

A logical approach to role-based access control in a distributed environment

Philippe Balbiani, Yannick Chevalier and Marwa El Hourri

Université Paul Sabatier, IRIT

COPS - February 2008

- Express access control policies in distributed systems.
- Take into consideration the RBAC structure with its extensions to include role hierarchy, delegation, separation of duties and obligations in a structured manner.
- Express the dynamic aspect of a policy (evolution in time)

- Express access control policies in distributed systems.
- Take into consideration the RBAC structure with its extensions to include role hierarchy, delegation, separation of duties and obligations in a structured manner.
- Express the dynamic aspect of a policy (evolution in time)
- We present a language based on a set of rules (static rules and dynamic rules).

Description of the language

- Domain: $D = \langle S, R, A, O \rangle$ where S = set of subjects, R = set of roles, A = set of actions and O = set of objects.
- Security state: $\mathcal{S} = \langle \Omega, \Pi \rangle$ such that $\Omega \subseteq \Pi$ with:
 - $O(s, r, a, o) \in \Omega$ "subject s has in S the obligation to execute action a on object o through role r "
 - $P(s, r, a, o) \in \Pi$ "subject s has in S the permission to execute action a on object o through role r "

We define

- A formula

$$\phi := \top \mid A \mid \phi \wedge \phi$$

where A is $P(s, r, a, o)$ or $O(s, r, a, o)$

- A static clause based on D is an expression of the form

$$A \leftarrow \phi$$

We define

- A formula

$$\phi := \top | A | \phi \wedge \phi$$

where A is $P(s, r, a, o)$ or $O(s, r, a, o)$

- A static clause based on D is an expression of the form

$$A \leftarrow \phi$$

Example

$$P(x, user, read, file) \leftarrow P(x, user, write, file)$$

x has the permissions to read the file through role user if he/she has the permission to write on the file through role user.

Definition

A static policy SP is a finite set of static clauses based on D

$S \models SP$ iff for all interpretations I for D and all static clauses $A \leftarrow \phi$ in SP , if $S, I \models \phi$ then $S, I \models A$.

Problem: Starting from a security state \mathcal{S}

- Some actions are permitted in \mathcal{S}
- A user executes a subset \mathcal{A} of these permitted actions
- These changes may affect the security state

We have to specify these changes.

- A dynamic clause is an expression of the form:

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

where

- ϕ , ψ_1 and ψ_2 are conjunctions of permissions and/or obligations

- A dynamic clause is an expression of the form:

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

where

- ϕ , ψ_1 and ψ_2 are conjunctions of permissions and/or obligations
- Read:
 - if** all permissions/obligations in ϕ are true in \mathcal{S}
then **if** all the "actions in ϕ " are executed
then ψ_1 will be true in the next state \mathcal{S}'
else ψ_2 will be true in the next state \mathcal{S}'
else the rule is not applied.

Definition

An access control policy based on D is a tuple

$$P = \langle SP, DP \rangle$$

Transition

For all subsets \mathcal{A} of Π we define the transition $S \Rightarrow_{DP}^{\mathcal{A}} S'$ iff

- for all I and all dynamic clauses $\phi \rightarrow (\psi_1, \psi_2)$ in DP , if $S, I \models \phi$ then either $\langle \mathcal{A}, \mathcal{A} \rangle, I \models \phi$ and $S', I \models \psi_1$ or $\langle \mathcal{A}, \mathcal{A} \rangle, I \not\models \phi$ and $S', I \models \psi_2$
- $S' \models SP$

Example

- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \leftarrow$
- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \rightarrow$
 $(O(\text{Mary}, \text{cardiologist}, \text{follow - up}, \text{patient}), \top)$

Mary has the permission to *operate* on a *patient* through role *cardiologist* ($P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient})$ is in \mathcal{S}) then

Example

- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \leftarrow$
- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \rightarrow$
 $(O(\text{Mary}, \text{cardiologist}, \text{follow} - \text{up}, \text{patient}), \top)$

Mary has the permission to *operate* on a *patient* through role *cardiologist* ($P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient})$ is in \mathcal{S}) then

- if the action *operate* is executed *Mary* obtains the obligation to *follow – up* on *patient* through role *cardiologist* ($O(\text{Mary}, \text{cardiologist}, \text{follow} - \text{up}, \text{patient})$ true in \mathcal{S}')

Example

- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \leftarrow$
- $P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient}) \rightarrow$
 $(O(\text{Mary}, \text{cardiologist}, \text{follow} - \text{up}, \text{patient}), \top)$

Mary has the permission to *operate* on a *patient* through role *cardiologist* ($P(\text{Mary}, \text{cardiologist}, \text{operate}, \text{patient})$ is in \mathcal{S}) then

- if the action *operate* is executed *Mary* obtains the obligation to *follow – up* on *patient* through role *cardiologist* ($O(\text{Mary}, \text{cardiologist}, \text{follow} - \text{up}, \text{patient})$ true in \mathcal{S}')
- if this action is not executed nothing happens (\top true in \mathcal{S}').

Role Activation

- User-role relations
- Permission-role relation
- Obligation-role relation

- To have the permission of playing a role: $can - play(s, r)$

Example

$can - play(Mary, Doctor) \leftarrow$

expresses that *Mary* has the permission to play the role doctor

- To have the permission of playing a role: $can - play(s, r)$

Example

$can - play(Mary, Doctor) \leftarrow$

expresses that *Mary* has the permission to play the role doctor

- To be active in a role: $is - active(s, r)$

Example

$can - play(Mary, Doctor) \rightarrow (is - active(Mary, Doctor), \top)$ If *Mary* chooses to activate role Doctor, *Mary* becomes active as a doctor.

Permission-role and Obligation-role relations

- $Acquire_{perm}(s, r)$ assigns the permissions for the role r to a subject s .
- $Acquire_{obl}(s, r)$ assigns obligations for the role r to a subject s .

Example

$Acquire_{perm}(x, Doctor) \leftarrow is - active(x, Doctor)$

$Acquire_{obl}(x, Doctor) \leftarrow is - active(x, Doctor)$

Example

$$Acquire_{perm}(x, Doctor) \leftarrow is - active(x, Doctor)$$
$$Acquire_{obl}(x, Doctor) \leftarrow is - active(x, Doctor)$$

Permission and obligation acquisition

$$P(x, Doctor, a, o) \leftarrow Acquire_{perm}(x, Doctor)$$
$$O(x, Doctor, a, o) \leftarrow Acquire_{obl}(x, Doctor)$$

In RBAC

- A role is a set of permissions associated with a set of users
- A role hierarchy is such that a subject acquires permissions to a role r if:
 - the subject is a member of role r , or
 - the role r is junior to the subject's role.

In RBAC

- A role is a set of permissions associated with a set of users
- A role hierarchy is such that a subject acquires permissions to a role r if:
 - the subject is a member of role r , or
 - the role r is junior to the subject's role.

In our model:

- A role is a set of **permissions** and **obligations** associated with a set of users
- Two hierarchies (permissions and obligations)

Hierarchy relative to permissions

A *cardiologist* is likely to inherit the permissions for the role *doctor* but less likely to inherit the obligations for the role *doctor*.

A *cardiologist* is likely to inherit the permissions for the role *doctor* but less likely to inherit the obligations for the role *doctor*.

- We define an order on roles relative to permissions

Definition

$r_2 <_{perm} r_1$ says that role r_1 can inherit the permissions associated with role r_2 .

Rules

- $is - active(Mary, cardiologist)$
- $doctor <_{perm} cardiologist$
- $intern <_{perm} doctor$
- $Acquire_{perm}(x, r_1) \leftarrow Acquire_{perm}(x, r_2) \wedge (r_1 <_{perm} r_2)$

Rules

- $is - active(Mary, cardiologist)$
- $doctor <_{perm} cardiologist$
- $intern <_{perm} doctor$
- $Acquire_{perm}(x, r_1) \leftarrow Acquire_{perm}(x, r_2) \wedge (r_1 <_{perm} r_2)$

Consequences:

- $Acquire_{perm}(Mary, cardiologist)$
- $Acquire_{perm}(Mary, doctor)$
- $Acquire_{perm}(Mary, intern)$

Definition

Executing an action by delegation is to execute it on one's behalf.

Two types of delegation

- Allowing an entity to be delegatee of a role by action
p – delegate
- Forcing an entity to be delegatee of a role by action
o – delegate

We suppose:

$$P(x, r, p - \text{delegate}, y) \leftarrow \text{Acquire}_{\text{perm}}(x, r_1) \wedge \text{can} - \text{play}(y, r_2)$$

We suppose:

$$P(x, r, p - \text{delegate}, y) \leftarrow \text{Acquire}_{\text{perm}}(x, r_1) \wedge \text{can} - \text{play}(y, r_2)$$

Example

$$P(x, \text{pers} - \text{assistant}, p - \text{delegate}, y) \leftarrow \\ \text{Acquire}_{\text{perm}}(x, \text{cardiologist}) \wedge \text{can} - \text{play}(y, \text{intern})$$

We suppose:

$$P(x, r, p - \textit{delegate}, y) \leftarrow \textit{Acquire}_{\textit{perm}}(x, r_1) \wedge \textit{can} - \textit{play}(y, r_2)$$

Example

$$P(x, \textit{pers} - \textit{assistant}, p - \textit{delegate}, y) \leftarrow \\ \textit{Acquire}_{\textit{perm}}(x, \textit{cardiologist}) \wedge \textit{can} - \textit{play}(y, \textit{intern})$$

- *Mary* is a *cardiologist*, she has the permission to delegate the role personal assistant to an intern.
- *Mary's* personal assistant is sick and she wishes to delegate this role to the intern *John*

Delegating a role

$P(\text{Mary}, \text{pers} - \text{assistant}, p - \text{delegate}, \text{John}) \rightarrow$
 $(P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John}), \top)$

Delegating a role

$$P(\text{Mary}, \text{pers} - \text{assistant}, p - \text{delegate}, \text{John}) \rightarrow (P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John}), \top)$$

- When *Mary* executes the action *p – delegate* *John* acquires the permission $P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John})$

Delegating a role

$$P(\text{Mary}, \text{pers} - \text{assistant}, p - \text{delegate}, \text{John}) \rightarrow (P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John}), \top)$$

- When *Mary* executes the action *p – delegate* *John* acquires the permission $P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John})$

$$P(\text{John}, \text{pers} - \text{assistant}, d - \text{play}, \text{John}) \rightarrow (\text{is} - \text{active}(\text{John}, \text{pers} - \text{assistant}) \wedge O(\text{John}, \text{pers} - \text{assistant}, \text{write}, \text{report}), \top)$$

- If *John* accepts the delegated role then he will be active in the role personal assistant but he will have an additional obligation to write a report on his work.

Delegation revocation

- *Mary*'s personal assistant recovers from his sickness
- *Mary* stops the delegation by choosing not to execute the action p – *delegate*
- *John* loses all the privileges of the delegates role at the next state.

- Relations of core RBAC can be encoded
- We can express hierarchies (for permissions and obligations)
- Delegation: we can dynamically alter the role inheritance relation and the user-role assignment.
- We can also express static and dynamic separation of duty and synchronization of actions.

Example

- The information in the client information file cannot be accessed without the client authorization.
- The client can grant authorization to a file that he/she owns.
- A clerk is responsible for receiving and identifying the client.
- A clerk can modify data in the client information file if he/she has access to the file.

The need to add new entities:

Add the entity *View*:

The *View* is an entity that gathers objects of the same "type"

- Views are used to express a policy on objects that share the same characteristics.
- Example: A file *John – info.doc* is an object in the *View file*.
- $Is - a(John - info.doc, file)$

Add the entity Context:

The context describes the constraints or relations between the subject, action and object within a permission (or obligation)

- Example: *Ownership* is a context that associate the ownership of the object to the subject.
- $Define(x, client, own, f, file, Ownership) \longleftrightarrow name(x) = owner(f)$
- *name* is an attribute for the subject and *owner* is an attribute for the object.

Define:

- A set of facts of the form $P(r, a, v, c)$
- A general rule $perm(s, r, a, o, v, c) \leftarrow is - active(s, r) \wedge is - a(o, v) \wedge P(r, a, v, c) \wedge Define(s, r, a, o, v, c)$
- Dynamic rules of the form $\phi \rightarrow (C_1, C_2)$ where
 - ϕ is a conjunction of $perm$, or $is - active$
 - C_1 and C_2 are conjunctions of $perm$, $is - active$, $is - a$ or \top

In static policy

The permissions of executing an action by a role on a view are stated:

- $P(\text{client}, \text{send}, \text{signature}, \text{ownership})$
- $P(\text{clerk}, \text{receive}, \text{signature}, \text{Access})$
- $P(\text{clerk}, \text{access}, \text{file}, \text{Access}) \leftarrow P(\text{clerk}, \text{receive}, \text{signature}, \text{Access})$
- $P(\text{clerk}, \text{modify}, \text{file}, \text{Access}) \leftarrow P(\text{clerk}, \text{access}, \text{file}, \text{Access})$
- $\text{Define}(s, r, a, o, v, \text{Access}) \longleftrightarrow is - \text{active}(x, \text{client}) \wedge \text{name}(x) = \text{owner}(o)$

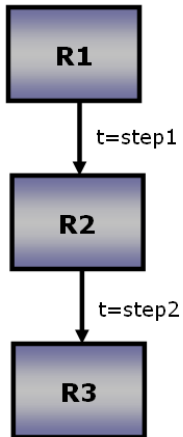
In dynamic policy

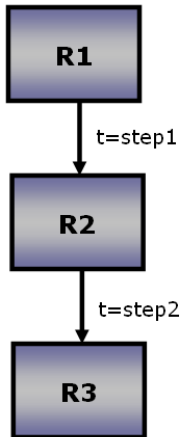
The permissions of executing an action by a subject in a role on an object in a view are acquired

- $perm(x, client, send, o, signature, ownership) \rightarrow (perm(y, pre - clerk, receive, o, signature, Access), \top)$
- $perm(y, post - clerk, access, o, file, Access) \rightarrow perm(y, post - clerk, modify, o, file, Access), \top)$

- An action in a dynamic rule is "executed by a subject on an object"
- The execution of an action can lead to another permission

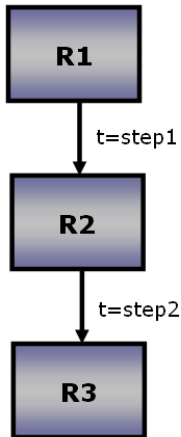
State transition: Modeling of a workflow





Static rules

- $P(R_1, \text{move}, \text{step1}, \text{context}) \leftarrow$
- $P(R_2, \text{move}, \text{step2}, \text{context}) \leftarrow$
 $P(R_1, \text{move}, \text{step1}, \text{context})$
- $P(R_3, \text{move}, \text{end}, \text{context}) \leftarrow$
 $P(R_2, \text{move}, \text{step2}, \text{context})$



Static rules

- $P(R_1, move, step1, context) \leftarrow$
- $P(R_2, move, step2, context) \leftarrow P(R_1, move, step1, context)$
- $P(R_3, move, end, context) \leftarrow P(R_2, move, step2, context)$

Dynamic rules

- $perm(s, R_1, move, t, step1, context) \rightarrow (is - active(s, R_2) \wedge is - a(t, step2), *)$
- $perm(s, R_2, move, t, step2, context) \rightarrow (is - active(s, R_3) \wedge is - a(t, end), *)$

- We described a language capable of expressing a policy that can evolve over time according to the actions performed by users
- We presented how we can express RBAC and its extensions into our language
- We gave example of an extension to a more complex environment involving views and contexts

- Test the expressive power of the language and possible extension via examining a case study
- Examine the notion of communication between roles
- Extend the language to support temporal constraints