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, ...);

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, ...);
- 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.

The Objectives

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 Dπ 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	$a, b, c, \ldots \in \textsc{Act}$		
Localities	$l, h, k, \ldots \in \textsc{Loc}$		
Agents	P, Q, R	$::=$	$\mathbf{nil} \mid a.P \mid \mathbf{go}_T l.P \mid P \mid Q \mid !P$
Systems	N	$::=$	$\mathbf{0} \mid l[\![M \rangle P]\!] \mid N_1 \parallel N_2$

where

A Calculus for Migrations

A minimal calculus (Turing not an issue here)

$$\begin{array}{rrcl} \textit{BasicActions} & a, b, c, \ldots \in \textsc{Act} \\ \textit{Localities} & l, h, k, \ldots \in \textsc{Loc} \\ \textit{Agents} & P, Q, R & ::= & \mathbf{nil} \mid a.P \mid \mathbf{go}_T l.P \mid P \mid Q \mid\ !P \\ \textit{Systems} & N & ::= & \mathbf{0} \mid l[\![M \,\rangle\, P]\!] \mid N_1 \parallel N_2 \end{array}$$

where

- $l[\![M \,\rangle\, P]\!]$ is a site with address l, membrane M and hosting process P;
- $\mathbf{go}_T l.P$ is an agent willing to migrate on l, whose body is P and exhibiting as PCC the policy T.

Dynamic Semantics – local

Local behaviours:

$$I[\![\, M \,\rangle\!\rangle\, a.P|Q \,]\!] \rightarrow I[\![\, M \,\rangle\!\rangle\, P|Q \,]\!]$$

Remark: we are not really interested in the local computations.

Dynamic Semantics – migration

Migration:

$$k[\![\, M \,\rangle\!\rangle\; \mathbf{go}_T\, l.P|Q \,]\!] \;\; \| \;\; l[\![\, M' \,\rangle\!\rangle\; R \,]\!] \quad \longrightarrow \quad k[\![\, M \,\rangle\!\rangle\; Q \,]\!] \;\; \| \;\; l[\![\, M' \,\rangle\!\rangle\; P|R \,]\!]$$

This reduction may happen only if *P complies with M'*.

Dynamic Semantics – migration

Migration:

$$k[\![\, M \,\rangle\!\rangle\; \mathbf{go}_T l.P|Q \,]\!] \;\|\; l[\![\, M' \,\rangle\!\rangle\; R \,]\!] \;\rightarrow\; k[\![\, M \,\rangle\!\rangle\; Q \,]\!] \;\|\; l[\![\, M' \,\rangle\!\rangle\; P|R \,]\!]$$

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)

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)

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."

Trust

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

- $M_t : \textsc{Loc} \rightarrow \{\mathtt{good}, \mathtt{bad}, \mathtt{unknown}\}$;
- M_p is an upper bound to the local actions of incoming agents.

The Migration Rule – revised

$$k[\![\, M \,\rangle\!\rangle\, \mathbf{go}_T\, l.P|Q \,]\!] \quad \| \quad l[\![\, M' \,\rangle\!\rangle\, R \,]\!]$$
$$\longrightarrow \quad k[\![\, M \,\rangle\!\rangle\, Q \,]\!] \quad \| \quad l[\![\, M' \,\rangle\!\rangle\, P|R \,]\!] \qquad \text{if } M' \vdash^k_T P$$

where $M' \vdash^k_T P$ is

$$\textbf{if } M'_t(k) = \texttt{good} \textbf{ then } (T \ \texttt{enforces} \ M'_p\,) \textbf{ else } \vdash P : M'_p$$

and

- predicate `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 $\textsc{Act} \cup \textsc{Loc}$ 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 $\textsc{Act} \cup \textsc{Loc}$ 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 $\textsc{Act} \cup \textsc{Loc}$ 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

$$\frac{\vdash P : T}{\vdash a.P : T}\ a \in T \qquad \frac{\vdash P : T'}{\vdash \mathbf{go}_{T'}\, l.P : T}\ l \in T$$

Policies as Constraints on Legal Actions (ctd)

- a system N is *well-formed*, written $\vdash N : \mathbf{ok}$, if "good" nodes only hosts "good" agents. Formally:

$$\frac{\vdash P : M_p}{\vdash l[\![M \,\rangle\, P]\!] : \mathbf{ok}} \; l \text{ good} \qquad\qquad \frac{}{\vdash l[\![M \,\rangle\, P]\!] : \mathbf{ok}} \; l \text{ not good}$$

Policies as Constraints on Legal Actions (ctd)

- a system N is *well-formed*, written $\vdash N : \mathbf{ok}$, if "good" nodes only hosts "good" agents. Formally:

$$\frac{\vdash P : M_p}{\vdash l[\![M \,\rangle\, P]\!] : \mathbf{ok}} \; l \text{ good} \qquad \frac{}{\vdash l[\![M \,\rangle\, P]\!] : \mathbf{ok}} \; l \text{ not good}$$

- **Subject Reduction:** If $\vdash N : \mathbf{ok}$ and $N \longrightarrow N'$, then $\vdash N' : \mathbf{ok}$.

Counting Legal Actions

- 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.

Counting Legal Actions

- 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 *multisets* containing elements from $\textsc{Act} \cup \textsc{Loc}$;
- T `enforces` T' is multisets inclusion;
- $\vdash$ adapts straightforwardly from the case of sets:

$$\frac{\vdash P : T}{\vdash a.P : T \cup \{a\}} \qquad \frac{\vdash P : T'}{\vdash \mathbf{go}_{T'} l.P : T \cup \{l\}} \qquad \frac{\vdash P : T_1 \quad \vdash Q : T_2}{\vdash P \mid Q : T_1 \cup T_2}$$

Counting Legal Actions (ctd)

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 $P \mid Q$ can send *more than* K mails (actually, it can send $2K$ mails)

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

$$\frac{\forall i\,.\,(P_i \text{ a thread and } \vdash P_i : M_p)}{\vdash l[\![\, M \,\|\, P_1 | \ldots | P_n \,]\!] : \mathbf{ok}} \quad l \text{ good}$$

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 loggin out, you must have saved the status of the transaction.

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

$$\mathtt{usr.pwd.(list + send + retr + del + reset)^*.quit}$$

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 loggin out, you must have saved the status of the transaction.

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

$$\texttt{usr.pwd.}(\texttt{list} + \texttt{send} + \texttt{retr} + \texttt{del} + \texttt{reset})^{*}\texttt{.quit}$$

- Policies are DFAs;
- T $\texttt{enforces}$ T' is inclusion of DFAs's languages;
- $\vdash P : T$ holds if the language of P is accepted by T.

Sequencing Legal Actions (ctd)

- 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?

Sequencing Legal Actions (ctd2)

- an agent can be easily associated to a *concurrent regular expression*: regular exprs with *shuffle* $\otimes$ and *shuffle closure* $^{\odot}$.
- e.g., agent $!(a.b \mid c.\mathbf{go}_T l.P)$ can be represented as

$$((a \cdot b) \otimes (c \cdot l))^{\odot}$$

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* $^{\odot}$.
- e.g., agent $!(a.b \mid c.\mathbf{go}_{l}\, l.P)$ can be represented as

$$((a \cdot b) \otimes (c \cdot l))^{\odot}$$

  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!

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 l, 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 l, 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.

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

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 l, 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.
  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. . .

Controlling Coalitions at a Site (ctd)

A new migration rule:

$$k[\![\, M \,\rangle\, \mathbf{go}_T l.P|Q \,]\!] \quad \| \quad l[\![\, M' \,\rangle\, R \,]\!]$$
$$\rightarrow \quad k[\![\, M \,\rangle\, Q \,]\!] \quad \| \quad l[\![\, M'' \,\rangle\, P|R \,]\!] \qquad \text{if } M' \vdash^k_T P \succ M''$$

where $M' \vdash^k_T P \succ M''$:

- 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 Θ associating each good site to a initial policy.

$$\frac{}{\Theta \vdash l[\![\, M \,\rangle\, P \,]\!] : \mathbf{ok}} \quad \begin{array}{l} l \text{ good} \\ (pol(P) \sqcup M_p) \ \mathtt{enforces}\ \Theta(l) \end{array}$$

where

- $pol(P)$ returns the minimal policy satisfied by P;
- $\sqcup$ merges together two policies.

Subject Reduction: If $\Theta \vdash N : \mathbf{ok}$ and $N \rightarrow N'$, then $\Theta \vdash N' : \mathbf{ok}$.

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`