A Distributed Calculus for Role-Based Access Control

Chiara Braghin

joint work with D. Gorla and V. Sassone

MyThS Meeting, Venice, June, 14th, 2004

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.

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.

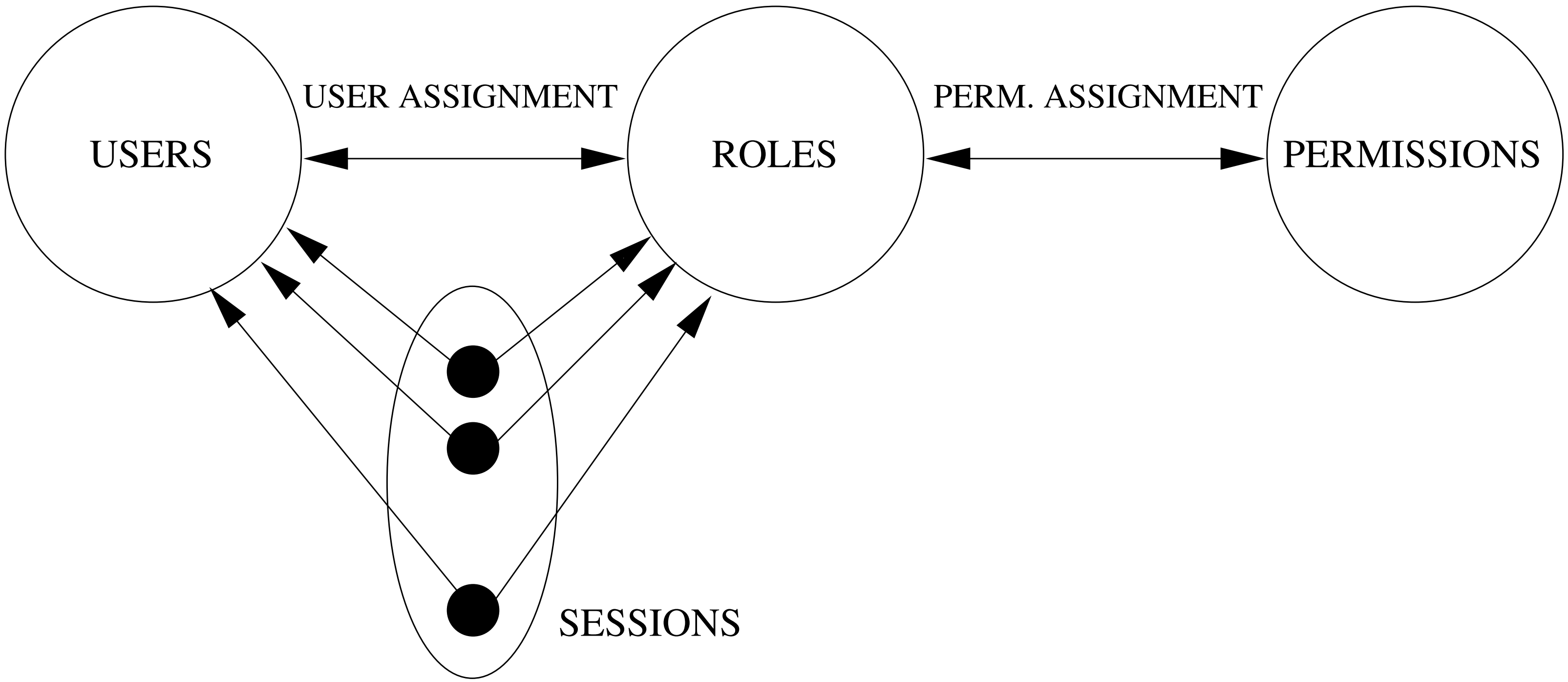
Our work: Formalize the behaviour of concurrent and distributed systems under security policies defined in a RBAC fashion, similar to

- the types developed in Dπ and KLAIM to implement discretionary access control
- the types developed for Boxed Ambients to implement mandatory access control

Contents

- the *RBAC96* model
- a *formal framework* for concurrent systems running under a RBAC policy: an extension of the π-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.

The Basic RBAC model



The starting point: π-calculus

Concurrent processes communicating on *channels*.

$$\textsc{Processes:}\quad P, Q \ ::= \ a(x).P \ \big| \ u\langle v\rangle.P \ \big| \ [u = v]P \ \big| \ (\nu a\!:\!R)P \\ \big| \ \mathbf{nil} \ \big| \ P|Q \ \big| \ !P$$

The Syntax of our Calculus

Concurrent processes communicating on *channels*.

$$\textsc{Processes:}\quad P, Q \ ::= \ a(x).P \ \big|\ u\langle v\rangle.P \ \big|\ [u = v]P \ \big|\ (\nu a\!:\!R)P \ \big|\ \mathbf{nil} \ \big|\ P|Q \ \big|\ !P \ \big|\ \mathbf{role}\, R.P \ \big|\ \mathbf{yield}\, R.P$$

The Syntax of our Calculus

Concurrent processes communicating on *channels*.

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

$\textsc{User Sessions:} \qquad r\{\!|P|\!\}_\rho$

The Syntax of our Calculus

Concurrent processes communicating on *channels*.

$$\textsc{Processes:}\quad P, Q \ ::= \ a(x).P \ \big| \ u\langle v\rangle.P \ \big| \ [u = v]P \ \big| \ (\nu a\!:\!R)P \ \big| \ \mathbf{nil} \ \big| \ P|Q \ \big| \ !P \ \big| \ \mathbf{role}\, R.P \ \big| \ \mathbf{yield}\, R.P$$

$$\textsc{Systems:}\quad A, B \ ::= \ \mathbf{0} \ \big| \ r\{\!|P|\!\}_\rho \ \big| \ A \parallel B \ \big| \ (\nu a^r\!:\!R)A$$

The Syntax of our Calculus

Concurrent processes communicating on *channels*.

$$\textsc{Processes:} \quad P, Q \;::=\; a(x).P \;\big|\; u\langle v\rangle.P \;\big|\; [u = v]P \;\big|\; (\nu a\!:\!R)P \;\big|\; \mathbf{nil} \;\big|\; P|Q \;\big|\; !P \;\big|\; \mathbf{role}\, R.P \;\big|\; \mathbf{yield}\, R.P$$

$$\textsc{Systems:} \quad A, B \;::=\; \mathbf{0} \;\big|\; r\{\!|P|\!\}_\rho \;\big|\; A \parallel B \;\big|\; (\nu a^r\!:\!R)A$$

Channels are allocated to users to enable a distibuted implementation

Dynamic Semantics

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

Communication:

$$s\{\!|a^r\langle n\rangle.P|\!\}_\rho \ \| \ r\{\!|a(x).Q|\!\}_{\rho'}$$

Dynamic Semantics

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

Communication:

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

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:

$$r\{\!|\mathbf{role}\; R.P|\!\}_\rho$$

Dynamic Semantics

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

Communication:

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

Rôle activation:

$$r\{\!|\mathbf{role}\,R.P|\!\}_\rho \;\longmapsto\; r\{\!|P|\!\}_{\rho\cup\{R\}}$$

Dynamic Semantics

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

Communication:

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

Rôle activation:

$$r\{\!|\mathbf{role}\, R.P|\!\}_\rho \;\longmapsto\; r\{\!|P|\!\}_{\rho\cup\{R\}}$$

Rôle deactivation:

$$r\{\!|\mathbf{yield}\, R.P|\!\}_\rho$$

Dynamic Semantics

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

Communication:

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

Rôle activation:

$$r\{\!|\mathbf{role}\,R.P|\!\}_\rho \longmapsto r\{\!|P|\!\}_{\rho\cup\{R\}}$$

Rôle deactivation:

$$r\{\!|\mathbf{yield}\,R.P|\!\}_\rho \longmapsto r\{\!|P|\!\}_{\rho-\{R\}}$$

RBAC schema

- Permissions are *capabilities* that enable process actions. Thus, $\mathcal{A} \stackrel{\triangle}{=} \{R^{\uparrow}, R^{?}, R^{!}\}_{R \in \mathcal{R}}$ 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

$$\mathcal{U} \subseteq_{\text{fin}} (\mathcal{N}_u \cup \mathcal{C}) \times \mathcal{R} \qquad\qquad \mathcal{P} \subseteq_{\text{fin}} \mathcal{R} \times \mathcal{A}$$

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`.

$$r\{\!|\textbf{role}\,\texttt{client}.enqueue^s\langle r\rangle.dequeue(z).z\langle req_1\rangle.\cdots.z\langle req_k\rangle.z\langle stop\rangle.\textbf{yield}\,\texttt{client}|\!\}_\rho \quad \|$$

$$s\{\!|(\nu\, free)(!enqueue(x).free(y).dequeue^x\langle y\rangle \quad | \quad \Pi_{i=1}^n free^s\langle c_i^s\rangle \quad |$$

$$\Pi_{i=1}^n\ !c_i(x).(\ [x = withdrw\_req] < \textsf{handle withdraw request} > \ |$$

$$[x = dep\_req] < \textsf{handle deposit request} > \ | \ \ldots \ |$$

$$[x = stop] free^s\langle c_i^s\rangle)\ )|\!\}_{\rho'}$$

Static Semantics - Types

- The syntax of types:

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

Static Semantics - Types

- The syntax of types:

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

- $\Gamma; \rho \vdash_r^{\mathcal{P}} P$ states that P respects Γ and $\mathcal{P}$ when it is run in a session of r with rôles ρ activated

Static Semantics - Types

- The syntax of types:

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

- $\Gamma; \rho \vdash_r^{\mathcal{P}} P$ states that P respects Γ and $\mathcal{P}$ when it is run in a session of r with rôles ρ activated

- A typing environment is a mapping from user names and variables to user types that respects the assignments in $\mathcal{U}$

Static Semantics - The Type System

An example: performing input actions.

$$\textsc{(T-Input)}\quad \frac{\Gamma \vdash a : R(T) \qquad R^{?} \in \mathcal{P}(\rho) \qquad \Gamma, x \mapsto T; \rho \vdash_r^{\mathcal{P}} P}{\Gamma; \rho \vdash_r^{\mathcal{P}} a(x).P}$$

Static Semantics - The Type System

An example: performing input actions.

(T-INPUT)

$$\frac{\Gamma \vdash a : R(T) \qquad R^{?} \in \mathcal{P}(\rho) \qquad \Gamma, x \mapsto T; \rho \vdash_{r}^{\mathcal{P}} P}{\Gamma; \rho \vdash_{r}^{\mathcal{P}} a(x).P}$$

Type Safety: Let A be a well-typed system for $(\mathcal{U}; \mathcal{P})$. Then, whenever $A \equiv (\nu\, \widetilde{a^r : R})(A' \;\|\; r\{\!|b(x).P|\!\}_{\rho})$, it holds that

- either $b^r : S \in \widetilde{a^r : R}$ and $S^{?} \in \mathcal{P}(\rho)$,
- or $b^r \notin \widetilde{a^r}$ and $S^{?} \in \mathcal{P}(\rho)$, where $\{S\} = \mathcal{U}(b^r)$

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 $\{(\texttt{rich\_client}, \texttt{cc}^!), (\texttt{rich}, \texttt{rich\_client}^\uparrow)\} \subseteq \mathcal{P}$.

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 $\{(\texttt{rich\_client}, \texttt{cc}^{!}), (\texttt{rich}, \texttt{rich\_client}^{\uparrow})\} \subseteq \mathcal{P}$.

$$\not\vdash r\{\!|\textbf{role}\,\texttt{client}.enqueue^{s}\langle r\rangle.dequeue(z).z\langle creditcard\_req\rangle.cc^{s}\langle signature\rangle.z\langle stop\rangle|\!\}_{\{\texttt{user}\}}$$

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 $\{(\mathtt{rich\_client}, \mathtt{cc}^{!}), (\mathtt{rich}, \mathtt{rich\_client}^{\uparrow})\} \subseteq \mathcal{P}$.

$$\not\vdash r\{\!|\, \mathbf{role}\, \mathtt{client}.enqueue^{s}\langle r\rangle.dequeue(z).z\langle creditcard\_req\rangle.cc^{s}\langle signature\rangle.z\langle stop\rangle \,|\!\}_{\{\mathtt{user}\}}$$

$$\vdash r\{\!|\, \mathbf{role}\, \mathtt{rich\_client}.enqueue^{s}\langle r\rangle.dequeue(z).z\langle creditcard\_req\rangle.cc^{s}\langle signature\rangle.z\langle stop\rangle \,|\!\}_{\{\mathtt{rich}\}}$$

LTS Semantics

- The labels of the LTS are derived from those of the π-calculus:

$$\mu \quad ::= \quad \tau \quad | \quad a^r n \quad | \quad a^r n : R \quad | \quad \overline{a}^r n \quad | \quad \overline{a}^r n : R$$

- the LTS relates *configurations*, i.e. pairs $(\mathcal{U}\,;\mathcal{P}) \triangleright A$ made up of a RBAC schema $(\mathcal{U}\,;\mathcal{P})$ and a system A.
- Example:

$$\frac{\text{(LTS-F-INPUT)} \qquad \mathcal{U}(a^r) = \{R\} \qquad R^? \in \mathcal{P}(\rho) \qquad n \notin dom(\mathcal{U})}{(\mathcal{U}\,;\mathcal{P}) \triangleright r\{\!|a(x).P|\!\}_\rho \xrightarrow{a^r n:S} (\mathcal{U} \uplus \{n : S\}; \mathcal{P}) \triangleright r\{\!|\, P[{}^{n}\!/\!{}_{x}]\,|\!\}_\rho}$$

Bisimulation Equivalence

- We can define a standard bisimulation over the LTS
- (Bisimulation) It is a binary symmetric relation $\mathcal{S}$ between configurations such that, if $(D, E) \in \mathcal{S}$ and $D \xrightarrow{\mu} D'$, there exists a configuration E' such that $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ and $(D', E') \in \mathcal{S}$. *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:

$$r\{\!|\alpha.P|\!\}_\rho \approx \mathbf{0} \qquad \text{if } \mathcal{P}(\rho) \text{ does not enable } \alpha$$

Some Algebraic Laws

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

$$r\{\!|\alpha.P|\!\}_\rho \approx \mathbf{0} \qquad \text{if } \mathcal{P}(\rho) \text{ does not enable } \alpha$$

- Differently from some distributed calculi, a terminated session does not affect the evolution of the system:

$$r\{\!|\mathbf{nil}|\!\}_\rho \approx \mathbf{0}$$

Some Algebraic Laws

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

$$r\{\!|\alpha.P|\!\}_\rho \approx \mathbf{0} \qquad \text{if } \mathcal{P}(\rho) \text{ does not enable } \alpha$$

- Differently from some distributed calculi, a terminated session does not affect the evolution of the system:

$$r\{\!|\mathbf{nil}|\!\}_\rho \approx \mathbf{0}$$

- the user performing an output action is irrelevant; the only relevant aspect is the set of permissions activated when performing the action:

$$r\{\!|b^s\langle n\rangle.\mathbf{nil}|\!\}_\rho \approx t\{\!|b^s\langle n\rangle.\mathbf{nil}|\!\}_\rho$$

Finding the “Minimal” Schema

- Goal: to look for a ‘minimal’ schema to execute a given system A while mantaining its behaviour w.r.t. $(\mathcal{U}\,;\mathcal{P})$
- Algorithm:
  - fix a *metrics* (number of rôles in the schema, permissions associated to each rôle, etc.)
  - define the set $CONF_A = \{(\mathcal{U}';\mathcal{P}') \triangleright A : (\mathcal{U}';\mathcal{P}')$ is a RBAC schema$\}$ of configurations for A
  - partition $CONF_A$ w.r.t. $\approx$ and consider the equivalence class containing $(\mathcal{U}\,;\mathcal{P}) \triangleright A$
  - 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 $(\mathcal{U}\,;\mathcal{P})$
- we want a rôle to be active only when needed
- the refining procedure replaces any input/output prefix α occurring in session $r\{\!|\cdots|\!\}_\rho$ with the sequence of prefixes $\mathbf{role}\ \vec{R}.\alpha.\mathbf{yield}\ \vec{R}$ where $\vec{R}$ is formed by rôles assigned to r, activable when having activated ρ and enabling the execution of α
- 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:

$$(\mathcal{U}\,;\mathcal{P}) \triangleright r\{\!|P|\!\}_\rho \approx (\mathcal{U}\,;\mathcal{P}) \triangleright s\{\!|P|\!\}_\rho$$

  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/~dbraghin/publications.html`