A Distributed Calculus for Role-Based Access Control

Chiara Braghin

joint work with D. Gorla and V. Sassone

MyThS Meeting, Venice, June, 14th, 2004
Why: 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.
RBAC

**Why:** 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, similar to
- the types developed in D\(\pi\) and KLAIM to implement discretionary access control
- the types developed for Boxed Ambients to implement mandatory access control
the \textit{RBAC96} model

a \textit{formal framework} for concurrent systems running under a RBAC policy: an extension of the \(\pi\text{-calculus}\)

a \textit{type system} ensuring that the specified policy is respected during computations

a \textit{bisimulation} to reason on systems’ behaviours

some useful applications of the theory:

- finding the \textit{‘minimal’ schema} to run a given system
- \textit{refining a system} to be run under a given schema
- \textit{minimize the number of users} in a given system.
The Basic RBAC model
The starting point: \( \pi \)-calculus

Concurrent processes communicating on *channels*.

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

Concurrent processes communicating on *channels*.

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

Concurrent processes communicating on channels.

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

\text{User Sessions: } \quad r\{P\}_\rho
The Syntax of our Calculus

Concurrent processes communicating on *channels*.

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

**Systems:**  
\[ A, B ::= 0 \mid r\{P\}_\rho \mid A \parallel B \mid (\nu a^r : R)A \]
Concurrent processes communicating on *channels*.

**Processes:**

\[ P, Q ::= a(x).P \mid u\langle v\rangle.P \mid [u = v]P \mid (\nu a : R)P \]

\[ \mid \text{nil} \mid P | Q \mid !P \mid \text{role } R.P \mid \text{yield } R.P \]

**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*

Communication:

$$s\{a^r (n).P\}_\rho \parallel r\{a(x).Q\}_\rho'$$
Dynamic Semantics

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

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

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

**Communication:**

\[
\begin{align*}
&\not\upharpoonright \quad \not\upharpoonright \\
&s\{a^r\langle n\rangle.P\}_\rho \parallel r\{a(x).Q\}_\rho' \quad \rightarrow \quad s\{P\}_\rho \parallel r\{Q[n/x]\}_\rho'
\end{align*}
\]

**Rôle activation:**

\[
\begin{align*}
&\not\upharpoonright \\
r\{\text{role } R.P\}_\rho
\end{align*}
\]
Dynamic Semantics

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

**Communication:**

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

**Rôle activation:**

\[ r\{\text{role } R . P\}_\rho \quad \longmapsto \quad r\{P\}_\rho \cup \{R\} \]
Dynamic Semantics

It is given in the form of a reduction relation

**Communication:**

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

**Rôle activation:**

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

**Rôle deactivation:**

\[ r\{\text{yield } R.P\}_\rho \]
Dynamic Semantics

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

**Communication:**

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

**Rôle activation:**

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

**Rôle deactivation:**

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

Permissions are capabilities that enable process actions. Thus,
\[ \mathcal{A} \triangleq \{ R^\uparrow, R^?, R^! \}_{R \in \mathcal{R}} \] is the set of permissions.

In our framework, the **RBAC schema** is a pair of finite relations \( (\mathcal{U}; \mathcal{P}) \), such that

\[
\begin{align*}
\mathcal{U} & \subseteq_{\text{fin}} (\mathcal{N}_u \cup \mathcal{C}) \times \mathcal{R} \\
\mathcal{P} & \subseteq_{\text{fin}} \mathcal{R} \times \mathcal{A}
\end{align*}
\]
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.

$$
\begin{align*}
  & r\{ \textbf{role}\text{ client.}\text{enqueue}^s(r).\text{dequeue}(z).z⟨\text{req}_1⟩.\ldots.z⟨\text{req}_k⟩.z⟨\text{stop}⟩.\textbf{yield}\text{ client}\} \rho \quad || \\
  & s\{ (ν\text{ free})(!\text{enqueue}(x).\text{free}(y).\text{dequeue}^x(y) \mid \prod_{i=1}^n \text{free}^s⟨c_i^s⟩) \mid \\
  & \quad \prod_{i=1}^n !c_i(x).([x = \text{withdraw}_\text{req}] < \text{handle withdraw request} > \mid \\
  & \quad [x = \text{dep}_\text{req}] < \text{handle deposit request} > \mid \ldots \mid \\
  & \quad [x = \text{stop}] \text{free}^s⟨c_i^s⟩)) \} \rho' \\
\end{align*}
$$
The syntax of types:

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

The syntax of types:

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

\[\Gamma; \rho \models^\varphi_r P\] states that $P$ respects $\Gamma$ and $\varphi$ when it is run in a session of $r$ with rôles $\rho$ activated.
Static Semantics - Types

- The syntax of types:

\[ Types \quad T ::= UT \mid C \]

\[ User \ Types \quad UT ::= \rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)] \]

\[ Channel \ Types \quad C ::= R(T) \]

- \( \Gamma; \rho \vdash^P_r P \) states that \( P \) respects \( \Gamma \) and \( P \) 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 \( \mathcal{U} \)
An example: performing input actions.

\[
\begin{align*}
\text{(T-INPUT)} \\
\Gamma \vdash a : R(T) & \quad R^? \in \varphi(\rho) & \quad \Gamma, x \mapsto T; \rho \vdash_{\varphi}^r P \\
\hline
\Gamma; \rho \vdash_{\varphi}^r a(x).P
\end{align*}
\]
Static Semantics - The Type System

An example: performing input actions.

\[(T\text{-}INPUT)\]
\[
\begin{array}{c}
\Gamma \vdash a : R(T) \\
R \in \wp(\rho) \\
\Gamma, x \leftrightarrow T; \rho \vdash^x_P P
\end{array}
\]
\[
\Gamma; \rho \vdash^x_P a(x).P
\]

Type Safety: Let \( A \) be a well-typed system for \((\mathcal{U}; \wp)\). Then, whenever
\[
A \equiv (\nu \underaccent{\tilde{\ }}{a} : R)(A' \parallel r\{b(x).P\}_\rho),
\]
it holds that
\[
\begin{array}{c}
\text{either } b : S \in \underaccent{\tilde{\ }}{a} : R \text{ and } S^? \in \wp(\rho), \\
\text{or } b \notin \underaccent{\tilde{\ }}{a} \text{ and } S^? \in \wp(\rho), \text{ where } \{S\} = \mathcal{U}(b^r)
\end{array}
\]
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 \(\{(\text{rich\_client}, cc'), (\text{rich}, \text{rich\_client}^\dagger)\} \subseteq \mathcal{P}\).
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 \( \{(\text{rich}_\text{client}, cc'), (\text{rich}, \text{rich}_\text{client}^{\uparrow})\} \subseteq \mathcal{P} \).

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

The banking scenario again:

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

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

\[ \vdash r \{ \text{role rich}_\text{client}.enqueue^s(r).dequeue(z).z\langle \text{creditcard}_\text{req}\rangle.cc^s(\text{signature}).z\langle \text{stop}\rangle \} \{ \text{rich} \} \]
LTS Semantics

The labels of the LTS are derived from those of the $\pi$-calculus:

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

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

Example:

(LTS-F-INPUT)

$$u(\alpha^r) = \{R\} \quad R^? \in \varphi(\rho) \quad n \notin \text{dom}(u)$$

$$(u; \varphi) \triangleright r\{a(x).P\}_{\rho} \xrightarrow{\alpha^r n:S} (u \uplus \{n : S\}; \varphi) \triangleright r\{P[n/x]\}_{\rho}$$
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 $E \xrightarrow{\mu} E'$ and $(D', E') \in S$. Bisimilarity, $\simeq$, 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:

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

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

- 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:
  \[ r\{\alpha . P\}_\rho \approx 0 \quad \text{if } \wp(\rho) \text{ does not enable } \alpha \]

• 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. $(U; P)$

- **Algorithm:**
  - fix a *metrics* (number of rôles in the schema, permissions associated to each rôle, etc.)
  - define the set $CONF_A = \{(u'; \varphi') \triangleright A : (u'; \varphi')$ is a RBAC schema$\}$ of configurations for $A$
  - partition $CONF_A$ w.r.t. $\approx$ and consider the equivalence class containing $(u; \varphi) \triangleright A$
  - 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
Relocating Activities

- **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:

  \[(u; \varphi) \triangleright r\{[P]\}_\rho \approx (u; \varphi) \triangleright s\{[P]\}_\rho\]

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

- Thus, the session \(r\{[P]\}_\rho\) 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;
- study information flow in terms of RBAC?

http://www.dsi.unive.it/~dbraghan/publications.html