A Distributed Calculus for Role-Based Access Control

Daniele Gorla

joint work with C. Braghin and V. Sassone

17th IEEE CSFW
Pacific Grove (California – USA), June, 28th, 2004
Contents

- the RBAC96 model
- a *formal framework* for concurrent systems running under a RBAC policy: an extension of the $\pi$-calculus
- a *type system* ensuring that the specified policy is respected during computations
- a *bisimulation* to reason on systems’ behaviours
- some useful applications of the theory:
  - finding the ‘**minimal**’ schema to run a given system
  - *refining a system* to be run under a given schema
  - *minimize the number of users* in a given system.
Access Control Models

“Techniques used to define or restrict the rights of individuals or application programs to obtain data from, or place data onto, a storage device”
(American National Standard, Telecom Glossary)

3 well-known models:
- Discretionary access control
- Mandatory access control
- Rôle-based access control
The Basic RBAC model
Role-Based Access Control is attracting increasing attention because:

- it reduces complexity and cost of security administration;
- permission’s management is less error-prone;
- it is flexible (rôle’s hierarchy, separation of duty, etc.);
- it is *least privilege*-oriented.
Our work

Formalize the behaviour of concurrent and distributed systems under security policies defined in a RBAC fashion.

This is similar to

- the types developed in $D\pi$ and $K\text{LAIM}$ to implement discretionary access control
- the types developed for Boxed Ambients to implement mandatory access control
The starting point: $\pi$-calculus

Concurrent processes communicating on *channels*.

\[
\text{PROCESSES: } P, Q ::= a(x).P \mid u\langle v \rangle.P \mid [u = v]P \mid (\nu a : R)P \\
\mid \text{nil} \mid P|Q \mid !P
\]
The Syntax of our Calculus

Concurrent processes communicating on channels.

\[
\text{Processes: } P, Q ::= a(x).P \mid u\langle v \rangle.P \mid [u = v]P \mid (va : R)P
\]
\[
\mid \text{nil} \mid P|Q \mid !P \mid \text{role } R.P \mid \text{yield } R.P
\]
The Syntax of our Calculus

Concurrent processes communicating on *channels*.

**Processes:**

\[ P, Q ::= a(x).P \mid u\langle v\rangle.P \mid [u = v]P \mid (v a : R)P \]
\[ \mid \text{nil} \mid P|Q \mid !P \mid \text{role } R.P \mid \text{yield } R.P \]

**User Sessions:**

\[ r\{P\}_\rho \]
The Syntax of our Calculus

Concurrent processes communicating on *channels*.

\[
\begin{align*}
\text{Processes:} & \quad P, Q ::= a(x).P \mid u\langle v \rangle .P \mid [u = v]P \mid (v_a : R)P \\
& \quad \mid \text{nil} \mid P|Q \mid !P \mid \text{role } R . P \mid \text{yield } R . P
\end{align*}
\]

\[
\begin{align*}
\text{Systems:} & \quad A, B ::= 0 \mid r\{P\}_\rho \mid A \parallel B \mid (v_{a^r} : R)A
\end{align*}
\]
The Syntax of our Calculus

Concurrent processes communicating on channels.

**Processes:** \( P, Q ::= a(x).P \mid u\langle v\rangle.P \mid [u = v]P \mid (\nu a : R)P \)
\( \mid \text{nil} \mid P\mid Q \mid !P \mid \text{role} R.P \mid \text{yield} R.P \)

**Systems:** \( A, B ::= 0 \mid r\{P\}_\rho \mid A \parallel B \mid (\nu a^r : R)A \)

Channels are allocated to users to enable a distributed implementation.
Dynamic Semantics

It is given in the form of a reduction relation

\[
\text{Communication: } \quad s\{a^r\langle n\rangle.P\}_\rho \parallel r\{a(x).Q\}_\rho'
\]
Dynamic Semantics

It is given in the form of a reduction relation

\[ s\{a^r\langle n\rangle.\, P\}_\rho \parallel r\{a(x)\cdot Q\}_\rho' \quad \leftrightarrow \quad s\{P\}_\rho \parallel r\{Q[n/x]\}_\rho' \]
Dynamic Semantics

It is given in the form of a \textit{reduction relation}.

\textbf{Communication:}
\[ s \{ a^r \langle n \rangle . P \}_\rho \parallel r \{ a(x) . Q \}_\rho' \quad \mapsto \quad s \{ P \}_\rho \parallel r \{ Q[n/x] \}_\rho' \]

\textbf{Rôle activation:}
\[ r \{ \text{role } R . P \}_\rho \]
Dynamic Semantics

It is given in the form of a *reduction relation*

**Communication:**

\[
\begin{align*}
    s\{a^r\langle n\rangle. P\}_\rho \ || \ r\{a(x). Q\}_\rho' & \longmapsto s\{P\}_\rho \ || \ r\{Q[\langle n/x\rangle]\}_\rho'
\end{align*}
\]

**Rôle activation:**

\[
\begin{align*}
    r\{\text{role } R. P\}_\rho & \longmapsto r\{P\}_\rho \cup \{R\}
\end{align*}
\]
Dynamic Semantics

It is given in the form of a *reduction relation*

**Communication:**
\[
\begin{align*}
\langle a^r \langle n \rangle . P \rangle_\rho & \parallel \langle a(x) . Q \rangle_\rho' \\ & \longmapsto \langle P \rangle_\rho \parallel \langle Q[n/x] \rangle_\rho'
\end{align*}
\]

**Rôle activation:**
\[
\begin{align*}
\langle \text{role } R . P \rangle_\rho & \longmapsto \langle P \rangle_\rho \cup \{R\}
\end{align*}
\]

**Rôle deactivation:**
\[
\begin{align*}
\langle \text{yield } R . P \rangle_\rho
\end{align*}
\]
Dynamic Semantics

It is given in the form of a *reduction relation*

**Communication:**

\[
s\{a^r\langle n\rangle.P\}_\rho \parallel r\{\alpha(x).Q\}_\rho' \quad \longrightarrow \quad s\{P\}_\rho \parallel r\{Q[n/x]\}_\rho'
\]

**Rôle activation:**

\[
r\{\text{role } R.P\}_\rho \quad \longrightarrow \quad r\{P\}_\rho \cup \{R\}
\]

**Rôle deactivation:**

\[
r\{\text{yield } R.P\}_\rho \quad \longrightarrow \quad r\{P\}_\rho \setminus \{R\}
\]
Permissions are *capabilities* that enable process actions. Thus, \( A \triangleq \{ R^\uparrow, R^?, R^! \}_{R \in \mathcal{R}} \) is the set of permissions.
Permissions are *capabilities* that enable process actions. Thus,

\[ A \triangleq \{ R^\uparrow, R^?, R^! \}_{R \in R} \] is the set of permissions.

In our framework, the **RBAC schema** is a pair of finite relations \((u \ ; \ p)\), such that

\[
u \subseteq_{\text{fin}} (\mathcal{N}_u \cup \mathcal{C}) \times \mathcal{R} \quad \quad \quad p \subseteq_{\text{fin}} \mathcal{R} \times A
\]
An Example

A banking scenario:

- two users, the client \( r \) and the bank \( s \)
- cashiers are modelled as channels \( c_1, \ldots, c_n \) of user \( s \)
- the rôles available are client and cashier.

\[
\begin{align*}
 r & \{ \text{role client.enqueue}^s \langle r \rangle . \text{enqueue}(z).z \langle \text{req}_1 \rangle . \cdots . z \langle \text{req}_k \rangle . z \langle \text{stop} \rangle . \text{yield} \text{ client} \} \rho & \parallel \\
 s & \{ (\nu \text{free})(!\text{enqueue}(x).\text{free}(y).\text{enqueue}^x \langle y \rangle \mid \Pi_{i=1}^n \text{free}^s \langle c_i^s \rangle \mid \\
 & \Pi_{i=1}^n !c_i(x).( [x = \text{withdrw}_\text{req}] \langle \text{handle withdraw request} \rangle \mid \\
 & [x = \text{dep}_\text{req}] \langle \text{handle deposit request} \rangle \mid \ldots \mid \\
 & [x = \text{stop}]\text{free}^s \langle c_i^s \rangle )) \} \rho'
\end{align*}
\]
Static Semantics - Types

The syntax of types:

\[
\begin{align*}
Types & \quad T ::= UT \mid C \\
User Types & \quad UT ::= \rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)] \\
Channel Types & \quad C ::= R(T)
\end{align*}
\]
Static Semantics - Types

The syntax of types:

\[ Types \quad T ::= \begin{array}{c} UT \mid C \\ User\ Types \quad UT ::= \rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)] \\ Channel\ Types \quad C ::= R(T) \end{array} \]

\[ \Gamma; \rho \vdash^p_r P \] states that \( P \) respects \( \Gamma \) and \( \rho \) when it is run in a session of \( r \) with rôles \( \rho \) activated.
Static Semantics - Types

- The syntax of types:

\[
\begin{align*}
Types & \quad T ::= U T \mid C \\
User Types & \quad UT ::= \rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)] \\
Channel Types & \quad C ::= R(T)
\end{align*}
\]

- \(\Gamma ; \rho \vdash^\varphi_r P\) states that \(P\) respects \(\Gamma\) and \(\varphi\) when it is run in a session of \(r\) with rôles \(\rho\) activated.

- A typing environment is a mapping from user names and variables to user types that respects the assignments in \(u\).
Static Semantics - The Type System

An example: performing input actions.

\[(T\text{-INPUT})\]
\[
\begin{array}{c}
\Gamma \vdash a : R(T) \\
R^? \in P(\rho) \\
\Gamma, x \mapsto T; \rho \vdash \rho \ P
\end{array}
\]
\[
\Gamma; \rho \vdash \rho \ a(x).P
\]
Static Semantics - The Type System

An example: performing input actions.

\[
\begin{array}{c}
\frac{
\Gamma \vdash a : R(T) \quad R^? \in \mathcal{P}(\rho) \quad \Gamma, x \mapsto T ; \rho \vdash^\rho_r P
}{\Gamma ; \rho \vdash^\rho_r a(x).P}
\end{array}
\]

Type Safety: Let \( A \) be a well-typed system for \((u; \mathcal{P})\). Then, whenever

\[
A \equiv (\nu \tilde{a}^r : \tilde{R})(A' \parallel r\{b(x).P\}_\rho),
\]

it holds that

- either \( b^r : S \in \tilde{a}^r : \tilde{R} \) and \( S^? \in \mathcal{P}(\rho) \),
- or \( b^r \not\in \tilde{a}^r \) and \( S^? \in \mathcal{P}(\rho) \), where \( \{S\} = u(b^r) \)
The Example Again

The banking scenario again:

- now each available operation is modelled as a different channel
  \( wdrw = \text{withdraw}, \ opn = \text{open account}, \ cc = \text{credit card request} \)
- the communication among different channels requires different rôles
- \( \mathcal{P} \) is such that \( \{(\text{rich}\_\text{client}, \text{cc'}), (\text{rich}, \text{rich}\_\text{client}^\uparrow)\} \subseteq \mathcal{P} \).
The Example Again

The banking scenario again:

- now each available operation is modelled as a different channel
  \( (\text{wdrw} = \text{withdraw}, \text{opn} = \text{open account}, \text{cc} = \text{credit card request}) \)
- the communication among different channels requires different rôles
- \( \mathcal{P} \) is such that \( \{(\text{rich\_client}, \text{cc}'), (\text{rich}, \text{rich\_client}^\uparrow)\} \subseteq \mathcal{P}. \)

\[ \forall r \{\text{role\_client.enqueue}^s(r) . \text{dequeue}(z) . z(\text{creditcard\_req}) . \text{cc}^s(\text{signature}) . z(\text{stop})\} \{\text{user}\} \]
The Example Again

The banking scenario again:

- now each available operation is modelled as a different channel
  \(wdrw = \text{withdraw}, \ opn = \text{open account}, \ cc = \text{credit card request}\)
- the communication among different channels requires different rôles
- \(\mathcal{P}\) is such that \{(rich\_client, cc\'), \ (\text{rich, rich_client}^\dagger)\} \subseteq \mathcal{P}.

\[
\forall r \{ \text{role client}.\ enqueue^s(r).\ dequeue(z).z(\text{creditcard req}).cc^s(\text{signature}).z(\text{stop}) \} \{\text{user}\}
\]

\[
\vdash r \{ \text{role rich_client}.\ enqueue^s(r).\ dequeue(z).z(\text{creditcard req}).cc^s(\text{signature}).z(\text{stop}) \} \{\text{rich}\}
\]
The labels of the LTS are derived from those of the $\pi$-calculus:

$$\mu ::= \tau \mid a^n \mid a^n : R \mid \overline{a}^n \mid \overline{a}^n : R$$

the LTS relates configurations, i.e. pairs $(\mu; \varphi) \triangleright A$ made up of a RBAC schema $(\mu; \varphi)$ and a system $A$.

Example:

$$(LTS-F-INPUT)$$

$$\mu(\overline{a}^n) = \{R\} \quad R^n \in \varphi(\rho) \quad n \notin \text{dom}(\mu)$$

$$(\mu; \varphi) \triangleright r\{a(x).P\}_\rho \xrightarrow{a^n:S} (\mu \cup \{n : S\}; \varphi) \triangleright r\{P[n/x]\}_\rho$$
Bisimulation Equivalence

- We can define a standard bisimulation over the LTS
- (Bisimulation) It is a binary symmetric relation $S$ between configurations such that, if $(D, E) \in S$ and $D \xrightarrow{\mu} D'$, there exists a configuration $E'$ such that $E \xrightarrow{\hat{\mu}} E'$ and $(D', E') \in S$. **Bisimilarity**, $\approx$, is the largest bisimulation.
- the bisimulation is adequate with respect to a standardly defined (typed) barbed congruence.
Some Algebraic Laws

- if an action is not enabled, then the process cannot evolve:

\[
\rho \models r \{ \alpha.P \} \approx 0 \quad \text{if } P(\rho) \text{ does not enable } \alpha
\]
Some Algebraic Laws

- if an action is not enabled, then the process cannot evolve:
  \[ r\{[\alpha.P]\}_\rho \approx 0 \quad \text{if } \mathcal{P}(\rho) \text{ does not enable } \alpha \]

- Differently from some distributed calculi, a terminated session does not affect the evolution of the system:
  \[ r\{\text{nil}\}_\rho \approx 0 \]
Some Algebraic Laws

- if an action is not enabled, then the process cannot evolve:
  \[ r\{\alpha. P\}_\rho \approx 0 \text{ if } \mathcal{P}(\rho) \text{ does not enable } \alpha \]

- Differently from some distributed calculi, a terminated session does not affect the evolution of the system:
  \[ r\{\text{nil}\}_\rho \approx 0 \]

- the user performing an output action is irrelevant; the only relevant aspect is the set of permissions activated when performing the action:
  \[ r\{b^s\langle n\rangle.\text{nil}\}_\rho \approx t\{b^s\langle n\rangle.\text{nil}\}_\rho \]
Finding the “Minimal” Schema

Goal: to look for a ‘minimal’ schema to execute a given system $A$ while maintaining its behaviour w.r.t. $(u; \mathcal{P})$

Algorithm:

- fix a metrics (number of rôles in the schema, permissions associated to each rôle, etc.)
- define the set $CONF_A = \{(u'; \mathcal{P}') \triangleright A : (u'; \mathcal{P}') \text{ is a RBAC schema}\}$ of configurations for $A$
- partition $CONF_A$ w.r.t. $\approx$ and consider the equivalence class containing $(u; \mathcal{P}) \triangleright A$
- choose the minimal schema according to the chosen metrics
**Refining Systems**

- **Goal**: to add rôle activations/deactivations within a system in such a way that the resulting system can be executed under a given schema \((U; P)\).
- We want a rôle to be active only when needed.
- The refining procedure replaces any input/output prefix \(\alpha\) occurring in session \(r\{\cdots\}\rho\) with the sequence of prefixes \(\text{role } \vec{R}.\alpha.\text{yield } \vec{R}\)
  where \(\vec{R}\) is formed by rôles assigned to \(r\), activable when having activated \(\rho\) and enabling the execution of \(\alpha\).
- The refining procedure adapts the type system.
- **Improvement**: we can give an algorithm to minimize the number of these actions added.
Goal: to transfer a process from one user to another without changing the overall system behaviour, in order to minimize the number of users in a system.

it is possible to infer axiomatically judgments of the form:

\[(u; ϕ) ⊲ r\{P\}_ρ ≃ (u; ϕ) ⊲ s\{P\}_ρ\]

This judgment says that the process \(P\) can be executed by \(r\) and \(s\) without affecting the overall system behaviour.

Thus, the session \(r\{P\}_ρ\) can be removed. If no other session of \(r\) is left in the system, then \(r\) is a useless user and is erased.
Conclusion

- We have defined a **formal framework** for reasoning about concurrent systems running under an RBAC schema;
- a number of papers deal with the specification and verification of RBAC schema;
- **Future Works:**
  - extend the framework to deal with more complex RBAC models;
  - prove that bisimilarity is complete for barbed congruence;

http://www.dsi.uniroma1.it/~gorla/publications.htm