Risk Management in Dynamic Role Based Access Control Systems

J. Ma†, K. Adi†, L. Logrippo†, Serge Mankovski‡,

† *Department of Computer Science and Engineering*
Université du Québec en Outaouais
Québec, Canada.
‡ *CA Labs, 125 Commerce Valley DR W, Thornhill ON, Canada.*

Email: {ji.ma, kamel.adi,luigi}@uqo.ca, Serge.Mankovskii@ca.com

***Abstract*—We present a risk management framework which allows to reason about and manage risk for role based access control systems. The framework expresses essential characteristics of risk management in dynamic environments, and can be used for assessing risk and decision making; it is flexible, and able to handle different access control requirements. This framework provides a basis for designing and implementation of access control systems.**

Keywords: Risk management, dynamic, RBAC, access control, framework.

I. Introduction

Access control systems are entrusted with the task of determining whether specific requests to access data or resources should be permitted. Normally such decisions are taken with consideration of risks involved. It is often considered risky to allow data access to untrusted parties, and so access may be denied to them. In role based access control systems, access decisions especially depend on the roles of individual users. For example, a user with role "accountant" normally has different access rights than a user with role "manager". The process of assigning roles is usually based on a thorough analysis of risks. Further, as risk perceptions change in time, access control policies may also change dynamically. In this paper, we intend to investigate risk management methods and techniques for role based access control systems in dynamic environments.

Current research considers many approaches for the specification and verification of risk management systems. Some related research can be seen on Fault Tree Analysis (FTA) [16], Event Tree analysis (ETA) [10], Probabilistic Risk Assessment (PRA) [15], Failure Mode and Effects Analysis (FMEA) [9] etc. However, since these approaches generally restrict the notion of risk management to static environments, there is a need for building trustworthy systems in dynamic environments. This is also a motivation of our work in this paper.

Trust is an important issue for role based access control systems, and it changes dynamically. However, there are only few papers that discuss the dynamics of trust. Jonker and Treur [11] proposed two functions, *trust evolution function* and *trust update function*. Dimmock *et al.* [7] discussed how to extend existing access control architectures to incorporate trust-based evaluation and reasoning. Bhargava *et al.* [2] proposed an approach enhancing role-based access control with trust ratings. Asnar *et al.* [1] proposed an approach to assess risk on the basis of trust relations among actors. Based on the method proposed in [13], if we formalise polices to establish a theory for a given RBAC system, then this theory can be used for the system in access decision making.

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 a large amount of assumed knowledge or pre-configuration are unnecessary and inflexible. Generic methods and techniques for handling various access control scenarios are highly desirable. In 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, the access control system must know: which object is requested and what action may

be applied by the user on the object. In our method, an access permission is defined as a pair consisting of an action and an object; a role is assigned a set of permissions. The first contribution of this paper is to provide a formalisation of RBAC systems. This formalisation allows us to reason about RBAC systems.

The second contribution of this paper is to provide a risk management framework for RBAC systems. This framework includes: decision and enforcement functions, and policy revision in dynamic environments. We also discuss the correctness of the framework.

The rest of this paper is organized as follows. Section 2 discusses dynamic role based access control model. Section 3 proposes a risk management framework for dynamic access control systems. Section 4 discusses policy management and, especially, presents a policy revision example. Section 5 concludes this paper and discusses further works.

II. Role Based Access Control Model

Mandatory access control (MAC) and *discretionary access control* (DAC) are traditional techniques for restricting system access to authorised users. Nowadays a new alternative approach to access control, called *role based access control* (RBAC), is widely discussed and applied to computer security.

Definition 1 (RBAC Model)**.** *A Role Based Access Control(RBAC) system is a 6-tuple,*

$$\mathcal{M} = \langle \mathcal{U}, \mathcal{R}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{AR} \rangle,$$

where

- *$\mathcal{U}$: a set of subjects or users,*
- *$\mathcal{R}$: a set of roles,*
- *$\mathcal{O}$: a set of objects,*
- *$\mathcal{A}$: a set of actions,*
- *$\mathcal{P}$: a set of permissions, $P \subseteq \mathcal{A} \times \mathcal{O}$, and*
- *$\mathcal{AR}$: assignment relations.*

The set of assignment relations, $\mathcal{AR}$, includes the following relations:

- RA: role assignment relation, $RA \subseteq \mathcal{U} \times \mathcal{R}$, and
- PA: permission assignment relation, $PA \subseteq \mathcal{R} \times \mathcal{P}$.

With the definition above, in the RBAC system a user may hold one or more roles and a role may possess one or more permissions.

With RBAC systems, as we said before, for security considerations the system may not trust anyone, but it trusts:

- those facts that come from system configuration (i.e., role assignment, permission assignment, etc. in our model), and
- access control policies, which are precisely specified and verified.

The basic aim of our approach is to establish a theory. The system can then reason with this theory for decision making. Since the system changes dynamically, the theory may also need to be revised when the system changes.

We will see that our model is appropriate for the formalisation of policies. It can also help system designers to capture the requirements for a given access control systems at the implementation stage. In the following we present the method for formalizing access control policies through an example.

Definition 2 (Access State)**.** *An access state for a given RBAC system is a particular assignment of the model.*

As an example, with a financial system, we assume that, at the initial state, denoted S_0, the formal assignment of the model is given as follow:

$\mathcal{U} = \{bob, lisa, tom\}$,
$\mathcal{R} = \{manager, admin, clerk\}$,
$\mathcal{O} = \{records, loans, \ldots\}$,
$\mathcal{A} = \{read, modify, approve, \ldots\}$,
$\mathcal{P} = \{(read, records), (modify, records), \ldots\}$,
$\mathcal{AR} = \{RA, PA\}$,

where

$$RA = \left\{ \begin{array}{l} (bob, manager), \\ (lisa, admin), \\ (tom, clerk) \end{array} \right\}$$

$$PA = \left\{ \begin{array}{l} (manager, (read, records)), \\ (admin, (modify, records)), \\ (manager, (approve, loans)), \\ (clerk, (read, records)) \end{array} \right\}$$

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

- $holds(U, R)$: User U holds role R,
- $possesses(R, A, O)$: Role R possesses permission (A, O).
- $permitted(U, A, O)$: User U is permitted to perform action A on object O.

- $can\_access(U, O)$: User U can access object O.
- $is\_user(X)$: X is a user.

The first and second predicates correspond to the assignment relations RA and PA, respectively, in the model. In fact, we can directly define the two predicates as: $holds(U, R)$ iff $(U, R) \in RA$ and $possesses(R, A, O)$ iff $(R, (A, O)) \in PA$. The predicate can_access is like the predicate $permitted$, except that actions involved are not considered.

In our discussion, we also need several auxiliary predicates given as follows:

- $is\_in(U, Dept)$: User U is in the department $Dept$.
- $can\_modify(U, O)$: User U can modify object O.
- $can\_approve(U, P)$: User U can approve P.
- $can\_delegate(U_1, U_2, A, O)$: User U_1 can delegate user U_2 to perform A on O.
- $can\_co\_approve(U_1, U_2, P)$: User U_1 and user U_2 can co-approve P.

Note that the second and third predicates are only specific cases of predicate $permitted(U, A, O)$. We define:

$$can\_modify(U, O) \equiv permitted(U, modify, O),$$
$$can\_approve(U, P) \equiv permitted(U, approve, P).$$

In the initial access state (S_0) defined above, the following facts hold:

F1. $holds(bob, manager)$.
F2. $holds(lisa, admin)$.
F3. $holds(tom, clerk)$.
F4. $possesses(manager, approve, loans)$.
F5. $possesses(admin, modify, records)$.
F6. $possesses(manager, read, records)$.

In other words, at the access state S_0, F1 - F6 are all true. We denote this set of facts by $\mathcal{F}$:

$$\mathcal{F} = \{F1, F2, F3, F4, F5, F6\}.$$

Thus, these facts will be trusted by the system unless the state changes.

Definition 3 (Access Policy)**.** *An access policy is in the following form:*

$$permitted(U, A, O) \leftrightarrow C_1 \wedge \ldots \wedge C_n,$$

where $U \in \mathcal{U}$ and $(A, O) \in \mathcal{P}$. It is read as "User U is permitted to perform action A on object O if and only if conditions C_1 through C_n hold".

Definition 4 (Policy Set)**.** *A Policy Set for a given system is the formal representation of the access control mechanisms of the system, where each policy is formally represented with a logical formula as shown in Definition 3.*

For the financial system, we consider the following policies:

Policy 1 (Access)**.** *Object "records" can be accessed only by users with associated roles.*

The access policy can be formalised as:

P1. $holds(U, R) \wedge possesses(R, A, records) \leftrightarrow can\_access(U, records)$.

Policy 2 (Modification)**.** *The financial system allows administrators only to modify "records".*

The modification policy can be formalised as:

P2. $holds(U, admin) \leftrightarrow can\_modify(U, record)$.

Policy 3 (Loan)**.** *The financial system allows only managers to approve loans.*

The loan policy can be formalised as:

P3. $holds(U, manager) \leftrightarrow can\_approve(U, loan)$.

Policy 4 (Delegation)**.** *A user can delegate an authorization to another user if and only if the user is permitted to perform the action on the object corresponding to the delegation.*

The delegation policy can be formalised as:

P4. $is\_user(U_1) \wedge is\_user(U_2) \wedge possesses(U_1, A, O) \leftrightarrow can\_delegate(U_1, U_2, A, O)$.

Policy 5 (Co-approval)**.** *A "contract" must be approved by two users coming from different departments.*

The co-approval policy can be formalised as:

P5. $is\_in(U_1, D_1) \wedge is\_in(U_2, D_2) \wedge D_1 \neq D_2 \leftrightarrow can\_co\_approve(U_1, U_2, contract)$.

Thus, we have established a policy set for the financial system that includes five policies.

$$\mathcal{PS} = \{P1, P2, P3, P4, P5\}.$$

The policy set provides a foundation for reasoning about the security properties of the system. For example, based on $\mathcal{PS}$ with the facts in $\mathcal{F}$, we can prove that "$Lisa\ can\ modify\ the\ records$". The logical proof outline is given as follows:

Example 1 (Policy Proof)**.**

(1) $holds(lisa, admin)$. *(F2)*
(2) $holds(X, admin) \leftrightarrow\ can\_modify(X, record)$. *(P2)*
(3) $holds(lisa, admin) \leftrightarrow can\_modify(lisa, record)$.
(from (1),(2), by Variable instantiation)
(4) $can\_modify(lisa, record)$.
(from (1), (3), by $\leftrightarrow$_elimination)
□

III. Risk Management Framework

The purpose of managing risks is to minimise, monitor, and control the probability of unfortunate events. Risk for a system may come from a variety of reasons. However, for RBAC systems, two major aspects that may cause risks are:

- Access control policies are not correctly implemented, and
- Policies are not updated in time when access state is changed.

In the following, we discuss risk management, focusing on the methods and techniques applied for policy implementation and revision.

Figure 1 presents a generic risk management framework for access control systems. There are six components that are usually involved in a given request:

- $User$: in access control systems, a user needs a permission to access a resource.
- $Object$: an accessible resource.
- $Access\ control\ enforcement\ point$: controls access based on decision function.
- $Access\ control\ decision\ point$: makes decisions based on policies and risk analysis.
- $Policy\ set$: specifies access control mechanisms.
- $Database$: provides information for risk analysis.

Definition 5 (Risk Management Framework)**.** *The Risk Management Framework for a given RBAC system is a six tuple:*

$$\langle \mathcal{U}, \mathcal{O}, \mathcal{PS}, \mathcal{DF}, \mathcal{EF}, \mathcal{D} \rangle,$$

where $\mathcal{U}$ is a set of users, $\mathcal{O}$ is a set of objects, $\mathcal{PS}$ is a set of access control policies of the system, $\mathcal{DF}$ is a set of access decision functions, $\mathcal{EF}$ is a set of policy enforcement functions, $\mathcal{D}$ is a database that contains necessary information for decision making.

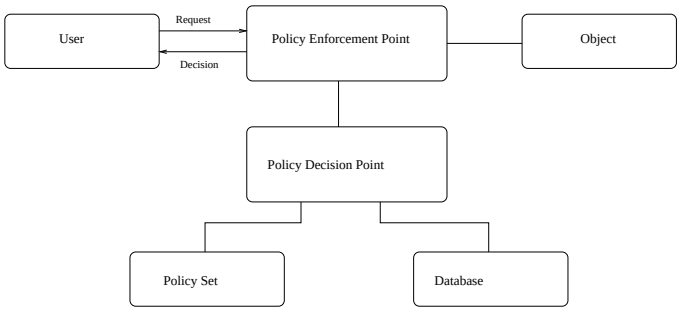

Figure 1. Generic risk management framework

We call $\langle \mathcal{U}, \mathcal{O}, \mathcal{PS}, \mathcal{DF}, \mathcal{EF}, \mathcal{D} \rangle$ a dynamic risk management framework, since the access control policy set as well as other components of the framework may change over time. It manages RBAC systems, for which the access state changes dynamically.

Within this framework, we have to mention the following aspects:

(1) A user $u \in \mathcal{U}$ could be a single user or a group of users acting as a single user.
(2) For any object $o \in \mathcal{O}$, 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: $(u, serv_i^o)$, and for any access request $(u, serv_i^o)$, there is a data set $d_o \in \mathcal{D}$ used for decision making that relates to object o. For example, for a home loan approval, a bank may consider the applicant's occupation, annual income, and the amount he wants to borrow, all these factors form a data set for risk analysis.
(4) for any request $(u, serv_i^o)$, there is a policy $p_o \in \mathcal{PS}$ that relates to object o.
(5) for any request $(u, serv_i^o)$, there is a decision function $df_o \in \mathcal{DF}$ that relates to object o,
(6) for any request $(u, serv_i^o)$, there is a policy enforcement function $ef_o \in \mathcal{EF}$ that relates to object o.

Definition 6 (Risk Analysis Data Set)**.** *For any access request, there is a data set that is used for risk analysis.*

$$d_o = \{d_1^o, \ldots, d_n^o\}.$$

Definition 7 (Access Decision Function)**.** *Given an object $o \in \mathcal{O}$, let $serv_o$ be the set of all services that object o provides. Then, the access control decision function related to object o, denoted by df_o, is a mapping of the form:*

$$df_o: \quad \mathcal{U} \times Serv_o \xrightarrow{p_o(d_o)} \mathcal{D},$$

where $\mathcal{U} \times serv_o$ is the request set, any pair $(u, serv_i^o) \in \mathcal{U} \times serv_o$ is a request. p_o and d_o are the access control policy and risk analysis data set, respectively, related to object o. $\mathcal{D} = \{permit, deny\}$ is called the decision set*.*

We assume that, if $df_o(u, serv_i^o) = 1$, then the request $(u, serv_i^o)$ is accepted, otherwise it is refused.

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

Here $c_1, ..., c_n$ are conditions of the policy that relates to the request $(u, serv_i^o)$.

Definition 8 (Policy Enforcement Function)**.** *Given an object $o \in \mathcal{O}$, the enforcement function associated with o, denoted by ef_o, is a mapping of the form:*

$$ef_o: \quad \mathcal{U} \times Serv_o \xrightarrow{df_o} \mathcal{D} \rightarrow \mathcal{E},$$

where df_o is the decision function, $\mathcal{D}$ is the decision set defined as above, and $\mathcal{E}$ is the execution set, which consists of all actions that the system may take.

After the decision is taken, an execution e will start, e consist of a sequence of activities that the system will take, that is, $e = \{a_1, \ldots, a_i\}$, $(1 \leq i \leq n)$.

Suppose that for loan approval, the system considers three factors: the applicant's identity, the applicant's reputation, and the amount he wants to borrow. The risk analysis data set is $d_{loan} = (identity, reputation, amount)$. We assume $identity$ has two possible states, *verified* (1) and *unverified* (0); $reputation$ has two states, *satisfied* (1), and *unsatisfied* (0); and $amount$ has two states, *satisfied* (1) and *excess* (0). For example, if Alice wants to borrow $\$10,000$, her identity is verified, her reputation is satisfied, but the amount is excess.

In this case, the policy decision function returns a false value:

$$df_{loan}(Alice, loan\$10,000) = 0.$$

And the policy enforcement function ef_{loan} starts the sequence of activities corresponding to the deny decision:

$ef_{loan}(Alice, loan\$10,000) =$
$\{deny\_notification, record, termination\}.$

IV. Policy Management

Blaze *et al.* [3] first identified the system management problem as a distinct and important component of agent systems. Further discussion of system management engines, including PolicyMaker and KeyNote, has appeared in their continuing work [4], [5].

We consider policy management as the basis of access control management. The security of the system is based on how the policies work in real situations. Systems usually operate in dynamic environments. Policies need to be timely updated.

Considering policy management, we need to note the following points:

- Access control systems change dynamically. A *state change* of the system causes a transition from the current access state S to the new state (next) S'. A state change could be related to components of the system model, such as "adding a user", "deleting a user", "designing a new role" etc.
- Corresponding to a state change, there could be a *policy change*. For example, when the system adds a new role, it should have a new policy or modify an existing policy for the new role.
- A *policy change* will cause a corresponding transition from the current policy set $\mathcal{PS}$ to a new policy set $\mathcal{PS}'$.

In the following, we use an example to show how to revise the policy set when the access state of a RBAC system changes.

Example 2 (Policy Revision)**.** *Recall the financial system. Assume that a new employee, Emma, arrives. Currently the system only allows administrators to modify financial records. Suppose that, after Emma arrives, the system adds a new role "admin_assist" (administrator assistant), and Emma is assigned this role. The system also allows administrator assistants to modify financial records. Then, we have the new access state S', where the state changes are underlined.*

$\mathcal{U}$ = $\{bob, lisa, tom, \underline{emma}\}$,
$\mathcal{R}$ = $\{manager, admin, clerk, \underline{admin\_assist}\}$,

$\mathcal{O}$ = $\{records, loans, ...\}$,
$\mathcal{A}$ = $\{create, read, write, delete, ...\}$,
$\mathcal{P} = \{(read, records), (modify, records), \ldots\}$,
$\mathcal{AR}$ = $\{RA, PA\}$,

where

$$RA = \left\{ \begin{array}{l} (bob, manager), \\ (lisa, admin), \\ (tom, clerk), \\ \underline{(emma, admin\_assist)} \end{array} \right\}$$

$$PA = \left\{ \begin{array}{l} (manager, (read, records)), \\ (admin, (modify, records)), \\ \underline{(admin\_assist, (modify, records)),} \\ (manager, (approve, loans)), \\ (clerk, (read, records)) \end{array} \right\}$$

Within the new access state of the financial system, we have the following new facts:

F7. $holds(emma, admin\_assist)$.
F8. $possesses(emma, modify, records)$.

Thus, we have:

$$\mathcal{F}' = \{F1, F2, F3, F4, F5, F6, R7, F8\}.$$

Based on the state change, we need to modify the policy P2:

$$\text{P2}'.holds(U, admin) \vee holds(U, admin\_assist) \leftrightarrow can\_modify(U, record).$$

A *policy change* to a given policy set $\mathcal{PS}$ can be viewed as consisting of one of the following two types of activities:

- adding a policy to $\mathcal{PS}$, and
- retracting a policy from $\mathcal{PS}$.

With this view, in the above example we first retract the policy P2 from $\mathcal{PS}$, then add the policy P2′ to the policy set and obtain the new policy set $\mathcal{PS}'$.

Using the notation proposed in [13] for theory revision, we define a *policy change* as a sequence of formulas with the signs $\oplus$ or $\ominus$, regarded as the operations *addition* and *retraction*, respectively. Thus, if $\oplus p_i$ is in the sequence, then the change contains the activity of adding p_i to $\mathcal{PS}$ (i.e.,$\mathcal{PS} \cup \{p_i\}$); and if $\ominus p_i$ is in the sequence, then the change contains the activity of retracting p_i from $\mathcal{PS}$, (i.e.,$\mathcal{PS} \setminus \{p_i\}$). Formally, we have

Definition 9. *A policy change, Δ, to a given policy set $\mathcal{PS}$ is a sequence having the following form:*

$$\Delta = \langle *_1 p_1, \ldots, *_n p_n \rangle$$

where each $*_i$ is $\oplus$ or $\ominus$, $p_1, \ldots, p_n$ are formulas representing single policies. If $*_i$ is $\ominus$, then $p_i \in \mathcal{PS}$; and if $*_1$ is $\oplus$, then $p_i \notin \mathcal{PS}$.

With policy revision, we do as follows: let p be a policy and $\mathcal{PS}$ be a policy set, then

- When $\mathcal{PS} \nvdash p$, i.e., p does not belong to $\mathcal{PS}$, we may add p to $\mathcal{PS}$ to obtain a new policy set $\mathcal{PS}'$, such that $\mathcal{PS}' = \mathcal{PS} \oplus p$ and $\mathcal{PS}' \vdash p$.
- When $\mathcal{PS} \vdash p$, i.e., the policy p is in $\mathcal{PS}$, we may retract p from $\mathcal{PS}$ to form a new policy set $\mathcal{PS}'$, such that $\mathcal{PS}' = \mathcal{PS} \ominus p$ and $\mathcal{PS}' \nvdash p$.

Thus, let $\Delta = \langle *_1 p_1, \ldots, *_n p_n \rangle$ be a policy change to the policy set $\mathcal{PS}$, then the new policy set $\mathcal{PS}'$ can be expressed with the following formula:

$$\mathcal{PS}' = \mathcal{PS} *_1 p_1, \ldots, *_n p_n.$$

In our example, the policy change is $\langle \ominus P2, \oplus P2' \rangle$. Therefore, $\mathcal{PS}' = \mathcal{PS} \setminus P2 \cup P2'$. That is, we have

$$\mathcal{PS}' = \{P1, P2', P3, P4, P5\}.$$

The new policy set $\mathcal{PS}'$ is obtained from the new access state S'.

In the process of forming or revising the policy set for a given RBAC system, it is important to guarantee (1) policy consistency, i.e., a new rule must be consistent with those policies that are already in the policy set; and (2) policy completeness, i.e., a required policy can be derived from the policy set for any access request. Formally, we say that:

- Policy set $\mathcal{PS}$ is $consistent$ if $\mathcal{PS} \not\models \perp$, that is, there is no contradiction derived from $\mathcal{PS}$.
- Policy set $\mathcal{PS}$ is $complete$ if any policy p required for the system is included in $\mathcal{PS}$, i.e., we have $\mathcal{PS} \vdash p$.

Another important issue is the correctness of policy implementation, which is based on the risk management framework. We have

Definition 10. *Let $\Gamma = \langle \mathcal{U}, \mathcal{O}, \mathcal{PS}, \mathcal{DF}, \mathcal{EF}, \mathcal{D} \rangle$ be a risk management framework for a given RBAC system. We say that the policy implementation of the system based on the framework is correct, if for any $u \in \mathcal{U}$ and any $o \in \mathcal{O}$, we have*

$$\Gamma \models permitted(u, s_i^o) \leftrightarrow C_1 \wedge \ldots \wedge C_n,$$

where s_i^o is a service that object o provides, and $C_1, \ldots, C_n$ are the conditions in the policy rule $permitted(u, s_i^o) \leftrightarrow C_1 \wedge \ldots \wedge C_n$.

In the risk management framework, for access decision function, for any $o \in \mathcal{O}$, there is a set, s^o, that contains all services o can provide. Further, for any service s_i^o there is a policy p_o expressed as $permitted(u, s_i^o) \leftrightarrow C_1 \wedge \ldots \wedge C_n$. For any access request, there is a data set, $d_o = (d_1^o, \ldots, d_n^o)$, that must be considered when making decisions for access requests.

$$df_o(u, s_i^o)(v_o) = \begin{cases} deny: \text{ iff } c_1 \wedge ... \wedge c_n = \bot. \\ permit: \text{ iff } c_1 \wedge ... \wedge c_n = \top. \end{cases}$$

Based on this definition, if $C_1 \wedge \ldots \wedge C_n$ = $\bot$, the decision is $deny$. If $C_1 \wedge \ldots \wedge C_n$ = $\top$, the decision is $permit$.

Similarly, the access enforcement function is defined as:

$$ef_o(u, s_i^o)(df_o) = \begin{cases} e_d: \text{ iff } df_o = deny. \\ e_p: \text{ iff } df_o = permit. \end{cases}$$

Here e_d is the execution set corresponding to the deny decision, such as $\{deny\_notification, record, termination\}$. And e_p is the execution set corresponding to the permit decision, such as $\{permit\_notification, ...\}$.

Based on the definition of decision and enforcement functions, the risk management satisfies the correctness of policy implementation.

V. Conclusion

In this paper, we have presented a dynamic risk management framework (DRMF) for access control systems. It expresses the essential characteristics of access control systems. This framework is highly desirable for handling access control scenarios in dynamic environments.

There are no existing general and systematic techniques or tools for risk analysis in access control policies. Therefore the methods and techniques proposed in this paper have potential to be applicable in many diverse applications, such as E-commerce, web services, and service oriented systems.

Future work includes the implementation of dynamic management of access control systems. It is important to investigate the associated security mechanisms to achieve the required security goals. The threats and attacks to access control systems will also be considered.

There are several methods and techniques for belief revision that could be helpful for policy revision. We plan to investigate a variety of belief revision techniques [6], [12], [14] that can be applied for the revision of trust theories. The controlled revision approach of Gabbay et al. [8] may be particularly useful for practical applications.

Acknowledgement

This research has been funded in part by grants from PROMPT Québec and from CA Labs, Canada. The authors would like to thank Dr. Hemanth Khambhammettu and Dr. Riaz Ahmed Shaikh for their useful comments and suggestions.

References

[1] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone. From trust to dependability through risk analysis. In *ARES*, pages 19–26, 2007.

[2] B. K. Bhargava and L. Lilien. Vulnerabilities and threats in distributed systems. In *ICDCIT*, pages 146–157, 2004.

[3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Computer Society Symposium on research in Security and Privacy*, pages 164–173, 1996.

[4] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the policymaker trust management system. In *Financial Cryptography*, pages 254–274, 1998.

[5] M. Blaze, J. Ioannidis, and A. D. Keromytis. Experience with the keynote trust management system: Applications and future directions. In *iTrust*, pages 284–300, 2003.

[6] M. R. Cravo and J. P. Martins. A practical system for defeasible reasoning and belief revision. In *ECSQARU*, pages 65–72, 1993.

[7] N. Dimmock, A. Belokosztolszki, D. M. Eyers, J. Bacon, and K. Moody. Using trust and risk in role-based access control policies. In *SACMAT*, pages 156–162, 2004.

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

[9] L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. In *QEST*, pages 119–128, 2007.

[10] D. Huang, T. Chen, and M. J. Wang. A fuzzy set approach for event tree analysis. *Fuzzy Sets and Systems*, 118(1):153–165, 2001.

[11] C. M. Jonker and J. Treur. Formal analysis of models for the dynamics of trust based on experiences. In *Proceedings of Multi-Agent System Engineering'99*, volume 1647 of *LNAI*, pages 221–231. Springer, 1999.

[12] V. Kessler and H. Neumann. A sound logic for analysing electronic commerce protocols. In *Proceedings of the ESORICS'98*, pages 345–360, 1998.

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

[14] M. Mazzieri and A. F. Dragoni. Ontology revision as non-prioritized belief revision. In *ESOE*, pages 58–69, 2007.

[15] H. Nejad, D. Zhu, and A. Mosleh. Hierarchical planning and multi-level scheduling for simulation-based probabilistic risk assessment. In *Winter Simulation Conference*, pages 1189–1197, 2007.

[16] H. Sun, M. Hauptman, and R. R. Lutz. Integrating product-line fault tree analysis into aadl models. In *HASE*, pages 15–22, 2007.