Mobile Access Control

Adriana Compagnoni
Stevens Institute of Technology

Joint work with
Elsa L Gunter (UI-UC)

Rutgers, February 3, 2006
Boxed Ambients

Network

Classroom

Instructor

Mail

Destination

Mailer
Boxed Ambients

Network

Classroom

Instructor

Mail

Destination

Mailer

msg
Boxed Ambients

Network

Classroom

Instructor

Mail

Destination

Mailer
Boxed Ambients
Boxed Ambients

Network

Classroom

Instructor

Destination

Mailer

Mail

route
Boxed Ambients

Network

Classroom

Instructor

Mailer

Destination

Mail

Mail

Instructor

Boxed Ambients
Boxed Ambients

Network

Classroom

Instructor

Destination

Mail

Mailer
Boxed Ambients

Network

Classroom

Instructor

Destination

Mail

Mailer

msg
Boxed Ambients

Network

Classroom

Student

Mail

Destination

Mailer
Boxed Ambients
Security is Necessary for Correct Functionality

- Embedded devices often need to receive data (and increasingly new code) from remote sources.
- If data (or new code) is corrupt, the functionality of the device is at risk.
- Need methods to verify security of communications.
CPAP Machines - Current Method

- Doctor sends you a Smart Card
- You insert the smart card into your machine
- When the machine is done interacting with the smart card, you take it out
- You mail the card back to the doctor
- The doctor places the smart card in his reader
- Security derives from the “belief” that the card is secure
- Networking is the way of the future
Boxed Ambients

Network

Patient1

Doctor

GetInfo

Patient2

CPAP

CPAP
Boxed Ambients

Network

Patient1

CPAP

Doctor

GetInfo

Patient2

CPAP
Boxed Ambients

Network

Patient1

CPAP

Doctor

Patient2

GetInfo

CPAP
Boxed Ambients

Network

Patient1

CPAP

Doctor

Patient2

GetInfo

CPAP
Boxed Ambients

Network

Patient1

CPAP

Doctor

Patient2

CPAP

GetInfo
Boxed Ambients

Network

Patient1

CPAP

Doctor

Patient2

GetInfo

CPAP
Boxed Ambients

Network

Patient1
CPAP

Doctor

GetInfo

Patient2
CPAP
Boxed Ambients

Network

Patient1

Doctor

GetInfo

Patient2

CPAP

CPAP
Privacy

How do we guarantee that only authorized agents access your CPAP?
Boxed Ambients with Security

Network

Patient1

GetInfo

CPAP

Doctor

Patient2

CPAP
Boxed Ambients with Security

Network

Patient1

CPAP

GetInfo

Patient2

CPAP

Doctor
Role-Based Access Control

- Separate control into roles for users and access privileges for roles
- Give one relation of users (and possibly active roles) to roles (that can be activated)
- Give separate relation of roles to privileges

- Access privileges: \( P : \text{Role set} \rightarrow \text{Acc set} \)
- User roles: \( UserPolicy : \text{User} \times \text{Role set} \rightarrow \text{Role set} \)
Local Role-Based Access Control

- Have a notion of a location (boxed ambient)
- Each ambient assigns privileges to the resources it controls:
  - Entry into itself
  - Read access to its channel
  - Write access to its channel
- $\text{Priv} : \text{Amb} \rightarrow \text{Role set} \times \text{Role set} \times \text{Role set}$
  
  enter \hspace{1cm} read \hspace{1cm} write
Ambient

- Assume set of (public) ambient names $Amb$
- Ambients given by:
  \[ A ::= m_u[P]@\rho \]
  - Where $m \in Amb$
  - $\rho \in Roles$ (active roles for that process)
  - $u \in Users$
  - $P$ is a Process
Processes (simplified)

- Similar to $\pi$-calculus
- $\eta ::= * \mid \uparrow c \mid \downarrow c$ (local | with parent | with child)
- $P ::= \text{nil} \mid (P_1 \mid P_2) \mid !P$
  | $\nu(n:\tau).P$ (creates a new ambient $n$)
  | $<M>\eta.P$ (send message $M$ on $\eta$)
  | $(x)\eta.P$ (receive message into $x$ on $\eta$)
  | activate($r$).$P$ (activate role $r$ for $P$)
  | deactivate($r$).$P$ (deactivate role $r$ for $P$)
  | $C(c).P$ (execute capability $C$, creating local channel $c$)

- Message is a capability or variable (containing a capability)
Capabilities

- Two main kinds of capabilities: communicating and non-communicating (quiet)

- Quiet Capabilities:
  \[ Q ::= \text{inQ} m \mid \text{outQ} m \mid \text{inQ} \mid \text{outQ} \]

- Capabilities:
  \[ C ::= \text{in} m \mid \text{out} m \mid \text{in} \mid \text{out} \mid Q \mid Q.C \]

- Capabilities are the content of messages (M) and actions of processes
Dynamic Semantics

- to activate or deactivate a role.

- to describe when one ambient may enter or exit another.

- to describe local communication, and communication across ambient boundaries.
Dynamic Semantics: activate

$$m_u[activate\ (r) .P]@\rho \rightarrow m_u[P]@(\rho \cup \{r\})$$

$$m_u[deactivate\ (r) .P]@\rho \rightarrow m_u[P]@(\rho - \{r\})$$
Dynamic Semantics

\[ <M>^*.P \mid (x)^*.R \rightarrow P\mid R \{x:=M\} \]

(local communication)

\[ m_u [M] \downarrow^c .P \mid n_v [x] \uparrow^c .R \rho_n] \rho_m \rightarrow m_u [P \mid n_v [R \{x:=M\}] \rho_n] \rho_m \]

(to child)

Similarly to parent
Dynamic Semantics: in

- The capabilities in $m (c_1)$ and in $(c_2)$ are consumed.
- $m$ and $n$ now share a new communication channel $c$. 
Dynamic Semantics: out

p_u[n_vmw[out p (c_1). P_1 | R_1]@\rho_m | R_2] @\rho_n
| out (c_2).P_2 | R_3] @\rho_p

→

p_u[n_v[R_2] @\rho_n | m_w[P_1{c_1 := c} | R_1]@\rho_m
| P_2 {c_2 := c} | R_3] @\rho_p

-The capabilities out p (c_1) and out (c_2) are consumed.
- m and p now share a new communication channel c.
Our Type System prevents two forms of security violations:

» Attempting to enter an ambient without proper authorization, and

» Attempting to read from or write to channels without the corresponding permissions.
What can we do statically?

- Give static types to channels and ambients
- Ambient types: \( \tau :: = \text{amb} (\rho_{\text{in}}, \sigma) \)
- Channel types: \( \sigma :: = (\rho_r, \rho_w, \tau) \mid \text{ssh} \)
- Being in \( \rho_{\text{in}} \) guarantees you can enter the ambient
- Being in \( \rho_r \) guarantees you can read from the channel
- Being in \( \rho_w \) guarantees you can write to the channel
- \text{shh} means you cannot read or write to the channel
Typing Judgements

\[ \Gamma, \rho_{\text{here}}, \rho_{\text{deact}}, m, u \vdash P: \rho_{\text{act}} \]

Where

» P is a process
» m is the enclosing ambient
» u is the user that owns m
» \( \rho_{\text{here}} \) is the set of roles authorizing P to be in m
» \( \rho_{\text{deact}} \) is the set of roles that P can deactivate
» \( \rho_{\text{act}} \) is the set of currently active roles.
» \( \Gamma \) typing environment for message identifiers and channel names
Typing Judgements

- Other typing judgements have similar forms.

- The typing judgement for actions reflect how the different role sets are modified.

\[
\Gamma, \rho_{\text{here}}, \rho_{\text{deact}}, \rho_{\text{act}}, m, u \vdash \text{a : } (\Gamma, \rho_{\text{here}}, \rho_{\text{act}})
\]
Typing Rules: Role Activation

\[ r \in U(u, \rho_{act}) \]
\[ \Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u \vdash \text{activate}(r): (\Gamma, \rho_{here}, \rho_{act} \cup \{r\}) \]

\[ r \not\in \rho_{deact} \quad (\rho_{act} - \{r\} - \rho_{deact}) \cap \rho_{here} \neq \emptyset \]
\[ \Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u \vdash \text{deactivate}(r): (\Gamma, \rho_{here}, \rho_{act} - \{r\}) \]
### Typing Rules: Data Exchange

**Input**

\[
\Gamma,m \vdash \eta : (\rho_r, \rho_w, \tau) \\
(\rho_{act} - \rho_{deact}) \cap \rho_r \neq \emptyset
\]

\[
\Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u \vdash (x)^\eta : (\Gamma + x: \tau, \rho_{here}, \rho_{act})
\]

**Output**

\[
\Gamma,m \vdash \eta : (\rho_r, \rho_w, \tau) \\
(\rho_{act} - \rho_{deact}) \cap \rho_w \neq \emptyset \\
\Gamma \vdash M : \tau
\]

\[
\Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u \vdash <M>^\eta : (\Gamma, \rho_{here}, \rho_{act})
\]
Type Rules: Entrance

In

\( \Gamma(n) = \text{amb}(\rho_{in}, \sigma) \)

\((\rho_{act} - \rho_{deact}) \cap \rho_{in} \neq \emptyset\)

\(\Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u |- \text{in} \ n \ (c) : (\Gamma + c : \sigma, \rho_{in}, \rho_{act})\)

Co-in

\(\Gamma(m) = \text{amb}(\rho_{in}, \sigma)\)

\(\Gamma, \rho_{here}, \rho_{deact}, \rho_{act}, m, u |- \text{in}(c) : (\Gamma + c : \sigma, \rho_{here}, \rho_{act})\)
Example

- Previous example can now work:
- Give members of doctor’s office the *doctor* role
- Patient allows GetInfo procedures with *doctor* role to enter, but not GetInfo procedures from other *patients*
- Patients can’t (in general) activate the *doctor* role
CPAP Example

- No matter how we specify types for the ambients, the Patient1 GetInfo process will not type check if it requests to enter Patient2

- We can find types that allow the Doctor GetInfo program to type check
Results

- We defined an un-typed and a typed (not shown) transitional semantics.
- We show that on well-typed processes both transitional semantics coincide.
- The typed transitional semantics is of independent interest, and it is relevant to situations where the access control policy is only known at runtime.
Future Work

- Trusted and untrusted locations
- Role hierarchies
- Subtyping: Can a more (or less) restrictive type be used than the one given?
- Multiple channels between communicating ambients
- Design a programming language based on this calculus
Related Work

- Bonelli, Compagnoni, Dezani, and Garralda (MFCS04)
  - The calculus splits communication and mobility by using ambient names and port names.

- Braghin, Gorla, and Sassone (CSFW04)
  - They develop a type system for statically (and dynamically) checking code in the $\pi$-calculus with roles.

- Hennessy (TGC05)
  - Type system for the $D\pi$-calculus
  - Uses dependent types to allow privileges to vary by the message received
  - No nesting of different user code or locations
  - No movement of locations, only code
Contributions

- We defined a boxed ambient calculus with Distributed Role-Based Access Control, where the privileges associated to processes change during computation.
- Privileges depend on location, owner, activated roles, and security policy.
- First calculus with distributed RBAC mechanism where the location of a process conditions its ability to move and communicate.
Thank You!