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

\[ P = (\text{new } a)\bar{a}\langle b \rangle \mid a(x).\overline{p}\langle a \rangle \quad P \rightarrow (\text{new } a)\overline{p}\langle a \rangle \]

Communication invisible by the context:

\[ P \approx (\text{new } a)\overline{p}\langle a \rangle \quad (\ast) \]

Secure implementation

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

Equation (\ast) preserved
Resource access control

Relevant both for design and security

- e.g. mailbox

\[ C = \overline{m}(mail) \quad M = m(x).P \]

- 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

\[ C = \overline{m}(mail) \quad M = m(x).P \]

- no guarantee mail not read by context

\[ C | M | m(y).D \rightarrow M | D\{mail/y\} \]

Pi calculus solution [PS’96]:

- channels have read/write polarities

- (static) typechecking enforces access control

<table>
<thead>
<tr>
<th>Type</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : T_{rw}</td>
<td>read/write</td>
</tr>
<tr>
<td>a : T_{r}</td>
<td>read</td>
</tr>
<tr>
<td>a : T_{w}</td>
<td>write</td>
</tr>
</tbody>
</table>
Access control by static typing

Typed mailbox:

\[ M = \text{(new } m) \parallel p\langle m \rangle \mid m(y).P \]

\begin{itemize}
  \item Provided \( I \vdash p : (\text{string}^w)^{rw} \)
  \item mail channel obtained by contexts at type \( \text{string}^w \)
\end{itemize}
Access control by static typing

Typed mailbox:

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

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

**Fact**: type of distribution channel regulates how contexts acquire capabilities

- formalization: typed labelled transitions

\[
\begin{align*}
I \vdash p : (\text{string}^w)^{rw} \\
\vdash (m)\overline{p}\langle m \rangle \\
I \triangleright M \quad \xrightarrow{(m)\overline{p}\langle m \rangle} \quad I, m : \text{string}^w \triangleright M'
\end{align*}
\]
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

\[ \Pi \models P \approx^{\pi} Q \]

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

\[ b : T^r \models b(x).P \approx^{\pi} 0 \]
Preserving typed equations

We want to preserve typed equations of the form

\[ I \models P \approx^\pi Q \]

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

\[ b : T^r \models b(x).P \approx^\pi 0 \]

- 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

  \[ \bar{p}\langle s@T \rangle \quad \text{type-coerced output} \]

  \[ T ::= \text{rw} \mid \text{w} \mid \text{r} \mid \top \quad \text{Types} \]

- Semantics

  \[ \bar{a}\langle b@T \rangle \mid a(x@S).P \rightarrow P\{b/x\} \quad \text{provided} \quad T <: S \]
What we have done

Developed typed pi calculus with dynamically typed synchronization

- Syntax

  \[ \overline{p}\langle s @ T \rangle \quad \text{type-coerced output} \]

  \[ T ::= \text{rw} | w | r | \top \quad \text{Types} \]

- Semantics

  \[ \overline{a}\langle b @ T \rangle | a(x @ S).P \rightarrow P\{b/x\} \quad \text{provided} \quad T <: S \]

Secure implementation of typed pi calculus

- full abstraction

  \[ I \models P \approx^{\pi} Q \iff [I] \models [P] \approx^{A\pi} [Q] \]
**Dynamic vs static typing**

**Dynamic approach**
- type $S$ decided by *coercion* type

\[
\begin{align*}
I(p) &= r \\
\vdash M &\xrightarrow{(m)\overline{p}\langle m@S \rangle} I, m : S \triangleright M'
\end{align*}
\]

**Static approach**
- type $S$ decided by *transmission channel* type

\[
\begin{align*}
I(p) &= S^r \\
\vdash M &\xrightarrow{(m)\overline{p}\langle m \rangle} I, m : S \triangleright M'
\end{align*}
\]
Towards the implementation

\[ P = (\text{new } a) \overline{a} \langle b \rangle \mid a(x) \cdot p \langle a @ r \rangle \quad p : r \models P \equiv_{r} (\text{new } a) \overline{p} \langle a @ r \rangle \]

Naive solution

- represent channel as couple formed by encryption and decryption key

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

- forward secrecy open problem  [Abadi, ICALP‘98]
Towards the implementation

\[ P = (\text{new } a)\overline{a} \langle b \rangle \mid a(x).p\langle a@r \rangle \quad p : r \models P \approx \pi (\text{new } a)\overline{p}\langle a@r \rangle \]

Naive solution

- represent channel as couple formed by encryption and decryption key

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

- forward secrecy open problem \cite{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
  \[
  \lfloor a@rw \rfloor = a_w^+, a_r^+ \quad \lfloor a@w \rfloor = a_w^+ \quad \lfloor a@r \rfloor = a_r^+
  \]

- 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
  \[
  Chan_a = (\text{new } a^\circ)WS_a | RS_a
  \]
Write protocol

Client

Packages requests with $a^+_w$ containing a fresh nonce

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

Client
Packages requests with $a^+_w$ containing a fresh nonce

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

Server
Stores (fresh) write requests in a secret local buffer $a^\circ$

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

Notation
A filter on $k^-$ discards all packets non-encrypted under $k^+$

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

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

$$\boxed{a(x@T).P} = (\text{new } k) \text{ Emit}(\{k, T\}_{a^+}) \mid !\text{filter } \tilde{x} \text{ with } k \text{ in } \boxed{P}$$
Read Protocol

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

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

Server
Filters packets from the buffer \(a^\circ\) at given types

\[ RS_a = \text{! filter } (y, t, z) \text{ with } a_r^- \text{ in } \]
if \(z\) fresh then filter \(\tilde{x}\) from \(a^\circ@t\) in \(!net\langle\{\tilde{x}\}_y\rangle\)

Notation
A filter from \(n\) at \(t\) pick up messages from \(n\) at a “subtype” of \(t\)
filter \(\tilde{x}\) from \(n@t\) in \(P = n(\tilde{x})\).if \(wf(\tilde{x}, t)\) then \(P\) else \(\overline{n}\langle\tilde{x}\rangle\)
Encoding of pi calculus processes

\[(\text{new } a)P] = (\text{new } a)\text{Chan}_a \mid [P]\]

\[\bar{u}\langle v@T \rangle] = \text{Emit}\{[v@T]\}_{w^+}\]

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

\[P \mid Q\] = \[P\] \mid \[Q\]

\[!!P\] = ![P]

\[[u = v]P; Q\] = if \(u_{ID} = v_{ID}\) then \[P\] else \[Q\]

\[0\] = 0
Soundness of the implementation

A closed type environment of the pi calculus

Computing environment for translated processes:

\[ E_I[-] = -|W| \prod_{n \in \text{dom}(I)} \text{Chan}_n \]

Theorem

\[ I \models P \equiv^\pi Q \iff E_I[[P]] \simeq^{\text{tr}}_{\pi} E_I[[Q]] \]

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:

\[ E_I[-] = -|W| \prod_{n \in \text{dom}(I)} \text{Chan}_n \]

**Theorem**

\[ I \models P \equiv^\pi Q \iff E_I[[P]] \equiv_{\text{tr}}^{A\pi} E_I[[Q]] \]

Closed only under translated contexts: not satisfying

**Example**  \( a(x).\overline{a}(x) \equiv^\pi 0 \not\Rightarrow E_I[[a(x).\overline{a}(x)]] \equiv_{\text{tr}}^{A\pi} E_I[0] \)

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 $k_P^+$

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

\[
\text{link } (M, \tilde{y}) \text{ in } P \triangleq (\text{new } h)\text{Emit} \left( \{h, M\}^{k_p^+} \right) | \text{filter } \tilde{y} \text{ with } h \text{ in } P
\]

- read/write protocol same rationale
Refined encoding

\[
\begin{align*}
\text{[ (new } a \text{) } P] & = (\text{new } a)[P] \\
\text{[ } a \langle v @ T \rangle \text{]} & = \text{link } ([a @ w], y) \text{ in } \text{Emit}([v @ T])_{y^+}^{\overline{w}} \\
\text{[ } a(x @ T).P \text{]} & = \text{link } ([a @ r], y) \text{ in } (\text{new } k) \text{ Emit}(\{k, T\})_{y^+}^{\overline{r}} \\
& \quad | \! \text{! filter } \tilde{x} \text{ with } k \text{ in } [P] \\
\text{[ } P \mid Q \text{]} & = [P] \mid [Q] \\
\text{[ !P] } & = \! [P] \\
\text{[ } u = v ] P; Q \text{]} & = \text{if } u_{ID} = v_{ID} \text{ then } [P] \text{ else } [Q] \\
\text{[ 0 ]} & = 0
\end{align*}
\]
Full Abstraction

Centralized implementation fully abstract:

Computing environment: $\text{CE}[-] = Proxy \mid W \mid -$

$I \models P \cong^\pi Q \iff [I] \models \text{CE}[[P]] \cong^{A\pi} \text{CE}[[Q]]$
Full Abstraction

Centralized implementation fully abstract:

- Computing environment: \( \text{CE}[\_] = \text{Proxy} | W | \_ \)

\[
I \models P \cong^\pi Q \iff [I] \models \text{CE}[[P]] \cong^{A\pi} \text{CE}[[Q]]
\]

- long and difficult proof
- bisimulation-based techniques
- main tool: notion \textit{administrative} steps first characterized then abstracted away
A distributed implementation

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

\[ S, T ::= \delta\{P\} \mid S \mid T \mid (\text{new } n : A)S \mid \text{stop} \]

- each domain mapped in a trusted proxy

- \( I \models S \cong^{\pi} T \) 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)
  
  \[
  S, T ::= \delta\{P\} \mid S \mid T \mid (\text{new } n : A)S \mid \text{stop}
  \]

- each domain mapped in a trusted proxy

- \(I \models S \simeq^\pi T\) closed under contexts using known domains

- proxies coordinate to create virtual single queue for channel manager

- distributed implementation fully abstract

  \[
  I \models S \simeq^\pi T \iff [I] \models \prod_{d \in \text{fd}(S,T)} \text{Proxy}_d \mid W \mid [S] \simeq^{A\pi} \prod_{d \in \text{fd}(S,T)} \text{Proxy}_d \mid W \mid [T]
  \]
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 desirable
- 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!