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

Philippe Balbiani, Yannick Chevalier and Marwa El Houri

Université Paul Sabatier, IRIT

COPS - February 2008
Motivation

- 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).
Outline
A logical approach to RBAC
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**: $S = \langle \Omega, \Pi \rangle$ 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 $r$"
Static policy

We define

- A formula
  \[ \phi := T | A | \phi \land \phi \]
  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

\[ \phi := T | A | \phi \land \phi \]

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**

\[ P(x, \text{user}, \text{read}, \text{file}) \leftarrow P(x, \text{user}, \text{write}, \text{file}) \]

\( x \) 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 $S, I \models A$. 
Stateful security policies

Problem: Starting from a security state $S$

- Some actions are permitted in $S$
- A user executes a subset $A$ 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
- $\phi, \psi_1$ and $\psi_2$ are conjunctions of permissions and/or obligations
A dynamic clause is an expression of the form:

\[ \phi \rightarrow (\psi_1, \psi_2) \]

where
- \( \phi, \psi_1 \) and \( \psi_2 \) are conjunctions of permissions and/or obligations

Read:

if all permissions/obligations in \( \phi \) are true in \( S \)
then if all the "actions in \( \phi \)" are executed
then \( \psi_1 \) will be true in the next state \( S' \)
else \( \psi_2 \) will be true in the next state \( S' \)
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 $A$ of $\Pi$ we define the transition $S \Rightarrow_{DP}^{A} S'$ iff

- for all $I$ and all dynamic clauses $\phi \rightarrow (\psi_1, \psi_2)$ in $DP$, if $S, I \models \phi$ then either $\langle A, A \rangle, I \models \phi$ and $S', I \models \psi_1$ or $\langle A, A \rangle, I \models \psi_2$

- $S' \models SP$
Example

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

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

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

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

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

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

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

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

- User-role relations
- Permission-role relation
- Obligation-role relation
To have the permission of playing a role: $\text{can} \rightarrow \text{play}(s, r)$

**Example**

$\text{can} \rightarrow \text{play}(\text{Mary}, \text{Doctor}) \leftarrow$

expresses that Mary has the permission to play the role doctor
To have the permission of playing a role: \(\text{can} \rightarrow \text{play}(s, r)\)

**Example**

\(\text{can} \rightarrow \text{play}(\text{Mary}, \text{Doctor}) \leftarrow\)

expresses that Mary has the permission to play the role doctor

To be active in a role: \(\text{is} \rightarrow \text{active}(s, r)\)

**Example**

\(\text{can} \rightarrow \text{play}(\text{Mary}, \text{Doctor}) \rightarrow (\text{is} \rightarrow \text{active}(\text{Mary}, \text{Doctor}), \top)\)

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

\[ \text{Acquire}_{\text{perm}}(x, \text{Doctor}) \leftarrow \text{is} - \text{active}(x, \text{Doctor}) \]

\[ \text{Acquire}_{\text{obl}}(x, \text{Doctor}) \leftarrow \text{is} - \text{active}(x, \text{Doctor}) \]
Example

\[ \text{Acquire}_{\text{perm}}(x, \text{Doctor}) \leftarrow \text{is} - \text{active}(x, \text{Doctor}) \]

\[ \text{Acquire}_{\text{obl}}(x, \text{Doctor}) \leftarrow \text{is} - \text{active}(x, \text{Doctor}) \]

Permission and obligation acquisition

\[ P(x, \text{Doctor}, a, o) \leftarrow \text{Acquire}_{\text{perm}}(x, \text{Doctor}) \]

\[ O(x, \text{Doctor}, a, o) \leftarrow \text{Acquire}_{\text{obl}}(x, \text{Doctor}) \]
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)
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 $r_2$. 
Example

Rules

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

Rules

- is $\rightarrow$ active(Mary, cardiologist)
- doctor $\prec_{\text{perm}}$ cardiologist
- intern $\prec_{\text{perm}}$ doctor
- $\text{Acquire}_{\text{perm}}(x, r_1) \leftarrow \text{Acquire}_{\text{perm}}(x, r_2) \land (r_1 \prec_{\text{perm}} r_2)$

Consequences:

- $\text{Acquire}_{\text{perm}}(Mary, \text{cardiologist})$
- $\text{Acquire}_{\text{perm}}(Mary, \text{doctor})$
- $\text{Acquire}_{\text{perm}}(Mary, \text{intern})$
Delegation

**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 $p \rightarrow \text{delegate}$
- Forcing an entity to be delegatee of a role by action $o \rightarrow \text{delegate}$
We suppose:

\[ P(x, r, \textit{p - delegate}, y) \leftarrow \text{Acquire}_{\text{perm}}(x, r_1) \land \text{can - play}(y, r_2) \]
We suppose:

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

**Example**

\[ P(x, \text{pers - assistant}, p - \text{delegate}, y) \leftarrow \text{Acquire}_{perm}(x, \text{cardiologist}) \land \text{can - play}(y, \text{intern}) \]
We suppose:

\[ P(x, r, p \rightarrow \text{delegate}, y) \leftarrow \text{Acquire}_{\text{perm}}(x, r_1) \land \text{can} \rightarrow \text{play}(y, r_2) \]

Example

\[ P(x, \text{pers} \rightarrow \text{assistant}, p \rightarrow \text{delegate}, y) \leftarrow \text{Acquire}_{\text{perm}}(x, \text{cardiologist}) \land \text{can} \rightarrow \text{play}(y, \text{intern}) \]

- *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

$P(Mary, pers\;\text{–}\;\text{assistant}, p\;\text{–}\;\text{delegate}, John) \rightarrow (P(John, pers\;\text{–}\;\text{assistant}, d\;\text{–}\;\text{play}, John), T)$

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.
Delegating a role

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

- When Mary executes the action \( p - \text{delegate} \) John acquires the permission \( P(John, \text{pers} - \text{assistant}, d - \text{play}, John) \)
Delegating a role

When Mary executes the action $p − delegate$ John acquires the permission $P(John, pers − assistant, d − play, John)$.

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.
Mary’s personal assistant recovers from his sickness

Mary stops the delegation by choosing not to execute the action $p \rightarrow \text{delegate}$

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.
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 $John \rightarrow info.doc$ is an object in the View $file$.
- $ls \rightarrow a(John \rightarrow info.doc, file)$
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 \((x, \text{client}, \text{own}, f, \text{file}, \text{Ownership}) \iff \text{name}(x) = \text{owner}(f)\)
- *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 $perm(s, r, a, o, v, c) \leftarrow is-active(s, r) \land is-a(o, v) \land P(r, a, v, c) \land Define(s, r, a, o, v, c)$
- Dynamic rules of the form $\phi \rightarrow (C_1, C_2)$ where
  - $\phi$ is a conjunction of $perm$, or $is-active$
  - $C_1$ and $C_2$ are conjunctions of $perm$, $is-active$, $is-a$ or $\top$
Example

In static policy

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

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

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

- 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

\[
\begin{align*}
P(R_1, \text{move}, \text{step} 1, \text{context}) & \leftarrow P(R_2, \text{move}, \text{step} 2, \text{context}) \leftarrow P(R_1, \text{move}, \text{step} 1, \text{context}) \\
\text{perm}(s, R_1, \text{move}, t, \text{step} 1, \text{context}) & \rightarrow (\text{is} - \text{active}(s, R_2) \land \text{is} \rightarrow \text{a}(t, \text{step} 2), \ast) \\
\text{perm}(s, R_2, \text{move}, t, \text{step} 2, \text{context}) & \rightarrow (\text{is} - \text{active}(s, R_3) \land \text{is} \rightarrow \text{a}(t, \text{end}), \ast)
\end{align*}
\]
State transition: Modeling of a workflow

Static rules

- $P(R_1, \text{move, step1, context}) \leftarrow$
- $P(R_2, \text{move, step2, context}) \leftarrow$
  $P(R_1, \text{move, step1, context})$
- $P(R_3, \text{move, end, context}) \leftarrow$
  $P(R_2, \text{move, step2, context})$
State transition: Modeling of a workflow

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

Dynamic rules
- \( \text{perm}(s, R_1, \text{move}, t, \text{step1}, \text{context}) \rightarrow \)
  \( (\text{is} \leftarrow \text{active}(s, R_2) \land \text{is} \leftarrow a(t, \text{step2}), *) \)
- \( \text{perm}(s, R_2, \text{move}, t, \text{step2}, \text{context}) \rightarrow \)
  \( (\text{is} \leftarrow \text{active}(s, R_3) \land \text{is} \leftarrow a(t, \text{end}), *) \)
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.
Future work

- 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