Type-Based Discretionary Access Control

Silvia Crafa

joint work with

M. Bugliesi and D. Colazzo

MYTHS

Dipartimento di Informatica

Università di Venezia, Ca' Foscari

Access Control in the pi calculus

Printing jobs via a spooler:

$$\text{Print Spooler} \quad S \quad \triangleq \quad !\,\mathit{spool}\,(x).\,\overline{\mathit{print}}\langle x\rangle$$

$$\text{Print Client} \quad C \quad \triangleq \quad \overline{\mathit{spool}}\langle\, j_1\,\rangle.\,\overline{\mathit{spool}}\langle\, j_2\,\rangle.\ldots$$

- Spooling channel *spool* publicly known
- **Can we guarantee that client jobs are printed?**
- **No!** ... clients may steel jobs: $S \mid C \mid \;!\,\mathit{spool}(x).\mathbf{0}$

Types for Access Control

Associate names with capabilities

- deliver spooling channel with read-only capabilities

$$S \triangleq (\boldsymbol{\nu} spool) \;\; !(\, \overline{p}\langle spool\rangle \;\mid\; spool(y).\overline{print}\langle y\rangle \,)$$

$$C \triangleq p(\, x : T^{\mathsf{w}} \,).\, \overline{x}\langle\, j_1 \,\rangle.\, \overline{x}\langle\, j_2 \,\rangle.\ldots$$

- p is the connecting port, publicly known;
  spool is the spooling channel, now private
- **Can we guarantee that client jobs are printed?**
- **Yes:** $S \mid C \mid\; p(x).!\,x(y).\mathbf{0}$ is not type correct
  ... in all contexts to which p is known as $p : ((T)^{\mathsf{w}})^{\mathsf{rw}}$

Stronger guarantees may be desirable

- **Client jobs should not be logged or leaked**
  $\Longrightarrow$ disallow leaking spoolers like

$$!spool(x).\overline{log}\langle x\rangle.\overline{print}\langle x\rangle \quad | \quad log(y).SPY$$

Stronger guarantees may be desirable

- **Client jobs should not be logged or leaked**

  $\Longrightarrow$ must disallow leaking spoolers like

$$!spool(x).\,\overline{log}\langle x\rangle.\,\overline{print}\langle x\rangle \quad | \quad log(y).SPY$$

- **Clients want to receive reliable ackowledgements as in**

$$\underbrace{!spool(x).\,\overline{print}\langle x\rangle}_{spooler} \;\; | \;\; \underbrace{!print(x).(\mathsf{P} \mid \overline{ack}\langle x\rangle)}_{printer}$$

  $\Longrightarrow$ must disallow cheating spoolers like

$$!spool(x).\,\overline{ack}\langle x\rangle$$

Need more informative types

- Control the flow of names among system components

- One needs the ability to express/enforce **discretionary policies of access control** governing
  - which authorities may legally receive values of a given type
  - what (type) capabilities should be passed along with the values

- Capability types, à la Pierce-Sangiorgi, do not help provide the intended guarantees

Controlling delivery of names

Associate names with delivery policies

- Capability-based control system $+$ new information to describe/prescribe the ways that values may be exchanged among system components.

- the new types generalize **Group Types** [CGG00]

$$\mathsf{G}[\ T \parallel \Delta\ ]$$

  - G : the authority in control of the values of the type
  - T : structural information about values
  - Δ : delivery policy, to control how values are passed around (to which authorities, with which capabilities)

Type based control of the spooler

$$j : \mathsf{Job}[\ \mathsf{fd}\ \|\ \mathsf{Spool} \rightarrow \mathsf{Print} \rightarrow \mathsf{Client}\ \ ]$$

j is a file descriptor

to be *first* delivered to the spooler,
then passed on to the printer, and
only then sent back to clients for notification.

Type based control of the spooler

$$j : \mathsf{Job}[\ \mathsf{fd}\ \|\ \mathsf{Spool} \rightarrow \mathsf{Print} \rightarrow \mathsf{Client}\ ]$$

j is a file descriptor

to be *first* delivered to the spooler,
then passed on to the printer, and
only then sent back to clients for notification.

$$spool : \mathsf{Spool}[\ (\mathsf{Job}[\mathsf{fd}\ \|\ \mathsf{Print} \rightarrow \mathsf{Client}])^{\mathsf{rw}}\ \|\ \Delta\ ]$$

spool is a (r/w) channel,
controlled by the spooler

spool itself should be delivered
as dictated by Δ

it carries file desc. which may be passed on to a client
only after having been transmitted to the printer

Type based control of the spooler

$$\mathsf{J} \quad = \quad \mathsf{Job}[\ \mathsf{fd}\ \|\ \mathsf{Spool} \longrightarrow \mathsf{Print} \longrightarrow \mathsf{Client}\ ]$$

$$\mathsf{S} \quad = \quad \mathsf{Spool}[\ \underbrace{(\mathsf{Job}[\mathsf{fd}\ \|\ \mathsf{Print} \longrightarrow \mathsf{Client}])^{\mathsf{rw}}}_{\mathsf{SJ}}\ \ \|\ \ \Delta\ ]$$

$$j : \mathsf{J}\ ,\ \ spool : \mathsf{S} \vdash\ \underbrace{spool(x : \mathsf{SJ}).\dots}_{spooler} \mid \underbrace{\overline{spool}\langle\ j\ \rangle.\dots}_{client}$$

- x (hence j) may only be delivered as prescribed by SJ
- there is no possibility of logging/cheating:
  $!spool(x{:}\mathsf{SJ}).\,\overline{ack}\langle x\rangle$ and $!spool(x{:}\mathsf{SJ}).\,\overline{log}\langle x\rangle.\,\overline{print}\langle x\rangle$

Note: j must be given different types as it is delivered:

$\mathsf{Job}[\mathsf{fd}\ \|\ \mathsf{Spool} \rightarrow \mathsf{Print} \rightarrow \mathsf{Client}]$ $\quad$ $\mathsf{Job}[\mathsf{fd}\ \|\ \mathsf{Print} \rightarrow \mathsf{Client}]$ $\quad$ $\mathsf{Job}[\mathsf{fd}\ \|\ \mathsf{Client}]$

Type Based DAC Policies

Our types support powerful policies

- delivery chains of bounded/unbounded depth;
- multiple (branching) chains along alternative paths

$$\mathsf{G}[\ T\ \ \|\ \mathsf{G}_1 \rightarrow \mathsf{G}_2 \rightarrow \mathsf{G}_3\ \ ;\ \ \mathsf{G}'_1 \rightarrow (\mathsf{G}'_2\,;\,\mathsf{G}'_3 \rightarrow \mathsf{G}'_4)\ ]$$

- delivery at different (super) types depending on recipients

$$n : \mathsf{G}[\ (\mathsf{int})^{\mathsf{rw}}\ \|\ \mathsf{G}_1@(\mathsf{int})^{\mathsf{w}} \rightarrow \mathsf{G}_2@(\mathsf{int})^{\mathsf{w}};\ \ \mathsf{G}_3@(\mathsf{int})^{\mathsf{r}}\ ]$$

Main Result (Safety)

In well-typed processes all names flow according to the delivery policies specified by their types, and are received at the intended sites with the intended capabilities.

A typed pi calculus with groups

Syntax as in [CGG00]

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid a(x_1 : \tau_1, \ldots, x_n : \tau_n).P \mid \overline{a}\langle b_1, \ldots, b_n\rangle.P \\ & \mid & (\boldsymbol{\nu} n : \tau)P \mid (\boldsymbol{\nu} G)P \mid P|P \mid !P \end{array}$$

Types generalize those in [CGG00]

$$\begin{array}{rrcll} \textit{Structural Types} & T & ::= & \mathsf{B} \mid (\tau_1, \ldots, \tau_n)^{\nu} & (\tau_i \ \textit{closed}) \\ \textit{Resource Types} & \tau & ::= & \mathsf{G}[T \parallel \Delta] \mid X \mid \mu X.\mathsf{G}[T \parallel \Delta\{X\}] & \\ \textit{Delivery Policies} & \Delta & ::= & [\mathsf{G}_i \to \tau_i]_{i \in I} & (\mathsf{G}_i = \mathsf{G}_j \Rightarrow i = j) \end{array}$$

Sample Types

- Channels of group G that may be received and re-transmitted at group F only as write-only channels.

$$\mu X.\ \mathsf{G}[\ (\mathsf{nat})^{\mathsf{rw}}\ \ \|\ \ \mathsf{G} \rightarrow X;\ \ \ \mathsf{F} \rightarrow \mu Y.\mathsf{G}[(\mathsf{nat})^{\mathsf{w}} \parallel \mathsf{F} \rightarrow Y]\ \ ]$$

Sample Types

- Channels of group G that may be received and re-transmitted at group F only as write-only channels.

$$\mu X.\ \mathsf{G}[\ (\mathsf{nat})^{\mathsf{rw}}\ \ \|\ \ \mathsf{G} \to X;\ \ \mathsf{F} \to \mu Y.\mathsf{G}[(\mathsf{nat})^{\mathsf{w}} \parallel \mathsf{F} \to Y]\ \ ]$$

- Default entries also allowed:

$$\mu X.\ \mathsf{G}[\ (\mathsf{nat})^{\mathsf{rw}}\ \ \|\ \ \mathsf{G} \to X;\ \ \mathsf{Default} \to \mu Y.\mathsf{G}[(\mathsf{nat})^{\mathsf{w}} \parallel \mathsf{Default} \to Y]\ \ ]$$

Sample Types

- Channels of group G that may be received and re-transmitted at group F only as write-only channels.

$$\mu X.\, \mathsf{G}[\,(\mathsf{nat})^{\mathsf{rw}} \;\|\; \mathsf{G} \to X;\;\; \mathsf{F} \to \mu Y.\mathsf{G}[(\mathsf{nat})^{\mathsf{w}} \,\|\, \mathsf{F} \to Y]\;\,]$$

- Default entries also allowed:

$$\mu X.\, \mathsf{G}[\,(\mathsf{nat})^{\mathsf{rw}} \;\|\; \mathsf{G} \to X;\;\; \mathsf{Default} \to \mu Y.\mathsf{G}[(\mathsf{nat})^{\mathsf{w}} \,\|\, \mathsf{Default} \to Y]\;\,]$$

- Two parties, Alice and Bob, establish a private exchange. Alice sends a fresh name c_{AB} to a trusted Server and delegates it to forward it to Bob. The Server should only act as a forwarder, and not interfere with the exchanges between Alice and Bob.

$$c_{AB} : \mathsf{Alice}[\,(\mathsf{data})^{\mathsf{rw}} \;\|\; \mathsf{Server} \to \mathsf{Alice}[(\mathsf{data}) \,\|\, \mathsf{Bob} \to \mathsf{Alice}[(\mathsf{data})^{\mathsf{rw}} \;\|]]\,]$$

Operational Semantics

Different occurrences of the same name may flow along different paths:

Let $n_1{:}\mathsf{G}_1[\ldots], n_2{:}\mathsf{G}_2[\ldots], n_3{:}\mathsf{G}_3[\ldots]$ and $m : \mathsf{G}[\mathsf{B} \parallel \mathsf{G}_1 \rightarrow \mathsf{G}_2 \ ; \ \mathsf{G}_3]$.

$$P \triangleq \overline{n_1}\langle m \rangle \ \mid \ \overline{n_3}\langle m \rangle \ \mid \ n_1(x).\, n_3(y).\, \overline{n_2}\langle x \rangle$$

$$Q \triangleq \overline{n_1}\langle m \rangle \ \mid \ \overline{n_3}\langle m \rangle \ \mid \ n_1(x).\, n_3(y).\, \overline{n_2}\langle y \rangle$$

Operational Semantics

Different occurrences of the same name may flow along different paths:

Let $n_1{:}\mathsf{G}_1[\ldots], n_2{:}\mathsf{G}_2[\ldots], n_3{:}\mathsf{G}_3[\ldots]$ and $m : \mathsf{G}[\mathsf{B} \parallel \mathsf{G}_1 \rightarrow \mathsf{G}_2 \ ; \ \mathsf{G}_3]$.

$$P \ \triangleq \ \overline{n_1}\langle m\rangle \ \mid \ \overline{n_3}\langle m\rangle \ \mid \ n_1(x).\,n_3(y).\,\overline{n_2}\langle x\rangle$$

$$Q \ \triangleq \ \overline{n_1}\langle m\rangle \ \mid \ \overline{n_3}\langle m\rangle \ \mid \ n_1(x).\,n_3(y).\,\overline{n_2}\langle y\rangle$$

P should be safe, Q unsafe, but $\quad P \rightarrow\rightarrow \ \overline{n_2}\langle m\rangle \ \leftarrow\leftarrow \ Q$.

Operational Semantics

Use names that are tagged to record their flow history: $m_{[npq]}$

$$\overline{n_{[\varphi]}}\langle m_{1\,[\varphi_1]}, \ldots, m_{k\,[\varphi_k]}\rangle.A \;\mid\; n_{[\psi]}(x_1 : \tau_1, \ldots, x_k : \tau_k).B$$

$$\longrightarrow \quad A \;\mid\; B\{x_i := m_{i\,[\varphi_i\, n]}\}$$

Now the computation exhibits different flows for P and Q:

$$P = \overline{n_1}\langle m\rangle \mid \overline{n_3}\langle m\rangle \mid n_1(x).n_3(y).\overline{n_2}\langle x\rangle \;\rightarrow\rightarrow\; \overline{n_2}\langle m_{[n_1]}\rangle$$

$$Q = \overline{n_1}\langle m\rangle \mid \overline{n_3}\langle m\rangle \mid n_1(x).n_3(y).\overline{n_2}\langle y\rangle \;\rightarrow\rightarrow\; \overline{n_2}\langle m_{[n_3]}\rangle$$

Theorem

- If $A \longrightarrow^* B$ then $|A| \mapsto^* |B|$.
- If $|A| \mapsto^* Q$, then $\exists B$ such that $A \longrightarrow^* B$ and $|B| \equiv Q$.

Type formation and Subtyping

Good types

$$\mathsf{G}[\ T \parallel \qquad\qquad\qquad\qquad\qquad\qquad \ ]$$

Type formation and Subtyping

Good types

$$\mathsf{G}[\ T \parallel \mathsf{G}_1 \to \mathsf{G}[T_1 \parallel \qquad\qquad\qquad]\ ]$$

Type formation and Subtyping

Good types

$$\mathsf{G}[\ T \parallel \mathsf{G}_1 \rightarrow \mathsf{G}[T_1 \parallel \mathsf{G}_2 \rightarrow \mathsf{G}[T_1 \parallel \Delta]\ ]\ ]$$

- delivery preserves the authority in control of values
- T_i are supertypes of T

Type formation and Subtyping

Good types

$$\mathsf{G}[\ T \parallel \mathsf{G}_1 \rightarrow \mathsf{G}[T_1 \parallel \mathsf{G}_2 \rightarrow \mathsf{G}[T_1 \parallel \Delta]\,]\ ]$$

- delivery preserves the authority in control of values
- T_i are supertypes of T

Subtyping

$$(\mathcal{T}\text{-}\textsc{Type})\quad \frac{\Gamma \vdash T \leq T'}{\Gamma \vdash \mathsf{G}[T \parallel \Delta] \quad \leq \quad \mathsf{G}[T' \parallel \Delta]} \qquad\qquad (\mathcal{T}\text{-}\textsc{Policy})\quad \frac{\Gamma \vdash \Delta \preccurlyeq \Delta'}{\Gamma \vdash \mathsf{G}[T \parallel \Delta] \quad \leq \quad \mathsf{G}[T \parallel \Delta']}$$

- $\Delta \preccurlyeq \Delta'$ implies Δ' is at least as restrictive as Δ

Core Typing Rules

Good Messages

(Delivery)

$$\frac{\Gamma \vdash n_{[\varphi]}{:}\mathsf{G}[T \parallel \Delta] \quad \Gamma \vdash m{:}\mathsf{G}_1[T_1 \parallel \Delta_1] \quad (\mathsf{G}_1 \to \mathcal{T} \in \Delta) \text{ 'or' } (\mathsf{Default} \to \mathcal{T} \in \Delta)}{\Gamma \vdash n_{[\varphi m]} : \mathcal{T}}$$

Good Processes

(Input)

$$\frac{\Gamma \vdash a : \mathsf{G}[(\tau_1, \ldots, \tau_k)^{\mathsf{r}}] \quad \Gamma, x_1 : \tau_1, \ldots, x_k : \tau_k \vdash P}{\Gamma \vdash a(x_1 : \tau_1, \ldots, x_k : \tau_k).P}$$

(Output)

$$\frac{\Gamma \vdash a : \mathsf{G}[(\tau_1, \ldots, \tau_k)^{\mathsf{w}}] \quad \Gamma \vdash P \quad \Gamma \vdash b_i : G_i[T_i \parallel \Delta_i] \quad \Gamma \vdash \Delta_i(\mathsf{G}) \preccurlyeq \tau_i}{\Gamma \vdash \overline{a}\langle b_1, \ldots, b_k \rangle.P}$$

Type soundness

Theorem: **Access Control**

If $\quad \Gamma \vdash \overline{n_{[\varphi]}}\langle a_1, \ldots, a_k \rangle.A' \mid n_{[\varphi']}(x_1 : \rho_1, \ldots, x_l : \rho_l).B' \quad$ then

Type soundness

Theorem: **Access Control**

If $\Gamma \vdash \overline{n_{[\varphi]}}\langle a_1, \ldots, a_k\rangle.A' \mid n_{[\varphi']}(x_1 : \rho_1, \ldots, x_l : \rho_l).B'$ then

$\Gamma \vdash n_{[\varphi]} : \mathsf{G}[(\tau_1, \ldots, \tau_k)^{\mathsf{w}} \;\; \| \; \Delta]$

The write capability
has been granted
to the writer process

$\Gamma \vdash n_{[\varphi']} : \mathsf{G}[(\rho_1, \ldots, \rho_k)^{\mathsf{r}} \;\; \| \; \Delta']$

The read capability
has been granted
to the reader process

Type soundness

Theorem: **Access Control**

If $\quad \Gamma \vdash \overline{n_{[\varphi]}}\langle a_1, \ldots, a_k \rangle.A' \mid n_{[\varphi']}(x_1 : \rho_1, \ldots, x_l : \rho_l).B' \quad$ then

$\Gamma \vdash n_{[\varphi]} : \mathsf{G}[(\tau_1, \ldots, \tau_k)^{\mathsf{w}} \;\; \| \; \Delta]$

$\Gamma \vdash n_{[\varphi']} : \mathsf{G}[(\rho_1, \ldots, \rho_k)^{\mathsf{r}} \;\; \| \; \Delta']$

$\Gamma \vdash a_i : \sigma_i \qquad \sigma_i \downarrow \mathsf{G} \leq \tau_i \qquad \tau_i \leq \rho_i$

The types σ_i of the emitted values
allow a_i to be delivered to G at a subtype
of the type ρ_i at which they are received

Type soundness

n flowed
as described in φ

Theorem: **Flow Control**

Let $\Gamma \vdash A$ be a derivable judgement
depending on the judgement $\Gamma' \vdash n_{[\varphi]} : \tau$

Type soundness

n flowed
as described in φ

Theorem: **Flow Control**

Let $\Gamma \vdash A$ be a derivable judgement
depending on the judgement $\Gamma' \vdash n_{[\varphi]} : \tau$
then

$$\Gamma'(n) = \rho \text{ such that } \rho \downarrow \varphi \leq \tau.$$

The type ρ allows n
to be delivered at a subtype of τ
after flowing according to φ

Type soundness

Accees Control + Flow Control

+ Subject Reduction

$$\Downarrow$$

Safety properties preserved along the computation

... **Secrecy** as in [CGG00] observing that

$$[\![\, \mathsf{G}[T_1, \ldots, T_n] \,]\!] = \mu X.\ \mathsf{G}[\, ([\![\, T_1 \,]\!], \ldots, [\![\, T_n \,]\!])^{\mathsf{rw}} \ \| \ \mathsf{Default} \to X \ ]$$

Conclusions

What we have done

- developed a new type foundation for discretionary policies for access control
- flexible/powerful and provides strong security guarantees
- a conservative extension of the type system by [CGG00]

A lot to be done

- allow changes in the ownership of names, account for ordering relationships over authorities,
- accommodate declassification mechanisms
- study import of type-based policies with typed behavioral equivalences