
Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 99 (2004) 295–317
www.elsevier.com/locate/entcs

Checking Integrity via CoPS and Banana: the E-Commerce Case Study*

Chiara Braghin[1] and Carla Piazza[2]

Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Venezia, Italy

Abstract

We consider two different approaches to security issues. In the first one bisimulation equivalences (dynamic verifications) are exploited to verify non-interference security properties on a CCS-like process algebra calculus. In the second approach control flow analysis (static analysis) is applied to verify security properties in Mobile Ambient calculus. We analyze how a simple electronic commerce case study can be modeled and its integrity verified using the two techniques. The tools CoPS and Banana are used to perform the computations.

Keywords: Security Process Algebra, Non-Interference, Mobile Ambients, Nesting Analysis, Integrity.

1 Introduction

In the last decades the protection of confidential data from undesired accesses has been widely investigated both on systems and networks. Information is typically protected via some access control policy, limiting accesses of entities (e.g., users, processes) to data. There are different levels of flexibility of access control policies depending on the possibility for one entity to change rights to its own data. In the case of mandatory policies, entities cannot change access

* Partially supported by the MIUR Project "Modelli formali per la sicurezza", the EU Contract IST-2001-32617 "MyThS", and the FIRB project (RBAU018RCZ) "Interpretazione astratta e model checking per la verifica di sistemi embedded".

[1] Email: braghin@dsi.unive.it
[2] Email: piazza@dsi.unive.it

doi:10.1016/j.entcs.2004.02.013

rights. For example *Multilevel Security* [1] imposes that entities and data are associated to (ordered) security levels and no access to data at higher level is ever possible. This strong security policy has been designed to avoid internal attacks performed by *Trojan Horse* programs. Unfortunately, even with a multilevel security policy data can be *indirectly* leaked by Trojan Horses (see, e.g., [18,22]).

The necessity of controlling information flow as a whole (both direct and indirect) motivated Goguen and Meseguer in introducing the notion of *Non-Interference* [14]. Non-Interference formalizes the absence of information flow within deterministic systems. Given a system in which *confidential* (i.e., high level) and *public* (i.e., low level) information may coexist, non-interference requires that confidential inputs never affect the outputs of the public interface of the system, i.e., they never interfere with low level users. If such a property holds, one can conclude that no information flow is ever possible from high to low level. In [11], Focardi and Gorrieri express the concept of non-interference in the *Security Process Algebra* (SPA) language, in terms of bisimulation semantics: a system is secure if what a low level user sees of the system is not modified by composing any high level attacker with the system. In [13] a persistent version of the security property defined in [11] has been introduced. Persistency has been proved to be fundamental to deal with processes in dynamic contexts, i.e., contexts that can be reconfigured at runtime. Algorithms to efficiently verify this and other bisimulation based persistent security properties have been presented in [2] and implemented in the tool CoPS, Checker of Persistent Security, available at `http://www.dsi.unive.it/~mefisto/CoPS/`.

The calculus of *Mobile Ambients* has been introduced in [6,7] with the main aim of explicitly modeling mobility. Ambients are arbitrarily nested entities which can move around through suitable capabilities. Big efforts have been devoted to the study of *Control Flow Analysis* (CFA) of such a calculus [16,20]. In particular, some analysis have been applied to the verification of security properties [3,8,17]. The idea of [3,17] is to compute an over-approximation of ambient nestings that may occur during process computation, thus detecting possible intrusions and unwanted information flows. Since the computation of ambient nesting analysis like [3,16,20] requires considerably high complexities, the design of efficient techniques turns out to be very important. Time and space efficient algorithms to compute the control flow analysis described in [3,16] have been presented in [4] and implemented in the tool Banana, Boundary Ambient Nesting ANAlysis, available at `http://www.dsi.unive.it/~mefisto/BANANA/`.

However, many applications require models of security not focused on the complete absence of information flow, but on the preservation of the *integrity* of the resources, that guarantees that information flowing from one place to another must traverse specific points of the path. For instance, consider the E-Commerce Processing System described in [15]. The system represents a process where:

- an order is submitted electronically by a client;
- an e-sales process ensures that the order is correct (e.g., the prices and discounts are correct), and, if so, passes it to the process accounts receivable;
- accounts receivable interacts with a credit card clearing house and, if everything is fine, it passes the order to the shipping process;
- the shipping process sends the items to the client.

In [15] the authors use Linear Temporal Logic to specify information flow policies for SELinux, which can then be checked via model-checking. The E-Commerce example is used to illustrate the technique. In particular, in this example it is important to ensure that, if the internal channels of communication are secure, then the action's casual chain is always the same even in presence of a malicious attacker (e.g., it is not possible that an unpaid order is shipped).

In this paper we show how to use both non-interference security properties and control flow analysis to check the integrity of the E-Commerce Processing System. In the two approaches we take a common starting point, then we concentrate on different aspects. In particular, in Section 2 we describe the non-interference approach by introducing the language and the security properties (Section 2.1), briefly describing the tool CoPS (Section 2.2), modeling the E-Commerce system (Section 2.3). Section 3 is devoted to the control flow analysis approach: we present the language and the nesting analysis (Section 3.1); we outline the features of the tool Banana (Section 3.2); we model and study the E-Commerce system (Sections 3.3 and 3.4). The paper ends with Section 4 containing some comparisons and conclusions.

2 Non-Interference Approach

2.1 Persistent Security Properties

The *Security Process Algebra* (SPA) [11] is a variation of Milner's CCS [19], where the set of visible actions is partitioned into high level and low level actions in order to specify multilevel systems. The syntax of SPA *processes* is

as follows:

$$E ::= \mathbf{0} \mid a.E \mid E + E \mid E|E \mid E \setminus v \mid E[f] \mid Z$$

The semantics is the same as in CCS. Intuitively, $\mathbf{0}$ is the empty process; $a.E$ performs an action a and continues as E; $E_1 + E_2$ is the nondeterministic choice between E_1 and E_2; $E_1|E_2$ is the parallel composition of E_1 and E_2, where executions are interleaved, possibly synchronized on complementary input/output actions, producing the silent action τ; $E \setminus v$ prevents E from performing actions in the set v; $E[f]$ is the process E whose actions are renamed *via* the relabeling function f; the constant Z is associated with a definition $Z \stackrel{\text{def}}{=} E$ and is used to define recursive processes.

As an example, a binary memory cell which initially contains the value 0 and is accessible by both high and low level users through the read and write operations (e.g., r_h0 represents the high read of 0) can be formalized as follows:

$$M0 \stackrel{\text{def}}{=} \overline{r_h0} \,.\, M0 + w_h0 \,.\, M0 + w_h1 \,.\, M1 + \overline{r_l0} \,.\, M0 + w_l0 \,.\, M0 + w_l1 \,.\, M1$$
$$M1 \stackrel{\text{def}}{=} \overline{r_h1} \,.\, M1 + w_h0 \,.\, M0 + w_h1 \,.\, M1 + \overline{r_l1} \,.\, M1 + w_l0 \,.\, M0 + w_l1 \,.\, M1$$

$M0$ and $M1$ are totally insecure processes: no access control is implemented and a high level malicious entity may write confidential information into the memory cell which can be then read by any low level user. Our security property will aim at detecting this kind of flaws, even in more subtle and interesting situations.

The non-interference property named *Persistent Bisimulation-based Non Deducibility on Composition* first introduced in [13] can be defined in terms of unwinding conditions (see [2]): if a state F of a secure process performs a high level action moving to a state G, then F also performs a sequence of silent actions moving to a state K which is equivalent to G for a low level user. We report here the definition of *P_BNDC* denoting by $(\stackrel{\tau}{\rightarrow})^*$ a sequence of zero or more silent actions. We also use $\approx$ for weak bisimulation (see [19]).

Definition 2.1 [P_BNDC [2]] A process E is *P_BNDC* if for all F reachable from E, if $F \stackrel{h}{\rightarrow} G$, then $F(\stackrel{\tau}{\rightarrow})^*K$ and $G \setminus H \approx K \setminus H$.

The memory cell defined above is not *P_BNDC*. In fact, there is a direct information flow from high to low level. We can redefine the cell by eliminating any low level read operation as follows:

$$M0 \stackrel{\text{def}}{=} \overline{r_h0} \,.\, M0 + w_h0 \,.\, M0 + w_h1 \,.\, M1 + w_l0 \,.\, M0 + w_l1 \,.\, M1$$
$$M1 \stackrel{\text{def}}{=} \overline{r_h1} \,.\, M1 + w_h0 \,.\, M0 + w_h1 \,.\, M1 + w_l0 \,.\, M0 + w_l1 \,.\, M1$$

Now the memory cell is *P_BNDC*.

In [2] an efficient polynomial algorithm to verify *P_BNDC* is described. The algorithm is based on the reduction of the problem of checking the security property to the problem of checking strong bisimulation between two graphs. The tool CoPS implements such algorithm. As far as the strong bisimulation underlying algorithm is concerned, CoPS allows the user to choose between the Paige and Tarjan's algorithm [21] and the fast bisimulation algorithm described in [9]. This choice does not affect the worst-case complexities.

2.2 CoPS

The tool CoPS, available at `http://www.dsi.unive.it/~mefisto/CoPS/`, is an automatic checker of multilevel system security properties. It implements the polynomial algorithm described in [2] to check *P_BNDC*. Moreover, it allows to check other security properties studied in [2].

CoPS consists of a *graphical interface* and a *kernel module*. The graphical interface has been implemented in Java to get a large portability and allows to:

- *Insert the process(es)* to be checked in the editor pane. The process(es) can be either typed or loaded from a file. A tree is automatically drawn to facilitate the navigation among processes. The syntax is highlighted to get a better readability. Both fonts and colors can be changed by the user.
- *Select the security property* to be checked and start the verification. It is also possible to check whether two processes are strongly or weakly bisimilar.
- *Read the verification results.* Some time/space statistics are shown together with the security result. Moreover, syntax errors are reported.
- *View the graph* representing the semantics of the process(es). This can be also saved in a file whose type (e.g., jpg, gif, eps) can be chosen by the user.

The kernel module has been implemented in standard C to obtain good performances and consists of:

- A *parser* which checks for syntax errors and builds the syntax tree out of the SPA process.
- A *semantics graph generator* which elaborates the syntax tree to generate an adjacency-list representation of the graph associated to the process.
- A *verifier* which transforms the graph in order to use a strong bisimulation algorithm to perform the security check.

The graph visualization requires the installation of Graphviz, available at `http://www.research.att.com/sw/tools/graphviz/`. The kernel execution can be personalized by using the Settings option in the Edit menu. Many options can be chosen, e.g. by setting the path of Graphviz and the format of the generated graph, choosing the bisimulation algorithm to be used

(the Paige and Tarjan's one [21] or the one presented in [9]), avoiding the graph generation, or setting the use/dimension of an hash table which speeds up the graph generation.

It is possible to avoid the use of the graphical interface and use directly the kernel via command line (`checker --help` shows the help).

Installation and configuration instructions together with a tutorial on the use of CoPS can be found in the tool site.

2.3 E-Commerce Integrity in CoPS via P_BNDC

Let us model the E-Commerce Processing System, described in Section 1, in the SPA language and use CoPS to check that the casual chain remains the same even in presence of a malicious attacker.

In order to model the E-Commerce System in SPA and exploit *P_BNDC* to check its integrity we use a standard technique introduced in the modeling of cryptographic protocols [10,12]. *P_BNDC* is based on the assumption that, given two groups of users, H and L, there is no information flow from H to L iff there is no way for a user of level H to modify the behavior of a user of level L. In the case of protocols we can assume that the users of level L are the honest participants, while H is the external, possibly malicious, environment. Since an intruder may have complete control of the network, we assume that the channels are of level H. In this way all the protocol actions are of level H, while the actions of level L are extra observable actions that are added to the protocol to check its properties. Hence, the way in which the low level actions are added depends on the security property we are interested in. For instance, in the case of integrity, if we want to check that an action h_2 can be performed only after the execution of an action h_1, we add a low level action l_1 after each h_1 and a low level action l_2 after each h_2 and we check that in the low level behavior l_2 occurs only after l_1.

Notice that *P_BNDC* exploits bisimulation as observational equivalence. Similarly, the *Non Deducibility on Composition* (*NDC*) property introduced in [11], instantiates the non-interference schema using trace equivalence instead of bisimulation. In [12] it has been shown that trace equivalence, and hence *NDC*, is sufficient for verifying several protocol properties, such as authentication, and bisimulation is necessary in other cases, such as fairness. The bisimulation relation is dead-lock sensitive, while trace equivalence is not. Hence, in the case of integrity the use of bisimulation instead of trace equivalence allows us to check that also the dead-locks in the casual chain are unalterable, i.e. the process under attack reaches a dead-lock after the action sequence s iff the process in isolation reaches a dead-lock after the sequence s. In particular, the use of bisimulation ensures that all the branching points

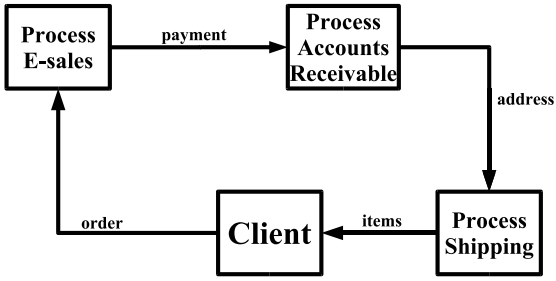

Fig. 1. E-Commerce: basic case.

in the execution chains do not change under attacks.

We start modeling only some basic aspects of the process, then we proceed adding details to the model. In Figure 1 we give a graphical representation of the E-Commerce Processing System assuming that the agents involved communicate directly through channels (the arrows in the picture). *Client* sends an order containing prices, payment data and address. Let us assume that *Client* sends a correct order, i.e., both prices and payment are correct. We use the action *itemprice&credcardnum&address* to represent this fact [3]. Then *Client* waits to receive the items. The process *E-sales* receives the order, and if the prices are correct it sends the payment data and the address to the process *AccountsRec*. We use the action *credcardnum&address* to model the fact that the correct payment data are sent to *AccountsRec*. Similarly, if the payment is correct *AccountsRec* asks, through the action *address*, to the process *Shipping* to send the items. As said above, since an intruder could have control over the network, all the involved channels are of level high. Hence, in the SPA language we get the following processes.

$$
\begin{array}{ll}
\mathit{E\text{-}Commerce} & \stackrel{\text{def}}{=} (\mathit{Client} | \mathit{Server}) \\
\mathit{Client} & \stackrel{\text{def}}{=} \overline{\mathit{itemprice\&credcardnum\&address}}.\mathit{items}.\mathbf{0} \\
\mathit{Server} & \stackrel{\text{def}}{=} \mathit{E\text{-}sales} | \mathit{AccountsRec} | \mathit{Shipping} \\
\mathit{E\text{-}sales} & \stackrel{\text{def}}{=} \mathit{itemprice\&credcardnum\&address}.\overline{\mathit{credcardnum\&address}}.\ \mathbf{0} \\
 & + \mathit{itemprice\&wrongcardnum\&address}.\overline{\mathit{wrongcardnum\&address}}.\mathbf{0} \\
\mathit{AccountsRec} & \stackrel{\text{def}}{=} \mathit{credcardnum\&address}.\overline{\mathit{address}}.\mathbf{0} \\
\mathit{Shipping} & \stackrel{\text{def}}{=} \mathit{address}.\overline{\mathit{items}}.\mathbf{0}
\end{array}
$$

Notice that if the prices are correct while the payment data are wrong the process *E-sales* reads the prices and outputs the remaining set of data (i.e., payment and address). In this case *AccountsRec* blocks the process since it cannot read a wrong payment. Another possibility would be to add to

[3] In the case of an order with wrong payment data we use the action *itemprice&wrongcardnum&address*.

AccountsRec the alternative behavior corresponding to the synchronization with actions denoting the reception of wrong data. This change would not affect the considerations which follows.

We want to check that in the execution chains only the orders with correct prices are sent to *AccountsRec* and only the orders with correct price and payment data are shipped. To verify this property, we add the low level actions *okl1*, *okl2*, and *okl3* in different execution points.

$$
\begin{array}{ll}
\textit{E-Commerce} & \stackrel{\text{def}}{=} (\textit{Client}|\textit{Server}) \\
\textit{Client} & \stackrel{\text{def}}{=} \overline{\textit{itemprice\&credcardnum\&address}}.\textit{items}.\mathbf{0} \\
\textit{Server} & \stackrel{\text{def}}{=} \textit{E-sales}|\textit{AccountsRec}|\textit{Shipping} \\
\textit{E-sales} & \stackrel{\text{def}}{=} \textit{itemprice\&credcardnum\&address}.\textit{okl1}.\overline{\textit{credcardnum\&address}}.\ \mathbf{0} \\
 & + \textit{itemprice\&wrongcardnum\&address}.\textit{okl1}.\overline{\textit{wrongcardnum\&address}}.\mathbf{0} \\
\textit{AccountsRec} & \stackrel{\text{def}}{=} \textit{credcardnum\&address}.\textit{okl2}.\overline{\textit{address}}.\mathbf{0} \\
\textit{Shipping} & \stackrel{\text{def}}{=} \textit{address}.\textit{okl3}.\overline{\textit{items}}.\mathbf{0}
\end{array}
$$

When there are no intruders interacting with the process *E-Commerce* the low level behavior is of the form $\textit{okl1}.\textit{okl2}.\textit{okl3}.\mathbf{0}$. This can be easily verified in CoPS by checking that $\textit{E-Commerce}\backslash H$ is weak bisimilar to the latter. The process *E-Commerce* is not *P_BNDC*. In particular, the state *Shipping* which is reachable from *E-Commerce* does not satisfy the unwinding condition of Definition 2.1: $\textit{Shipping} \stackrel{\textit{address}}{\longrightarrow} \textit{okl3}.\overline{\textit{items}}.\mathbf{0}$, and *Shipping* does not reach with any sequence of τ transitions any state E such that $\textit{okl3}.\overline{\textit{items}}.\mathbf{0}\backslash H \approx E \setminus H$. Such a situation, automatically detected by CoPS, represents the fact that a high level intruder could directly synchronize with the process *Shipping* and force the system to ship orders which have not been paid. Hence, the lack of integrity is due to the fact that the intruder can communicate with process *Shipping*.

If we believe that our channels are secure and the intruder cannot directly communicate with the processes inside the *E-Commerce* system, we can model this fact using a restriction over the channels. Hence, we get the following model.

$$
\begin{array}{ll}
\textit{E-Commerce} & \stackrel{\text{def}}{=} (\textit{Client}|\textit{Server})\backslash H \\
\textit{Client} & \stackrel{\text{def}}{=} \overline{\textit{itemprice\&credcardnum\&address}}.\textit{items}.\mathbf{0} \\
\textit{Server} & \stackrel{\text{def}}{=} \textit{E-sales}|\textit{AccountsRec}|\textit{Shipping} \\
\textit{E-sales} & \stackrel{\text{def}}{=} \textit{itemprice\&credcardnum\&address}.\textit{okl1}.\overline{\textit{credcardnum\&address}}.\ \mathbf{0} \\
 & + \textit{itemprice\&wrongcardnum\&address}.\textit{okl1}\,\overline{\textit{wrongcardnum\&address}}.\mathbf{0} \\
\textit{AccountsRec} & \stackrel{\text{def}}{=} \textit{credcardnum\&address}.\textit{okl2}.\overline{\textit{address}}.\mathbf{0} \\
\textit{Shipping} & \stackrel{\text{def}}{=} \textit{address}.\textit{okl3}.\overline{\textit{items}}.\mathbf{0}
\end{array}
$$

In this case the process *E-Commerce* satisfies the three security properties checked by CoPS, which implies that the casual chain order is always respected.

We can now increase the power of the client by adding the option of querying the system through an untrusted channel, as represented in Figure 2. In

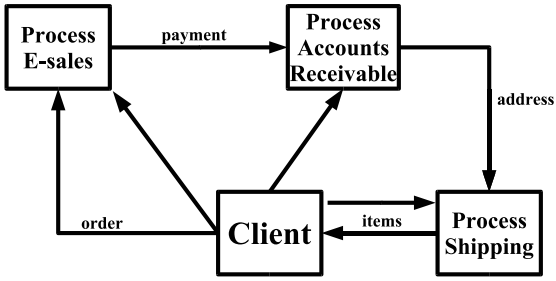

Fig. 2. E-Commerce: queries.

this case, the queries are not restricted high level actions, named *okh1*, *okh2*, *okh3*. In order to obtain a secure system, we need to add timeouts (τ actions), i.e., the system does not wait the queries forever.

$$
\begin{array}{ll}
\textit{E-Commerce} & \stackrel{\text{def}}{=} (\textit{Client}|\textit{Server})\backslash Hc \\
\textit{Client} & \stackrel{\text{def}}{=} \overline{\textit{itemprice\&credcardnum\&address}}.\textit{items}.\mathbf{0} \\
\textit{Server} & \stackrel{\text{def}}{=} \textit{E-sales}|\textit{AccountsRec}|\textit{Shipping} \\
\textit{E-sales} & \stackrel{\text{def}}{=} \textit{itemprice\&credcardnum\&address}. \\
 & \quad (\overline{\textit{okh1}}.\textit{okl1}.\overline{\textit{credcardnum\&address}}.\mathbf{0} \\
 & \qquad + \tau.\textit{okl1}.\overline{\textit{credcardnum\&address}}.\mathbf{0}) + \\
 & \quad \textit{itemprice\&wrongcardnum\&address}. \\
 & \quad (\overline{\textit{okh1}}.\textit{okl1}.\overline{\textit{wrongcardnum\&address}}.\mathbf{0} \\
 & \qquad + \tau.\textit{okl1}.\overline{\textit{wrongcardnum\&address}}.\mathbf{0}) \\
\textit{AccountsRec} & \stackrel{\text{def}}{=} \textit{credcardnum\&address}. \\
 & \quad (\overline{\textit{okh2}}.\textit{oklow2}.\overline{\textit{address}}.\mathbf{0} + \tau.\textit{oklow2}.\overline{\textit{address}}.\mathbf{0}) \\
\textit{Shipping} & \stackrel{\text{def}}{=} \textit{address}.(\overline{\textit{okh3}}.\textit{oklow3}.\overline{\textit{items}}.\mathbf{0} + \tau.\textit{oklow3}.\overline{\textit{items}}.\mathbf{0})
\end{array}
$$

where $Hc \subseteq H$ is the set of actions $\{\textit{item}, \textit{itemprice\&credcardnum\&address},$

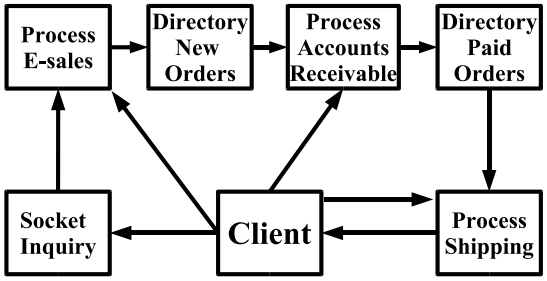

Fig. 3. E-Commerce: files and queries.

itemprice&wrongcardnum&address, *credcardnum&address*, *address*}. We can use CoPS to verify that the systems is *P_BNDC*. Moreover, we can ask to CoPS a graphical representation of *E-Commerce*$\backslash H$. In the graphical representation we can see the sequences of low level actions when the system is not under attack. Since the system is secure, the sequences remain the same even in presence of any malicious attacker.

Finally, another interesting variation in the modeling of the system consists of introducing intermediate files in which the information are stored (see Figure 3). In this case it is important to model the files carefully: each order has to be processed exactly once. For this reason each file has a state *empty* in which it is reset after the information have been read. We show below part of the model written in the SPA language.

$$
\begin{array}{lll}
\textit{E-Commerce} & \stackrel{\text{def}}{=} & (\textit{Client}|\textit{Server})\backslash Hc \\
\textit{Client} & \stackrel{\text{def}}{=} & \overline{\textit{itemprice\&credcardnum\&address}}.\textit{items}.\mathbf{0} \\
\textit{Server} & \stackrel{\text{def}}{=} & \textit{Socket-empty}|\textit{E-sales}|\textit{Filepaydata-empty} \\
 & & \quad |\textit{AccountsRec}|\ \textit{Fileaddress-empty}|\ \textit{Shipping} \\
\textit{Socket-empty} & \stackrel{\text{def}}{=} & \textit{itemprice\&credcardnum\&address}.\textit{Socket-full-okok} \\
 & & +\textit{wrongitemprice\&credcardnum\&address}.\textit{Socket-full-nook} \\
 & & +\textit{itemprice\&wrongcardnum\&address}.\textit{Socket-full-okno} \\
 & & +\textit{wrongitemprice\&wrongcardnum\&address}.\textit{Socket-full-nono} \\
\textit{Socket-full-okok} & \stackrel{\text{def}}{=} & \overline{\textit{socket-itemprice\&credcardnum\&address}}.\textit{Socket-empty} \\
\textit{Socket-full-nook} & \stackrel{\text{def}}{=} & \overline{\textit{socket-wrongitemprice\&credcardnum\&address}}.\textit{Socket-empty} \\
\textit{Socket-full-okno} & \stackrel{\text{def}}{=} & \overline{\textit{socket-itemprice\&wrongcardnum\&address}}.\textit{Socket-empty} \\
\textit{Socket-full-nono} & \stackrel{\text{def}}{=} & \overline{\textit{socket-wrongitemprice\&wrongcardnum\&address}}.\textit{Socket-empty} \\
\textit{E-sales} & \stackrel{\text{def}}{=} & \textit{socket-itemprice\&credcardnum\&address}. \\
 & & \quad (\overline{\textit{okh1}}.\textit{okl1}.\textit{credcardnum\&address}.\mathbf{0}+\tau.\ldots)+ \\
 & & \textit{socket-wrongitemprice\&credcardnum\&address}. \\
 & & \quad (\overline{\textit{noh1}}.\textit{nol1}.\mathbf{0}+\tau.\textit{nol1}.\mathbf{0})+\ldots
\end{array}
$$

...

Also in this case the system is secure, since it is *P_BNDC*.

3 Control Flow Analysis Approach

3.1 Mobile Ambients and Nesting Analysis

The Mobile Ambient calculus has been introduced in [6] 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 $n \in \mathbf{Amb}$ denotes an ambient name.

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

Intuitively, the restriction $(\nu n)P$ introduces the new name n and limits its scope to P; $P \mid Q$ is P and Q running in parallel; replication provides recursion and iteration. 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 a sibling ambient n. Labels on ambients and on transitions are introduced as it is customary in static analysis to indicate "program points". They will be useful in the next paragraphs when developing the analysis.

The operational semantics 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 (see [6] for more details).
For instance, let P_1 be a process modeling an *envelope* sent from *venice* to *pisa*:

$$venice[\ envelope[\mathbf{out}\ venice.\mathbf{in}\ pisa.\mathbf{0}\,]\ \mid Q\,]\ \mid\ pisa[\mathbf{open}\ envelope.\mathbf{0}\,]$$

Initially, *envelope* is in site *venice*. Then, it exits *venice* and enters site *pisa* by applying its capabilities $\mathbf{out}\ venice$ and $\mathbf{in}\ pisa$, respectively. Once site *pisa* receives *envelope*, it reads its content by consuming its $\mathbf{open}\ envelope$ capability. Finally, P_1 reaches the state $venice[\,Q\,]\ \mid\ pisa[\,\mathbf{0}\,]$.

To deal with security issues, information is classified into different levels of confidentiality. This is obtained by exploiting the labeling of the ambients. In particular, the set of ambient labels is partitioned into three disjoint sets: *high*, *low* and *boundary* labels. Ambients labeled with boundary labels (*boundary ambients*) are the ones responsible for confining confidential information. Information leakage occurs if a high level ambient exits a boundary, thus becoming possibly exposed to a malicious ambient attack. In the following examples, we will use b to label boundaries, h for high level ambients, and c for capabilities. For instance, let P_2 be a labeled extension of process P_1, in which the *envelope* contains confidential data *hdata* (labeled *high*) which needs to be safely sent from *venice* to *pisa*.

$$\begin{aligned} venice^{b_1}[\ envelope^{b_2}[\mathbf{out}^{c_1}\ venice.\mathbf{in}^{c_2}\ pisa.\mathbf{0}\ \mid \\ \qquad\qquad hdata^{h}[\mathbf{0}\,]\,]\,]\ \mid \\ pisa^{b_3}[\mathbf{open}^{c_4}\ envelope.\mathbf{0}\,] \end{aligned}$$

In this case, *venice*, *pisa* and *envelope* must be labeled *boundary* to protect *hdata* during the whole execution. See [3] for more detail.

In [16], Nielson et al. introduce a Control Flow Analysis of a process P aiming at modeling the possible ambient nestings occurring in the execution of P. It works on pairs $(\hat{I}, \hat{H})$. The first component $\hat{I}$ is a set of labels' pairs which records all nestings: if process P, during its execution, contains an ambient labeled ℓ^a having inside either a capability or an ambient labeled ℓ, then (ℓ^a, ℓ) is expected to belong to $\hat{I}$. The second component $\hat{H}$ keeps track of the correspondence between names and labels.

The analysis is defined as usual by a representation and a specification function. The representation function aims at mapping concrete values to their best abstract representation. It is given in terms of a function $\beta^{\mathrm{CF}}_{\ell}(P)$ which maps process P into a pair $(\hat{I}, \hat{H})$ corresponding to the initial state of P, with respect to an enclosing ambient labeled with ℓ. In particular, the representation of a process P is defined as $\beta^{\mathrm{CF}}_{env}(P)$, env being a special label to denote the external environment. Consider for example process P_2. Its representation function is

$$\beta^{\mathrm{CF}}_{env}(P_2) = (\{(env, b1), (env, b3), (b1, b2), (b2, h), (b2, c1), (b2, c2), (b3, c4)\},$$
$$\{(b1, venice), (b2, envelope), (h, hdata), (b3, pisa)\}),$$

where all ambient nestings are captured by the first component, while all the correspondences between ambients and labels of P_2 are kept by the second one.

The specification states a closure condition of a pair $(\hat{I}, \hat{H})$ with respect to all the possible moves executable on a process P. It mostly relies on recursive calls on subprocesses except for the three capabilities *open*, *in*, and *out*, where the result of performing that capability must be recorded in $\hat{I}$. Consider again process P_2. The first component of its least solution is

$$\hat{I} = \{(env, b1), (env, b2), (env, b3), (b1, b2), (b2, h), (b2, c1), (b2, c2),$$
$$(b3, b2), (b3, h), (b3, b3), (b3, c1), (b3, c2), (b3, c4)\}.$$

Notice that the analysis correctly captures through the pair $(env, b2)$ the possibility for *envelope* to exit from *venice*.

In [3], a more accurate abstract domain that separately considers nesting inside and outside security boundaries is proposed, yielding to a much more sophisticated control flow analysis for detecting unwanted boundary crossing, i.e., information leakage. The main idea is to distinguish among nestings either *protected* or *unprotected* by boundaries.

The notion of "boundary ambient" and the refined control flow analysis can be further enhanced to infer which ambients should be "protected" to guarantee the absence of information leakage for a given process. More specifically, in [5] we consider a process P wherein only high level data are known and the aim of the analysis is to detect which ambients among the "untrusted" ones should be protected and labeled "boundary" to guarantee that the system is secure. This problem can be properly addressed by re-executing the Control Flow Analysis presented in [3]. A successful analysis infers boundary

ambients until a fix-point is reached, returning the set of ambients that should be "protected".

3.2 Banana

Banana, available at `http://www.dsi.unive.it/~mefisto/BANANA`, analyses the information leakage in mobile agent specifications through the analysis described in [3,16]. In particular, all the analyses described in the previous section have been implemented in the tool, i.e., the nesting analysis by Nielson et al. [16] and the boundary nesting analysis of [3].

The main features of Banana are:

- A textual and graphical editor for Mobile Ambients, to specify and modify the process by setting ambient nesting capabilities and security attributes in a very user-friendly fashion.
- A parser which checks for syntax errors and builds the syntax tree out of the Mobile Ambient process.
- An analyzer which computes an over-approximation of all possible nestings occurring at run-time. The tool supports two different control flow analyses, namely the one of Nielson et al. [16] and the one by Braghin et al. [3].
- A post-processing module, that interprets the results of the analysis in terms of the boundary-based information-flow model proposed in [3], where information flows correspond to leakages of high-level (i.e., secret) ambients out of protective (i.e., boundary) ambients, toward the low-level (i.e., untrusted) environment.
- A detailed output window reporting both the analysis and the security results obtained by the post-processing module, and some statistics about the computational costs of the performed analysis.

Banana is implemented in Java and strongly exploits the modularity of object-oriented technology, thus allowing scalability to other analysis and extensions of the target language. Moreover, it is conceived as an applet based on AWT and thus compatible with the majority of current web browsers supporting Java. A tutorial about the use of Banana is available in the tool web pages.

3.3 E-Commerce Integrity in Banana *via CFA*

Let us now model the E-Commerce Processing System in the pure Ambient calculus, and then use the Banana tool to check the integrity of the process.

Notice that in the pure Ambient calculus there are not communication

primitives. In the absence of such primitives, the exchange of a *message* between a *Sender* and a *Receiver* may be modeled as a sequence of **out** *Sender* and **in** *Receiver* actions, performed by ambient *message*. In this way *message* moves from *Sender* to *Receiver*, where it will be **open**ed in order to read its contents. Hence, the opening of an ambient is used to model the synchronization among the *Sender* and the *Receiver* of the message.

We model the E-Commerce system in Mobile Ambients by representing each principle and each protocol phase with an ambient. More specifically, *Client* and *Server* ambients represent the main actors, while *Server*'s sub-ambients model the phases the order goes through. The submission of orders by clients, and the interaction among one protocol phase and the next one correspond to message exchanges which we model as explained above. For instance, *itemprice* moves from *Client* to *E-sales*, which is a *Server* sub-ambient, where it is opened. To check that the order is properly formatted (i.e., the item is available and the purchase price is correct), *E-sales* opens the ambient *itemprice*. When the order is correct, the opening of *itemprice* succeeds, while in the case of a wrong order this ambient is named *wrongitemprice* thus it cannot be opened (see also *itemprice&wrongcardnum&address* in Section 2.3). After the opening of the ambient *itemprice*, the interaction moves inside the ambient *AccountsRec*, which checks that the payment data are correct by opening the ambient *creditcardnum*. We summarize in the opening of *creditcardnum* all the interactions with the bank to receive the payment. Hence, when the opening succeeds, the payment has been received. Finally, the *Shipping* ambient uses the information contained in *address* to send the *items* to the *Client* . In this way we get that the *Server S* of the E-Commerce system is modeleed as follows (see also Figure 4):

$$
\begin{array}{l}
\mathit{Server}\,\mathbf{[} \\
\quad \mathit{E\text{-}sales}\,\mathbf{[}\,\mathbf{open}\,\mathit{itemprice}.\mathbf{0}\,\mathbf{]}\ \mid \\
\quad \mathit{AccountsRec}\,\mathbf{[}\,\mathbf{open}\,\mathit{creditcardnum}.\mathbf{0}\,\mathbf{]}\ \mid \\
\quad \mathit{Shipping}\,\mathbf{[}\,\mathbf{open}\,\mathit{address}.\mathit{item}\,\mathbf{[}\,\mathbf{out}\,\mathit{Shipping}.\mathbf{out}\,\mathit{Server}.\mathbf{in}\,\mathit{Client}.\mathbf{0}\,\mathbf{]}\,\mathbf{]}\,\mathbf{]}
\end{array}
$$

Notice that in our modeling each sub-ambient of the *Server* opens an ambient: each opening represents the fact that a checking phase has been passed. Also the client must be explicitly modeled. The whole system will be the parallel composition of the two. All the capabilities to move from one sub-ambient of the *Server* to another have to be included in the sub-ambients of the *Client*. In particular, *itemprice* has to exit from *Client*, enter into *Server* and then into *E-sales*. If the order is correct, *itemprice* is opened. After the opening of *itemprice*, an ambient *creditcardnum* has to enter into *AccountsRec*.

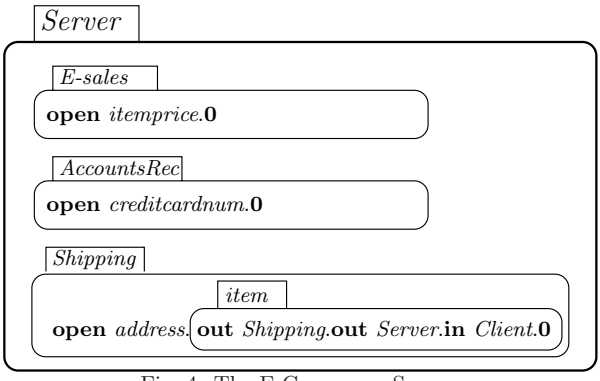

Fig. 4. The E-Commerce Server.

Once *creditcardnum* has been opened, an ambient *address* has to move inside *Shipping* where it is used to send the items. Hence, a correct client C_1 can be modeled as follows:

$$
\begin{array}{l}
\mathit{Client}[\,\\
\quad \mathit{itemprice}[\,\mathbf{out}\ \mathit{Client}.\mathbf{in}\ \mathit{Server}.\mathbf{in}\ \mathit{E\text{-}sales}.\\
\qquad \mathit{creditcardnum}[\,\mathbf{out}\ \mathit{E\text{-}sales}.\mathbf{in}\ \mathit{AccountsRec}.\\
\qquad\quad \mathit{address}[\,\mathbf{out}\ \mathit{AccountsRec}.\mathbf{in}\ \mathit{Shipping}.\mathbf{0}\,]\,]\,]\,]
\end{array}
$$

It is easy to see that in the execution of process $S \mid C_1$ the first checking phase is passed with the opening of *itemprice*, then the second checking phase is passed with the opening of *creditcardnum*, and finally the third checking phase is passed with the opening of *address*. We are interested in ensuring that the three checking phases are performed in the correct order, e.g., we do not want that the *address* is opened inside *Shipping* before *itemprice* is opened inside *E-sales*.

Using BANANA to compute an over-approximation of the nestings of $S \mid C_1$, we get that (*E-sales, itemprice*), (*AccountsRec, creditcardnum*), and (*Shipping, address*) belong to $\hat{I}$. The pair (*E-sales, itemprice*) represents the fact that the first check correctly starts and ends. Similarly, the other pairs are related to the second and the third checks. The correct order of the checks is assured by the nesting of the ambients inside the client, which forces *itemprice* to be opened before *creditcardnum*, and *creditcardnum* before *address*.

A honest client C_2 sending a wrong payment is similar to the correct client, but instead of the ambient *creditcardnum* it has an ambient *wrongcardnum*, i.e., it is of the form:

$$Client[\ itemprice[\ \mathbf{out}\ Client.\mathbf{in}\ Server.\mathbf{in}\ E\text{-}sales.\ wrongcardnum[\ \mathbf{out}\ E\text{-}sales.\mathbf{in}\ AccountsRec.\ address[\ \mathbf{out}\ AccountsRec.\mathbf{in}\ Shipping.\mathbf{0}\ ]\ ]\ ]\ ]$$

In this case, during the execution of process $S \mid C_2$, the first check is passed, while the second is not. Hence, the third check does not even start because the process is blocked at the **open** *creditcardnum* action. The $\hat{I}$ set computed by the tool captures correctly this fact, since the pairs (*AccountsRec, creditcardnum*), and (*Shipping, address*) are not in $\hat{I}$. Since this does not break the integrity of the E-Commerce system also this client is "correct".

On the other hand, an intruder is a process which tries to corrupt the casual chain, by skipping one of the checking phases. For example, the following intruder C_3 sends his *address* directly to the *Shipping* process. In this way he receives the items without paying for them.

$$Client[\ address[\ \mathbf{out}\ Client.\mathbf{in}\ Server.\mathbf{in}\ Shipping.\mathbf{0}\ ]\ ]$$

If we analyze $S \mid C_3$ using BANANA, we notice that the pair (*Shipping, address*) is in $\hat{I}$, while (*E−sales, itemprice*) and (*AccountsRec, creditcardnum*) are not in $\hat{I}$.

There are also more subtle intruders which we want to detect. For instance, consider the process C_4 of the form:

$$C_1 \mid Intruder[\ address[\ \mathbf{out}\ Intruder.\mathbf{in}\ Server.\mathbf{in}\ Shipping.\mathbf{0}\ ]\ ]$$

where C_1 is the correct client. In this case even if $\hat{I}$contains all the pairs which were in the analysis of C_1, the system can be corrupted, since there exists an execution in which the intruder receives the items the client has payed for, but unfortunately our analysis cannot detect this kind of attack.

Therefore, we start by giving a definition which aims at avoiding the latter kind of intruders. In particular, we impose that there is at most one address to which shipping the items. Moreover, we restrict the capabilities of the client, since we do not want that the client performs the checks by himself by opening some ambients. Finally, we impose some restrictions on the nesting of the ambients in the initial state of the client. This last condition is necessary to ensure that the checks are performed in the correct order. In the definition we use the notation $ambient1 \preccurlyeq ambient2$ to denote the fact that if there is at least one occurrence of *ambient2*, then all the occurrences of *ambient1* are nested in *ambient2*.

Definition 3.1 [Syntactic Correctness] A client C is *syntactically correct* if the following conditions hold:

(i) each ambient occurs in C at most once and C cannot have ambients with names *Server, E−sales, AccountsRec, and Shipping*;

(ii) C has not **open** capabilities, and the **out** and **in** capabilities in C are only referred to the *Client, Server, E−sales, AccountsRec, and Shipping* ambients;

(iii) $address \preccurlyeq creditcardnum \preccurlyeq itemprice$.

Notice that the conditions in the above definition are just syntactic checks. Condition *(iii)* can be checked exploiting BANANA, since, for instance, to check $creditcardnum \preccurlyeq itemprice$ it is sufficient to compute $\beta^{\mathrm{CF}}_{env}(C)$ and, if *itemprice* occurs in C, check that in the pairs of the form $(x, creditcardnum)$ x is always *itemprice*.

We have that clients C_1, C_2, and C_3 satisfy Definition 3.1, while C_4 does not. Another intruder which is not syntactically correct is

$$\begin{array}{l} Client[\\ \quad itemprice[\,\mathbf{out}\, Client.\mathbf{in}\, Server.\mathbf{in}\, E\text{-}sales.\mathbf{0}\,] \mid \\ \quad credicardnum[\,\mathbf{out}\, Client.\mathbf{in}\, Server.\mathbf{in}\, AccountsRec.\mathbf{0}\,] \mid \ldots\,] \end{array}$$

because it does not satisfy Condition *(i)*. In fact in this case there are executions in which the checks are not performed in the correct order.

We now want to use BANANA to distinguish between the honest clients C_1 and C_2 and the intruder C_3. First, we formalize the fact that the checks in the execution chains are performed in the correct order. A *prefix* of a given sequence of actions $s = \langle a_1, \ldots, a_n\rangle$ is a (possibly empty) sequence of the form $\langle a_1, \ldots, a_i\rangle$, with $i \leq 0 \leq n$. Intuitively we want to accept only the executions which are prefixes of the execution in which all the checks are performed in the right order.

Definition 3.2 [Integrity] Let S be the E-Commerce Server and C be a syntactically correct client. We say that $S \mid C$ satisfies *integrity* if in all the executions of $S \mid C$ the **open** capabilities of S are executed in an order which is a prefix of:

$$sopen = \langle \mathbf{open}\, itemprice, \mathbf{open}\, creditcardnum, \mathbf{open}\, address\rangle$$

If the client is syntactically correct, the integrity can be verified by reading the pairs in $\hat{I}$. The conditions on the initial nestings inside *Client* imposed in Definition 3.1 (condition *(iii)*) ensures that when the pairs of interest are in $\hat{I}$, they have been inserted in a certain order. Moreover, we prove that it is never the case that the such pairs are in $\hat{I}$ because of an over-approximation. In particular, in the case of a syntactically correct client, the following proposition

translates the prefix requirement of Definition 3.2 in a prefix requirement over $\hat{I}$.

Proposition 3.3 *Let S be the E-Commerce System and C be a syntactically correct client. If $\hat{I}$ contains an unordered prefix of:*

$$snesting = \langle (E\text{-}sales,\ itemprice), (AccountsRec,\ creditcardnum), (Shipping,\ address) \rangle$$

where $\hat{I}$ is such that $(\hat{I}, \hat{H}) \models^{\mathrm{CF}} S \mid C$, then $S \mid C$ satisfies integrity.

Proof. We consider all the possible prefixes $\langle a_1, \ldots, a_i \rangle$ of *snesting* and we prove that if $\{a_1, \ldots, a_i\}$ is a subset of $\hat{I}$ (while $a_{i+1}, \ldots, a_n$ are not in $\hat{I}$), then in all the executions in which a_j, with $j \leq i$, is a real nesting (i.e., a pair added not because of over-approximation) also $a_1, \ldots, a_{j-1}$ are real nesting.

If $i = 0$, then we have nothing to prove. Also the cases $i = 1$ and $i = 2, j = 1$ are trivial.

If $i = 2$ and $j = 2$, then in $\hat{I}$ we have the pairs (*E-sales, itemprice*) and (*AccountsRec, creditcardnum*). Let us consider an execution in which the nesting (*AccountsRec, creditcardnum*) occurs. Since (*E-sales, itemprice*) is in $\hat{I}$ the ambient *itemprice* occurs in C. Hence, since C is syntactically correct, *creditcardnum* is nested in *itemprice* in the initial state of C. Since C is syntactically correct, it has no open capabilities, and no out capabilities relatively to *itemprice*. Hence, *itemprice* has to be opened inside *E-sales*, i.e., also the nesting (*E-sales, itemprice*) occurs in the execution under consideration.

Similarly we can prove the thesis in all the remaining cases.

In order to prove the proposition we have to prove that if we consider an execution *ex* in which the prefix $\langle a_1, \ldots, a_j \rangle$ of *snesting* occurs, then in *ex* the open capabilities are executed in the order which corresponds to the prefix length j of *sopen*. This is again an immediate consequence of condition *(iii)* of Definition 3.1. □

Using BANANA we get that $S \mid C_1$ satisfies integrity, since $\hat{I}$ contains all the three pairs required by Proposition 3.3. Also $S \mid C_2$ satisfies integrity, since $\hat{I}$ contains only the first pair, which means that the execution has blocked after the first phase. On the other hand, if we consider $S \mid C_3$, $\hat{I}$ contains a suffix (instead of a prefix) of *snesting*. In fact $S \mid C_3$ does not satisfy integrity, since it skips two checks.

Notice that $S \mid C$ can satisfy integrity even if $\hat{I}$ does not contain a prefix of *snesting*, i.e., the vice-versa of Proposition 3.3 does not hold. Consider for instance the client C_5:

$$Client[\,\mathbf{out}\ Server.address[\,\mathbf{out}\ Client.\mathbf{in}\ Server.\mathbf{in}\ Shipping.\mathbf{0}\,]\,]$$

In this case $\hat{I}$ contains the pair (*Shipping,address*) and not the other two pairs. However, this is due to an over-approximation, since during the execution **out** *Server* blocks the system $S \mid C_5$.

3.4 Information Leakage

Let us now exploit the tool for what it was originally intended, that is, detecting information leakage. This security property is orthogonal to integrity, thus the results obtained are independent one another. Our starting point is the fact that the ordering information sent by the client cannot be disclosed to other untrusted users of the system. We model this fact by labeling as *high* ambients *itemprice*, *creditcardnum*, and *address*. Then, we apply the analysis of [5], also supported by the tool, to verify if the process is secure and, if not, which ambients should be labeled boundary in order to protect the system.

Using BANANA, we can check that the analysis fails. In fact, in order to make the execution of $S \mid C$ process secure, the ordering information (i.e., ambient *itemprice*) must be enclosed in a new boundary ambient which protects the data when it moves out of the client. This requirement corresponds to the SSL-protected network socket requirement of [15].

4 Comparisons and Conclusions

In [15] the authors informally described a simple E-Commerce case study. They showed how to: express it in a highly abstract model of the SELINUX operating system access control mechanism; express integrity in temporal logic; verify the integrity of the system using model checking.

In this paper we started from the informal description of the E-Commerce system presented in [15] and we considered two different approaches to study its integrity:

- a dynamic verification technique on the SPA language;
- a static analysis technique on Mobile Ambients.

As far as the modeling is concerned it is easy to see in the *E-Commerce* systems we proposed there are many similarities, even if: in the SPA language we had to add low level observable actions to check integrity; in Mobile Ambients we had to exploit in, out, and open capabilities to simulate communications. On the other hand, the main differences concern the verification techniques we used:

- the dynamic verification technique is correct, no approximation is explicitly introduced by the technique, and it does not require to model the intruder,

but in the worst-case it has a time/space exponential cost;

- the static analysis technique gives us a partial result (thanks to Proposition 3.3) and it requires an explicit modeling of the intruder, but it has a (low) polynomial complexity [4].

Notice that in order to exploit the static analysis technique to verify integrity we had to prove Proposition 3.3. In fact, we had to prove that, thanks to particular syntactic hypothesis on the processes, the over-approximation is limited. We are presently investigating a possible generalization of this result not depending on the particular system under analysis.

The basic results we get with the two analyses are comparable: the intruders we found in Mobile Ambients correspond to the fact that in SPA without restriction the model is not secure. After getting such common results, we moved into different directions in order to analyze as many aspects as possible: in SPA we applied a restriction over channels and we added other details to the model; in Mobile Ambients we studied syntactic restrictions which allow us to discriminate a priori a large number of intruders.

The above considerations suggest a natural combination of the two techniques: apply the dynamic verification technique when the static analysis one does not prove the integrity.

Comparing our analysis with the results in [15] we notice that the model presented in [15] aims at representing the access control policy of SELinux, while our models are more abstract: we decide/infer that some channels have to be secure without describing how to implement them. As a consequence also the integrity results discussed in [15] are strongly related to the implementation of the system in SELinux, e.g., the system satisfies the integrity goal assuming that the administrators are trustworthy.

Acknowledgements

We thank Annalisa Bossi, Agostino Cortesi, Riccardo Focardi, Flaminia Luccio, and Sabina Rossi for their suggestions and comments. We also thank Riccardo Cannas, Stefano Filippone, and Enrico Pivato who were involved in the developing of the tools. Finally we thank Roberto Zunino for the useful discussion during the Mefisto meeting in Pisa.

References

[1] D. Bell. Lapadula Secure Computer System: Unified Exposition and Multics, 1975.

[4] It is not interesting to report here the time/space statistics shown by the tools, since this example is not large/complex enough to show significant differences.

[2] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying Persistent Security Properties. *Computer Languages, Systems and Structures*, 2003. To appear. Available at `http://www.dsi.unive.it/~srossi/cl03.ps.gz`.

[3] C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages*, 28(1):101–127, 2002.

[4] C. Braghin, A. Cortesi, R. Focardi, F.L. Luccio, and C. Piazza. Nesting Analysis of Mobile Ambients. *Computer Languages, Systems and Structures*, 2003. To appear.

[5] C. Braghin, A. Cortesi, R. Focardi, and S. van Bakel. Boundary Inference for Enforcing Security Policies in Mobile Ambients. In Ricardo Baeza-Yates, Ugo Montanari, and Nicola Santoro, editors, *Proc. IFIP Conference on Theoretical Computer Science (IFIP TCS'02)*, pages 383–395. Kluwer Academic Publishers, 2002.

[6] L. Cardelli and A. D. Gordon. Mobile Ambients. In M. Nivat, editor, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS'98)*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.

[7] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[8] P. Degano, F. Levi, and C. Bodei. Safe Ambients: Control Flow Analysis and Security. In Jifeng He and Masahiko Sato, editors, *Proc. of Advances in Computing Science - Asian Computing Science Conference, Penang, Malaysia (ASIAN'00)*, volume 1961 of *LNCS*, pages 199–214. Springer-Verlag, 2000.

[9] A. Dovier, C. Piazza, and A. Policriti. A Fast Bisimulation Algorithm. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. of Int. Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 79–90. Springer-Verlag, 2001.

[10] A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):488–528, October 2000.

[11] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.

[12] R. Focardi, R. Gorrieri, and F. Martinelli. Non Interference for the Analysis of Cryptographic Protocols. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Proc. of Int. Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 744–755. Springer-Verlag, 2000.

[13] R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. In *Proc. of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 307–319. IEEE Computer Society Press, 2002.

[14] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society Press, 1982.

[15] J. D. Guttman, A. L. Herzog, and J. D. Ramsdell. Information Flow in Operating Systems: Eager Formal Methods. Presented at Workshop on Issues in the Theory of Security 2003 (WITS'03).

[16] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. In A. Cortesi and G. File', editors, *Proc. of Static Analysis Symposium (SAS'99)*, volume 1694 of *LNCS*, pages 134–148. Springer-Verlag, 1999.

[17] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Validating Firewalls in Mobile Ambients. In J. C. M. Baeten and S. Mauw, editors, *Proc. of Int. Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 463–477. Springer-Verlag, 1999.

[18] J. K. Millen. "Finite-State Noiseless Covert Channels". In *Proc. of the Computer Security Foundations Workshop II*, pages 81–86. the MITRE Corporation, IEEE Computer Society Press, 1989.

[19] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[20] F. Nielson, R. R. Hansen, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming - Special Issue on Static Analysis edited by A. Cortesi and G. File'*, 47(2-3):145–175, 2003.

[21] R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[22] C. R. Tsai, V. D. Gligor, and C. S. Chandersekaran. "On the Identification of Covert Storage Channels in Secure Systems". *IEEE Transactions on Software Engineering*, pages 569–580, June 1990.