Secure Implementations of Typed Channel Abstractions

Marco Giunti

Dep. of Informatics, University of Lisbon

April 16, 2008

(joint work with Michele Bugliesi)

Abstract

Analysis distributed computer systems

- process algebras techniques
- formal tools to control and reason about their behaviour

Such tools adequate to describe distributed systems?

- model should be implementable

Example

Private communications in the pi calculus

$$P = (\text{new } a)\overline{a}\langle b\rangle \mid a(x).\overline{p}\langle a\rangle \qquad P \longrightarrow (\text{new } a)\overline{p}\langle a\rangle$$

- Communication invisible by the context:

$$P \approx (\text{new } a)\overline{p}\langle a\rangle \quad (*)$$

Secure implementation

- model using open communications and cryptography in applied pi calculus
- Dolev-Yao intruder

Equation $(*)$ preserved

Resource access control

Relevant both for design and security

- e.g. mailbox

$$C \;=\; \overline{m}\langle mail \rangle \qquad M = m(x).P$$

- no guarantee mail not read by context

$$C \,|\, M \,|\, m(y).D \longrightarrow M \,|\, D\{mail/y\}$$

Resource access control

Relevant both for design and security

- e.g. mailbox

$$C \quad = \quad \overline{m}\langle mail\rangle \qquad M = m(x).P$$

- no guarantee mail not read by context

$$C \,|\, M \,|\, m(y).D \longrightarrow M \,|\, D\{mail/y\}$$

Pi calculus solution [PS'96]:

- channels have read/write polarities
- (static) typechecking enforces access control

Type	Mode
$a : T^{\mathsf{rw}}$	read/write
$a : T^{\mathsf{r}}$	read
$a : T^{\mathsf{w}}$	write

Access control by static typing

Typed mailbox:

$$M \;=\; (\mathsf{new}\ m)\quad \overline{p}\langle m\rangle \mid m(y).P$$

- Provided $\mathrm{I} \vdash p : (\mathsf{string}^{\mathsf{w}})^{\mathsf{rw}}$
- mail channel obtained by contexts at type $\mathsf{string}^{\mathsf{w}}$

Access control by static typing

Typed mailbox:

$$M \;=\; (\text{new } m) \quad \overline{p}\langle m\rangle \mid m(y).P$$

- Provided $\mathrm{I} \vdash p : (\mathsf{string}^{\mathsf{w}})^{\mathsf{rw}}$
- mail channel obtained by contexts at type $\mathsf{string}^{\mathsf{w}}$
- Fact : type of distribution channel regulates how contexts acquire capabilities
- formalization: typed labelled transitions

$$\frac{\mathrm{I} \vdash p : (\mathsf{string}^{\mathsf{w}})^{\mathsf{rw}}}{\mathrm{I} \triangleright M \xrightarrow{(m)\overline{p}\langle m\rangle} \mathrm{I}, m : \mathsf{string}^{\mathsf{w}} \triangleright M'}$$

Implementing typed access control

Motivation

- needed to use typed process calculi as specification tool for distributed systems

Difficulties

- Source level: behaviour of contexts enforced by static typing, i.e. “enemies” respect the game’s rules
- Implementation level: no assumptions on behaviour or trust of contexts

Preserving typed equations

We want to preserve typed equations of the form

$$\mathrm{I} \models P \approx^{\pi} Q$$

- types have semantics consequences
- e.g. secret buffer

$$b : T^{\mathsf{r}} \models b(x).P \approx^{\pi} \mathbf{0}$$

Preserving typed equations

We want to preserve typed equations of the form

$$\mathrm{I} \models P \approx^{\pi} Q$$

- types have semantics consequences
- e.g. secret buffer

$$b : T^{\mathsf{r}} \models b(x).P \approx^{\pi} \mathbf{0}$$

- in contrast low-level untyped contexts gain more capabilities on the buffer

What we have done

Developed typed pi calculus with dynamically typed synchronization

- Syntax

  $\overline{p}\langle s@T \rangle$ *type-coerced output*

  $T ::= \mathsf{rw} \mid \mathsf{w} \mid \mathsf{r} \mid \top$ *Types*

- Semantics

$$\overline{a}\langle b@T \rangle \mid a(x@S).P \longrightarrow P\{b/x\} \quad \textit{provided} \quad T <: S$$

What we have done

Developed typed pi calculus with dynamically typed synchronization

- Syntax

$\overline{p}\langle s@T \rangle$	*type-coerced output*
$T ::= \mathsf{rw} \mid \mathsf{w} \mid \mathsf{r} \mid \top$	*Types*

- Semantics

$$\overline{a}\langle b@T \rangle \mid a(x@S).P \longrightarrow P\{b/x\} \quad \textit{provided} \quad T <: S$$

Secure implementation of typed pi calculus

- full abstraction

$$\mathrm{I} \models P \approx^{\pi} Q \Leftrightarrow [\![\mathrm{I}]\!] \models [\![P]\!] \approx^{A\pi} [\![Q]\!]$$

Dynamic vs static typing

Dynamic approach

- type S decided by coercion type
- $$\frac{\mathrm{I}(p) = \mathsf{r}}{\mathrm{I} \triangleright M \xrightarrow{(m)\overline{p}\langle m@S\rangle} \mathrm{I}, m : S \triangleright M'}$$

Static approach

- type S decided by transmission channel type
- $$\frac{\mathrm{I}(p) = S^{\mathsf{r}}}{\mathrm{I} \triangleright M \xrightarrow{(m)\overline{p}\langle m\rangle} \mathrm{I}, m : S \triangleright M'}$$

Towards the implementation

$$P = (\mathsf{new}\ a)\overline{a}\langle b\rangle \mid a(x).\overline{p}\langle a@\mathsf{r}\rangle \qquad p : \mathsf{r} \models P \approx^{\pi} (\mathsf{new}\ a)\overline{p}\langle a@\mathsf{r}\rangle$$

Naive solution

- represent channel as couple formed by encryption and decryption key

$$[\![P]\!] = (\mathsf{new}\ a^{+}, a^{-})$$

$$!\,net\langle\{b^{+}, b^{-}\}_{a^{+}}\rangle \mid net(y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{\tilde{x}\}_{a^{-}}\ \mathsf{in}\ !\,net\langle\{a^{+}, a^{-}\}_{p^{+}}\rangle$$

- forward secrecy *open problem* [Abadi, ICALP‘98]

Towards the implementation

$$P = (\text{new } a)\overline{a}\langle b\rangle \,|\, a(x).\overline{p}\langle a@\text{r}\rangle \qquad p : \text{r} \models P \approx^{\pi} (\text{new } a)\overline{p}\langle a@\text{r}\rangle$$

Naive solution

- represent channel as couple formed by encryption and decryption key

$$[\![P]\!] = (\text{new } a^{+}, a^{-})$$
$$!\,net\langle\{b^{+}, b^{-}\}_{a^{+}}\rangle \,|\, net(y).\text{decrypt } y \text{ as } \{\tilde{x}\}_{a^{-}} \text{ in } !\,net\langle\{a^{+}, a^{-}\}_{p^{+}}\rangle$$

- forward secrecy *open problem* [Abadi, ICALP‘98]

Our solution

- represent channel as process does not leak decryption key

A sound implementation

Client /server scheme with a read/write protocol

- Types mapped into read/write encryption keys

$$[\![a@\mathsf{rw}]\!] = a^+_{\mathsf{w}}, a^+_{\mathsf{r}} \quad [\![a@\mathsf{w}]\!] = a^+_{\mathsf{w}} \quad [\![a@\mathsf{r}]\!] = a^+_{\mathsf{r}}$$

- Input/output source processes implemented as clients using encryption keys
  - translated output processes use a^+_{w} (write protocol)
  - translated input processes use a^+_{r} (read protocol)
- Decryption keys stored in secure channel manager servers

$$Chan_a \quad = \quad (\mathsf{new}\ a^\circ) WS_a \,|\, RS_a$$

Write protocol

Client

Packages requests with a^+_{w} containing a fresh nonce

$$[\![\overline{a}\langle v@T \rangle]\!] = Emit\{[\![v@T]\!]\}_{a^+_w} \triangleq (\mathsf{new}\ c)\ !net\langle \{[\![v@T]\!], c\}_{a^+_w} \rangle$$

Write protocol

Client

Packages requests with a_{w}^{+} containing a fresh nonce

$$[\![\, \overline{a}\langle v@T\rangle \,]\!] = Emit\{[\![\, v@T \,]\!]\}_{a_w^+} \triangleq (\mathsf{new}\ c)\ !\,net\langle\{[\![\, v@T \,]\!], c\}_{a_w^+}\rangle$$

Server

Stores (fresh) write requests in a secret local buffer a°

$$WS_a \quad = \quad !\,\mathsf{filter}\ (\tilde{x}, z)\ \mathsf{with}\ a_w^-\ \mathsf{in}\ \mathsf{if}\ z\ \mathrm{fresh}\ \ \mathsf{then}\ \overline{a^{\circ}}\langle\tilde{x}\rangle$$

Notation

A filter on k^{-} discards all packets non-encrypted under k^{+}

$$\mathsf{filter}\ \tilde{y}\ \mathsf{with}\ k^{-}\ \mathsf{in}\ P \ = \ net(x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\tilde{y}\}_{k^-}\ \mathsf{in}\ P\ \mathsf{else}\ net\langle x\rangle$$

Read Protocol

Client

Package requests (w.r.t. types) containing a session key for the answer

$$[\![a(x@T).P]\!] = (\mathsf{new}\ k)\, Emit(\{k, T\}_{a_r^+}) \mid\ !\,\mathsf{filter}\ \tilde{x}\ \mathsf{with}\ k\ \mathsf{in}\ [\![P]\!]$$

Read Protocol

Client

Package requests (w.r.t. types) containing a session key for the answer

$$[\![a(x@T).P]\!] = (\mathsf{new}\ k)\,Emit(\{k,T\}_{a_r^+}) \mid \,!\,\mathsf{filter}\ \tilde{x}\ \mathsf{with}\ k\ \mathsf{in}\ [\![P]\!]$$

Server

Filters packets from the buffer a° at given types

$$\begin{array}{rcl} RS_a & = & !\,\mathsf{filter}\ (y,t,z)\ \mathsf{with}\ a_r^-\ \mathsf{in} \\ & & \quad \mathsf{if}\ z\ \mathrm{fresh}\ \ \mathsf{then\ filter}\ \tilde{x}\ \mathsf{from}\ a^\circ@t\ \mathsf{in}\ !\,net\langle\{\tilde{x}\}_y\rangle \end{array}$$

Notation

A filter from n at t pick up messages from n at a “subtype” of t

$$\mathsf{filter}\ \tilde{x}\ \mathsf{from}\ n@t\ \mathsf{in}\ P \ = \ n(\tilde{x}).\mathsf{if}\ wf(\tilde{x},t)\ \mathsf{then}\ P\ \mathsf{else}\ \overline{n}\langle\tilde{x}\rangle$$

Encoding of pi calculus processes

$$
\begin{array}{lcl}
[\![(\mathsf{new}\ a)P]\!] & = & (\mathsf{new}\ a)Chan_a \mid [\![P]\!] \\
[\![\overline{u}\langle v@T \rangle]\!] & = & Emit\{[\![v@T]\!]\}_{u_w^+} \\
[\![u(x@T).P]\!] & = & (\mathsf{new}\ k)Emit(\{k, T\}_{u_r^+}) \mid !\,\mathsf{filter}\ \tilde{x}\ \mathsf{with}\ k\ \mathsf{in}\ [\![P]\!] \\
[\![P \mid Q]\!] & = & [\![P]\!] \mid [\![Q]\!] \\
[\![!P]\!] & = & ![\![P]\!] \\
[\![[u = v]P; Q]\!] & = & \mathsf{if}\ u_{ID} = v_{ID}\ \mathsf{then}\ [\![P]\!]\ \mathsf{else}\ [\![Q]\!] \\
[\![\mathbf{0}]\!] & = & \mathbf{0}
\end{array}
$$

Soundness of the implementation

I a closed type environment of the pi calculus

- Computing environment for translated processes:

$$\mathsf{E}_{\mathrm{I}}[-] \;=\; - \,|\, W \,|\, \prod_{n \in \mathrm{dom}(\mathrm{I})} Chan_n$$

Theorem

$$\mathrm{I} \models P \cong^{\pi} Q \quad \text{iff} \quad \mathsf{E}_{\mathrm{I}}[\llbracket P \rrbracket] \cong^{A\pi}_{\mathsf{tr}} \mathsf{E}_{\mathrm{I}}[\llbracket Q \rrbracket]$$

Closed only under translated contexts: not satisfying

Soundness of the implementation

I a closed type environment of the pi calculus

- Computing environment for translated processes:

$$\mathsf{E}_{\mathrm{I}}[-] \;=\; - \,|\, W \,|\, \prod_{n \in \mathrm{dom}(\mathrm{I})} Chan_n$$

Theorem

$$\mathrm{I} \models P \cong^{\pi} Q \quad \text{iff} \quad \mathsf{E}_{\mathrm{I}}[\llbracket P \rrbracket] \cong^{A\pi}_{\mathsf{tr}} \mathsf{E}_{\mathrm{I}}[\llbracket Q \rrbracket]$$

Closed only under translated contexts: not satisfying

Example $\quad a(x).\overline{a}\langle x\rangle \cong^{\pi} \mathbf{0} \not\Rightarrow \mathsf{E}_{\mathrm{I}}[\llbracket a(x).\overline{a}\langle x\rangle \rrbracket] \cong^{A\pi} \mathsf{E}_{\mathrm{I}}[\mathbf{0}]$

- If a generated by the context the channel manager for a is not secure

Enhancing the design

Channel servers created by trusted centralized authority (Proxy)

- separation among client (unsafe) and server (safe) names
- client names associated to server names in Proxy's table
- client names tokens for server names requested using proxy's public key k_p^+

$$\text{link } (M, \tilde{y}) \text{ in } P \triangleq (\text{new } h)Emit(\{h, M\}_{k_p^+}) \,|\, \text{filter } \tilde{y} \text{ with } h \text{ in } P$$

Enhancing the design

Channel servers created by trusted centralized authority (Proxy)

- separation among client (unsafe) and server (safe) names
- client names associated to server names in Proxy's table
- client names tokens for server names requested using proxy's public key k_p^+

$$\text{link } (M, \tilde{y}) \text{ in } P \triangleq (\text{new } h) Emit(\{h, M\}_{k_p^+}) \mid \text{filter } \tilde{y} \text{ with } h \text{ in } P$$

- read/write protocol same rationale

Refined encoding

$$
\begin{array}{lcl}
[\![(\mathsf{new}\ a)P]\!] & = & (\mathsf{new}\ a)[\![P]\!] \\
[\![\overline{a}\langle v@T\rangle]\!] & = & \mathsf{link}\ ([\![a@\mathsf{w}]\!], \underline{y})\ \mathsf{in}\ \mathit{Emit}\{[\![v@T]\!]\}_{\underline{y}_w^+} \\
[\![a(x@T).P]\!] & = & \mathsf{link}\ ([\![a@\mathsf{r}]\!], \underline{y})\ \mathsf{in}\ (\mathsf{new}\ k)\,\mathit{Emit}(\{k, T\}_{\underline{y}_r^+}) \\
 & & \quad \mid\ !\,\mathsf{filter}\ \tilde{x}\ \mathsf{with}\ k\ \mathsf{in}\ [\![P]\!] \\
[\![P \mid Q]\!] & = & [\![P]\!] \mid [\![Q]\!] \\
[\![!P]\!] & = & ![\![P]\!] \\
[\![[u = v]P; Q]\!] & = & \mathsf{if}\ u_{ID} = v_{ID}\ \mathsf{then}\ [\![P]\!]\ \mathsf{else}\ [\![Q]\!] \\
[\![\mathbf{0}]\!] & = & \mathbf{0}
\end{array}
$$

Full Abstraction

Centalized implementation fully abstract:

- Computing environment: $\mathsf{CE}[-] = \mathit{Proxy} \,|\, W \,|\, -$

$$\mathrm{I} \models P \cong^{\pi} Q \Leftrightarrow [\![\mathrm{I}]\!] \models \mathsf{CE}[[\![P]\!]] \cong^{A\pi} \mathsf{CE}[[\![Q]\!]]$$

Full Abstraction

Centalized implementation fully abstract:

- Computing environment: $\mathsf{CE}[-] = \mathit{Proxy} \,|\, W \,|\, -$

$$\mathrm{I} \models P \cong^{\pi} Q \Leftrightarrow [\![\mathrm{I}]\!] \models \mathsf{CE}[[\![P]\!]] \cong^{A\pi} \mathsf{CE}[[\![Q]\!]]$$

- long and difficult proof
- bisimulation-based techniques
- main tool: notion *administrative* steps first characterized then abstracted away

A distributed implementation

- Pi calculus with domain labels (no impact on types/semantics)

$$S, T \; ::= \; \delta\{P\} \;\mid\; S \,|\, T \;\mid\; (\mathsf{new}\; n : A)S \;\mid\; \mathbf{stop}$$

- each domain mapped in a trusted proxy
- $\mathrm{I} \models S \cong^{\pi} T$ closed under contexts using known domains
- proxies coordinate to create virtual single queue for channel manager

A distributed implementation

- Pi calculus with domain labels (no impact on types/semantics)

$$S, T \ ::= \ \delta\{P\} \ | \ S \,|\, T \ | \ (\mathsf{new}\ n : A)S \ | \ \mathbf{stop}$$

- each domain mapped in a trusted proxy
- $\mathrm{I} \models S \cong^{\pi} T$ closed under contexts using known domains
- proxies coordinate to create virtual single queue for channel manager
- distributed implementation fully abstract

$$\begin{aligned} \mathrm{I} \models S \cong^{\pi} T \Leftrightarrow [\![\mathrm{I}]\!] \quad \models \quad & \Pi_{d \in fd(S,T)} Proxy_d \,|\, W \,|\, [\![S]\!] \cong^{A\pi} \\ & \Pi_{d \in fd(S,T)} Proxy_d \,|\, W \,|\, [\![T]\!] \end{aligned}$$

Conclusions

- Revised access control by subtyping in the pi calculus to make implementation possible in untyped networks
- In fact: source calculus result of “reverse engineering” of the implementation
- Given a secure implementation of typed abstractions
  - first result of this kind for typed process calculi
  - solves open problems (forward secrecy)
  - first implementation of pi with matching

Limitations

All proxies participating in distributed implementation fully trusted

- model sub-sytems as physical locations trusting each other
- some form of guarantees in the presence of malicious proxies desiderable
- seems achievable by strengthening protocols that govern interactions among proxies

Noise's presence hardly realistic

- move to models consider semantic probabilistic equations [Palamidessi and al. '00, Mitchell and al. 06]

Thanks!