Role-based Access Control in a Mobile Environment

Elsa L Gunter (UI-UC)

Joint work with
Adriana Compagnoni (Stevens)
Pablo Garralda (Stevens)
Boxed Ambients

Network

Classroom

Instructor

Mail

Destination

Mailer
Boxed Ambients

Network

Classroom

Instructor

Destination

Mailer

Mail
Boxed Ambients

Network

Classroom

Instructor

Destination

Mailer

Mail
Boxed Ambients

Network

Classroom

Student

Mail

Destination

Mailer
Boxed Ambients

Network

Classroom

Student

Mail

msg

Destination

Mailer
Boxed Ambients

Network

Classroom

Student

Class Resp

Destination

Mailer
Security in 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 device is at risk
- Need methods to verify security of communications
CPAP Machines - Current Method

- Doctor send you a Smart Card
- You insert the smart card into your machine
- When the machine is done intercating 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
- CPAP
- Patient2
- CPAP
- Doctor
- GetInfo
Boxed Ambients

Network

Patient1

CPAP

Doctor

GetInfo

Patient2

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

CPAP

GetInfo

Doctor

Patient2

CPAP

CPAP
Boxed Ambients

Network

Patient1

Doctor

GetInfo

Patient2

CPAP

CPAP
Boxed Ambients

Network

Patient1

CPAP

Doctor

GetInfo

Patient2

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

- Patient1
  - GetInfo
  - CPAP

- Doctor

- Patient2
  - CPAP

- Network
Boxed Ambients with Security

Network

Patient1

GetInfo

CPAP

Doctor

Patient2

CPAP
Boxed Ambients with Security

Network

Patient1

CPAP

Doctor

GetInfo

Patient2

CPAP
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: \( U : \text{User} \times \text{Role set} \rightarrow \text{Role set} \)
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
- $Priv : Amb \rightarrow \text{Role set} \times \text{Role set} \times \text{Role set}$
  - enter
  - read
  - write
Ambient

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

- Similar to $\pi$-calculus
- $P ::= A \mid \textbf{nil} \mid (P_1 \mid P_2) \mid !P$
  - $\nu(n:\tau) P$ (creates a new ambient)
  - $c<M>.P$ (send message $M$ on channel $c$)
  - $c(x).P$ (receive message into $x$ on $c$)
  - 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

- Give Dynamic Semantics to describe when one ambient may enter, exit, or communicate with another.
- Requires run-time access to role-based access policy
Dynamic Semantics

- $\Gamma$ maps ids to types
- $m_u \ [\text{activate} \ (r) \ . \ P@\rho], \ \Gamma \rightarrow m_u \ [P \ @ (\rho \cup \ \{ \ r \} )], \ \Gamma$
  if $r \in U (u, \rho)$
- $m_u \ [\text{deactivate} \ (r) \ . \ P@\rho], \ \Gamma \rightarrow m_u \ [P \ @ (\rho \ - \ \{ \ r \} )], \ \Gamma$
- $m_u \ [\nu(n: \tau) \ . \ P \ @ \rho], \ \Gamma \rightarrow m_u \ [P \ @ \rho], \ \Gamma + (n: \tau)$
- $m_u \ [c<M> \ . \ P_1@\rho_1 \ \parallel \ c(x) \ . \ P_2@\rho_2], \ \Gamma$
  $\rightarrow$
  $m_u \ [P_1@\rho_1 \ \parallel \ P_2\{x:=M}@\rho_2], \ \Gamma$
  if $\Gamma(c) = (\rho_r, \rho_w)$ and $\rho_r \cap \rho_2 \neq \emptyset$ and $\rho_w \cap \rho_1 \neq \emptyset$
Dynamic Semantics: in

- \( n_u \[ \text{in } m \ (c_1).P_1@\rho_1 \sqcup R_1 \] @\rho_n \)
- \( \vdash m_v \[ \text{in } (c_2).P_2@\rho_2 \sqcup R_2 \] @\rho_m \ , \Gamma \)

\[ \rightarrow \]

\[ m_u[n_v[P_1\{c_1:=c\}@\rho_1 \sqcup R_1]@\rho_n \]
\[ \vdash P_2\{c_2:=c\}@\rho_2 \sqcup R_2]@\rho_m \ , \Gamma + (c : (\rho'_r \cap \rho''_r, \rho_w)) \]

where \( \Gamma(m) = (\rho'_m, \rho'_r, \rho'_w) \) and \( \Gamma(n) = (\rho''_m, \rho''_r, \rho''_w) \)

- in\( Q \), same but update \( \Gamma + (c : (\{ \}, \{ \})) \) (aka shh)
Dynamic Semantics: out

\[ \rho_u[n_v[m_w[\text{out } p (c_1). P_1@\rho_1 \triangleright R_1] \ @\rho_m \triangleright R_2] \ @\rho_n \]
\[ \triangleright \text{out } (c_2). P_2@\rho_2 \triangleright R_3] \ @\rho_p , \Gamma \]

\[ \rightarrow \]

\[ \rho_u[n_v[R_2] \ @\rho_n \ ] m_w[P_1\{c_1:= c\}@\rho_1 \triangleright R_1] \ @\rho_m \]
\[ \triangleright P_2\{c_2:=c\}@\rho_2 \triangleright R_3] \ @\rho_p , \Gamma + (c : (\rho'_r \cap \rho''_r, \rho_w)) \]

where \( \Gamma(m) = (\rho'_\text{in}, \rho'_r, \rho'_w) \) and \( \Gamma(p) = (\rho''_\text{in}, \rho''_r, \rho''_w) \)

- \( \text{outQ same, but update } \Gamma + (c : \text{shh}) \)

- Nested Ambients: \( m_u[R_1] \rightarrow m_u[R'_1] \) if \( R_1 \rightarrow R'_1 \)
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
What can we do statically?

- Give static types to channels and ambients
- Ambient types: \( \sigma ::= (\rho_{\text{in}}, \tau) \)
- Channel types: \( \tau ::= (\rho_r, \rho_w, \sigma) \mid \text{ssh} \)
- 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{ssh} means you cannot read or write to the channel
Static Semantics

- Develop type system that checks if access to local resources is authorized
- If process statically type checks, **activate** and **deactivate** may be ignored at runtime
- Requires complete static knowledge of local role-based access control policy
Some Typing Rules

- $\Gamma$ assigns types to channels and ambients
- Roled processes:
  \[
  \frac{\Gamma, u, \rho |- P}{\Gamma, u |- P \@ \rho}
  \]

  \[
  \frac{\Gamma, u |- R_1 \quad \Gamma, u |- R_2}{\Gamma, u |- R_1 \| R_2}
  \]

- Processes:
  \[
  \frac{\Gamma, v |- R}{\Gamma, u, \rho |- \text{nil}}
  \]

  \[
  \frac{\Gamma, v |- R}{\Gamma, u, \rho |- m_v[R]}
  \]
Typing Rules: Role Activation

\[ r \in U (u, \rho) \quad \Gamma, u, \rho \cup \{r\} \vdash P \]

\[ \Gamma, u, \rho \vdash \text{activate}(r).P \]

\[ \Gamma, u, \rho \vdash P \]

\[ \Gamma, u, \rho - \{r\} \vdash \text{deactivate}(r).P \]
Typing Rules: Data Exchange

\[ \Gamma |- c: (\rho_r, \rho_w, \sigma) \quad \rho_r \cap \rho \neq \emptyset \]
\[ \Gamma+ (x':\sigma), u, \rho |- P\{x:=x'\} \quad x' \text{ fresh} \]
\[ \Gamma, u, \rho |- c(x).P \]

\[ \Gamma |- c: (\rho_r, \rho_w, \sigma) \quad \rho_w \cap \rho \neq \emptyset \]
\[ \Gamma, u, \rho |- P \quad \Gamma|- M : \sigma \]
\[ \Gamma, u, \rho |- c<M>.P \]
Type Rules: Entrance

\[
\begin{align*}
\Gamma & \vdash m : (\rho_{\text{in}}, \tau) \quad \rho_{\text{in}} \cap \rho \neq \emptyset \\
\Gamma+(c' : \tau), u, \rho & \vdash P\{c := c'\} \quad c' \text{ fresh} \\
\hline
\Gamma & \vdash \text{in } m (c).P@\rho \\
\end{align*}
\]

\[
\begin{align*}
\Gamma & \vdash m : (\rho_{\text{in}}, \tau) \quad \rho_{\text{in}} \cap \rho \neq \emptyset \\
\Gamma+(c' : \text{ssh}), u, \rho & \vdash P\{c := c'\} \quad c' \text{ fresh} \\
\hline
\Gamma & \vdash \text{inQ } m (c).P@\rho \\
\end{align*}
\]
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
Limitations

- Coming up with the types for the ambients is difficult
  - They go in the code making writing code infeasible
- The size of the types quickly becomes enormous
- The type of a channel is fixed for all communication
  - Some form of session types seems possible
  - Would make the types in code even worse
Limitations

- The types have a finite nature, bounding the level of indirect learning from a channel
  - Some natural examples that are fine don’t type check
  - A form of recursive type might help here
  - Simplest solution: same type for channel and data
    - $\sigma = (\rho_{in}, (\rho_r, \rho_w, \sigma))$
    - Type system makes sense and can handle at least one example above can’t, but tends to force all ambients to have the same type
Future Work

- Subtyping: Can a more (or less) restrictive type be used than the one given?
  - Covariant / contravariant problem
- Support for “open” (delivering running code)
- Multiple channels between communicating ambients
- Implement on top of the Tunnel Calculus (Carl Gunter and Alwyn Goodloe)
Related Work

- Braghin, Gorla, and Sassone (CSFW04)
  - They develop a type system for statically (and dynamically) checking code in the $\pi$-calculus
  - Their type system has similar finitary limitation

- 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
  - No movement of locations, only code
  - Has residual run-time type checks