Security Policies as Membranes in Systems for Global Computing

Vladimiro Sassone

University of Sussex, UK

GC 2004: MyThS/MIKADO/DART Meeting
Venice 15.06.04

with D. Gorla, M. Hennessy
1. Why

2. What

3. How
   - Barring actions
   - Counting actions
   - Sequencing actions
   - Controlling coalitions

4. Conclusion
Why

- Most calculi/languages for GC rely on *code mobility* to model interprocesses interactions;

- This leads to *security concerns* (malicious agents can compromise ‘good’ sites through viruses, spammings, denial-of-service attacks, ...);
Most calculi/languages for GC rely on code mobility to model interprocesses interactions;

This leads to security concerns (malicious agents can compromise ‘good’ sites through viruses, spammings, denial-of-service attacks, ...);

Thus, code mobility usually equipped with security checks:

1. static checks: make the run-time as efficient as possible, but it may be not adequate in practice;

2. dynamic checks: make the runtime heavier, execution slower, but are flexible.
**Simple**

- **Systems** are (plain) collections of sites;

- **Sites** are places for computations, divided in at least two layers:
  - a computing body
  - a *membrane*, to carry on security related issues

- **membranes** regulate the interactions between the computing body and the environment around the site

- differently from Boudol’s and Stefani’s: our membranes are *not* fully-fledged computing entities. They only implement higher-level (type related) verification on incoming agents.
Run an initial investigation into what kind of security policies can be implemented through membranes, and how.

This is related to, and aims at generalizing for the specific application:

- the security types developed for $D_\pi$ and KLAIM;
- the session types by Honda et al;
- the generic types by Igarashi, Kobayashi.
What

1. a **formal framework** to formalize processes running in a GC system, whose activities are *local computations* and *migrations*;

2. **membranes** to implement advanced checks on incoming agents (including notions of *trust* and *proof-carrying code*);

3. **tools** to enforce different kind of policies.
A Calculus for Migrations

A minimal calculus (Turing not an issue here)

**BasicActions**  \( a, b, c, \ldots \in \text{Act} \)

**Localities**  \( l, h, k, \ldots \in \text{Loc} \)

**Agents**  \( P, Q, R ::= \text{nil} \mid a.P \mid \text{go}_l.P \mid P \mid Q \mid !P \)

**Systems**  \( N ::= 0 \mid l[[ M ] P ] \mid N_1 \parallel N_2 \)

where
A minimal calculus (Turing not an issue here)

**BasicActions** \( a, b, c, ... \in \text{Act} \)

**Localities** \( l, h, k, ... \in \text{Loc} \)

**Agents** \( P, Q, R \ ::= \text{nil} | a.P | \text{go}_l.P | P | Q | !P \)

**Systems** \( N ::= 0 | [l[M]P] | N_1 || N_2 \)

where
- \([l[M]P]\) is a site with address \(l\), membrane \(M\) and hosting process \(P\);
- \(\text{go}_l.P\) is an agent willing to migrate on \(l\), whose body is \(P\) and exhibiting as PCC the policy \(T\).
Local behaviours:

\[
I[ M \triangleright a.P | Q ] \rightarrow I[ M \triangleright P | Q ]
\]

Remark: we are not really interested in the local computations.
Dynamic Semantics – migration

Migration:

\[ k[[ M \parallel Q ] \parallel l[[ M' \parallel R ]] \rightarrow k[[ M \parallel Q ] \parallel l[[ M' \parallel P \parallel R ]] \]

This reduction may happen only if \( P \) complies with \( M' \).
Dynamic Semantics – migration

Migration:

\[
\begin{align*}
  k[M \triangleright go_t l.P | Q] & \parallel l[M' \triangleright R] & \rightarrow & & k[M \triangleright Q] & \parallel l[M' \triangleright P | R]
\end{align*}
\]

This reduction may happen only if \( P \) complies with \( M' \).

But checking whole processes at migration can be very expensive!

Solution: PCCs. A source-generated and certified ‘process outline’ accepted as such at destination.
The matter with certification

When can we consider PCCs?

- They are easy to verify (they are usually very small, if compared to the process they refer to), but

- they can be dangerous (if they don’t certify properly the process behaviour)
When can we consider PCCs?

- They are easy to verify (they are usually very small, if compared to the process they refer to), but

- they can be dangerous (if they don’t certify properly the process behaviour)

A compromise:

`we can safely consider PCCs of agents coming from trusted sites, i.e. sites that calculate the PCC attached to a migrating agent “properly.”`
Each site store the trust it has on other sites, as part of its membrane.

Thus, a membrane is a couple \((M_t, M_p)\), where

- \(M_t : \text{Loc} \rightarrow \{\text{good, bad, unknown}\};\)

- \(M_p\) is an upper bound to the local actions of incoming agents.
The Migration Rule – revised

\[
\begin{align*}
  k\left[ M \parallel \text{go}_{\syn{T}}.P\mid Q \right] & \rightarrow \quad l\left[ M' \parallel R \right] \\
  & \rightarrow \quad k\left[ M \parallel Q \right] \parallel l\left[ M' \parallel P \mid R \right] & \text{if } M' \vdash^k P
\end{align*}
\]

where \( M' \vdash^k P \) is

\[
\text{if } M'_t(k) = \text{good} \text { then } (T \text{ enforces } M'_p) \text{ else } \vdash P : M'_p
\]

and

- predicate \( \text{enforces} \) is a partial order on policies;
- \( \vdash \) is a compliance check of a process against a policy.
Policies as Constraints on Legal Actions

- A site only provides some methods (i.e. only some actions can be executed while running in it)

- A policy $T$ is a subset of $\text{Act} \cup \text{Loc}$ where
  - A process can only execute locally actions in $T$
  - A process can only migrate on sites in $T$
Policies as Constraints on Legal Actions

- a site only provides some methods (i.e. only some actions can be executed while running in it)

- a policy $T$ is a subset of $\text{Act} \cup \text{Loc}$ where
  
  - a process can only execute locally actions in $T$
  
  - a process can only migrate on sites in $T$

- $T$ enforces $T'$ is simply defined as $T \subseteq T'$;
Policies as Constraints on Legal Actions

- a site only provides some methods (i.e. only some actions can be executed while running in it)

- a policy $T$ is a subset of $\text{Act} \cup \text{Loc}$ where
  - a process can only execute locally actions in $T$
  - a process can only migrate on sites in $T$

- $T$ enforces $T'$ is simply defined as $T \subseteq T'$;

- judgment $\vdash$ is simple. The key rules are

\[
\begin{align*}
\vdash P : T \quad & \quad \vdash P : T' \\
\vdash a.P : T \quad & \quad \vdash \text{go}_{T'} l.P : T
\end{align*}
\]
a system $N$ is **well-formed**, written $\vdash N : \text{ok}$, if “good” nodes only hosts “good” agents. Formally:

\[
\begin{align*}
\vdash P : M_p \\
\vdash \llbracket M | P \rrbracket : \text{ok} \quad &\text{I good} \\
\vdash \llbracket M | P \rrbracket : \text{ok} \quad &\text{I not good}
\end{align*}
\]
a system $N$ is well-formed, written $\vdash N : \text{ok}$, if “good” nodes only host “good” agents. Formally:

$$
\vdash P : M_p \\
\vdash l[M \triangleright P] : \text{ok} \\
\vdash l[M \triangleright P] : \text{ok}
$$

Subject Reduction: If $\vdash N : \text{ok}$ and $N \rightarrow N'$, then $\vdash N' : \text{ok}$. 
sometimes, legal actions can be performed only a certain number of times. E.g.:

- a fair mail server allows its clients to send mails, but:
- it should block spamming activities of malicious clients; thus:
- it could allow sending at most $K$ mails for each login of each client.
sometimes, legal actions can be performed only a certain number of times. E.g.:

- a fair mail server allows its clients to send mails, but:
- it should block spamming activities of malicious clients; thus:
- it could allow sending at most $K$ mails for each login of each client.

Policies are \textit{multisets} containing elements from \textit{Act} $\cup \textit{Loc}$;

\textit{T enforces} \textit{T'} is multisets inclusion;

$\vdash$ adapts straightforwardly from the case of sets:

\[
\begin{align*}
\vdash P : T \\
\vdash \alpha.P : T \cup \{\alpha\} \\
\vdash P : T' \\
\vdash \text{go}_{T}, l.P : T \cup \{l\} \\
\vdash P : T_1 \\
\vdash Q : T_2 \\
\vdash P | Q : T_1 \cup T_2
\end{align*}
\]
This setting enforces a *thread-wise* property. Indeed,
- if two different agents $P$ and $Q$ individually send at most $K$ mails,
- when they both run in the mail server, the agent $P | Q$ can send more than $K$ mails (actually, it can send $2K$ mails)

Thus, the well-formedness predicate for good sites is changed as

$$
\forall i. (P_i \text{ a thread and } \vdash P_i : M_p) ~ \vdash l[[ M \parallel P_1 | \ldots | P_n ]] : \text{ok} ~ \text{\_ good}
$$

Subject reduction holds for this modified judgment
Sequencing Legal Actions

- sometimes, legal actions can be performed only in a certain order. E.g.
  - before exploiting the functionalities of a mail server, you must have logged in, and
  - before logging out, you must have saved the status of the transaction.

This can be easily formalized by *(deterministic) finite automata*

\[ \text{usr.pwd.}(\text{list + send + retr + del + reset}^*) . \text{quit} \]
Sequencing Legal Actions

- sometimes, legal actions can be performed only in a certain order. E.g.
  - before exploiting the functionalities of a mail server, you must have logged in, and
  - before logging out, you must have saved the status of the transaction.

  This can be easily formalized by (deterministic) finite automata

  \[ \text{usr.pwd.}(\text{list + send + retr + del + reset})^*.\text{quit} \]

- Policies are DFAs;
- \( T \) enforces \( T' \) is inclusion of DFAs’s languages;
- \( \vdash P : T \) holds if the language of \( P \) is accepted by \( T \).
As well-known, inclusion of regular languages can be calculated easily, once given the associated DFAs.

What about predicate $\vdash P : T$?

- we expect that calculating it is harder than verifying PCCs (i.e., verifying predicate enforces)
- But, how harder? Is it decidable?
- what is the language associated to an agent?
an agent can be easily associated to a *concurrent regular expression*: regular exprs with *shuffle* $\otimes$ and *shuffle closure* $\circ$.

e.g., agent $!(a.b \mid c.go_I.P)$ can be represented as

$$(((a \cdot b) \otimes (c \cdot I))\circ$$

we are only interested in the *local behaviour* of the agent.
Sequencing Legal Actions (ctd2)

- an agent can be easily associated to a *concurrent regular expression*: regular exprs with *shuffle* $\otimes$ and *shuffle closure* $\circ$.

- e.g., agent $!(a.b \mid c.go \cdot l.P)$ can be represented as $((a \cdot b) \otimes (c \cdot l))^{\circ}$

we are only interested in the *local behaviour* of the agent.

- we can derive the language associated to this CRE and check whether it is contained in the language accepted by the policy;

- CREs can be represented as Petri nets. Inclusion of a Petri net in a DFA is *decidable*, even if *super-exponential*;

- This is done by *static analysis algorithm*, not by a type system!
policies as multisets and as DFAs can only express thread-oriented properties;

Dealing with the overall behaviour of a site; Two options: When agent $P$ want to migrate on $I$, containing agent $R$
Controlling Coalitions at a Site

- policies as multisets and as DFAs can only express thread-oriented properties;

- Dealing with the overall behaviour of a site; Two options: When agent $P$ want to migrate on $I$, containing agent $R$

  1. freeze and retrieve the current content of the site, viz. $R$; check whether $P \mid R$ respects the policy of the site; reactivate $R$ and, according to the result of the checking phase, activate $P$. 

V. Sassone
Security Policies as Membranes
Controlling Coalitions at a Site

- policies as multisets and as DFAs can only express thread-oriented properties;

- Dealing with the overall behaviour of a site; Two options: When agent $P$ want to migrate on $I$, containing agent $R$

  1. freeze and retrieve the current content of the site, viz. $R$; check whether $P | R$ respects the policy of the site; reactivate $R$ and, according to the result of the checking phase, activate $P$.

  2. let membranes evolving at run-time: they are decreased with the privileges granted to $P$. 

V. Sassone
Security Polices as Membranes
Controlling Coalitions at a Site

- policies as multisets and as DFAs can only express thread-oriented properties;

- Dealing with the overall behaviour of a site; Two options: When agent $P$ want to migrate on $I$, containing agent $R$
  1. freeze and retrieve the current content of the site, viz. $R$; check whether $P | R$ respects the policy of the site; reactivate $R$ and, according to the result of the checking phase, activate $P$.
  2. let membranes evolving at run-time: they are decreased with the privileges granted to $P$.

- I’m sure you see that the first option is just crazy...
A new migration rule:

\[
k[ M \parallel go_1 . P \parallel Q ] \parallel I[ M' \parallel R ] \rightarrow k[ M \parallel Q ] \parallel I[ M'' \parallel P \parallel R ] \text{ if } M' \vdash^k P \succ M''
\]

where \( M' \vdash^k P \succ M'' \):

- verifies whether \( P \) respects \( M'_p \) (by examining its PCC \( T \) or its code, according to the trust level in its origin, \( k \));
- if \( P \) respects \( M'_p \), it decrease \( M'_p \) with the privileges granted to \( P \). This returns \( M''_p \).
Controlling Coalitions at a Site (ctd2)

Well-formed systems are now defined w.r.t. a function \(\Theta\) associating each good site to a initial policy.

\[
\Theta \vdash l \llbracket M \triangleright P \rrbracket : \text{ok}
\]

\(l\) good

\((\text{pol}(P) \triangleright M_P)\) enforces \(\Theta(l)\)

where

- \(\text{pol}(P)\) returns the minimal policy satisfied by \(P\);
- \(\triangleright\) merges together two policies.

**Subject Reduction:** If \(\Theta \vdash N : \text{ok}\) and \(N \rightarrow N'\), then \(\Theta \vdash N' : \text{ok}\).
Conclusions

- a formal framework to reason on the role of membranes as security policies

- several variations expressing finer and finer policies

- to be done:
  - a richer calculus (including communications, restrictions, ...)
  - more complex policies (not expressible with DFAs)
  - ...

- the paper is available at
  www.dsi.uniroma1.it/~gorla/papers/GHS-membranes.ps