for secure computer systems"; Technical report, MITRE Corp., (Apr 1977).

[Celikel et al. 2007] Celikel, E., Kantarcioglu, M., Thuraisingham, B., Bertino, E.: "Managing risks in rbac employed distributed environments"; Proc. OTM Conferences (2), (2007), 1548-1566.

[Chakraborty and Ray 2006] Chakraborty, S., Ray, I.: "Integrating trust relationships into the rbac model for access control in open systems"; Proc. SACMAT, (2006), 49-58.

[Chen et al. 2006] Chen, Y., Yang, S., Guo, L., Shen, K.: "I A dynamic access control scheme across multi-domains in grid environment"; Journal of Computer Research and Development, 43,11 (2006), 1863-1869.

[Denning 1976] Denning, D.: "A lattice model of secure information flow"; Commun. ACM, 19,5 (1976), 236-243.

[Dimmock et al. 2004] Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., Moody, K.: "Using trust and risk in role-based access control policies"; Proc. SACMAT, (2004), 156-162.

[Eyers et al. 2008] Eyers, D., Srinivasan, S., Moody, K., Bacon, J.: "Compile-time enforcement of dynamic security policies "; Proc. POLICY, (2008), 119-126.

[Gabbay et al. 2003] Gabbay, D., Pigozzi, G., Woods, J.: "Controlled revision - an algorithmic approach for belief revision", Journal of Logic and Computation, 13,1 (2003), 3-22.

[Gabbay and Rodrigues 1997] Gabbay, D., Rodrigues, O.: "Structured Belief Bases: A Practical Approach to Prioritised Base Revision", Proc. ECSQARU-FAPR, (1997), 267-281.

[Holzmann and Peled 1996] Holzmann, G., Peled, D.: "The state of spin", Proc. CAV, (1996), 385-389.

[Hinton et al. 1999] Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: "Prism: A tool for automatic verication of probabilistic systems", Proc. TACAS, (2006), 441-444.

[Jonker and Treur 1999] Jonker, C., Treur. J.: "Formal analysis of models for the dynamics of trust based on experiences", Proc. Multi-Agent System Engineering'99, volume 1647 of LNAI, Springer, (1999), 221-231.

[Kondo et al. 2008] Kondo, S., Iwaihara, M., Yoshikawa, M., Torato, M.: "Extending RBAC for large enterprises and its quantitative risk evaluation "; Proc. II3E, (2008), 99-112.

[Ma et al. 2010a] Ma, J., Adi, K., Logrippo, L., Mankovski, S.: "Risk analysis in access control systems based on trust theories"; Proc. The 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), IEEE Computer Society Press, (2010), 415-419.

[Ma et al. 2010b] Ma, J., Adi, K., Logrippo, L., Mankovski, S.: "Risk management in dynamic role based access control systems"; Proc. The Fifth International Conference on Digital Information Management (ICDIM), IEEE Computer Society Press, (2010), 442-451.

[Ma et al. 2010c] Ma, J., Adi, K., Mejri, M., Logrippo; L.: "Risk analysis in access control systems"; Proc. The 2010 International Conference on Privacy, Security and Trust (PST), IEEE Computer Society Press, (2010), 160-166.

[Ma and Orgun 2006] Ma, J., Orgun, M.: "Trust management and trust theory revision. IEEE Transactions on Systems"; Man and Cybernetics, Part A, 36,3 (2006) 451-460.

[Mazzieri and Dragoni 2007] Mazzieri, M., Dragoni, A.: "Ontology revision as non-prioritized belief revision"; Proc. ESOE, (2007), 58-69.

[Nissanke and Khayat 2004] Nissanke, N., Khayat, E.: "Risk based security analysis of permissions in RBAC"; Proc. WOSIS, (2004), 332-341.

[Sandhu 1993] Sandhu, R.: "Lattice-based access control models"; J. IEEE Computer, 26,11 (1993), 9-19.

[Thuraisingham et al. 2007] Thuraisingham, B., Kantarcioglu, M., Iyer, S.: "Extended RBAC-based design and implementation for a secure data warehouse"; J. IJBIDM, 2,4 (2007), 367-382.

[Zhang and Jia 2006] Zhang, Y., Jia, S.: "Common program analysis of two-party security protocols using SMV"; Proc. APWeb, (2006), 923-930.