A logical approach to dynamic role-based access control

Philippe Balbiani   Yannick Chevalier   Marwa El Houri

Abstract

Since its formalization RBAC has become the yardstick for the evaluation of access control formalisms. In order to meet organizational needs, it has been extended along several directions: delegation, separation of duty, history-based access control, etc. We propose in this paper an access control language in which RBAC and all the above-listed extensions can be encoded. In contrast with Cassandra, we have not promoted role management mechanism to first-class citizenship, and have based our model on the assumption that access control systems could be separated into a dynamic part that evolves according to actions performed by users and a static part. We solve in this paper decision problems related to access control for policies expressed in this language.

Keywords: Access control language, RBAC, privacy policy

1 Introduction

The academic foundations of the access control problems have been formalized in [4, 8], where an information system is defined by a set of *objects*, a set of *subjects* performing *operations* on objects, and a finite number of relations called *rights* between subjects and objects. This system has later been refined with *Role-based access control* (RBAC) [6], where subjects are organized in groups called roles, and these groups are *hierarchically* structured. This structure permits one to define an inheritance of rights from one role to another, and thus to express complex policies in a less error-prone way. Several extensions of this "core RBAC" policy expression language have been developed over the years to address needs arising from real-case problems such as role hierarchy, separation of duty and delegation.

RBAC, along with several of its extensions, was a core component in many highly expressive policy languages. The notion of delegation was explored in the works of [1, 11], in which delegation was seen as a subject who grants some access control rights to another one. In [9] one distinguished between delegation from a role to another and delegation from a subject active in a role to another subject active in another role to avoid potential inconsistencies. Actually, a study on a large information system [3] aiming at implementing a security policy in Cassandra [2] has shown that different flavors of delegation had to be employed in different parts of the system. Role hierarchy and separation of duty in its static and dynamic aspects were also subject of study, wether explicitly as extensions of the *RT* language [9] or within the Cassandra policy specification.

However we believe that such policy languages although highly expressive in terms of access control management, lack a dynamic aspect. In fact in Cassandra one can see a dynamic aspect in terms of activation and deactivation of a role as a mean to step in and out of a role and thus acquire the adequate rights, but the effect of performing an action other that activating a role cannot be specified within the policy language. We propose in this paper a language that takes into account RBAC extensions but also introduces a dynamic aspect to the expression of access control rules.

The concept underlying this language is that an access control system is characterised by decision contexts and an invariant datalog program. A decision context is defined by a set of permissions. Within each decision context, an access control decision is based on the computation of whether the requested permission is obtainable using the datalog program from the set of permissions defining the current decision context. The access control system evolves from one decision context to another according to actions performed by a user. A drawback of this simplicity is the declaration of every action that can alter the decision context, including *e.g.* the action stating that a client is active in a given role.

In spite of its simplicity, this model permits us to express seamlessly core RBAC policies as well as their different extensions presented above. It is similar in spirit to Cassandra, though we have not built-in role management: being active in a role is an action similar to the invocation of a service. Finally, in contrast with Transaction Logic [5] the side effects are not attached to a rule but to an effective action of the client. The consequence of this choice is the determinism of the system w.r.t. client's actions (instead of client's choice of rules to apply).

In Sect. 2 we present a novel language for expressing access control policies. Then, in Sect. 3 we present how RBAC can be encoded into this language. Section 4 is devoted to the definition and complexity analysis of decision problems related to access control in our language.

2 Access control policies

In this section, we present our framework for expressing role-based access control policies and their extensions.

2.1 Domains

Active processes acting on behalf of users are referred to as subjects whereas passive resources accessible on a computer system are referred to as objects. A key feature of our access control policies is that all actions are done through roles, i.e. subjects receive permissions to execute actions on objects only through the roles to which they are assigned. In our policies, following the notions considered in RBAC, subjects are organized in groups called roles, and these groups are hierarchically structured. In our setting, a domain is a tuple

$$\mathcal{D} = \langle S, A, O, R \rangle$$

such that S is a set of subjects, A is a set of actions, O is a set of objects and R is a set of roles. We will assume that:

- $S \subseteq O$ with S and R pairwise disjoint,
- A and O are pairwise disjoint.

2.2 Security states

Consider a domain $\mathcal{D} = \langle S, A, O, R \rangle$. A security state based on $\mathcal{D}$ is a subset Π of $S \times A \times O \times R$. For all s in S, for all a in A, for all o in O and for all r in R, we will write $\Pi(s, a, o, r)$ instead of $(s, a, o, r) \in \Pi$. $\Pi(s, a, o, r)$ means that "subject s has in Π the permission to execute action a on object o through role r". A primary relation of interest between security states is that of set inclusion, under which the set of security states based on $\mathcal{D}$ forms a complete lattice.

2.3 Atomic formulae and conditions

Consider a domain $\mathcal{D} = \langle S, A, O, R \rangle$ and a security state Π based on $\mathcal{D}$. We assume an alphabet of variable symbols: X, Y, etc, possibly with subscripts. A term based on $\mathcal{D}$ is either an element of $S \cup A \cup O \cup R$ or a variable symbol. An interpretation function for $\mathcal{D}$ is a function I mapping the variable symbols to elements of $S \cup A \cup O \cup R$. The value $\widetilde{I}(t)$ of a term t is defined as follows:

- if t is an element of $S \cup A \cup O \cup R$ then $\widetilde{I}(t) = t$,
- if t is a variable symbol then $\widetilde{I}(t) = I(t)$.

A 4-tuple (t_1, t_2, t_3, t_4) of terms based on $\mathcal{D}$ is said to be correct iff the following conditions are satisfied:

- t_1 is either a variable symbol or an element of S,
- t_2 is either a variable symbol or an element of A,
- t_3 is either a variable symbol or an element of O,
- t_4 is either a variable symbol or an element of R.

An atomic formula based on $\mathcal{D}$ is an expression of the form $P(t_1, t_2, t_3, t_4)$ were (t_1, t_2, t_3, t_4) is a correct 4-tuple of terms based on $\mathcal{D}$. We define the well-formed conditions (ϕ, ψ, etc, possibly with subscripts) based on $\mathcal{D}$ by the rule

$$\phi ::= P(t_1, t_2, t_3, t_4) \mid \bot \mid \top \mid (\phi_1 \vee \phi_2) \mid (\phi_1 \wedge \phi_2).$$

The satisfiability relation $\Pi, I \models \phi$ between a security state Π, an interpretation function I and a condition ϕ is defined as follows:

- $\Pi, I \models P(t_1, t_2, t_3, t_4)$ iff $\Pi(\widetilde{I}(t_1), \widetilde{I}(t_2), \widetilde{I}(t_3), \widetilde{I}(t_4))$,
- $\Pi, I \not\models \bot$,
- $\Pi, I \models \top$,
- $\Pi, I \models \phi_1 \vee \phi_2$ iff $\Pi, I \models \phi_1$ or $\Pi, I \models \phi_2$,
- $\Pi, I \models \phi_1 \wedge \phi_2$ iff $\Pi, I \models \phi_1$ and $\Pi, I \models \phi_2$.

2.4 Static clauses and static policies

Security states are dynamic in nature, i.e. they are likely to change over time in reflection of ever evolving environmental conditions. This observation leads us to the central concept of the paper: rule-based access control policies. Rule-based access control policies will be built up using static clauses and dynamic clauses. We define in this section the concept of static clauses whereas we define in the next section the concept of dynamic clauses. Consider a domain $\mathcal{D} = \langle S, A, O, R \rangle$ and a security state Π based on $\mathcal{D}$. A static clause based on $\mathcal{D}$ is an expression of the form

$$P(t_1, t_2, t_3, t_4) \leftarrow \phi.$$

For example, the expression $P(X, a, Y, r) \leftarrow P(X, a, o, r)$ is a static clause. It says that "if X has the permission to execute a on o through r then X has the permission to execute a on Y through r". A static policy based on $\mathcal{D}$ is a finite set SP of static clauses based on $\mathcal{D}$. We shall say that Π is a model of SP, in symbols $\Pi \models SP$, iff

- for all interpretation functions I for $\mathcal{D}$ and for all static clauses $P(t_1, t_2, t_3, t_4) \leftarrow \phi$ in SP, if $\Pi, I \models \phi$ then $\Pi, I \models P(t_1, t_2, t_3, t_4)$.

The reader may easily verify that the set $\{\Pi : \Pi \models SP\}$ has a least element under set inclusion. Let $l(SP)$ be this least element.

2.5 Dynamic clauses and dynamic policies

So far, there is nothing really new. But a simple idea is going to ensure that security states change over time: dynamic policies. If an action is permitted, it is not necessarily executed. As such the dynamic policy manages the consequence of executing (or not) a permitted action.

Consider a domain $\mathcal{D} = \langle S, A, O, R \rangle$ and security states Π, Π' based on $\mathcal{D}$. A dynamic clause based on $\mathcal{D}$ is an expression of the form

$$\phi \rightarrow (\psi_1, \psi_2)$$

where neither ψ_1 nor ψ_2 contain occurrences of the Boolean connective $\vee$. The dynamic clause $\phi \rightarrow (\psi_1, \psi_2)$ is said to be consistent iff neither ψ_1 nor ψ_2 contain occurrences of the Boolean connective $\perp$. For example, the dynamic clause $P(X, a, Y, r) \rightarrow (P(X, a, o, r), \top)$ is consistent. It says that "if X has the permission to execute a on Y through r then either X executes a on Y through r and X next obtain the permission to execute a on o through r or X does not execute a on Y through r". Informally, each dynamic clause $\phi \rightarrow (\psi_1, \psi_2)$ defines a transition relation from a security state Π to a security state Π' as follows:

- If ϕ is satisfied in Π then
  - if every action in ϕ is executed then ψ_1 will be true in the next state Π'
  - if some action in ϕ is not executed then ψ_2 will be true in Π'
- else the rule is not applied.

As such, let $\mathcal{A} \subseteq \Pi$ denote the set of the permitted actions that are actually executed. A (consistent) dynamic policy based on $\mathcal{D}$ is a finite set DP of (consistent) dynamic clauses based on $\mathcal{D}$. We shall say that the pair (Π, Π') is a transition of DP through $\mathcal{A}$, in symbols $\Pi \Longrightarrow^{\mathcal{A}}_{DP} \Pi'$, iff

- for all interpretation functions I for $\mathcal{D}$ and for all dynamic clauses $\phi \rightarrow (\psi_1, \psi_2)$ in DP, if $\Pi, I \models \phi$ then either $\mathcal{A}, I \models \phi$ and $\Pi', I \models \psi_1$ or $\mathcal{A}, I \not\models \phi$ and $\Pi', I \models \psi_2$.

The reader may easily verify that if the set $\{\Pi' : \Pi \Longrightarrow^{\mathcal{A}}_{DP} \Pi'\}$ is not empty, then it has a least element under set inclusion. Let $L(\Pi, DP, \mathcal{A})$ be this least element.

2.6 Rule-based policies

Consider a domain $\mathcal{D} = \langle S, A, O, R \rangle$ and security states Π, Π' based on $\mathcal{D}$. A (consistent) rule-based access control policy based on $\mathcal{D}$ is a tuple

$$\mathcal{P} = \langle SP, DP \rangle$$

whose first component is a static policy based on $\mathcal{D}$ and second component is a (consistent) dynamic policy based on $\mathcal{D}$. For all subsets $\mathcal{A}$ of Π, we shall say that the pair (Π, Π') is a transition of $\mathcal{P}$ through $\mathcal{A}$, in symbols $\Pi \Longrightarrow^{\mathcal{A}}_{\mathcal{P}} \Pi'$, iff

- for all interpretation functions I for $\mathcal{D}$ and for all dynamic clauses $\phi \rightarrow (\psi_1, \psi_2)$ in DP, if $\Pi, I \models \phi$ then either $\mathcal{A}, I \models \phi$ and $\Pi', I \models \psi_1$ or $\mathcal{A}, I \not\models \phi$ and $\Pi', I \models \psi_2$,
- $\Pi' \models SP$.

The reader may easily verify that the if the set $\{\Pi' : \Pi \Longrightarrow^{\mathcal{A}}_{\mathcal{P}} \Pi'\}$ is not empty, then it has a least element under set inclusion. Let $L(\Pi, \mathcal{P}, \mathcal{A})$ be this least element.

3 RBAC features

In this section, we present an encoding of RBAC and some of its extensions into our access control language.

3.1 Terminology

In the rest of this section, for the purpose of characterizing RBAC features, we consider special actions. Namely, the special actions can-play and is-active express role membership and role activation respectively. The special action Acquire denotes the acquiring of permissions relative to a role. The special actions delegate and d-play express delegation of a role and playing the role by delegation respectively.

3.2 Role activation

The essential notion in RBAC is that permissions are associated with roles and users are assigned to appropriate roles. To this end we define role membership by the predicate $P(X, \text{can-play}, X, r)$ saying that subject X has the permission to play role r. On the other hand $P(X, \text{is-active}, X, r)$ expresses that X is currently active in role r. The assignment of permissions to roles is expressed with the special action Acquire in the body of rules as follows:

$$P(X, a, o, r) \leftarrow P(X, \text{Acquire}, X, r)$$

That is, if user X has acquired the permissions associated with role r then X will have the permission to do action a on the object o. There is one rule for each triple (a, o, r) in the permission-role assignment relation.

One way for user X to acquire the permissions associated with role r is to activate the role r. This is done by executing the action can-play, and induces the addition of $P(X, \text{is-active}, X, r)$ at the next state. This is expressed by the two rules:

$$\begin{cases} P(X, \text{can-play}, X, r) \rightarrow (P(X, \text{is-active}, X, r), \top) \\ P(X, \text{Acquire}, X, r) \leftarrow P(X, \text{is-active}, X, r) \end{cases}$$

The user X can step out of a role r by simply choosing not to activate $P(X, \text{can-play}, X, r)$. In this case she automatically looses all privileges associated with role r in the next state.

Finally we impose that when user X becomes active in the role r, she must acquire the associated permissions, and this acquisition is modeled by explicit actions. This is guaranteed by the following two dynamic rules:

$$\begin{cases} P(X, \text{is-active}, X, r) \rightarrow (\top, \bot) \\ P(X, \text{Acquire}, X, r) \rightarrow (\top, \bot) \end{cases}$$

If the actions is-active and Acquire are not executed the system enters in an inconsistent state. We note that in a real system, these mandatory actions can be performed on a server as a consequence of the explicit actions of a client.

3.3 Role hierarchy

Role hierarchy is very useful in structuring roles within a certain organization. As such role, r_1 will be junior to role r_2 if the permissions associated with r_1 are inherited by members of r_2. We express this by the rule

$$P(X, \text{Acquire}, X, r_1) \leftarrow P(X, \text{Acquire}, X, r_2)$$

For example, a cardiologist can inherit the permissions associated with role intern. This can be expressed by

$$P(X, \text{Acquire}, X, \text{intern}) \leftarrow P(X, \text{Acquire}, X, \text{cardiologist})$$

If $P(\text{Mary}, \text{can-play}, \text{Mary}, \text{cardiologist})$ is true in SP and if Mary activates the role cardiologist, then $P(\text{Mary}, \text{Acquire}, \text{Mary}, \text{cardiologist})$ will be true in the next state Π'. In this case $P(\text{Mary}, \text{AcquireMary}, \text{intern})$ will also be true in Π'. That is Mary automatically acquires the permissions for role intern.

3.4 Role Delegation

Delegation is the act of authorizing or requesting someone to act on one's behalf. In order to be able to delegate a role r, an entity should be active in some role r_1 or allowed to the set of permissions associated with that role:

$$P(X, \text{delegate}, Y, r) \leftarrow P(X, \text{Acquire}, X, r_1) \wedge P(Y, \text{can-play}, Y, r_2)$$

The dynamic rules

$$\begin{cases} P(X, \text{delegate}, Y, r) \rightarrow (P(Y, \text{d-play}, Y, r), \top) \\ P(X, \text{d-play}, X, r) \rightarrow (P(X, \text{Acquire}, X, r), \top) \end{cases}$$

grant the permission for Y to play the role by delegation. If Y chooses to activate this delegation, Y will be active in the role r at the next state. Note that if X chooses not to activate the action delegate, Y looses his privileges at the next state.

3.5 Separation of duties

The separation of duty principle can be seen in both its static and dynamic aspects. In the dynamic separation of duty, a subject may have the permission to play two mutually exclusive roles, but can become active in only one of them. For example, a subject X can be both a doctor and a patient at a hospital, however X will not have the right to activate the role doctor if X is currently playing the role patient in the same hospital. We express this constraint as follows:

$$P(X, \text{Acquire}, X, \text{doctor}) \wedge P(X, \text{Acquire}, X, \text{patient}) \rightarrow (\bot, \top)$$

Note that $P(X, \text{Acquire}, X, r)$ is true only if a role r is activated, inherited or delegated, that is if X is active in the role. In that case no matter what X does, the system enters into an inconsistent state.

In the static separation of duty, a subject having the right to play the role teller in a bank for example will not be allowed to be a member of the role auditor of the same bank. A subject is considered as member of a role if she has the permission to play the role, was delegated the role or inherited the permissions associated with the role. Taking that into consideration, we define a special action $\widehat{\text{play}}$ such that

$$P(X, \widehat{\text{play}}, X, r) \leftarrow P(X, a, X, r) \text{ for } a \in \{\text{play}, \text{d-play}, \text{Acquire}\}$$

Then the static separation of duty can be expressed by the rule:

$$P(X, \widehat{\text{play}}, X, \text{teller}) \wedge P(X, \widehat{\text{play}}, X, \text{auditor}) \rightarrow (\bot, \bot)$$

If the subject X acquires both permissions at the same state of the dynamic model, then the system will enter into an inconsistent state, no matter what X decides to do.

3.6 Synchronization of actions

By synchronization, we mean the necessity to execute two actions at the same time. For example, to open the safe in a bank both an agent and a manager should enter a password otherwise the system would block:

$$P(X, \text{enter}, \text{pswrd}, \text{agent}) \wedge P(Y, \text{enter}, \text{pswrd}, \text{manager}) \rightarrow (\top, \bot)$$

In this case we do not take into consideration the difference between not executing any action or executing only one action. However, in real life, executing only one may be considered as an intrusion in the system and should not be accepted. To express this case, the above rule must be replaced by the following set of rules:

$$P(X, \text{enter}, \text{pswrd}, \text{agent}) \wedge P(Y, \text{enter}, \text{pswrd}, \text{manager}) \rightarrow (\top, p),$$

$$P(X, \text{enter}, \text{pswrd}, \text{agent}) \rightarrow (p_{\text{agent}}, \top),$$

$$P(Y, \text{enter}, \text{pswrd}, \text{manager}) \rightarrow (p_{\text{manager}}, \top),$$

$$p \wedge (p_{\text{agent}} \vee p_{\text{manager}}) \rightarrow (\bot, \bot).$$

where p, p_{agent} and p_{manager} are ground predicates that will only be used in the last rule above. This will guarantee that if the agent or the manager execute the action alone, then the system will enter into an inconsistent state.

4 Assigning permissions

This section is an encounter with the computational complexity of assigning permissions. The proofs of our results are given in the appendix. See [10] for details concerning computational complexity.

4.1 Static assignments

The $STATIC(\exists)$ problem is the following decision problem:

- $STATIC(\exists)$: given a domain $\mathcal{D}$, a static policy SP based on $\mathcal{D}$ and a condition ϕ based on $\mathcal{D}$, determine whether there exists an interpretation function I for $\mathcal{D}$ such that $l(SP), I \models \phi$.

Proposition 4.1 *$STATIC(\exists)$ is NP-complete.*

4.2 Dynamic assignments

The $DYNAMIC(\exists, \exists)$ problem and the $DYNAMIC_{con}(\exists, \exists)$ problem are the following decision problems:

- $DYNAMIC(\exists, \exists)$ ($DYNAMIC_{con}(\exists, \exists)$): given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$, a (consistent) dynamic policy DP based on $\mathcal{D}$ and a condition ϕ based on $\mathcal{D}$, determine whether there exists a subset $\mathcal{A}$ of Π, there exists an interpretation function I for $\mathcal{D}$ such that $L(\Pi, DP, \mathcal{A}), I \models \phi$.

Proposition 4.2 *$DYNAMIC_{con}(\exists, \exists)$ is NP-complete.*

Proposition 4.3 *$DYNAMIC(\exists, \exists)$ is in $\Sigma_2 P$.*

The $DYNAMIC^{path}(\exists, \exists)$ problem and the $DYNAMIC^{path}_{con}(\exists, \exists)$ problem are the following decision problems:

- $DYNAMIC^{path}(\exists, \exists)$ ($DYNAMIC^{path}_{con}(\exists, \exists)$): given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$, a (consistent) dynamic policy DP based on $\mathcal{D}$ and a condition ϕ based on $\mathcal{D}$, determine whether there exists an integer $n \geq 0$ and there exists security states $\Pi_0, \ldots, \Pi_n$ based on $\mathcal{D}$ such that
  - $\Pi_0 = \Pi$,
  - for all integers $i \geq 0$, if $1 \leq i \leq n$ then there exists a subset $\mathcal{A}$ of Π_{i-1} such that $\Pi_i = L(\Pi_{i-1}, DP, \mathcal{A})$,
  - there exists an interpretation function I for $\mathcal{D}$ such that $\Pi_n, I \models \phi$.

Proposition 4.4 *$DYNAMIC^{path}_{con}(\exists, \exists)$ is $PSPACE$-complete.*

Proposition 4.5 *$DYNAMIC^{path}(\exists, \exists)$ is $PSPACE$-complete.*

4.3 Rule-based assignments

The $RULEBASED(\exists,\exists)$ problem and the $RULEBASED_{con}(\exists,\exists)$ problem are the following decision problems:

- $RULEBASED(\exists,\exists)$ ($RULEBASED_{con}(\exists,\exists)$): given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$, a (consistent) rule-based policy $\mathcal{P}$ based on $\mathcal{D}$ and a condition ϕ based on $\mathcal{D}$, determine whether there exists a subset $\mathcal{A}$ of Π, there exists an interpretation function I for $\mathcal{D}$ such that $L(\Pi,\mathcal{P},\mathcal{A}), I \models \phi$.

Proposition 4.6 *$RULEBASED_{con}(\exists,\exists)$ is NP-complete.*

Proposition 4.7 *$RULEBASED(\exists,\exists)$ is in $\Sigma_2 P$.*

The $RULEBASED^{path}(\exists,\exists)$ problem and the $RULEBASED^{path}_{con}(\exists,\exists)$ problem are the following decision problems:

- $RULEBASED^{path}(\exists,\exists)$ ($RULEBASED^{path}_{con}(\exists,\exists)$): given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$, a (consistent) rule-based policy $\mathcal{P}$ based on $\mathcal{D}$ and a condition ϕ based on $\mathcal{D}$, determine whether there exists an integer $n \geq 0$ and there exists security states Π_0, ..., Π_n based on $\mathcal{D}$ such that
  - $\Pi_0 = \Pi$,
  - for all integers $i \geq 0$, if $1 \leq i \leq n$ then there exists a subset $\mathcal{A}$ of Π_{i-1} such that $\Pi_i = L(\Pi_{i-1},\mathcal{P},\mathcal{A})$,
  - there exists an interpretation function I for $\mathcal{D}$ such that $\Pi_n, I \models \phi$.

Proposition 4.8 *$RULEBASED^{path}_{con}(\exists,\exists)$ is $PSPACE$-complete.*

Proposition 4.9 *$RULEBASED^{path}(\exists,\exists)$ is $PSPACE$-complete.*

5 Conclusion

In our framework, a role-based access control policy consists in a set of static clauses and a set of dynamic clauses defined in terms of permissions. Static clauses characterize what remains true during the life of a system whereas dynamic clauses characterize the different ways according to which the system can change. We have provided examples on how to express RBAC features using static clauses and dynamic clauses and we have addressed the complexity issue of some decision problems related to the assignment of permissions with respect to such-or-such policy.
In other respect, we have seen how the Boolean connective $\perp$ in the right-hand side of well-formed conditions can be used in order to express RBAC features such as separation of duties and synchronisation of actions. Seeing that it is essential for the system to never enter into an inconsistent state, we plan to

study the computational complexity of the decision problem consisting, given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$ and a rule-based policy $\mathcal{P}$ based on $\mathcal{D}$, to determine whether there exists an infinite sequence of transitions of $\mathcal{P}$ beginning with Π. Finally, permissions are often associated to temporal constraints: permissions are given for such-or-such periods of time. Extending our current language with such temporal constraints would permit the expression of more useful policies.

Acknowledgements

The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 216471, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (www.avantssar.eu). We are also grateful for the support of the project ANR-05-SSIA-0007-01 *Cops* financed by the "Agence nationale de la recherche" (www.irit.fr/COPS/Accueil.htm).

References

[1] Abadi, M. *Logic in access control.* In 18th IEEE Symposium on logic in Computer Science (LICS 2003). IEEE Computer Society (2003) 228–233.

[2] Becker, M., Sewell, P. *Cassandra: distributed access control policies with tunable expressiveness.* In 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04). IEEE Computer Society (2004) 159–168.

[3] Becker, M., Sewell, P. *Cassandra: flexible trust management, applied to electronic health records.* In Proceedings of the 17th IEEE Computer Science Foundations Workshop (CSFW 2004). IEEE Computer Society (2004) 139–154.

[4] Bell, D., LaPadula, L. *Secure Computer Systems: Mathematical Foundations.* MITRE Corporation (1973).

[5] Bonner, A.J., Kifer, M. *An overview of transaction logic.* Theoretical Computer Science 133(2), 205-265 (1994).

[6] Ferraiolo, D., Kuhn, D. *Role-based access controls.* In 15th NIST-NCSC National Computer Security Conference. (1992) 554–563.

[7] Garey, M., Johnson, D. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freeman (1979).

[8] Harrison , M., Ruzzo, W., Ullman, J. *On protection in operating systems.* Communications of the ACM **19** (1976) 461–471.

[9] Li, N., Mitchell, J., Winsborough, W. *Design of a role-based trust-management framework.* In Symposium on Security and Privacy. IEEE Computer Society (2002) 114–130.

[10] Papadimitriou, C. *Computational Complexity.* Addison-Wesley (1994).

[11] Zhang, X., Oh, S., Sandhu, R. *PDBM: A flexible delegation model in RBAC.* In Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003). ACM (2003) 149 – 157.

Appendix

In this appendix, we provide the proofs of propositions 4.1 to 4.9.

Proof of proposition 4.1. $STATIC(\exists)$ is in NP. It suffices to prove the existence of an algorithm in NP that solves $STATIC(\exists)$. Let us consider the following algorithm:

1. First, choose an enumeration A_1, ..., A_n of some subset of the set of all ground atomic formulas based on $\mathcal{D}$.
2. Second, choose a ground instance ϕ' of ϕ based on $\mathcal{D}$.
3. Third, for all integers $i \geq 0$, if $1 \leq i \leq n$ then check whether A_i can be inferred in one step from A_1, ..., A_{i-1} and some static clause in SP.
4. Fourth, check whether $\{A_1, \ldots, A_n\} \models \phi'$.

The reader may easily verify that this algorithm can be executed in nondeterministic polynomial time.
$STATIC(\exists)$ is NP-hard. It suffices to prove the existence of a reduction from an NP-hard problem to $STATIC(\exists)$. We shall say that a connected graph $G = (V, E)$ is 3-colorable iff there exists a function f in $\{0, 1, 2\}^V$ such that for all u, v in V, if (u, v) is in E then $f(u) \neq f(v)$. Let us consider the following decision problem:

- 3-$COLORABILITY$: given a connected graph G, determine whether G is 3-colorable.

It is well-known that 3-$COLORABILITY$ is NP-hard [7]. Given a connected graph $G = (V, E)$, the instance $\rho(G)$ of the $STATIC(\exists)$ problem that we construct is given by the domain $\mathcal{D}_G$, the static policy SP_G based on $\mathcal{D}_G$ and the condition ϕ_G based on $\mathcal{D}_G$ defined by

- $\mathcal{D}_G = \langle \{0, 1, 2\}, \{a\}, \{0, 1, 2, \}, \{r\} \rangle$,
- $SP_G = \{P(i, a, j, r) :\ 0 \leq i, j \leq 2 \text{ and } i \neq j\}$,
- $\phi_G = \bigwedge \{P(X_u, a, X_v, r) :\ (u, v) \text{ is in } E\}$.

This completes the construction. Obviously, ρ can be computed in logarithmic space. Moreover, the reader may easily verify that G is 3-colorable iff there exists an interpretation function I for $\mathcal{D}_G$ such that $l(SP_G), I \models \phi_G$. Hence, ρ is a reduction from 3-$COLORABILITY$ to $STATIC(\exists)$.

Proof of proposition 4.2. $DYNAMIC_{con}(\exists, \exists)$ is in NP. It suffices to prove the existence of an algorithm in NP that solves $DYNAMIC_{con}(\exists, \exists)$. Let us consider the following algorithm:

1. First, choose a subset $\mathcal{A}$ of Π.
2. Second, choose an enumeration $A_1, \ldots, A_n$ of some subset of the set of all ground atomic formulas based on $\mathcal{D}$.
3. Third, choose a ground instance ϕ' of ϕ based on $\mathcal{D}$.
4. Fourth, for all integers $i \geq 0$, if $1 \leq i \leq n$ then check whether A_i can be inferred in one step from Π, $\mathcal{A}$ and some dynamic clause in DP.
5. Fifth, check whether $\{A_1, \ldots, A_n\} \models \phi'$.

The reader may easily verify that this algorithm can be executed in nondeterministic polynomial time.
$DYNAMIC_{con}(\exists, \exists)$ is NP-hard. It suffices to prove the existence of a reduction from an NP-hard problem to $DYNAMIC_{con}(\exists, \exists)$. Let us consider the 3-$COLORABILITY$ problem. Given a connected graph $G = (V, E)$, the instance $\rho(G)$ of the $DYNAMIC_{con}(\exists, \exists)$ problem that we construct is given by the domain $\mathcal{D}_G$, the security state Π_G based on $\mathcal{D}_G$, the consistent dynamic policy DP_G based on $\mathcal{D}_G$ and the condition ϕ_G based on $\mathcal{D}_G$ defined by

- $\mathcal{D}_G = \langle \{0, 1, 2\}, \{a\}, \{0, 1, 2, r\}, \{r\} \rangle$,
- $\Pi_G = \emptyset$,
- $DP_G = \{\top \to (\bigwedge \{P(i, a, j, r) : \ 0 \leq i, j \leq 2 \text{ and } i \neq j\}, \top)\}$,
- $\phi_G = \bigwedge \{P(X_u, a, X_v, r) : \ (u, v) \text{ is in } E\}$.

This completes the construction. Obviously, ρ can be computed in logarithmic space. Moreover, the reader may easily verify that G is 3-colorable iff there exists a subset $\mathcal{A}$ of Π_G, there exists an interpretation function I for $\mathcal{D}_G$ such that $L(\Pi_G, DP_G, \mathcal{A}), I \models \phi_G$. Hence, ρ is a reduction from 3-$COLORABILITY$ to $DYNAMIC_{con}(\exists, \exists)$.

The $INCONSISTENT$ problem is the following decision problem:

- $INCONSISTENT$: given a domain $\mathcal{D}$, a security state Π based on $\mathcal{D}$, a subset $\mathcal{A}$ of Π and a dynamic policy DP based on $\mathcal{D}$, determine whether there exists a dynamic clause $\phi \to (\psi_1, \psi_2)$ in DP and there exists an interpretation function I for $\mathcal{D}$ such that $\Pi, I \models \phi$ and either $\mathcal{A}, I \models \phi$ and $\psi_1 = \bot$ or $\mathcal{A}, I \not\models \phi$ and $\psi_2 = \bot$.

We will use the following result in the proof of proposition 4.3.

Proposition .1 *$INCONSISTENT$ is NP-complete.*

Proof of proposition .1. $INCONSISTENT$ is in NP. It suffices to prove the existence of an algorithm in NP that solves $INCONSISTENT$. Let us consider the following algorithm:

1. First, choose a dynamic clause $\phi \to (\psi_1, \psi_2)$ in DP.
2. Second, choose an interpretation function I for $\mathcal{D}$.
3. Third, check whether $\Pi, I \models \phi$ and either $\mathcal{A}, I \models \phi$ and $\psi_1 = \bot$ or $\mathcal{A}, I \not\models \phi$ and $\psi_2 = \bot$.

The reader may easily verify that this algorithm can be executed in nondeterministic polynomial time.
$INCONSISTENT$ is NP-hard. It suffices to prove the existence of a reduction from an NP-hard problem to $INCONSISTENT$. Let us consider the 3-*COLORABILITY* problem. Given a connected graph $G = (V, E)$, the instance $\rho(G)$ of the $INCONSISTENT$ problem that we construct is given by the domain $\mathcal{D}_G$, the security state Π_G based on $\mathcal{D}_G$, the subset $\mathcal{A}_G$ of Π_G and the dynamic policy DP_G based on $\mathcal{D}_G$ defined by

- $\mathcal{D}_G = \langle \{0, 1, 2\}, \{a\}, \{0, 1, 2\}, \{r\} \rangle$,
- $\Pi_G = \{(i, a, j, r) : \ 0 \leq i, j \leq 2 \text{ and } i \neq j\}$,
- $\mathcal{A}_G = \{0, 1, 2\} \times \{a\} \times \{0, 1, 2\} \times \{r\}$,
- $DP_G = \{\bigwedge\{P(X_u, a, X_v, r) : \ (u, v) \text{ is in } E\} \to (\bot, \top)\}$.

This completes the construction. Obviously, ρ can be computed in logarithmic space. Moreover, the reader may easily verify that G is 3-colorable iff there exists a dynamic clause $\phi \to (\psi_1, \psi_2)$ in DP_G and there exists an interpretation function I for $\mathcal{D}_G$ such that $\Pi_G, I \models \phi$ and either $\mathcal{A}, I \models \phi$ and $\psi_1 = \bot$ or $\mathcal{A}, I \not\models \phi$ and $\psi_2 = \bot$. Hence, ρ is a reduction from 3-*COLORABILITY* to $INCONSISTENT$.

Proof of proposition 4.3. It suffices to prove the existence of an algorithm in NP^{NP} that solves $DYNAMIC(\exists, \exists)$. Let us consider the following algorithm:

1. First, choose a subset $\mathcal{A}$ of Π.
2. Second, choose an enumeration $A_1, \ldots, A_n$ of some subset of the set of all ground atomic formulas based on $\mathcal{D}$.
3. Third, choose a ground instance ϕ' of ϕ based on $\mathcal{D}$.
4. Fourth, check whether $INCONSISTENT(\Pi, \mathcal{A}, DP)$ returns “no”.

5. Fifth, for all integers $i \geq 0$, if $1 \leq i \leq n$ then check whether A_i can be inferred in one step from Π, $\mathcal{A}$ and some dynamic clause in DP.

6. Sixth, check whether $\{A_1, \ldots, A_n\} \models \phi'$.

The reader may easily verify that this algorithm with oracle $INCONSISTENT$ in NP can be executed in nondeterministic polynomial time. In contrast to the aforementioned decision problems, we still do not know if $DYNAMIC(\exists, \exists)$ is complete with respect to the class $\Sigma_2 P$.

Proof of proposition 4.4. $DYNAMIC_{con}^{path}(\exists, \exists)$ is in $PSPACE$. It suffices to prove the existence of an algorithm in $NPSPACE$ that solves $DYNAMIC_{con}^{path}(\exists, \exists)$. Let us informally describe such an algorithm. First, choose an interpretation function I for $\mathcal{D}$. Second, check whether $\Pi, I \models \phi$. Third, if a negative answer is returned then we choose a subset $\mathcal{A}$ of Π, we let $\Pi := L(\Pi, DP, \mathcal{A})$ and we move to the first step. The reader may easily verify that this algorithm can be executed in nondeterministic polynomial space.
$DYNAMIC_{con}^{path}(\exists, \exists)$ is $PSPACE$-hard. It suffices to prove that any language in $PSPACE$ can be reduced to $DYNAMIC_{con}^{path}(\exists, \exists)$. Let $L \subseteq \{0, 1\}^\star$ be a language in PSPACE. Hence, there is a polynomial-space-bounded Turing machine M such that for all x in $\{0, 1\}^\star$, x is in L iff M accepts x. The machine M has three components (Q, Σ, δ) where Q is the set of states of M, Σ is the set of symbols of M, and δ is the transition function of M. We assume that there is q_0 in Q, the start state of M, and there is q_1 in Q, the final state of M, such that $q_0 \neq q_1$. We suppose that 0 is in Σ and 1 is in Σ. We also assume that there is B in Σ, the blank symbol of M, such that $0 \neq B$ and $1 \neq B$. The transition function δ assigns $\delta(q, A)$ in $Q \times \Sigma \times \{L, R\}$ to each q in Q and each A in Σ. If $\delta(q, A) = (q', A', L)$ or $\delta(q, A) = (q', A', R)$ then it means that whenever the machine is in state q and scans an A on its tape, it changes its state to q', replaces the A by a A' and moves its tape head leftward or rightward. We suppose that q_0 is not in the range of δ and q_1 is not in the domain of δ. We also assume that M never falls off the left end of its input string $x \in \{0, 1\}^\star$. The machine accepts x in $\{0, 1\}^\star$ iff starting in state q_0 and scanning the left end of x on its tape, preceded and followed by an infinity of blanks, M finally enters in state q_1. The machine M is allowed to use an amount of space that is polynomial in the size of its input, no matter how much time it uses. Consequently, there is a polynomial $p(n)$ such that when given input x in $\{0, 1\}^\star$ of length n, M never visits more than $p(n)$ cells of its tape. Hence, there is a positive integer k such that for all x in $\{0, 1\}^\star$, M uses at most 2^{N^k} positions on its tape when given input x in $\{0, 1\}^\star$ of length N. We are now ready to reduce the following decision problem to $DYNAMIC_{con}^{path}(\exists, \exists)$:

- $P(L)$: given x in $\{0, 1\}^\star$, determine whether M accept x.

Given x in $\{0, 1\}^\star$, it is convenient to write $x = x_1 \ldots x_N$ where N is the length of x. For all q in Q and for all A in Σ, if $\delta(q, A) = (q', A', L)$ then for all A'' in Σ and for all i in $\{1, \ldots, N^k\}$, we let $dc_L(q, A, A'', i)$ be the following dynamic clause:

$$\phi_L(q, A, A'', i) \to (\psi_L(q, A, A'', i), \psi_L(q, A, A'', i))$$

where

$$\phi_L(q, A, A'', i) = P(X_1, 1, X_1, r) \wedge \ldots \wedge P(X_{i-2}, i-2, X_{i-2}, r)$$

$$\wedge P(A'', i-1, A'', r) \wedge P((q, A), i, (q, A), r)$$

$$\wedge P(X_{i+1}, i+1, X_{i+1}, r) \wedge \ldots \wedge P(X_{N^k}, N^k, X_{N^k}, r)$$

and

$$\psi_L(q, A, A'', i) = P(X_1, 1, X_1, r) \wedge \ldots \wedge P(X_{i-2}, i-2, X_{i-2}, r)$$

$$\wedge P((q', A''), i-1, (q', A''), r) \wedge P(A', i, A', r)$$

$$\wedge P(X_{i+1}, i+1, X_{i+1}, r) \wedge \ldots \wedge P(X_{N^k}, N^k, X_{N^k}, r).$$

For all q in Q and for all A in Σ, if $\delta(q, A) = (q', A', R)$ then for all A'' in Σ and for all i in $\{1, \ldots, N^k\}$, we let $dc_R(q, A, A'', i)$ be the following dynamic clause:

$$\phi_R(q, A, A'', i) \to (\psi_R(q, A, A'', i), \psi_R(q, A, A'', i))$$

where

$$\phi_R(q, A, A'', i) = P(X_1, 1, X_1, r) \wedge \ldots \wedge P(X_{i-1}, i-1, X_{i-1}, r)$$

$$\wedge P((q, A), i, (q, A), r) \wedge P(A'', i+1, A'', r)$$

$$\wedge P(X_{i+2}, i+2, X_{i+2}, r) \wedge \ldots \wedge P(X_{N^k}, N^k, X_{N^k}, r)$$

and

$$\psi_R(q, A, A'', i) = P(X_1, 1, X_1, r) \wedge \ldots \wedge P(X_{i-1}, i-1, X_{i-1}, r)$$

$$\wedge P(A', i, A', r) \wedge P((q', A''), i+1, (q', A''), r)$$

$$\wedge P(X_{i+2}, i+2, X_{i+2}, r) \wedge \ldots \wedge P(X_{N^k}, N^k, X_{N^k}, r).$$

The instance $\rho(x)$ of the $DYNAMIC^{path}_{con}(\exists, \exists)$ problem that we construct is given by the domain $\mathcal{D}_x$, the security state Π_x based on $\mathcal{D}_x$, the consistent dynamic policy DP_x based on $\mathcal{D}_x$ and the condition ϕ_x based on $\mathcal{D}_x$ defined by

- $\mathcal{D}_x = \langle (Q \times \Sigma) \cup \Sigma, \{1, \ldots, N^k\}, (Q \times \Sigma) \cup \Sigma, \{r\} \rangle$,
- $\Pi_x = \{((q_0, x_1), 1, (q_0, x_1), r), (x_2, 2, x_2, r), \ldots, (x_{N^k}, N^k, x_{N^k}, r)\}$,
- $DP_x = \{dc_L(q, A, A'', i) :$ q is in Q, A is in Σ, $\delta(q, A) = (q', A', L)$, A'' is in Σ and i in $\{1, \ldots, N^k\}\} \cup \{dc_R(q, A, A'', i) :$ q is in Q, A is in Σ, $\delta(q, A) = (q', A', R)$, A'' is in Σ and i in $\{1, \ldots, N^k\}\}$,
- $\phi_x = \bigvee \{P((q_1, A), i, (q_1, A), r) :$ A is in Σ and i is in $\{1, \ldots, N^k\}\}$.

This completes the construction. Obviously, ρ can be computed in logarithmic space. Moreover, the reader may easily verify that M accepts x iff there exists an integer $n \geq 0$ and there exists security states $\Pi_0, \ldots, \Pi_n$ based on $\mathcal{D}_x$ such that

- $\Pi_0 = \Pi_x$,
- for all integers $i \geq 0$, if $1 \leq i \leq n$ then there exists a subset $\mathcal{A}$ of Π_{i-1} such that $\Pi_i = L(\Pi_{i-1}, DP_x, \mathcal{A})$,
- there exists an interpretation function I for $\mathcal{D}_x$ such that $\Pi_n, I \models \phi_x$.

Hence, ρ is a reduction from $P(L)$ to $DYNAMIC_{con}^{path}(\exists, \exists)$.

Proof of proposition 4.5. Similar to the proof of proposition 4.4.

Proof of proposition 4.6. Similar to the proof of proposition 4.2.

Proof of proposition 4.7. Similar to the proof of proposition 4.3.

Proof of proposition 4.8. Similar to the proof of proposition 4.4.

Proof of proposition 4.9. Similar to the proof of proposition 4.5.