# Boundary Inference for Enforcing Security Policies in Mobile Ambients

(Proceedings of IFIP-TCS'02, pp. 383-395, 2002)

Chiara Braghin, Agostino Cortesi, Riccardo Focardi[†], Steffen van Bakel

Dipartimento di Informatica, Università Ca' Foscari di Venezia, Via Torino 155, 30173 Venezia – Mestre (Italy),
Department of Computing, Imperial College, 180 Queen s Gate, London SW7 2BZ, (UK)
{braghin,cortesi,focardi}@dsi.unive.it svb@doc.ic.ac.uk

**Abstract**

The notion of "boundary ambient" has been recently introduced to model multilevel security policies in the scenario of mobile systems, within pure Mobile Ambients calculus. Information flow is defined in terms of the possibility for a confidential ambient/data to move outside a security boundary, and boundary crossings can be captured through a suitable Control Flow Analysis. We show that this approach can be further enhanced to infer which ambients should be "protected" to guarantee the lack of information leakage for a given process.

**keywords:** Mobile Ambients, Security, Static Analysis.

## 1 Introduction

A Trusted Computing Base is the set of protection mechanisms within a computer system the combination of which is responsible for enforcing a security policy [1]. One of the main challenges faced when building a TCB is deciding which parts of the system are security-critical. Our focus is on *Multilevel Security*, a particular *Mandatory Access Control* security policy: every entity is bound to a security level (for simplicity, we consider only two levels: *high* and *low*), and information may just flow from the low level to the high one. Typically, two access rules are imposed: $(i)$ *No Read Up*, a low level entity cannot access information of a high level entity; $(ii)$*No Write Down*, a high level entity cannot leak information to a low level entity.

In order to detect information leakages, a typical approach (see, e.g., [2, 8, 9, 10, 12, 13]) consists in directly defining what is an information flow from one level to another one. Then it is sufficient to verify that, in any system execution, no flow of information is possible from level *high* to level *low*. This is the approach we follow also in this paper.

To model information flow security, we adopt the scenario of mobile systems. This particular setting, where code may migrate from one security level to another one, complicates even further the problem of capturing all the possible information leakages. As an example, confidential data may be read by an authorized agent which, moving around, could expose them to unexpected attacks. Moreover, the code itself could be confidential, and so not allowed to be read/executed by lower levels.

[†] Work partially supported by MURST Projects "Interpretazione Astratta, Type Systems e Analisi Control-Flow", and MEFISTO, and EU Contract IST-2001-32617.

In order to study this problem in an as abstract manner as possible, we consider the "pure" Mobile Ambients calculus [5], in which no communication channels are present and the only possible actions are represented by the moves performed by mobile processes. This allows the study of a very general notion of information flow which should be applicable also to more "concrete" versions of the calculus.

The information flow property of interest is defined in terms of the possibility for a confidential ambient/data to move outside a *security boundary*. In [6], a very simple syntactic property is introduced that it is sufficient to imply the absence of unwanted information flow. In [3], a refinement of the control flow analysis defined in [11] is introduced that deals with the same property with improved accuracy.

As an example, consider two different sites *venice* and *montreal*, each with some set of confidential information that need to be protected. This can be modeled by just defining two boundary ambients, one for each site:

$$venice^b[\![\, P_1 \,]\!] \mid montreal^b[\![\, P_2 \,]\!] \mid Q^\ell,$$

where $Q$ is an untrusted process. In order to make the model applicable, a mechanism for moving confidential data from one boundary to another one is certainly needed. This is achieved through another boundary ambient which moves out from the first protected area and into the second one. In the example, label $b$ denotes a boundary, $h$ a high-level ambient, $\ell$ a low-level ambient and $c$ a capability. Consider the example depicted in Figure 1. Process

$$venice^b[\![\, send^b[\![\, \mathbf{out}^c\, venice.\mathbf{in}^c\, montreal \,]\!] \mid hdata^h[\![\, \mathbf{in}^c\, send \,]\!] \,]\!] \mid montreal^b[\![\, \mathbf{open}^c\, send \,]\!] \mid Q^\ell$$

may evolve to (step (b))

$$venice^b[\![\,]\!] \mid send^b[\![\, \mathbf{in}^c\, montreal \mid hdata^h[\![\,]\!] \,]\!] \mid montreal^b[\![\, \mathbf{open}^c\, send \,]\!] \mid Q^\ell$$

then to (step (c))

$$venice^b[\![\,]\!] \mid montreal^b[\![\, \mathbf{open}^c\, send \mid send^b[\![\, hdata^h[\![\,]\!] \,]\!] \,]\!] \mid Q^\ell$$

and finally to

$$venice^b[\![\,]\!] \mid montreal^b[\![\, hdata^h[\![\,]\!] \,]\!] \mid Q^\ell$$

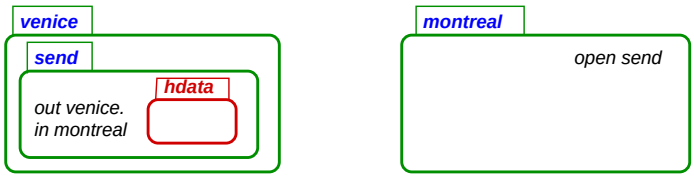
Note that *send* is labeled as a boundary ambient. Thus, the high level data *hdata* is always protected by boundary ambients, during the whole execution.

The analysis developed in [3] allows to verify that no leakage of secret data/ambients outside the boundary ambients is possible. When applied to this example, it shows that $h$ is always contained inside $b$, i.e., a boundary ambient. This basically proves that the system is secure and no leakage of $h$ data may happen.

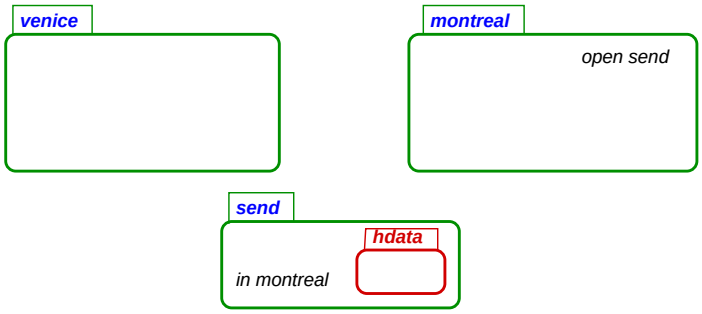
In this paper we are interested in merging these ideas towards the definition of a TCB, to a more ambitious perspective: which are the ambients that should be labeled "boundary", to guarantee that the system is secure, i.e. that no $h$ data may fall into an unprotected environment? Is there always a solution to this problem? Is there a minimal solution?

We show that these problems can be properly addressed by re-executing the Control Flow Analysis presented in [3]. A successful analysis infers boundary ambients until a fixed point is reached, returning the set of ambients that should be "protected".

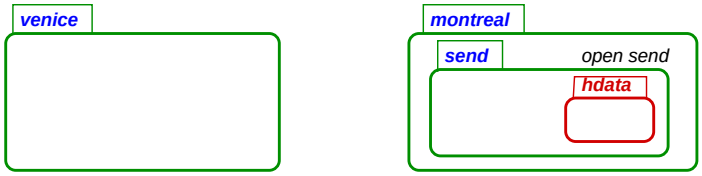
In the example above, all we know is that *hdata* is information that must be protected during the whole execution of the process; thus, a successful analysis should infer *venice*, *montreal* and *send* as ambients to be labeled "boundary".



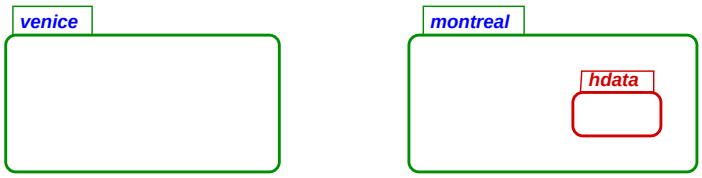

(a) *venice* needs to send confidential data *hdata* to *montreal*.




(b) The confidential data is sent inside the secure "envelope"*send*.




(c) The confidential data safely arrive in *montreal*.




(d) The envelope is dissolved to allow confidential data to be accessed in *montreal*.

Figure 1: Venice and Montreal exchange confidential information.

The rest of the paper is organized as follows. In Section 2 we introduce the basic terminology on ambient calculus, then we present the model of multilevel security for mobile agents and we show how to guarantee absence of unwanted information flows through the control flow analysis of [3]. In Section 3, we introduce the enhanced Control Flow Analysis. Section 4 concludes the paper.

## 2 Background

In this section we introduce the basic terminology on ambient calculus on multilevel security and we briefly recall the control flow analysis defined in [3].

### 2.1 Mobile Ambients

The Mobile Ambients calculus has been introduced in [5] with the main purpose of explicitly modeling mobility. Indeed, ambients are arbitrarily nested boundaries which can move around through suitable capabilities. The syntax of processes is given as follows, where $n$ denotes an ambient name.

| | | | |
|---|---|---|---|
| $P, Q$ | ::= | $(\nu n)P$ | restriction |
| | \| | $\mathbf{0}$ | inactivity |
| | \| | $P \mid Q$ | composition |
| | \| | $!P$ | replication |
| | \| | $n^{\ell^a}[\![ P ]\!]$ | ambient |
| | \| | $\mathbf{in}^{\ell^t} n.P$ | capability to enter $n$ |
| | \| | $\mathbf{out}^{\ell^t} n.P$ | capability to exit $n$ |
| | \| | $\mathbf{open}^{\ell^t} n.P$ | capability to open $n$ |

Labels $\ell^a \in \mathbf{Lab}^a$ on ambients and labels $\ell^t \in \mathbf{Lab}^t$ on transitions (capabilities), have been introduced in the control flow analysis proposed in [11]. This is just a way of indicating "program points" and will be useful in the next section when developing the analysis.

Intuitively, the restriction $(\nu n)P$ introduces the new name $n$ and limits its scope to $P$; process $\mathbf{0}$ does nothing; $P \mid Q$ is $P$ and $Q$ running in parallel; replication provides recursion and iteration as $!P$ represents any number of copies of $P$ in parallel. By $n^{\ell^a}[\![ P ]\!]$ we denote the ambient named $n$ with the process $P$ running inside it. The capabilities $\mathbf{in}^{\ell^t} n$ and $\mathbf{out}^{\ell^t} n$ move their enclosing ambients in and out ambient $n$, respectively; the capability $\mathbf{open}^{\ell^t} n$ is used to dissolve the boundary of a sibling ambient $n$. The operational semantics [5] of a process $P$ is given through a suitable reduction relation $\rightarrow$ and a structural congruence $\equiv$ between processes. Intuitively, $P \rightarrow Q$ represents the possibility for $P$ of reducing to $Q$ through some computation.

## 2.2 Modeling Multilevel Security

In order to define Multilevel security in Mobile Ambients we first need to classify information into different levels of confidentiality. We do that by exploiting the labeling of ambients. In particular, the set of ambient labels $\mathbf{Lab}^a$ will be partitioned into three mutually disjoint sets $\mathbf{Lab}^a_H, \mathbf{Lab}^a_L$ and $\mathbf{Lab}^a_B$, which stand for *high*, *low* and *boundary* labels. We denote by $\mathcal{L}$ the triplet $(\mathbf{Lab}^a_H, \mathbf{Lab}^a_L, \mathbf{Lab}^a_B)$.

Given a process, the multilevel security policy may be established by deciding which ambients are the ones responsible for confining confidential information. These will be labeled with boundary labels from set $\mathbf{Lab}^a_B$ and we will refer to them as *boundary ambients*. Thus, all the high level ambients must be contained in a boundary ambient, and labeled with labels from set $\mathbf{Lab}^a_H$. On the other side, all the external ambients are considered low level ones and consequently labeled with labels from set $\mathbf{Lab}^a_L$. This is how we will always label processes, and corresponds to defining the security policy (what is secret, what is not, what is a container of secrets). In all the examples, we will use the following notation for labels: $b \in \mathbf{Lab}^a_B, h \in \mathbf{Lab}^a_H$, $m, m' \in \mathbf{Lab}^a_L$ and $c, ch, cm, cm' \in \mathbf{Lab}^t$.

In [3] we introduced a refinement of the Control Flow Analysis of [11], in order to incorporate the ideas above, thus yielding to a more accurate tool for detecting unwanted boundary crossings. The main idea is to keep information about the nesting of boundaries, and about "unprotected" ambients.

**Definition 2.1** The refined control flow analysis works on triplet $(\hat{I}_B, \hat{I}_E, \hat{H})$, where:

$(\hat{I}_B)$: The first component is an element of $\wp(\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$. If a process contains either a capability or an ambient labeled $\ell$ inside an ambient labeled $\ell^a$ which is a boundary or an ambient nested inside a boundary (referred as *protected ambient*) then $(\ell^a, \ell)$ is expected

$$
\begin{array}{lll}
 & \beta^{\mathcal{L}}(P) & = \beta^{\mathcal{L}}_{env,False}(P) \\
(res) & \beta^{\mathcal{L}}_{\ell,Proct}((\nu n)P) & = \beta^{\mathcal{L}}_{\ell,Proct}(P) \\
(zero) & \beta^{\mathcal{L}}_{\ell,Proct}(\mathbf{0}) & = (\emptyset,\emptyset,\emptyset) \\
(par) & \beta^{\mathcal{L}}_{\ell,Proct}(P \mid Q) & = \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup \beta^{\mathcal{L}}_{\ell,Proct}(Q) \\
(repl) & \beta^{\mathcal{L}}_{\ell,Proct}(!P) & = \beta^{\mathcal{L}}_{\ell,Proct}(P) \\
(amb) & \beta^{\mathcal{L}}_{\ell,Proct}(n^{\ell^a}[\![P]\!]) & = \text{case } Proct \text{ of} \\
 & & \quad \text{True} : \beta^{\mathcal{L}}_{\ell^a,Proct}(P) \sqcup (\{(\ell,\ell^a)\},\emptyset,\{(\ell^a,n)\}) \\
 & & \quad \text{False: if } (\ell_a \in \mathbf{Lab}^a_B) \text{ then} \\
 & & \qquad \text{let } Proct' = \text{True else } Proct' = \text{False in} \\
 & & \qquad \beta^{\mathcal{L}}_{\ell^a,Proct'}(P) \sqcup (\emptyset,\{(\ell,\ell^a)\},\{(\ell^a,n)\}) \\
(\mathbf{in}) & \beta^{\mathcal{L}}_{\ell,Proct}(\mathbf{in}^{\ell^t} n.P) & = \text{case } Proct \text{ of} \\
 & & \quad \text{True} : \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\{(\ell,\ell^t)\},\emptyset,\emptyset) \\
 & & \quad \text{False: } \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\emptyset,\{(\ell,\ell^t)\},\emptyset) \\
(\mathbf{out}) & \beta^{\mathcal{L}}_{\ell,Proct}(\mathbf{out}^{\ell^t} n.P) & = \text{case } Proct \text{ of} \\
 & & \quad \text{True} : \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\{(\ell,\ell^t)\},\emptyset,\emptyset) \\
 & & \quad \text{False: } \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\emptyset,\{(\ell,\ell^t)\},\emptyset) \\
(\mathbf{open}) & \beta^{\mathcal{L}}_{\ell,Proct}(\mathbf{open}^{\ell^t} n.P) & = \text{case } Proct \text{ of} \\
 & & \quad \text{True} : \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\{(\ell,\ell^t)\},\emptyset,\emptyset) \\
 & & \quad \text{False: } \beta^{\mathcal{L}}_{\ell,Proct}(P) \sqcup (\emptyset,\{(\ell,\ell^t)\},\emptyset)
\end{array}
$$

Figure 2: Representation Function for the refined Control Flow Analysis

to belong to $\hat{I}_B$. As long as high level data is contained inside a protected ambient there is no unwanted information flow.

$(\hat{I}_E)$: The second component is also an element of $\wp(\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$. If a process contains either a capability or an ambient labeled $\ell$ inside an ambient labeled $\ell^a$ which is not protected, then $(\ell^a,\ell)$ is expected to belong to $\hat{I}_E$.

$(\hat{H})$: The third component keeps track of the correspondence between names and labels. If a process contains an ambient labeled $\ell^a$ with name $n$, then $(\ell^a,n)$ is expected to belong to $\hat{H}$.

The analysis is defined by a representation function and a specification, like in [11]. They are depicted, respectively, in Figure 2 and Figure 3, in which we consider a process $P_*$ executing at the top-level environment labeled *env*.

Observe that within the specification of the analysis (depicted in Figure 3), some predicates are used to enhance readability, namely

- $path_B(\ell^a,\ell) = \begin{cases} \text{True} & \text{if } \ell^a = \ell \vee \exists \ell_1,\ell_2,\ldots,\ell_n \notin \mathbf{Lab}^a_B : n \geq 0 \wedge \\ & (\ell^a,\ell_1),(\ell_1,\ell_2),\ldots,(\ell_n,\ell) \in \hat{I}_B \ \wedge \ \ell^a,\ell \notin \mathbf{Lab}^a_B, \\ \text{False} & \text{otherwise.} \end{cases}$
- $path_E(\ell^a,\ell) = \begin{cases} \text{True} & \text{if } \ell^a = \ell \vee \exists \ell_1,\ell_2,\ldots,\ell_n \notin \mathbf{Lab}^a_B : n \geq 0 \wedge \\ & (\ell^a,\ell_1),(\ell_1,\ell_2),\ldots,(\ell_n,\ell) \in \hat{I}_E \ \wedge \ \ell^a,\ell \notin \mathbf{Lab}^a_B, \\ \text{False} & \text{otherwise.} \end{cases}$

$$
\begin{array}{lll}
(\mathit{res}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} (\nu n)P & \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \\
(\mathit{zero}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} \mathbf{0} & \text{always} \\
(\mathit{par}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \mid Q & \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \ \wedge\ (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} Q \\
(\mathit{repl}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} !P & \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \\
(\mathit{amb}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} n^{\ell^a}[\![P]\!] & \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \\
(\mathbf{in}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} \mathbf{in}^{\ell^t} n.P & \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \wedge \\
& \forall \ell^a,\ell^{a'},\ell^{a''} \in \mathbf{Lab}^a : & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_B \ \wedge\ (\ell^{a'},n) \in \hat{H}) & \\
& \quad \Longrightarrow\ (\ell^{a'},\ell^a) \in \hat{I}_B & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \in \mathbf{Lab}^a_B & \\
& \qquad \wedge\ (\ell^{a'},n) \in \hat{H}) \Longrightarrow & \\
& \quad \text{if } (\ \ell^{a'} \in \mathbf{Lab}^a_B) \text{ then } (\ell^{a'},\ell^a) \in \hat{I}_B & \\
& \quad \text{else } (\ell^{a'},\ell^a) \in \hat{I}_E & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ (\ell^{a'},n) \in \hat{H}) \Longrightarrow & \\
& \quad \text{if } (\ \ell^{a'} \in \mathbf{Lab}^a_B) & \\
& \qquad \text{then } (\ell^{a'},\ell^a) \in \hat{I}_B \ \wedge\ \{(\ell,\ell') \in \hat{I}_E \mid \mathit{path}_E(\ell^a,\ell)\} \subseteq \hat{I}_B & \\
& \quad \text{else } (\ell^{a'},\ell^a) \in \hat{I}_E & \\
(\mathbf{out}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} \mathbf{out}^{\ell^t} n.P \quad \text{iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \wedge & \\
& \forall \ell^a,\ell^{a'},\ell^{a''} \in \mathbf{Lab}^a : & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_E \cup \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E & \\
& \qquad \wedge\ (\ell^{a'},n) \in \hat{H}) \Longrightarrow & \\
& \quad \text{if } (\ \ell^a \in \mathbf{Lab}^a_B) \text{ then } (\ell^{a''},\ell^a) \in \hat{I}_E & \\
& \quad \text{else } (\ell^{a''},\ell^a) \in \hat{I}_E \ \wedge\ \{(\ell,\ell') \in \hat{I}_B \mid \mathit{path}_B(\ell^a,\ell)\} \subseteq \hat{I}_E & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_B \ \wedge\ (\ell^{a'},n) \in \hat{H}) & \\
& \quad \Longrightarrow\ (\ell^{a''},\ell^a) \in \hat{I}_B & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ (\ell^{a'},n) \in \hat{H}) & \\
& \quad \Longrightarrow\ (\ell^{a''},\ell^a) \in \hat{I}_E & \\
(\mathbf{open}) & (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} \mathbf{open}^{\ell^t} n.P \ \text{ iff } (\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathcal{L}} P \wedge & \\
& \forall \ell^a,\ell^{a'} \in \mathbf{Lab}^a : & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^a,\ell^{a'}) \in \hat{I}_E \ \wedge\ (\ell^{a'},n) \in \hat{H}) \Longrightarrow & \\
& \quad \text{if } (\ \ell^{a'} \in \mathbf{Lab}^a_B) \text{ then } \left\{(\ell^a,\ell^{a''}) \mid (\ell^{a'},\ell^{a''}) \in \hat{I}_B\right\} \subseteq \hat{I}_E \wedge & \\
& \qquad \left\{(\ell,\ell') \mid (\ell,\ell') \in \hat{I}_B \ \wedge\ (\ell^{a'},\ell'') \in \hat{I}_B \ \wedge\ \mathit{path}_B(\ell'',\ell)\right\} \subseteq \hat{I}_E & \\
& \quad \text{else } \left\{(\ell^a,\ell) \mid (\ell^{a'},\ell) \in \hat{I}_E\right\} \subseteq \hat{I}_E & \\
& \text{case } ((\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^a,\ell^{a'}) \in \hat{I}_B \ \wedge\ (\ell^{a'},n) \in \hat{H}) & \\
& \quad \Longrightarrow \left\{(\ell^a,\ell) \mid (\ell^{a'},\ell) \in \hat{I}_B\right\} \subseteq \hat{I}_B &
\end{array}
$$

Figure 3: Specification of the Control Flow Analysis

The representation function maps processes to their abstract representation, i.e. a triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ representing process $P_*$.

*Example 2.2* Let $P$ be a process of the form: $P = n^{\ell_1^a}[\, m^{\ell_2^a}[\, \mathbf{out}^{\ell^t}\, n\,]\,]$, with $\ell_1^a \in \mathbf{Lab}_B^a$ and $\ell_2^a \in \mathbf{Lab}_L^a$, thus the representation function of $P$ is the following: $\beta^{\mathcal{L}}(P) = (\{(\ell_1^a, \ell_2^a), (\ell_2^a, \ell^t)\}, \{(env, \ell_1^a)\}, \{(\ell_1^a, n), (\ell_2^a, m)$
.

The specification of the analysis amounts to recursive checks of subprocesses, which provide constraints that the triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ should satisfy in order to be a correct solution for the analysis. It is possible to prove that a least solution of this analysis exists and it may be computed as follows: first apply the representation function to the process $P_*$, then apply the analysis to validate the correctness of the proposed solution, adding, if needed, new information to the triplet until a fixed point is reached.

*Example 2.3* Let $P$ be the process of Example 2.2. The least solution of $P$ is the triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ where $\hat{I_B} = \{(\ell_1^a, \ell_2^a), (\ell_2^a, \ell^t)\}$, $\hat{I_E} = \{(env, \ell_1^a), (env, \ell_2^a), (\ell_2^a, \ell^t)\}$, and $\hat{H} = \{(\ell_1^a, n), (\ell_2^a, m)\}$. Observe that $(\hat{I_B}, \hat{I_E}, \hat{H})$ strictly contains $\beta^{\mathcal{L}}(P)$, as expected being $(\hat{I_B}, \hat{I_E}, \hat{H})$ a safe approximation.

More formally, the fixed point algorithm works as follows:

[Algorithm 2.4 F ixed Point Algorithm]
**Input:** a process $P_*$ and a partition labeling $\mathcal{L}$.

(i) Apply the representation function $\beta^{\mathcal{L}}$ to process $P_*$ to get a triplet $(\hat{I_B^o}, \hat{I_E^o}, \hat{H})$;

(ii) for all the constraints of the specification of the analysis, validate the triplet $(\hat{I_B^i}, \hat{I_E^i}, \hat{H})$ generated in (i):
   *a*) if the constraint is satisfied, continue;
   *b*) else, in case the constraint is not satisfied, this is due to the fact that either $\hat{I_B}$ or $\hat{I_E}$ do not consider nestings that may actually occur. In this case, modify $\hat{I_B}$ and $\hat{I_E}$ by adding the "missing" pairs, thus getting a new triplet $(I_B^{\hat{i+1}}, I_E^{\hat{i+1}}, \hat{H})$. Then, go back to (ii) with $i = i + 1$.

The iterative procedure above computes the least solution independent of the iteration order. The result of the analysis should be read, as expected, in terms of information flows.

**Theorem 2.5** *No leakage of secret data/ambients outside the boundary ambients is possible if in the analysis no high level label appears in* $\hat{I_E}$.

*Example 2.6* Consider, for instance, a process, which allows an *application* (say, an applet) to be downloaded from the *web* within *montreal*; then, the application may open the ambient *send* and disappear.

$$P_4 = venice^{b_1}[\, send^{b_3}[\, \mathbf{out}^c venice.\mathbf{in}^c montreal \mid hdata^h\ [\, \mathbf{in}^{ch} filter]\,]\ \mid$$
$$\mid download^{m'}\ [\, \mathbf{out}^{cm'} venice.\mathbf{in}^{cm'} web.\mathbf{in}^{cm'} montreal]\,]\ \mid$$
$$\mid montreal^{b_2}[\, \mathbf{open}^c web.\mathbf{open}^c application]\ \mid$$
$$web^m[\, application^m[\, \mathbf{open}^{cm} send.filter^m[\,]\,]\ \mid \mathbf{open}^{cm} download]$$

In this case, there is no information flow, as the application is not exporting any data out of

the *montreal* boundary. In this case, the refined CFA yields to positive information, namely:

$$\begin{aligned}
\hat{I}_B &= \{(b_1,b_3),(b_1,m'),(b_3,h),(b_3,c),(h,ch),(m',cm'),(b_2,b_3),(b_2,h),\\
&\quad (b_2,m'),(b_2,b_2),(b_2,m),(b_2,c),(b_2,cm'),(b_2,cm),(m,h),(m,m'),\\
&\quad (m,b_2),(m,m),(m,cm'),(m,cm)\}\\
\hat{I}_E &= \{(env,b_1),(env,b_3),(env,m'),(env,b_2),(env,m),(m',cm'),(m,m'),\\
&\quad (m,b_2),(m,m),(m,cm'),(m,cm)\}\\
\hat{H} &= \{(b_1,venice),(b_3,send),(b_2,montreal),(h,hdata),(m',download),\\
&\quad (m,web),(m,application),(m,filter)\}
\end{aligned}$$

Observe that the result is also better than the Hansen-Jensen-Nielsons's CFA [11] as the latter does not capture the fact that $h$ enters $m$ only after it has crossed the boundary and can never return back.

## 3 Inferring Boundaries

Let us turn now to the boundary inference issue. By now, we consider a process $P$ wherein high level data are known, i.e. $\mathbf{Lab}_H$ is fixed. We are interested to partition the set of ambient labels into $\mathbf{Lab}_L$ and $\mathbf{Lab}_B$ so that $\mathbf{Lab}_B$ is the minimal labeling that guarantees the absence of direct information flow concerning confidential data. In other words, the aim of the analysis is to detect which ambients among the "untrusted ones" should be protected (let's say by a firewall or by encryption) as they may carry sensitive data.

Since we want to infer a minimal set of boundary ambients it makes sense to discriminate all the ambients belonging to process $P_*$, thus we assume that initially all ambient occurrences have different labels. Note that this condition may not be verified during the execution of process $P$ because of the replication operator. Given this initial labeling, a label has at most one parent, thus we can give the following definitions.

**Definition 3.1** (Border of an ambient) Given an ambient with label $\ell$ in a process $P$, we denote by $\mathcal{B}(\ell)$ the border of the ambient $n$ labeled $\ell$, i.e. the label of the ambient which $n$ belongs to. Observe that $\mathcal{B}(\ell)$ is defined for all ambients but the environment *env*.

For example, in process $P = p^t[\![\, m^k[\![\, n^\ell[\![\, \mathbf{0} \,]\!] \,]\!] \mid q^s[\![\, \mathbf{0} \,]\!] \,]\!]$, the border of the ambient labeled $\ell$ is $\mathcal{B}(\ell) = k$.

**Definition 3.2** (Upward closure) The upward closure of the border of an ambient labeled $\ell$, $\overline{\mathcal{B}}(\ell)$, is the minimal set that contains $\mathcal{B}(\ell)$ and such that $m \in \overline{\mathcal{B}}(\ell) \Rightarrow \mathcal{B}(m) \in \overline{\mathcal{B}}(\ell)$.

For instance, considering again process $P = p^t[\![\, m^k[\![\, n^\ell[\![\, \mathbf{0} \,]\!] \,]\!] \mid q^s[\![\, \mathbf{0} \,]\!] \,]\!]$, the upward closure of the border of $\ell$ is $\overline{\mathcal{B}}(\ell) = \{k, t, env\}$.

We have already observed that Algorithm 2.4 takes as input a labeling, where labels are partitioned into three distinct sets: high, low and boundary. Let us introduce this notion more formally in order to deal with a dynamic labeling, where only the high labels cannot change status.

**Definition 3.3** (i-th Label Partitioning $\mathcal{L}_i$) We denote by $\mathcal{L}_i$ and we call it the i-th Label Partitioning, the triplet $\mathcal{L}_i = (\mathbf{Lab}_H, \mathbf{Lab}^i_B, \mathbf{Lab}^i_L)$. We assume that $\mathbf{Lab}_H$, $\mathbf{Lab}_B$ and $\mathbf{Lab}_L$ are mutually disjoint, and that $\mathbf{Lab}_H \cup \mathbf{Lab}_B \cup \mathbf{Lab}_L = \mathbf{Lab}^a$.

### 3.1 The Algorithm

The algorithm described below analyses process $P$ starting from the initial labeling $\mathcal{L}_0$. It may either succeed (in this case a labeling $\mathcal{L}_k$ is reached that fulfills the security property we are interested in) or it may fail. The latter case simply means that the process $P$ cannot be guaranteed to be secure by our analysis.

**Initial Label Partitioning.** Given a set of high level labels $\mathbf{Lab}_H$, we initially partition the remaining ambients into the following sets:

$$\begin{array}{rcl} \mathbf{Lab}_B^0 & = & \{\ell \in \mathbf{Lab}^a \setminus \mathbf{Lab}_H \mid \exists h \in \mathbf{Lab}_H \ \wedge \ \ell = \mathcal{B}(h) \ \wedge \\ & & \qquad \nexists h' \in \mathbf{Lab}_H : \mathcal{B}(h') \in \overline{\mathcal{B}}(h)\} \\ \mathbf{Lab}_L^0 & = & \mathbf{Lab} \setminus (\mathbf{Lab}_H \cup \mathbf{Lab}_B^0) \end{array}$$

Through this step, the boundaries that guarantee the absence of information flow in the initial state of process $P$ are defined. Observe that in this way we avoid initial boundary nesting. This is how the boundaries are inferred:
For all $h$ ambients belonging to the process $P$:

*i*) compute the border $\mathcal{B}(h)$. If $\mathcal{B}(h) = env$, process $P$ is insecure by construction, then stop with failure;

*ii*) if $\mathcal{B}(h) \in \mathbf{Lab}^a \setminus \mathbf{Lab}_H$, compute the upward closure $\overline{\mathcal{B}}(h)$ and label $\mathcal{B}(h)$ as boundary iff $\nexists h' \in \mathbf{Lab}_H : \mathcal{B}(h') \in \overline{\mathcal{B}}(h)$.

[Algorithm 3.4 B oundary Inference Algorithm] The analysis is performed by the fixed point algorithm parameterized with respect to $\mathcal{L}_i$. At the beginning, $i = 0$.

(i) Compute Algorithm 2.4 with input $\mathcal{L}_i$ and $P_*$.

(ii) During the execution of Algorithm 2.4, whenever a high level ambient $n$ labeled $h$ gets into an unprotected environment, i.e. $\exists \ell \ : \ (\ell,h) \in \hat{I}_E$ do:

*a*) if $(env,h) \in \hat{I}_E$, the analysis terminates with failure, as it cannot infer a satisfactory labeling that guarantees absence of information leakage;

*b*) otherwise, if $(env,h) \notin \hat{I}_E$:

- a new labeling $\mathcal{L}_{i+1}$ should be considered, labeling every $\ell$ such that $(\ell,h) \in \hat{I}_E$ as a boundary. Let $\mathbf{L} = \{\ell |\ (\ell,h) \in \hat{I}_E\}$ then $\mathcal{L}_{i+1}$ = $(\mathbf{Lab}_H, \mathbf{Lab}_B^i \cup \{\mathbf{L}\}, \mathbf{Lab}_L^i \setminus \{\mathbf{L}\})$.
- go to (i) with $i = i + 1$.

**Refining the solution.** Through this step, a more precise label partitioning $\mathcal{L}_*$ might be computed. From the set of boundaries inferred by the analysis, we take away, if possible, the set of boundaries $\mathbf{B}$ that are not needed to guarantee absence of information leakage (i.e. boundaries nested inside other boundaries). Observe that the set of boundaries nested inside other boundaries can be empty. This refinement procedure can be seen as a narrowing step in the sense of Abstract Interpretation. More formally:

$$\begin{array}{rcl} \mathcal{L}_* & = & (\mathbf{Lab}_H, \mathbf{Lab}_B \setminus \mathbf{B}, \mathbf{Lab}_L \cup \mathbf{B}) \\ \mathbf{B} & = & \{\ell \in \mathbf{Lab}_B \mid \exists \ell' : (\ell',\ell) \in \hat{I}_B \ \wedge \ \nexists \ell'' : (\ell'',\ell) \in \hat{I}_E)\} \end{array}$$

Before addressing termination, soundness and minimality issues, let us try to understand the behavior of this algorithm by looking at an example.

*Example 3.5* Let us consider again the example given in the Introduction:

$$venice^x[\![\, send^y[\![\, \mathbf{out}^c\, venice.\mathbf{in}^c\, montreal\,]\!]\, |\, hdata^h[\![\, \mathbf{in}^c\, send\,]\!]\,]\!]\, | \\ |\, montreal^z[\![\, \mathbf{open}^c\, send\,]\!]\, |\, Q^\ell$$

- Given the set of high level labels $\mathbf{Lab}_H$ in $P$, the initial label partitioning $\mathcal{L}_0$ is computed. $\mathcal{L}_0 = (\mathbf{Lab}_H = \{h\}, \mathbf{Lab}^0_B = \{x\}, \mathbf{Lab}^0_L = \{y,z\})$
- Applying the representation function $\beta^{\mathcal{L}_0}$ to $P_*$, it returns the triplet $(\hat{I}^o_B, \hat{I}^o_E, \hat{H})$:

$$\begin{array}{rcl} \hat{I}^o_B & = & \{(x,y),(x,h),(y,c),(h,c)\} \\ \hat{I}^o_E & = & \{(env,x),(env,z),(z,c)\} \\ \hat{H} & = & \{(h,hdata),(x,venice),(y,send),(z,montreal)\} \end{array}$$

  Executing Algorithm 2.4, the pair $(y,h)$ is introduced in $\hat{I}_E$, reflecting the fact that ambient *send* leaves ambient *venice* during the execution of process $P$.
- At this point, a new label partitioning should be considered:
  $\mathcal{L}_1 = (\mathbf{Lab}_H = \{h\}, \mathbf{Lab}^1_B = \{x,y\}, \mathbf{Lab}^1_L = \{z\})$
  Algorithm 2.4 is computed again. During its execution, the pair $(z,h) \in \hat{I}_E$, reflecting the fact that the boundary *send* , containing confidential data, is opened inside the low ambient *montreal* during the execution of process $P$.
- At this point, the following new label partitioning is considered:
  $\mathcal{L}_2 = (\mathbf{Lab}_H = \{h\}, \mathbf{Lab}^2_B = \{x,y,z\}, \mathbf{Lab}^2_L = \emptyset)$
  Algorithm 2.4 is computed again, and a fixed point is finally reached. In this case, there is no need to refine the solution. Thus, the set of ambients that should be labeled as boundaries is $\{venice, send, montreal\}$.

### 3.2 Soundness and Minimality

In this final section we formally prove termination and correctness of the Boundary Inference Algorithm described in section 3.1. Moreover, we show a minimality result on the computed solution.

**Theorem 3.6** (Termination) *The algorithm always terminates.*

*Proof:* Straightforward, as the number of labels is finite.

**Theorem 3.7** (Soundness) *If there exists a label partitioning $\mathcal{L}_k$ such that the analysis of process $P$ (with initial label partitioning $\mathcal{L}_0$) terminates with success and, in the resulting triplet $(\hat{I}_B, \hat{I}_E, \hat{H})$, no high level ambient does appear in the pairs of $\hat{I}_E$, then the labeling $\mathcal{L}_k$ is sufficient to guarantee the absence of direct leakage within the process P.*

We introduce a new predicate to formalize the notion of *protected ambient*. Given a label $\ell^a \in \mathbf{Lab}^a$, Protected$(\ell^a)$ is true iff $\nexists \ell^{a'} : [(\ell^{a'}, \ell^a) \in \hat{I}_E \ \wedge\ \ell^a \neq env] \ \vee\ \ell^a \in \mathbf{Lab}^a_B$.

The following result guarantees a minimality condition of the refined solution computed by Algorithm 3.4.

*Lemma 3.8* In the solution $(\hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathcal{L}} P_*$:

$(i)$ $\ell \in \mathbf{B} \Rightarrow$ Protected$(\ell)$ both in $\mathcal{L}$ and $\mathcal{L}_*$.
$(ii)$ Protected$(\ell)$ in $\mathcal{L} \Leftrightarrow$ Protected$(\ell)$ in $\mathcal{L}_*$.
$(iii)$ $\beta^{\mathcal{L}}(P_*) = \beta^{\mathcal{L}_*}(P_*)$ and Protected$(\ell)$ in $\beta^{\mathcal{L}}(P_*) \Leftrightarrow$ Protected$(\ell)$ in $\beta^{\mathcal{L}_*}(P_*)$.

**Theorem 3.9** (MINIMALITY) *Let $\mathcal{L}$ be the triplet $(\mathbf{Lab}^a_H, \mathbf{Lab}^a_L, \mathbf{Lab}^a_B)$ and $\mathcal{L}_*$ the label partitioning generated by the Boundary Inference Algorithm 3.1. Then, the Fixed Point Algorithm 2.4 parameterized with respect to $\mathcal{L}$ and to $\mathcal{L}_*$ compute the same solution $(\hat{I}_B, \hat{I}_E, \hat{H})$.*

*Proof:* Essentially the proof simply amounts to observing that, if computing both the algorithms step by step, the pairs added to $\hat{I}_E$ or $\hat{I}_B$ are the same. It is proven by induction on the steps computed by the algorithms. Only the cases for capabilities are non-trivial.

Base of the induction: performing one step from $\beta^{\mathcal{L}}(P_*) = \beta^{\mathcal{L}_*}(P_*)$ with label partitioning $\mathcal{L}$ and $\mathcal{L}_*$ has the same effect.

(**in**): Protected$(\ell^{a''})$ is the same both in $\beta^{\mathcal{L}}(P_*)$ and $\beta^{\mathcal{L}_*}(P_*)$ from point (iii) of Lemma 3.8, thus $\hat{I}_B$ is modified exactly in the same way with $\mathcal{L}$ and $\mathcal{L}_*$.
$\neg$ Protected$(\ell^{a''}) \Rightarrow \ell^{a''}, \ell^{a'}, \ell^{a} \notin \mathbf{B}$, thus $\hat{I}_E$ is modified exactly in the same way with $\mathcal{L}$ and $\mathcal{L}_*$.

(**out**): Protected$(\ell^{a''})$ is the same both in $\beta^{\mathcal{L}}(P_*)$ and $\beta^{\mathcal{L}_*}(P_*)$ from point (iii) of Lemma 3.8, thus $\hat{I}_B$ is modified exactly in the same way with $\mathcal{L}$ and $\mathcal{L}_*$.

(**open**): the case is analogous to the **out** one.

The inductive step is proved by exploiting the fact that the predicate Protected$(\ell)$ is the same for $\mathcal{L}$ and $\mathcal{L}_*$ in each step. Observe that minimality within each step is guaranteed by the fact that Algorithm 2.4 computes the least solution.

## 4 Conclusions

As far as we know, the idea of inferring a security policy that avoids direct information leakage when modeling mobility through Ambients, has not been investigated in the literature yet. Major emphasis, in fact, has been put on Access Control issues [7, 4] than in Information Flow properties. Most of the works in this area, in fact, focus more on enhancing the language to control how ambients may move in and out of other ambients, than on looking at how to "protect" high-data information from untrusted environments.

A few interesting open issues are under investigation to complete the picture we draw in this paper. In particular, it would be interesting to see if there is an ordering among labeling w.r.t. which the analysis behaves monotonically, and if optimizations can be applied to our algorithm to reduce the overall complexity.

## References

[1] US Department of Defense. DoD Trusted Computer System Evaluation Criteria. *DOD 5200.28-STD*, 1985.

[2] C. Bodei, P. Degano, F. Nielson, and H.R.Nielson. Static Analysis of Processes for No Read-Up and No-Write-Down. In *Proc. FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 120–134, Springer-Verlag, 1999.

[3] Chiara Braghin, Agostino Cortesi, and Riccardo Focardi. Control Flow Analysis of Mobile Ambients with Security Boundaries. In Bart Jacobs and Arend Rensink, editors, *Proc. of Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'02)*, pages 197–212. Kluwer Academic Publisher, 2002.

[4] M. Bugliesi and G. Castagna. Secure Safe Ambients. In *Proc. 28th ACM Symposium on Principles of Programming Languages* (POPL'01), pp. 222-235, London. 2001.

[5] L. Cardelli and A. Gordon. Mobile Ambients. In *Proc. FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, Springer-Verlag, 1998.

[6] A. Cortesi, and R. Focardi. Information Flow Security in Mobile Ambients. In *Proc. of International Workshop on Cuncurrency and Coordination* CONCOORD'01, Lipari Island, July 2001, volume 54 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2001.

[7] P. Degano, F. Levi, C. Bodei. Safe Ambients: Control Flow Analysis and Security. In *Proceedings of ASIAN'00*, LNCS 1961, 2000, pages 199-214.

[8] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1): 5-33, 1995.

[9] R. Focardi and R. Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties, *IEEE Transactions on Software Engineering*, Vol. 23, No. 9, September 1997.

[10] R. Focardi, R. Gorrieri, F. Martinelli. Information Flow Analysis in a Discrete Time Process Algebra, in *Proc. of 13th IEEE Computer Security Foundations Workshop* (CSFW13), (P.Syverson ed), IEEE CS Press, 170-184, 2000.

[11] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. In *Proc. Static Analysis Symposium* SAS'99, volume 1694 of *Lecture Notes in Computer Science*, pages 134–148, Springer-Verlag, 1999.

[12] M. Hennessy, J. Riely. Information Flow vs. Resource Access in the Asynchronous Pi-Calculus. *ICALP 2000*: 415-427.

[13] G. Smith, D.M. Volpano, Secure Information Flow in a Multi-Threaded Imperative Language. *In Proc. of POPL 1998: 355-364*.