

## Chapter 1

# FORENSIC LEAK DETECTION FOR BUSINESS PROCESS MODELS

Rafael Accorsi and Claus Wonnemann

**Abstract** This paper presents a formal forensic technique based on information flow analysis to detect leaks in business processes models. The approach can be uniformly applied both for the analysis of process specifications and of the log files generated during processes' execution. Specifically, the special Petri net dialect **IFnet** provides a common basis for the formalization of isolation properties, the representation of business process specifications and their analysis. Focusing on the analysis of business process models, this paper illustrates the approach using a process from the eHealth setting.

**Keywords:** Business process forensics, Leak evidence, Information flow analysis.

## 1. Introduction

Forensic techniques for enterprise environments focus on the application layer, e.g. servers and browsers, and on the technical layer, e.g. virtual machines and operating systems. With the wide adoption of business process for automated enterprise operation, there is a substantial need to obtain business layer evidence. Indeed, up to 70% of the processes, including the core processes to manage ERP systems, customer-relationship and supply-chain, are operated in a fully automated manner. In contrast to the application and technical layers, tool support for business process forensics is missing.

Here, forensic auditors must gather evidence as to whether a process exhibits data and information leaks [11]. Specifically, they must demonstrate, on the one hand, the absence of harmful *data flows* across different enterprise domains. On the other hand, they might need to demonstrate that there has been no *information flow* over “covert channels”. Overall, demonstrating these properties is necessary to corroborate

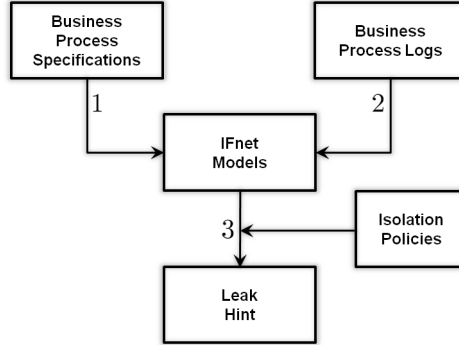


Figure 1. Approach for the formal analysis of business processes.

rate compliance with *isolation* properties [16] and its various instances in enterprise settings, e.g. Chinese wall [8] and separation of duties [6].

There are two complementary cases in which forensic auditors would enormously profit from automated tool support to gather such evidence. First, in the analysis of business process (BP) *specifications*. Second, in the automated analysis of the *log files* recording the execution of BP. The former is important as a means to demonstrate that a process allows for a leak; the latter is important to detect which leaks actually took place. Moreover, automated techniques are essential in highly flexible environments, such as Clouds and service networks, in which instances of the workflow may deliberately deviate from the original BP specification in a particular execution (so-called “multitenancy”).

This paper presents a well-founded, formal forensic technique that serves as a uniform basis for leak detection in both BP specifications and log files. Fig. 1 depicts the overall setting. Our technique employs a Petri net-based meta-model, the IFnet, as a formal underpinning tailored for the representation and leak detection analysis of BP. BP specifications are automatically translated into IFnet models (Arrow 1) [3]. Alternatively, process reconstruction techniques [25] are applied to mine IFnet models from business processes logs (Arrow 2). Given the representation of processes as IFnet models and isolation properties also expressed as Petri net patterns, information flow analysis based on Petri nets [9] is employed to detect data and information leaks (Arrow 3), thereby generating the corresponding evidence.

The use of a Petri net-based meta-model offers three advantages: first, it provides a uniform modeling formalism for a plethora of BP specification languages, such as BPMN, BPEL and EPC; second, it allows the well-founded formalization of structural isolation properties as Petri net

patterns [9]; and third, it provides a sound basis for the efficient isolation analysis, which boils down to determining whether a Petri net encompasses a leak pattern [13]. Moreover, the graphical notation and similarity with BP specification makes the approach suitable for the practical use for enterprise forensics.

**Related work.** The enterprise meta-model gives an abstraction to classify the previous work on the business process forensics. The model consists of three layers: the *business* layer contains BP specifications and business objects. The *application* layer provides for data objects and services. The *technical* layer contains software and hardware for service operation, i.e. the databases, operating systems and virtual machines.

Forensic techniques focus on the application and technical layers. Techniques for the application layer fall into the scope of network forensics [20], focusing on the choreography and usage of distributed web services [17]. Gunestas et al. introduce “forensic web services” that can securely maintain transaction records between web services [14]. Chandrasekaran et al. develop techniques for inferring sources of data leaks in document management systems [10]. The technical layer is the traditional terrain of computer forensics. In database systems, forensics attempt to detect, e.g., leaks in query answering [7], tampering attempts [19] and measure retention [22]. New aspects arise from the increasing virtualization of operating system abstractions [5], as well as the live evidence acquisition [15].

For the business layer, there are tools to analyze structural properties of the BP [1, 26], but no tools are available to obtain evidence on isolation properties. Focusing solely on data flows, Accorsi and Wonnemann [2] provide a forensic approach for log analysis based on propagation graphs (denoting how data items spread in the system). Sun et al. [23] provide a method for reasoning data flows and BP but do not relate it to security. Trčka et al. [24] provides a number of anti-patterns (formalized in Petri nets) which denote data flow flaws, but are not related to isolation or security. Atluri et al. [4] provide a model for the analysis of Chinese wall policies but do not address the detection of leaks.

Covert channels are channels not intended to transfer information but misused to this end [16]. While relevant for isolation properties [18], covert channels do not lie into the scope of forensic analysis of BP. The approach proposed in this paper aims to bridge this gap and provide powerful, automated tools for the BP analysis of both data and information flows.

**Paper structure.** §1.2 provides an overview of the approach. §1.3 presents the Petri net-based meta-model employed to model BP and isolation properties, as well as the verification procedures. This is illus-

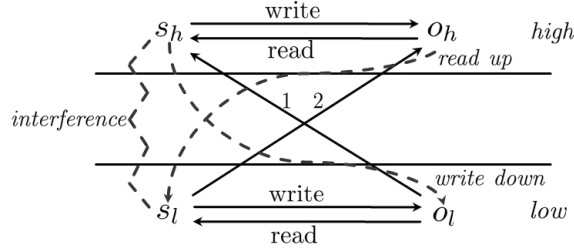


Figure 2. Security levels and leaks.

trated with a running example. §1.4 summarizes the paper and indicates possible extensions.

## 2. Approach Overview

There are two causes for leaks: *Data flows* characterize the direct access to pieces of data over legitimate channels, e.g. a request that circumvents the execution monitor and violates an access control policy. *Information flows* denote the indirect access of information over covert channels, which allow an attacker to derive confidential information.

The approach this paper presents aims to detect leaks that may arise from the interaction of different subjects with BP. In its simplest form, the abstract system model considers subjects  $s$  separated into two security classes (with regard to a particular object  $o$ ): *high* for those able to access  $o$ , *low* for those that cannot access  $o$ . Subjects in both classes may interact with the BP. An *information leak* happens when a “low” subject obtains information meant to be only visible to “high” subjects.

Fig. 2 depicts this setting. Formally, this model integrates mandatory access control [21] with information flow control [12]. The solid arrows denote the legitimate data flows. Within a security level, subjects may read and write to objects in that level. Besides that, high subjects  $s_h$  may read lower level objects  $o_l$  (Arrow 1) and low subjects  $s_l$  may write high objects  $o_h$  (Arrow 2). Forbidden data flows, denoted as dashed arrows, are those in which high subjects write low objects (write down) and those in which low subjects read high objects (read up). Further, the low subjects may perform observations and, knowing the operation of the BP, combine this information to derive classified information. This denotes an *interference*.

**Leak detection.** The approach to detect such leaks in BP encompasses three steps, as depicted in Fig 1. First, BP specifications or, alternatively, logs are translated into IFnet models suitable for the specification and analysis of isolation properties. In the second step, the resultant

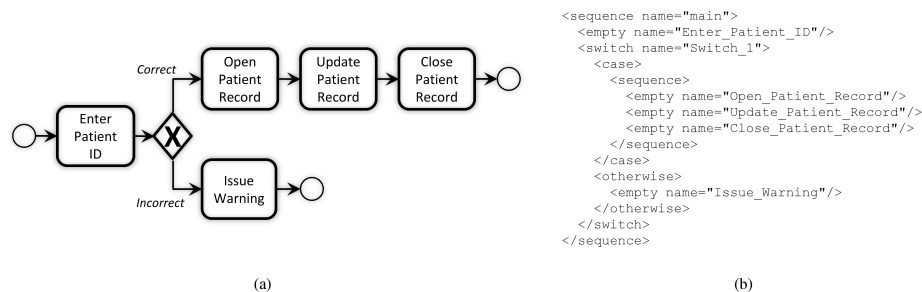


Figure 3. Update Patient Record process model and its BPEL specification

IFnet model is labeled with security levels “high” and “low” for the subsequent analysis. With Petri net patterns capturing illegal data and information flows, algorithms detect whether the patterns are active, i.e. reachable in the net representing the BP.

**Running example.** Fig. 3 presents the “Update Patient Record” BP and the corresponding BPEL code, which acts as a running example below. Albeit simple, it is a central fragment that recurs in several of the hospital’s information systems, e.g. for accounting, medical treatment and billing. It consists of five activities (boxes) and an exclusive choice (x-diamond) that denotes an if-statement. The goal of the forensic analysis is to detect whether this BP leaks information and, if so, over which channels this happens.

The choice for an eHealth scenario is interesting insofar as it is a setting in which strict confidentiality requirements hold, e.g. HIPAA privacy rules for electronic health records. Moreover, due to the necessary interconnection of different parties, e.g. hospitals, laboratories, and pharmacies, business processes play a relevant role for automation. Since each party has a difference clearance with regard to the data they can access, analyzing these processes is important.

### 3. The Approach and its Components

This section presents the main components of the approach and illustrates each component with the running example presented in §1.2.

#### 3.1 IFnet and Translations

The IFnet is an extension of (colored) Petri nets tailored for the specification and analysis of isolation properties in BP. Due to space con-

straints, this paper provides only a brief introduction to IFnet. See [3] for details.

**Petri nets.** This section introduces Colored Petri Net (CPN), on top of which IFnet is defined. CPN generalizes standard Petri net to support distinguishable tokens and which is used as the basis for the IFnet. Following the standard terminology, tokens are distinguished by their *color*, which is an identifier from the universe  $\mathcal{C}$  of token colors.

A CPN is a tuple  $N = (P, T, F, C, I, O)$ , where  $P$  is a finite set of *places*,  $T$  is a finite set of *transitions* such that  $P \cap T = \emptyset$ , and  $F \subseteq (P \times T) \cup (T \times P)$  is a set of directed arcs, called the *flow relation*. Let  $x, y \in (P \cup T)$ ,  $xFy$  denotes that there is an arc from  $x$  to  $y$ . The functions  $C$ ,  $I$  and  $O$  define the *capacity* of places and the *input* and *output* of transitions, respectively:

- The capacity function  $C$  defines the number of tokens a place can hold at a time:  $C \in P \rightarrow \mathbb{N}$ .
- The input function  $I$  defines for each transition  $t$ , each place  $i$  with  $iFt$  and each token color  $c$  the number of tokens that is expected:  $I \in T \times P \times \mathcal{C} \rightarrow \mathbb{N}$ .
- The output function  $O$  defines for each transition  $t$ , each place  $o$  with  $tFo$  and each token color  $c$  the number of produced tokens:  $O \in T \times P \times \mathcal{C} \rightarrow \mathbb{N}$ .

A place contains zero or more tokens. The *marking* (or *state*) is the distribution of tokens over places. A marking  $M$  is a *bag* over the Cartesian product of the set of places and the set of token colors, i.e. a function from  $P \times \mathcal{C}$  to the natural numbers:  $M \in P \times \mathcal{C} \rightarrow \mathbb{N}$ . A partial ordering is defined to compare states with regard to the number of tokens in places. For any two states  $M_1$  and  $M_2$ ,  $M_1 \leq M_2$  iff for all  $p \in P$  and for all  $c \in \mathcal{C}$ :  $M_1(p, c) \leq M_2(p, c)$ . The sum of two bags ( $M_1 + M_2$ ), the difference ( $M_1 - M_2$ ) and the presence of an element in a bag ( $a \in M_1$ ) are defined in a straightforward way. A *marked* CPN is a pair  $(N, M)$ , where  $N = (P, T, F, C, I, O)$  is a CPN and  $M$  is a bag over  $P \times \mathcal{C}$  denoting the marking of the net.

Elements of  $P \cup T$  are called *nodes*. A node  $x$  is an *input node* of another node  $y$  iff there is a directed arc from  $x$  to  $y$  (i.e.  $xFy$ ). Node  $x$  is an *output node* of  $y$  iff  $yFx$ . For any  $x \in P \cup T$ ,  $\overset{N}{\bullet}x = \{y \mid yFx\}$  and  $x \overset{N}{\bullet} = \{y \mid xFy\}$ ; the superscript  $N$  can be omitted if it is clear from the context.

The number of tokens may change during the execution of the net. Transitions are the active components in a CPN. They change the state of the net according to the following *firing rule*:

- 1 A transition  $t \in T$  is *enabled* in state  $M_1$  *iff* each input place contains sufficiently many tokens of each color and each output place has sufficient capacity to contain the output tokens:
  - $\forall i \in \bullet t, \forall c \in \mathcal{C} : I(t, i, c) \leq M_1(i, c)$  and
  - $\forall o \in t\bullet : \sum_{c \in \mathcal{C}} O(t, o, c) + \sum_{c \in \mathcal{C}} M_1(o, c) \leq C(o)$ .
- 2 An enabled transition may *fire*. If transition  $t$  fires, then  $t$  *consumes* the designated number of tokens from each of its input places and *produces* the designated number of tokens for each of its output places. Firing of transition  $t$  in state  $M_1$  results in some state  $M_2$  which is defined as follows:
  - $\forall i \in \bullet t, \forall c \in \mathcal{C} : M_2(i, c) = M_1(i, c) - I(t, i, c)$  and
  - $\forall o \in t\bullet, \forall c \in \mathcal{C} : M_2(o, c) = M_1(o, c) + O(t, o, c)$  and
  - $\forall p \in P \setminus (\bullet t + t\bullet), \forall c \in \mathcal{C} : M_2(p, c) = M_1(p, c)$ .

Given a CPN  $N$  and a state  $M_1$ , the following notation is defined:

- $M_1 \xrightarrow{t} M_2$ : transition  $t$  is enabled in  $M_1$  and firing  $t$  in  $M_1$  results in state  $M_2$ .
- $M_1 \longrightarrow M_2$ : there is a transition  $t$  such that  $M_1 \xrightarrow{t} M_2$ .
- $M_1 \xrightarrow{\sigma} M_n$ : the firing sequence  $\sigma = t_1 t_2 t_3 \dots t_{n-1}$  from state  $M_1$  leads to state  $M_n$  via a (possibly empty) set of intermediate states  $M_2, \dots, M_{n-1}$ , i.e.  $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} M_n$ .

$M_n$  is *reachable* from  $M_1$  (notation  $M_1 \xrightarrow{*} M_n$ ) *iff* there is a firing sequence  $\sigma$  so that  $M_1 \xrightarrow{\sigma} M_n$ . The empty firing sequence is also allowed, i.e.  $M_1 \xrightarrow{*} M_1$ . The set of states reachable from state  $M_1$  is denoted  $[M_1]$ .

**IFnet extension.** IFnet refines CPN by adding constructs required for BP modeling and IF analysis. An IFnet models BP activities through transitions and data items (including documents, messages, variables) through tokens. Tokens with color `black` (“black tokens”) have a special status. They do not stand for data items but indicate the triggering and termination of activities. The set of *colored tokens* (i.e. tokens that are not `black`) is denoted  $\mathcal{C}_c$ , i.e.  $\mathcal{C}_c = \mathcal{C} \setminus \{\text{black}\}$ .

Formally, an **IFnet** is a tuple  $N = ((P, T, F, C, I, O), S_{\mathcal{U}}, A, G, L_{\mathcal{SC}})$ , where  $(P, T, F, C, I, O)$  is a CPN and the further elements are as follows:

- The function  $S_{\mathcal{U}}$  assigns transitions *subjects* from a set  $\mathcal{U}$ :  $S_{\mathcal{U}} \in T \rightarrow \mathcal{U}$ . A subject denotes the acting entity on which behalf a corresponding BP activity is performed.
- The function  $A$  defines whether a transition  $t$  reads or writes an input datum  $i \in \bullet t$ :  $A \in T \times \mathcal{C}_c \rightarrow \{read, write\}$ .
- The function  $G$  assigns predicates (guards) to transitions:  $G \in T \rightarrow \mathcal{P}_{\mathcal{C}}$ , where  $\mathcal{P}_{\mathcal{C}}$  denotes the set of predicates over colored tokens. A predicate evaluates to either *true* or *false* and is denoted, for instance,  $\mathbf{p}(\mathbf{red}, \mathbf{green})$ , where  $\mathbf{p}$  is the name of the predicate and the identifiers in brackets indicate the tokens needed for its evaluation. For an enabled transition to fire, its guard must evaluate to *true*.
- The function  $L_{\mathcal{SC}}$  assigns security labels to transitions and colored tokens:  $L_{\mathcal{SC}} \in T \cup \mathcal{C}_c \rightarrow \mathcal{SC}$ .  $\mathcal{SC}$  is a finite set of security labels which forms a lattice under the relation  $\prec$ . Every set  $\mathcal{SC}$  contains an additional element `unlabeled`, which denotes that a transition or token does not hold a label.

Furthermore, an **IFnet** must meet the following structural conditions (we refrain from a formal definition here for simplicity). The first two conditions ensure that a BP has defined start and end points. The third condition prevents that there are “dangling” transitions or activities which do not contribute to the processing of the BP. The fourth condition requires transitions to signal their triggering and termination through black tokens. The last condition ensures that data items are passed through the BP according to the transitions.

**Translation to IFnet models.** The translation from BP specifications to **IFnet** models occurs automatically for BP specified in BPMN and BPEL. It consists of two steps: first, the structure of the process is translated in a **IFnet** model. Subsequently, the net is labeled for analysis, i.e. activities, places and resources are annotated with security labels (“high” and “low”). While the first step runs in a fully automated manner, for the latter we developed a number of strategies to automatically derive the labels. Below we show one strategy for the running example.

**Labeling strategy and running example.** A useful, fully-automated labeling strategy consists of “unfolding” an **IFnet** model as a means to investigate the interaction of two subjects (one each “high” and “low”) with the BP. Formally, for a marked Petri net  $(N, M) = ((P, T, F), M)$

and a resource relation  $\mathcal{D}$ , the corresponding **IFnet** is a tuple  $(P_L \cup P_H \cup P_{\mathcal{D}}, T_L \cup T_H, F_L \cup F_H \cup F_{\mathcal{D}}, M_L + M_H + M_{\mathcal{D}})$  where:

- $((P_L, T_L, F_L), M_L)$  corresponds to the net  $((P, T, F), M)$ .
- $((P_H, T_H, F_H), M_H)$  is an equivalent net to  $((P_L, T_L, F_L), M_L)$  with its elements renamed for distinction. The function  $\Upsilon_{T_L \rightarrow T_H} : T_L \rightarrow T_H$  maps the transitions from  $T_L$  to their counterparts in  $T_H$ .
- $P_{\mathcal{D}}$  is a set of places which model the blocking of resources. There exists exactly one  $p \in P_{\mathcal{D}}$  for each pair  $(t_0, t_1) \in \mathcal{D}$ . The function  $\Upsilon_{P_{\mathcal{D}} \rightarrow \mathcal{D}} : P_{\mathcal{D}} \rightarrow \mathcal{D}$  maps the places from  $P_{\mathcal{D}}$  to the corresponding pair of transitions in  $T$  (and therewith in  $T_L$ ).
- $M_{\mathcal{D}}$  denotes the initial marking of places  $P_{\mathcal{D}}$ .  $M_{\mathcal{D}}$  marks each  $p \in P_{\mathcal{D}}$  with exactly one token.
- $F_{\mathcal{D}}$  denotes the arcs which connect the places in  $P$  to the blocking and releasing transitions in  $T_L$  and  $T_H$ . For each  $p \in P_{\mathcal{D}}$  with  $\Upsilon_{P_{\mathcal{D}} \rightarrow \mathcal{D}}(p) = (t_0, t_1)$  it contains the following arcs:
  - $(p, t_0)$  denotes the blocking of the resource modeled by  $p$  through transition  $t_0$ .
  - $(t_1, p)$  denotes the release of that resource by transition  $t_1$ .
  - $(p, \Upsilon_{T_L \rightarrow T_H}(t_0))$  denotes the blocking of resource  $p$  through the transition in  $T_H$  which corresponds to  $t_0$ .
  - $(\Upsilon_{T_L \rightarrow T_H}(t_1), p)$  denotes the corresponding release of the resource.

This strategy, as well other strategies to obtain labels from access control lists and role-based access control policies, has been mechanized. Fig. 4 depicts the **IFnet** obtained from the BP in Fig. 3. The record is a resource shared by both the high subject (upper part of the net) and the low subject (lower part). The resultant **IFnet** is the subject of analysis.

### 3.2 Isolation Policies

Isolation policies formalize confidentiality properties. These are safety properties denoting leaks that shall not happen between the high and the low subjects (see Fig. 2). In our approach, these properties are captured using **IFnet** patterns. Below we demonstrate patterns that capture data flow and information flow violations. It is important to notice that these patterns stand for *extensional* policies, i.e. policies that capture

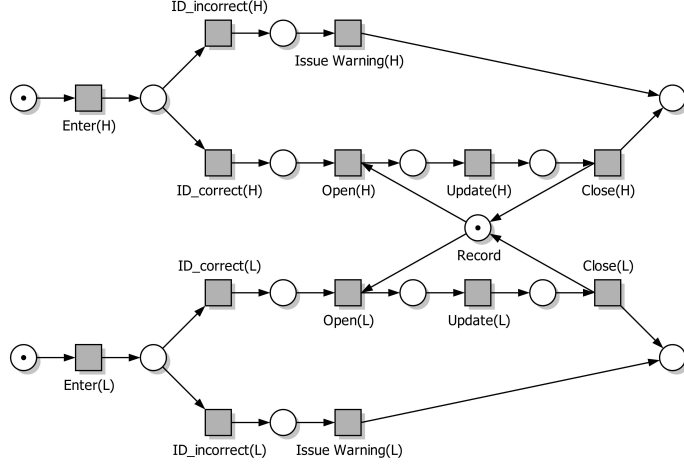


Figure 4. IFnet of the Update Patient Record BP.

leaks independently of the actual BP at hand. This allows us to compile a library of patterns and, depending on the concrete investigation purpose, flexibly select the patterns. *Intensional* policies, the counterpart of extensional policies, capture BP specific properties and would not be suitable for thorough isolation analysis.

**Data flow patterns.** These patterns generally capture the direct data leaks that happen in two situations: first, whenever a high subject *writes* on a low-level object; and second, whenever a low subject *reads* a high-level object. Correspondingly, the patterns formalize these leaks as IFnet patterns. As an example, the pattern in Fig. 5(a) formalizes the “write down” rule: The resource  $a$  (grey token) is written by high and then read by low. If reachable in an IFnet, this denotes a leak.

These policies capture mandatory access control policies over a lattice-based information flow model. With these policies, our approach can capture a number of industrial requirements, including Bell-LaPadula and Chinese wall models, as well as binding and separation of duties. Although intuitively correct, ongoing work is establishing the formal correspondence between the patterns and the original lattice-based policies.

**Information flow patterns.** The idea is to use patterns capture different interferences between the activities of high and low subjects. Each interference allows low subjects to derive information about high. Formally, patterns capture well-founded bisimulation-based information flow properties specified in terms of process algebra.

Busi and Gorrieri [9] demonstrate the correspondence of some patterns. For example, together, the patterns in Fig. 5(b) and (c) are

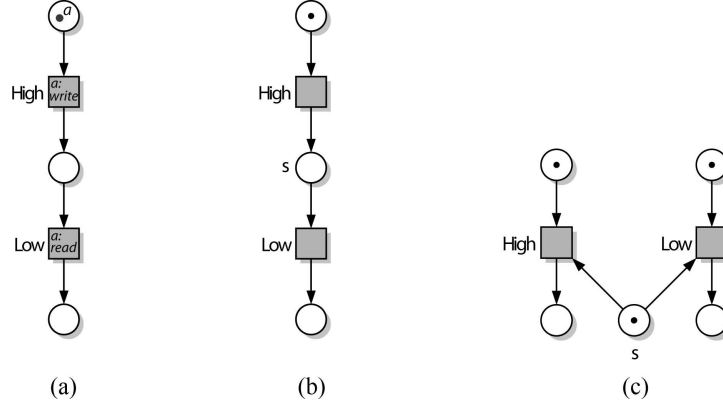


Figure 5. Isolation properties as IFnet patterns.

necessary, albeit apparently not sufficient, to capture the bisimulation-based non-deducibility property, which prohibits low from deriving *any* aspect of high. Intuitively, the place  $s$  in Fig. 5(b) stands for a *causal place*: whenever high fires an activity, low is able to observe it. The place  $s$  in Fig. 5(c) denotes a *conflict place*: both high and low compete for the control flow token in  $s$ . If high obtains it, low can derive that high has performed the corresponding activity. We have designed further patterns to capture other, both stronger and weaker bisimulation-based properties. Ongoing work aims at demonstrating equivalence between patterns and properties based on process algebra.

It is important to note that the non-interference properties capture *possibilistic* information leaks, i.e. they indicate BP vulnerabilities that allow for the derivation of information. Hence, they are weaker than data leaks, which indicate a concrete illegal data flow. Still, possibilistic statements are useful for forensic analysts as a means to substantiate a suspicion, indicating the possible source of an information leak.

### 3.3 Leak Detection Analysis

With both BP and policies formalized in the IFnet meta-model, the leak detection algorithms checks whether the policy patterns are reachable in the BP model. Focusing on causal and conflict places, this section overviews the corresponding verification procedure and shows that the running example exhibits information and data leaks.

**Verification procedure.** The verification procedure consists of two steps. The first step checks whether the BP model exhibits a harmful (here: causal and/or conflict place). If so, the second step determines

whether the detected places are reachable during an execution of the net. While the first step is a *static* check of the net, the second is *dynamic* and requires the analysis of the whole “state-space”, i.e. marking graph, generated by the BP model.

Let us assume that a BP model, i.e. IFnet,  $M$  exhibits a causal place  $s$ . The following informally outlines the decision procedure to perform the dynamic check. First, the marking graph is generated.

- 1 create the marking list  $L$  and add initial marking
- 2 for each marking  $m$  in  $L$ 
  - 2.1 for each enabled transition  $t$ 
    - compute the marking  $m'$  reachable from  $m$  by firing  $t$
    - if  $m'$  is not in  $L$ , add  $(m', \text{emptylist})$  to  $L$
    - add to the list associated to  $m$  the new pair  $(t, m')$

Hence, for each marking in the list, the procedure computes the reachable marking of every enabled transition and adds the corresponding pair to the currently examined marking. When the procedure finds a new marking, it adds it to the queue to be examined in later cycles.

Given the marking graph, the following procedure detects whether a causal place is active in this graph.

- 1 search the graph and select only markings reached by high transitions in the preset  $s$
- 2 for each such marking  $m$ 
  - create a new marking graph rooted in  $m$  and restricted on all transitions containing  $s$  in their postset, i.e. set of outgoing nodes
  - search among its markings for one enabling a low transition in the postset of  $s$

If found, add  $s$  to the list of *active* causal places.

Intuitively, the procedure traverses the marking graph, trying to find markings in which the potential causal place  $s$  reached by a high transition and, subsequently, leading to a low transition. If these conditions are met by  $s$ , then it is an active causal place.

The procedure to detect active conflict places is similar and omitted here due to space constraints. Currently, each pattern requires a special verification algorithm. Ongoing work generalizes and improves the performance of these algorithms, parameterizing them by policy patterns.

**Running example.** The IFnet in Fig. 4 exhibits both data and information leaks. A data leak violating the policy in Fig. 5(a) happens when high subjects updates a patient record (first execution of the BP) and, subsequently, a low subject opens the record. In this case, high has “written down” and, thus, leaked data to low.

The information leak occurs over the place labeled with “record”.

- 1 Firing of transition  $\text{Open(H)}$ , the patient record is removed from the storage place and transition  $\text{Open(L)}$  is blocked until the token is returned. Here, the “High” part of the net influences the “Low” part as it prevents the transition from firing. There is an information flow (through a so-called *resource exhaustion channel*) which allows the “Low” part to deduce that “High” currently holds the patient record.
- 2 Firing of transition  $\text{Close(H)}$ , the token representing the patient record is returned to its storage place and might be consumed by transition  $\text{Open(L)}$ . Hence, opening the patient record on the “Low” side requires its preceding return on the “High” side: this causality reveals to “Low” the fact that “High” has returned the record.

Further derivations, e.g. with regard to the time and duration of update, are also possible, however their concrete semantics depends on the purpose of evidence generation.

#### 4. Summary

This paper presented an approach for the formal forensic analysis of BP and exemplified its applicability with a simple running example focusing. The approach presented in the paper is based on IFnet, a meta-model tailored for the analysis of data and information leaks. While this paper focused on the evidence generation for existing BP specification, the approach we put forward is equally applicable for the analysis of log files generated by BP executions. For this purpose, process reconstruction algorithms are being investigated to mine IFnet models suitable for analysis. In particular, we are extending process reconstruction algorithms to produce not only a *single* model of the process, but a series of different models, thereby producing forensic tools more suitable to cope with the multitenancy approach present in cloud and grid environment. In particular, through the consideration of runtime information, we expect to be able to analyze other isolation properties based on the dynamics of the execution.

## References

- [1] R. Accorsi and L. Lowis. ComCert: Automated certification of cloud-based business processes. *ERCIM News*, (83):50–51, 2010.
- [2] R. Accorsi and C. Wonnemann. Auditing workflow executions against dataflow policies. *Business Information Systems*, vol. 47 of *LNBIP*, pp. 207–217. Springer, 2010.
- [3] R. Accorsi and C. Wonnemann. Information flow analysis of business processes for confidentiality requirements. *Workshop on Security and Trust Management*, to appear in *LNCS*. Springer, 2011.
- [4] V. Atluri, S. A. Chun, and P. Mazzoleni. A Chinese Wall security model for decentralized workflow systems. *ACM Conference on Computer and Communications Security*, pp. 48–57. ACM, 2001.
- [5] D. Bem. Virtual machine for computer forensics – the open source perspective. In *Open Source Software for Digital Forensics*, pp. 25–42. Springer, 2010.
- [6] R. Botha and J. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.
- [7] S. Böttcher and R. Steinmetz. Finding the leak: A privacy audit system for sensitive XML databases. *ACM Privacy Data Management*, pp. 100–110, 2006.
- [8] D. Brewer and M. Nash. The Chinese-wall security policy. *IEEE Symposium on Security and Privacy*, pp. 206–214, 1989.
- [9] N. Busi and R. Gorrieri. Structural non-interference in elementary and trace nets. *Mathematical Structures in Computer Science*, 19(6):1065–1090, 2009.
- [10] M. Chandrasekaran, V. Sankaranarayanan, and S. J. Upadhyaya. Inferring sources of leaks in document management systems. *IFIP Conference on Digital Forensics*, vol. 285 of *IFIP*, pp. 291–306. Springer, 2008.
- [11] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. *ACM Workshop on Cloud Computing Security*, pp. 85–90. 2009.
- [12] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.
- [13] S. Frau, R. Gorrieri, and C. Ferigato. Petri net security checker: Structural non-interference at work. *Formal Aspects in Security and Trust*, vol. 5491 of *LNCS*, pp. 210–225. Springer, 2008.

- [14] M. Gunestas, D. Wijsekera, and A. Singhal. Forensic web services. *IFIP Conference on Digital Forensics*, volume 285 of *IFIP*, pp. 163–176. Springer, 2008.
- [15] B. Hay, M. Bishop, and K. Nance. Live analysis: Progress and challenges. *IEEE Security & Privacy*, 7(2):30–37, 2009.
- [16] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.
- [17] L. Lowis and R. Accorsi. Finding vulnerabilities in SOA-based business processes. To appear in *IEEE Transactions on Service Computing*, 2011.
- [18] J. McHugh. *Handbook for the Computer Security Certification of Trusted Systems*, chapter Covert Channel Analysis, pp. 4–78. US Navy, 1995.
- [19] K. Pavlou and R. Snodgrass. Forensic analysis of database tampering. *ACM Transactions on Database Systems*, 33(4), 2008.
- [20] M. Ponec, P. Giura, H. Brönnimann, and J. Wein. Highly efficient techniques for network forensics. *ACM Conference on Computer and Communications Security*, pp. 150–160. 2007.
- [21] R. Sandhu and P. Samarati. Authentication, access control, and audit. *ACM Computing Surveys*, 28(1):241–243, 1996.
- [22] P. Stahlberg, G. Miklau, and B. Levine. Threats to privacy in the forensic analysis of database systems. *Conference on Management of Data*, pp. 91–102. ACM, 2007.
- [23] S. Sun, L. Zhao, J. Nunamaker, and O. L. Sheng. Formulating the data-flow perspective for business process management. *Information Systems Research*, 17(4):374–391, 2006.
- [24] N. Trčka, W. van der Aalst, and N. Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. *Advanced Information Systems Engineering*, volume 5565 of *LNCS*, pp. 425–439. Springer, 2009.
- [25] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
- [26] W. van der Aalst, K. van Hee, J. M. van der Werf, and M. Verdonk. Auditing 2.0: Using process mining to support tomorrow’s auditor. *IEEE Computer*, 43(3):90–93, 2010.