



HAL
open science

Monitoring Information flow by Diagnosis Techniques

Jérémy Dubreil, Thierry Jéron, Hervé Marchand

► **To cite this version:**

Jérémy Dubreil, Thierry Jéron, Hervé Marchand. Monitoring Information flow by Diagnosis Techniques. [Research Report] PI 1901, 2008, pp.15. inria-00312747v1

HAL Id: inria-00312747

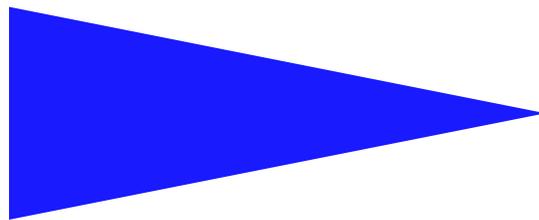
<https://inria.hal.science/inria-00312747v1>

Submitted on 26 Aug 2008 (v1), last revised 26 Aug 2008 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PUBLICATION
INTERNE
N° 1901



MONITORING INFORMATION FLOW BY DIAGNOSIS TECHNIQUES

JÉRÉMY DUBREIL, THIERRY JÉRON, HERVÉ MARCHAND

Monitoring Information flow by Diagnosis Techniques

Jérémy Dubreil, Thierry Jéron, Hervé Marchand

Systèmes communicants
 Projets VerTeCs

Publication interne n1901 — August 2008 — 15 pages

Abstract: In this paper, we are interested in constructing monitors for the detection of confidential information flow in the context of partially observable discrete event systems. We focus on the case where the secret information is given as a regular language. We first characterize the set of observations allowing an attacker to infer the secret behaviors. We consider the general case where the attacker and the administrator have different partial views of the system. Further, based on the diagnosis of discrete event systems, we provide necessary and sufficient conditions under which detection and prediction of secret information flow can be ensured and a construction of a monitor ensuring this task.

Key-words: security, opacity, discrete event systems, partial observation, diagnosis, on-line detection.

(Résumé : tsvp)

Supervision de fuites d'information via des techniques de diagnostic

Résumé : Nous nous intéressons à la construction de moniteurs permettant de détecter la fuite d'information confidentielle pour des systèmes partiellement observables, modélisés par des systèmes de transition finis. Nous considérons le cas où le secret peut se modéliser par des langages réguliers. Nous commençons par définir la notion d'opacité pour formaliser la fuite d'information et caractérisons l'ensemble des observations pour lesquelles un attaquant infère de l'information confidentielle. Ensuite, considérant le cas général où l'attaquant et l'administrateur ont des vues partielles potentiellement différentes du système, nous adaptons les techniques de diagnostic sur des systèmes à événement discrets, nous explicitons des conditions nécessaires et suffisantes sur le système pour permettre la détection et/ou la prédiction de cette fuite d'information et construisons un moniteur permettant un administrateur d'assurer cette détection. Nous considérons

Mots clés : Sécurité, opacité, systèmes à événements discrets, observation partielle, diagnostic, détection en ligne

1 Introduction

There has been an increasing interest in research about computer security in the past decades. Indeed, the emergence of web services and the improvements of the possibilities of mobile and embedded systems allow lots of new and interesting features. But some of these services such as online payment, medical information storage or e-voting system may deal with some critical information. In the meantime, having more applications and devices for accessing these services also increases the possibilities for such information to flow. To avoid security breach, using automatic tools based on formal methods for security analysis can be beneficial. In this context, there has been a growing interest in verification [3, 10] and testing of security properties [5] and monitoring security properties [12] in past years. In order to specify such automatic analysis methods, security properties are generally separated into three different categories: *availability* (a user can always perform the actions that are allowed by the security policy), *integrity* (something illegal cannot be performed by a user) and *confidentiality* (some secret information cannot be inferred by a user) [4].

In this paper, we focus on confidentiality and more particularly on the notion of opacity as defined in [4]. The general problem of confidentiality consists of determining whether an attacker having only a partial observations of the system, is able or not to discover some secret behaviors (e.g. a password stored in a file, the value of some hidden variables, etc) occurring during execution. The motivation of this paper is to provide an analysis method for detecting information flows. Therefore we proceed first from an attacker point of view, for generating the set of possible attacks, and second from the administrator point of view interested in monitoring this set of attacks.

Overview of the problem. We consider three components: a system G , an attacker \mathcal{A} and a monitor \mathcal{M} (modelling for example the administrator of the system or an intrusion detection system) (C.f. Figure 1). We assume that the system G is modeled by a finite transition system. Users interact with G through an interface $\Pi_{\mathcal{A}}$, corresponding to the inputs/outputs of the system.

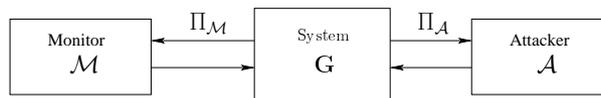


Figure 1: Architecture

For this system, one can define some confidentiality policies. Following the approach of [8] for the diagnosis and [1, 4], a secret is modeled by a property φ given as a regular language over the alphabet Σ of the system G . The secret is preserved as far as the attacker cannot surely infer that the property φ is satisfied by the current execution of the system based on the observations performed through the interface $\Pi_{\mathcal{A}}$. We characterize the set of observations allowing the attacker \mathcal{A} to infer the secret information. *A contrario*, the monitor \mathcal{M} tries to analyze the information flow between the system G and the attacker \mathcal{A} in order to raise an alarm whenever the secret has been revealed. \mathcal{M} can also try to predict the information flow. To do so, we assume that \mathcal{M} knows the power of the attacker (i.e. he knows the model of the system G and the interface $\Pi_{\mathcal{A}}$ of the attacker). He observes the system through the interface $\Pi_{\mathcal{M}}$ (we do not assume any link between the two interfaces). Further, based on the set of observations allowing the attacker to infer the secret information, we provide necessary and sufficient conditions under which detection and prediction of secret information flow can be ensured, and construct a monitor \mathcal{M} allowing an administrator to detect the attacks. This supervision is performed on-line, the monitor raising an alarm whenever an information flow occurs.

The structure of the document is as follows: In section 2, we define the mathematical terminology and notions used throughout the paper. In Section 3, we show how to build a monitor in charge of the supervision of the system according to a given property. In Section 4, we define the notion of opacity

formalizing information flow. With this notion, we can characterize the set of observations for which an attacker can infer confidential information. In Section 5, we use diagnosis techniques to exhibit necessary and sufficient conditions under which a monitor can diagnose and/or predict the information flow. Finally, we study in Section 6 how to deal with abstractions.

2 Models & Notations

Let Σ be a finite alphabet of events. A *string* is a finite-length sequence of events in Σ . ϵ denotes the empty string. Given a string s , the length of s is denoted by $|s|$. The set of all strings formed by events in Σ is denoted by Σ^* . Any subset of Σ^* is called a *language* over Σ . Let L be a language over Σ . Given a string $s \in L$, $L/s \triangleq \{t \in \Sigma^* \mid s.t \in L\}$ is called the *post-language* of L after s and defined as L/s . L is said to be *extention-closed* when $L.\Sigma^* = L$. We assume that the systems are modeled as Labelled Transitions Systems (LTS for short). The formal definition of an LTS is as follows.

Definition 1 (LTS) An LTS over Σ is defined by a 4-tuple $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$ where Q_G is a finite set of states, Σ is the set of events of G , $q_G^0 \in Q_G$ is the initial state, and $\rightarrow_G \subseteq Q_G \times \Sigma \times Q_G$ is the partial transition relation. \diamond

Notations In the remainder of this section, we consider a given LTS $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$.

- We write $q \xrightarrow{a}_G q'$ if $(q, a, q') \in \rightarrow_G$ and $q \xrightarrow{a}_G$ for $\exists q' \in Q_G, q \xrightarrow{a}_G q'$. We extend \rightarrow_G to arbitrary sequences by setting: $q \xrightarrow{\epsilon}_G q$ for all states q , and $q \xrightarrow{s\sigma}_G q'$ whenever $q \xrightarrow{s}_G q''$ and $q'' \xrightarrow{\sigma}_G q'$, for some $q'' \in Q_G$.
- $\Sigma(q) \triangleq \{a \in \Sigma \mid q \xrightarrow{a}_G\}$ corresponds to the set of events admissible in state q of G . G is said to be *complete* whenever $\forall q \in Q_G, \Sigma(q) = \Sigma$. It is said to be *live* if $\Sigma(q) \neq \emptyset$, for each $q \in Q_G$.
- We set $\Delta_G(q, l) \triangleq \{q' \in Q_G \mid q \xrightarrow{l}_G q'\}$. By a slight abuse of notation, for any language $L \subseteq \Sigma^*$, $\Delta_G(q, L) \triangleq \{q' \in Q_G \mid \exists s \in L, q \xrightarrow{s}_G q'\}$. For any $X \subseteq Q_G$, $\Delta_G(X, L) = \bigcup_{q \in X} \Delta_G(q, L)$. Also, X is said to be *stable* if $\Delta_G(X, \Sigma^*) \subseteq X$.
- We denote by $L(G) = \{l \in \Sigma^*, q_0 \xrightarrow{l}_G\}$ the set of trajectories of the system G .

Given a special set of states $F_G \subseteq Q_G$, the notions above are extended in this setting by letting the language $L_{F_G}(G) = \{l \in \Sigma^* \mid \exists q \in F_G, q_0 \xrightarrow{l}_G q\}$ be the set of trajectories that end in a state of F_G . Note that F_G is stable if $L_{F_G}(G)$ is extention-closed. Also, if G is complete and F_G is stable, then $L_{F_G}(G)$ is extention-closed.

We now define the synchronous product of two LTSs.

Definition 2 (Synchronous product) Let $G^i = (Q^i, \Sigma, \rightarrow_{G^i}, q_{G^i}^0)$, $i = 1, 2$ be two LTSs. The synchronous product between G^1 and G^2 is an LTS $G^1 \times G^2 = (Q^1 \times Q^2, \Sigma, \rightarrow_{G^1 \times G^2}, (q_{G^1}^0, q_{G^2}^0))$, where $(q^1, q^2) \xrightarrow{\sigma}_{G^1 \times G^2} (q'^1, q'^2)$ whenever $q^1 \xrightarrow{\sigma}_{G^1} q'^1$ and $q^2 \xrightarrow{\sigma}_{G^2} q'^2$.

Clearly, $L(G^1 \times G^2) = L(G^1) \cap L(G^2)$ and for $F_i \subseteq Q^i, i = 1, 2$, we also have $L_{F_1 \times F_2}(G^1 \times G^2) = L_{F_1}(G^1) \cap L_{F_2}(G^2)$. Also, if for $i = 1, 2$ the set F_i is stable in G^i , $F_1 \times F_2$ is stable in $G^1 \times G^2$.

Given a set of states $E \subseteq Q_G$ of an LTS G , the operators pre_G^\forall et pre_G^\exists are defined as follows:

$$\begin{aligned} Pre_G^\exists(E) &= \{q \in Q \mid \exists a \in \Sigma, \Delta_G(q, a) \cap E \neq \emptyset\} \\ Pre_G^\forall(E) &= \{q \in Pre_G^\exists(E) \mid \forall a \in \Sigma, \Delta_G(q, a) \subseteq E\} \end{aligned}$$

The states belonging to $Pre_G^{\forall}(E)$ are the states such that all immediate successors belong to E , while the states belonging to $Pre_G^{\exists}(E)$ are such that at least one immediate successor belongs to E .

Given a live LTS G , let $Inev_G(E)$ be the set of states that inevitably lead to a set E in a finite number of steps and $CoReach_G(E)$ the set of states from which E is reachable. These sets are given by the following least fix-points (lfp):

$$\begin{aligned} Inev_G(E) &= lfp(\lambda X. E \cup pre_G^{\forall}(X)) \\ CoReach_G(E) &= lfp(\lambda X. E \cup pre_G^{\exists}(X)) \end{aligned}$$

Observable behavior The key point of our approach concerns the ability of an user \mathcal{U} to deduce information from a system by observing only a subset of the events or only an abstraction of them. For this purpose, we introduce the concept of *observation mask*. An observation mask is a function $\Pi_{\mathcal{U}} : \Sigma \rightarrow \Sigma_{\mathcal{U}} \cup \{\epsilon\}$, where $\Pi_{\mathcal{U}}$ is defined for all $\sigma \in \Sigma$. The set $\Sigma_{\mathcal{U}}$ is another event set called the *observed events*. We denote by $\Sigma_{\mathcal{U}}^{-1} = \{\sigma \in \Sigma \mid \Pi_{\mathcal{U}}(\sigma) \neq \epsilon\}$ the set of *observable events*, i.e. the events of Σ inducing an observation for \mathcal{U} . The observation mask is extended to any trajectory by assigning $\Pi_{\mathcal{U}}(\epsilon) = \epsilon$ and $\forall s \in \Sigma^*, \sigma \in \Sigma, \Pi_{\mathcal{U}}(s\sigma) = \Pi_{\mathcal{U}}(s)\Pi_{\mathcal{U}}(\sigma)$. This is further extended to any language $L \subseteq \Sigma^*$ by letting:

$$\Pi_{\mathcal{U}}(L) = \{\Pi_{\mathcal{U}}(s) \mid s \in L\}$$

The inverse observation mask for $T \subseteq \Sigma_{\mathcal{U}}^*$ is given by:

$$\Pi_{\mathcal{U}}^{-1}(T) = \{l \in \Sigma^* \mid \Pi_{\mathcal{U}}(l) \in T\}.$$

We say that G is $\Sigma_{\mathcal{U}}$ -live if $\forall q \in Q, \exists s \in \Sigma^*, \sigma \in \Sigma_{\mathcal{U}}^{-1}, q \xrightarrow{s\sigma}$, meaning that there is no terminal loop of events that cannot be observed by the observation mask.

Starting from a system G and a set of observable events $\Sigma_{\mathcal{U}}$, the set of *observed traces* of G is simply given by $\mathcal{T}_{\mathcal{U}}(G) = \Pi_{\mathcal{U}}(L(G))$.

We define the *semantic* $\llbracket \mu \rrbracket_{\mathcal{U}}$ of a trace $\mu \in \mathcal{T}(G)$ as the set of trajectories of G that are *compatible* with the trace μ :

$$\llbracket \mu \rrbracket_{\mathcal{U}} \triangleq \begin{cases} \Pi_{\mathcal{U}}^{-1}(\mu) \cap L(G) \cap \Sigma^* \Sigma_{\mathcal{U}}^{-1} & \text{if } \mu \neq \epsilon \\ \{\epsilon\} & \text{otherwise.} \end{cases}$$

This means that (except for the empty trace), trajectories compatible with a trace μ are trajectories of G ending with an observable event and having trace μ . This is consistent with an on-line observation performed by a user of the system for whom the system is only seen through the interface given by the observation mask $\Pi_{\mathcal{U}}$. We suppose that the observers are reacting faster than the system. Therefore, when an observable event occurs, observers can take a decision or raise an alarm before the system proceeds with any unobservable event. This explains why we do not consider trajectories ending with unobservable events in the definition of the semantic.

An LTS G is said to be *deterministic* if for all $q \in Q_G$, for all $a \in \Sigma$, $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ implies $q' = q''$.

In order to build monitors in charge of the observation of the system, we will need to build, starting from a non-deterministic LTS G , a deterministic LTS $Det_{\mathcal{U}}(G)$ over the alphabet $\Sigma_{\mathcal{U}}$ preserving the set of traces, i.e. $L(Det_{\mathcal{U}}(G)) = \mathcal{T}_{\mathcal{U}}(G)$.

Definition 3 Let $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$ be an LTS and $\Pi_{\mathcal{U}}$ an observation mask. The determinization of G with respect to $\Pi_{\mathcal{U}}$ is the LTS $Det_{\mathcal{U}}(G) = (\mathcal{X}, \Sigma_{\mathcal{U}}, \rightarrow_d, X^0)$ where $\mathcal{X} = 2^{Q_G}$ (the set of subsets of Q_G called macro-states), $X^0 = \{q_G^0\}$ and $\rightarrow_d = \{(X, \Pi_{\mathcal{U}}(a), \Delta_G(X, (\Sigma \setminus \Sigma_{\mathcal{U}}^{-1})^* . a) \mid X \in \mathcal{X} \text{ and } a \in \Sigma_{\mathcal{U}}^{-1}\}$.

Notice that this definition is consistent with the above semantic of observations $\llbracket \cdot \rrbracket_{\mathcal{U}}$: the target macro-state X' of a transition $X \xrightarrow{\sigma} X'$ is composed of the set of states q' of G which are targets of sequences of transitions $q \xrightarrow{s.a} q'$ ending with an observable event a such that $\Pi_{\mathcal{U}}(a) = \sigma$, with $q \in X$.

From the definition of \rightarrow_d , we get $\Delta_{Det_{\mathcal{U}}(G)}(X^0, \mu) = \{\Delta_G(q_G^0, \llbracket \mu \rrbracket_{\mathcal{U}})\}$. This means that a macro-state that is reached from X^0 by μ in $Det_{\mathcal{U}}(G)$ is composed of states that are reached from q_G^0 by trajectories of $\llbracket \mu \rrbracket_{\mathcal{U}}$ in G .

3 Inference of properties under partial observation

In this section, we consider a user \mathcal{U} interacting with a system modeled by a LTS $G = (Q_G, \Sigma, \rightarrow, q_G^0)$ through an interface modeled by an observation mask $\Pi_{\mathcal{U}}$. We consider properties modeled by regular languages over Σ that are defined as follows.

Definition 4 *A property is given by a marked language $L_{F_{\psi}}(\psi) \subseteq \Sigma^*$ of a complete and deterministic LTS $\psi = (Q_{\psi}, \Sigma, \rightarrow_{\psi}, q_{\psi}^0)$ equipped with a distinguished set F_{ψ} .*

We say that a trajectory $s \in L(G)$ is recognized by ψ , noted $s \models \psi$ whenever $s \in L_{F_{\psi}}(\psi)$. As ψ is complete, we get $L(G \times \psi) = L(G)$ and $L_{Q_G \times F_{\psi}}(G \times \psi) = L(G) \cap L_{F_{\psi}}(\psi)$ is the set of trajectories of G satisfying ψ .

Let $s \in L(G)$ be a trajectory that has been triggered by the system. The user \mathcal{U} aims to infer whether s satisfies the property ψ by observing $\mu = \Pi_{\mathcal{U}}(s) \in \mathcal{T}_{\mathcal{U}}(G)$. However, the user cannot distinguish s from any trajectory $s' \in \llbracket \mu \rrbracket_{\mathcal{U}}$ compatible with the observation μ . Thus, \mathcal{U} can only infer partial information regarding $s \models \psi$ from $\llbracket \mu \rrbracket_{\mathcal{U}}$. For example, \mathcal{U} is sure that $s \models \psi$ if $\llbracket \mu \rrbracket_{\mathcal{U}} \subseteq L_{F_{\psi}}(\psi)$. Meanwhile, if there exists $s' \in \llbracket \mu \rrbracket_{\mathcal{U}}$ and $s' \not\models \psi$, then it is impossible for \mathcal{U} to know if the current trajectory is s or s' and then \mathcal{U} cannot infer whether $s \models \psi$ or not. To go further, \mathcal{U} might be also interested in the fact that after observing μ , ψ will be inevitably satisfied, or will not be satisfied anymore by the trajectories of G extending s .

Next, we formalize these ideas and propose a way to build a function $\mathcal{O}_{\mathcal{U}}^{\psi}$, inspired by [8], which gives access, for each observation $\mu \in \mathcal{T}(G)$ to what a user \mathcal{U} can infer on s and ψ . Formally, if s is the current execution of the system and $\mu = \Pi_{\mathcal{U}}(s)$ is the corresponding observation, the verdicts we are interested in are defined by the following function:

$$\mathcal{O}_{\mathcal{U}}^{\psi} : \Sigma_{\mathcal{U}}^* \rightarrow V = \{Yes, Inev, Inev_Yes, Never, No, ?\}$$

where the semantic of the verdicts is as follows:

- 1) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = Yes$ if \mathcal{U} knows that for the current execution s (s.t. $\Pi_{\mathcal{U}}(s) = \mu$), $s \models \psi$;
- 2) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = Inev$ if \mathcal{U} knows that $s \not\models \psi$ but also that ψ will be inevitably satisfied by all the possible extension of s ;
- 3) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = Inev_Yes$ if \mathcal{U} knows that $s \models \psi$ or that ψ will inevitably be satisfied in the future but cannot distinguish between the two cases so far
- 4) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = Never$ if \mathcal{U} knows that ψ will never be satisfied by the executions of G extending s ;
- 5) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = No$ if \mathcal{U} knows that $s \not\models \psi$, but ψ is neither unavoidable nor impossible;
- 6) $\mathcal{O}_{\mathcal{U}}^{\psi}(\mu) = ?$ in all the other cases, meaning that \mathcal{U} cannot infer any useful information with regards to s and ψ after the observation $\mu = \Pi_{\mathcal{U}}(s)$.

3.1 Construction of $\mathcal{O}_{\mathcal{U}}^{\psi}$

In this section, we now explain how to construct the function $\mathcal{O}_{\mathcal{U}}^{\psi} : \Sigma_{\mathcal{U}}^* \rightarrow V$:

Step 1. Construct the synchronous product $G_\psi = G \times \psi = (Q_{G_\psi}, \Sigma, \rightarrow_{G_\psi}, q_{G_\psi}^0)$ as well as the set of final states $F_{G_\psi} = Q^G \times F_\psi$. By the property of the synchronous product, and using the fact that ψ is complete, we get $L(G_\psi) = L(G)$ and $L_{F_{G_\psi}}(G_\psi) = L(G) \cap L_{F_\psi}(\psi)$. Thus, the accepted trajectories of G_ψ in F_{G_ψ} , $L_{F_{G_\psi}}(G_\psi)$, are exactly the trajectories of G accepted by ψ .

Step 2. Compute $Inev_{G_\psi}(F_{G_\psi})$ on G_ψ and consider the following partition: $Q_{G_\psi} = F_{G_\psi} \cup I_{G_\psi} \cup P_{G_\psi} \cup N_{G_\psi}$, where

- $I_{G_\psi} = Inev_{G_\psi}(F_{G_\psi}) \setminus F_{G_\psi}$ is the set of states not belonging to F_{G_ψ} but from which F_{G_ψ} is unavoidable;
- $P_{G_\psi} = Q_{G_\psi} \setminus CoReach_{G_\psi}(F_{G_\psi})$, i.e. the set of states from which F_{G_ψ} is unreachable;
- $N_{G_\psi} = Q_{G_\psi} \setminus (F_{G_\psi} \cup I_{G_\psi} \cup P_{G_\psi})$ is the set of all other states.

Step 3. Build $\chi_{\mathcal{U}}^\psi(G) = Det_{\mathcal{U}}(G_\psi) = (\mathcal{X}, \Sigma_{\mathcal{U}}, \rightarrow_d, X^0)$. We thus have $L(\chi_{\mathcal{U}}^\psi(G)) = \mathcal{T}_{\mathcal{U}}(G)$. For each observation $\mu \in \mathcal{T}_{\mathcal{U}}(G)$, we get $\Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) = \{\Delta_{G_\psi}(q_{G_\psi}^0, \llbracket \mu \rrbracket_{\mathcal{U}})\}$.

Step 4. We finally compute the observation function $\mathcal{O}_{\mathcal{U}}^\psi$ from $\chi_{\mathcal{U}}^\psi(G)$ and the sets $F_{G_\psi}, I_{G_\psi}, P_{G_\psi}, N_{G_\psi}$ as follows:
 $\forall \mu \in \mathcal{T}_{\mathcal{U}}(G)$,

$$\mathcal{O}_{\mathcal{U}}^\psi(\mu) = \begin{cases} \text{Yes,} & \text{if } \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq F_{G_\psi} \\ \text{Inev,} & \text{if } \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq I_{G_\psi} \\ \text{Inev_Yes,} & \text{if } \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq (I_{G_\psi} \cup F_{G_\psi}) \\ & \wedge \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \cap I_{G_\psi} \neq \emptyset \\ & \wedge \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \cap F_{G_\psi} \neq \emptyset \\ \text{No,} & \text{if } \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq N_{G_\psi} \\ \text{Never} & \text{if } \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq P_{G_\psi} \\ ? & \text{otherwise.} \end{cases}$$

It is easy to check that the construction of $\mathcal{O}_{\mathcal{U}}^\psi$ conforms to the informal definition previously introduced. For example, for the verdict *Yes*, consider an execution $s \in L(G)$ together with its corresponding observation $\mu = \Pi_{\mathcal{U}}(s)$ and $\mathcal{O}_{\mathcal{U}}^\psi(\mu) = \text{Yes}$. We thus have $\Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq F_{G_\psi}$. Now, according to the definition of $\chi_{\mathcal{U}}^\psi(G)$, for all $s' \in \llbracket \mu \rrbracket_{\mathcal{U}}$, $\Delta_{S_\psi}(q_{G_\psi}^0, s') \subseteq \Delta_{\chi_{\mathcal{U}}^\psi(G)}(X^0, \mu) \subseteq F_{G_\psi}$, thus $s' \models \psi$. Hence, for all trajectories $s' \in \llbracket \mu \rrbracket_{\mathcal{U}}$, $s' \models \psi$ and in particular $s \models \psi$. Similarly for $\mathcal{O}_{\mathcal{U}}^\psi(\mu) = \text{Inev}$. It implies that $\Delta_{S_\psi}(q_{G_\psi}^0, \llbracket \mu \rrbracket_{\mathcal{U}}) \subseteq I_{G_\psi}$. Then, the trajectories in $\llbracket \mu \rrbracket_{\mathcal{U}}$ are for sure not satisfying ψ and all their continuations will inevitably satisfy ψ . Then this also holds for s .

To conclude this section, given a system G that is observed by a user \mathcal{U} through the interface $\Pi_{\mathcal{U}}$, we know how to construct a function $\mathcal{O}_{\mathcal{U}}^\psi : \Sigma_{\mathcal{U}}^* \rightarrow V$ that gives access to all the information that the user \mathcal{U} can deduce with respect to the executions of G and the property ψ .

4 Characterization and verification of opacity

Assume now that the attacker \mathcal{A} is a user of a system G trying to infer confidential information. We assume that the attacker perfectly knows the model of G , but only observes it through the interface $\Pi_{\mathcal{A}}$. We now consider a secret φ given by a marked language of a complete deterministic LTS, $\varphi = (Q_{\varphi}, q_{\varphi}^0, \Sigma, \rightarrow, F_{\varphi})$. We do assume that \mathcal{A} knows how to build an observational function as described in the preceding section and our aim is to know if the attacker can know that the current execution $s \in L(G)$ reveals the secret φ . Here is a very simple example to illustrate the approach.

Example 1 Let G be a LTS with $\Sigma = \{h, p, l_1, l_2, l_3\}$, $\Sigma_{\mathcal{A}} = \{l_1, l_2, l_3\}$ (the observation Mask is reduced to the natural projection). The secret under consideration is the occurrence of the event p . This should not be revealed to the users of the system, knowing that p is not observable. However,

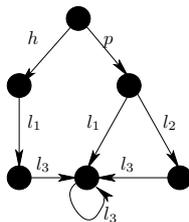


Figure 2: An example of interference

users can infer that p has occurred by observing the event l_2 . Such a system is then not secure because the fact that p occurs during execution is modifying what \mathcal{A} can observe. Note however that for a different observation mask such that $\Pi_{\mathcal{A}}(l_1) = \Pi_{\mathcal{A}}(l_2)$, then the occurrence of p does not change the observations and G is safe. \diamond

4.1 Definition of Opacity

Intuitively, a secret φ is said to be opaque with respect to a system G and an observation mask $\Pi_{\mathcal{A}}$ if the attacker \mathcal{A} can never be sure that the current execution of G satisfy φ [1, 4, 2].

Definition 5 [Opacity] Given a system G and a secret φ , φ is said to be opaque w.r.t. G and $\Pi_{\mathcal{A}}$ if

$$\forall s \in L(S), \llbracket \Pi_{\mathcal{A}}(s) \rrbracket_{\mathcal{A}} \not\subseteq L_{F_{\varphi}}(\varphi) \quad (1)$$

In other words, φ is opaque w.r.t. G and $\Pi_{\mathcal{A}}$ if and only if

$$\forall \mu \in \mathcal{T}_{\mathcal{A}}(G), \llbracket \mu \rrbracket_{\mathcal{A}} \not\subseteq L_{F_{\varphi}}(\varphi),$$

and φ is non-opaque w.r.t. G and $\Pi_{\mathcal{A}}$ if and only if

$$\exists \mu \in \mathcal{T}_{\mathcal{A}}(G), \llbracket \mu \rrbracket_{\mathcal{A}} \subseteq L_{F_{\varphi}}(\varphi)$$

Based on the semantics of $\mathcal{O}_{\mathcal{A}}^{\varphi}$ described in the preceding section, one can say that φ is opaque with respect to G and $\Pi_{\mathcal{A}}$ if

$$\forall s \in L(G), \mathcal{O}_{\mathcal{A}}^{\varphi}(\Pi_{\mathcal{A}}(s)) \neq Yes$$

4.2 Verification of Opacity

In this section, we are interested in checking whether a secret φ is opaque with respect to a system G and an interface $\Pi_{\mathcal{A}}$. This happens to be a particular case of the inference of property that we presented in Section 3. To do so, consider $\chi_{\mathcal{A}}^{\varphi}(G) = \text{Det}_{\mathcal{A}}(G \times \varphi) = (\mathcal{X}, \Sigma_{\mathcal{A}}, \rightarrow_d, X^0)$ equipped with the set of final states $F = 2^{Q^G \times F_{\varphi}}$. By construction of $\chi_{\mathcal{A}}^{\varphi}(G)$, we get the following property

$$\llbracket L_F(\chi_{\mathcal{A}}^{\varphi}(G)) \rrbracket_{\mathcal{A}} = \{s \in L(S) \cap \Sigma.\Sigma_{\mathcal{A}}^{-1} \mid \llbracket \Pi_{\mathcal{A}}(s) \rrbracket_{\mathcal{A}} \subseteq L_{F_{\varphi}}(\varphi)\}$$

which gives a characterization of opacity:

Proposition 1 φ is opaque with respect to G and the interface $\Pi_{\mathcal{A}}$ if and only if $L_F(\chi_{\mathcal{A}}^{\varphi}(G)) = \emptyset$. \diamond

Hence, checking the opacity of a secret φ consists of checking that the set of states F is not reachable in $\chi_{\mathcal{A}}^{\varphi}(G)$. If it is reachable, then φ is not opaque and there exists at least one observation allowing the attacker to infer that φ is satisfied. In other words, $L_F(\chi_{\mathcal{A}}^{\varphi}(G))$ corresponds to the set of observations for which the attacker \mathcal{A} knows that the current execution reveals φ . In that case, the attacker \mathcal{A} , based on the preceding techniques, can compute the LTS $\chi_{\mathcal{A}}^{\varphi}(G)$ and deduce an observation function $\mathcal{O}_{\mathcal{A}}^{\varphi}$ such that, for a given observation μ of $\mathcal{T}(S)$:

- if $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = \text{Yes}$, then $\mu \in L_F(\chi_{\mathcal{A}}^{\varphi}(G))$ and $\llbracket \mu \rrbracket_{\mathcal{A}} \subseteq L_{F_{\varphi}}(\varphi)$; the attacker, based on this observation, can deduce that φ is satisfied on G and there is an information flow;
- if $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = ?_{\mathcal{A}}$, \mathcal{A} cannot deduce φ and there is no information flow, where $?_{\mathcal{A}} = \{\text{No}, \text{Inev}, \text{Inev_Yes}, \text{Never}, ?\}^1$.

Example 2 Consider the system G described in Fig. 3 (a). The alphabet of G is $\Sigma = \{a, b, c, X, Y, Z, a_{\varphi}, \tau, \delta\}$. We assume here that the secret property is given by the LTS described in Fig. 3 (b); The marked state is represented by the black state. In this example, the attacker tries to infer the occurrence of the event a_{φ} in the system.

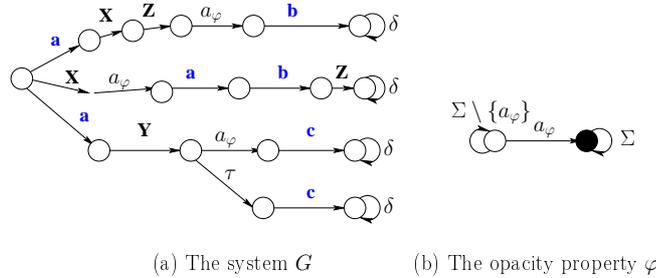


Figure 3: G and φ

For simplicity, we assume that the projection mask is reduced to the natural projection. The interface of the attacker is reduced to $\Sigma_{\mathcal{A}} = \{a, b, c, \delta\}$. The observer $\mathcal{O}_{\mathcal{A}}^{\varphi}$ that the attacker \mathcal{A} can build is given by the LTS depicted in Fig. 4.

If \mathcal{A} observes $a.b.\delta^*$ then φ is revealed \mathcal{A} is then sure that the event a_{φ} occurred in \mathcal{S} (the set of compatible trajectories is $a.X.Z.a_{\varphi}.b.\delta^*$ and $X.a_{\varphi}.a.b.Z.\delta^*$). A contrario, if \mathcal{A} simply observes a or $a.c.\delta^*$, then he is not sure that φ is satisfied or not. Some of the compatible trajectories satisfy the secret and some other do not, thus \mathcal{A} cannot infer the secret.

Remark 1 It is also possible to consider other kinds of opacity:

¹Compared with (1), we consider here that the attacker \mathcal{A} is only interested by the detection of the satisfaction of the secret.

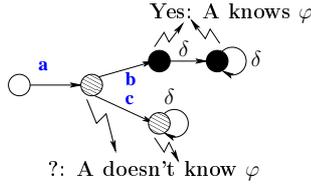


Figure 4: The function \mathcal{O}_A^φ based on χ_φ^A

- In some cases, \mathcal{A} might be interested by the information: " φ is satisfied by the current execution of the system G or will inevitably be in the future". In that case, we will say that the secret is opaque if and only if $\forall \mu \in \mathcal{T}_A(G), \llbracket \mu \rrbracket_{\mathcal{A}} \not\subseteq L_{Inev_{S_\varphi}}(S_\varphi)$. The verification of opacity and the construction of the associated observer are similar.
- It is also possible to consider the case where there is an information flow as soon as the attacker knows that either φ is satisfied or $\neg\varphi$ is satisfied (c.f. [1]). In other words, being opaque (for this definition), means that φ has to be opaque as well as $\neg\varphi$ (according to definition 5).

The associated observer will have three verdicts $\{Yes, No_A, ?_A\}$, where the verdict *Yes* corresponds to the verdict described in (1), the verdict *No_A* encompasses the verdicts *No*, *Inev*, *Never* whereas *?_A* corresponds to the other cases. \diamond

5 Monitoring Opacity

Given a secret φ , based on the techniques described in the preceding sections, it is possible to check whether φ is opaque w.r.t. G and the interface Π_A . When φ is not opaque, it can be important for an administrator to supervise the system on-line by means of a monitor \mathcal{M} and raise an alarm as soon as an information flow occurs.

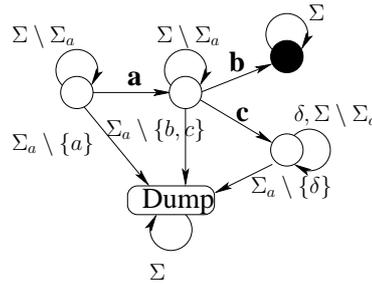
For this, we assume that \mathcal{M} knows the model of the system G and observes it through the interface $\Pi_{\mathcal{M}}$. Moreover, \mathcal{M} knows the ability of the attacker \mathcal{A} , meaning that the monitor knows that \mathcal{A} observes the system via the interface Π_A and that he can construct an observation function \mathcal{O}_A^φ . We do not assume any relation between Π_A and $\Pi_{\mathcal{M}}$. Thus, \mathcal{M} has to infer the attacker's knowledge based on the observation of $\mathcal{T}_{\mathcal{M}}(G) \subseteq \Sigma_{\mathcal{M}}^*$.

If φ is not opaque w.r.t. the system G and the interface Π_A , an administrator can build an observation function to diagnose the fact that the secret has been revealed. One can also be more accurate and try to predict the fact that the secret will be inevitably known by the attacker strictly before the information flow, or that the secret will never be revealed anymore.

Note that it is not necessary to diagnose the fact that the system performed a sequence satisfying the secret if this sequence does not correspond to a non-opaque execution (this sequence does not reveal anything to the attacker); only the executions that lead to an information flow have to be taken into account. Indeed, the secret φ is revealed to the attacker by an execution $s \in L(G)$ if and only if $\Pi_A(s) \in L_F(\chi_{\mathcal{M}}^\varphi(G))$. In other words, we are interested in diagnosing the property: "The secret φ has been revealed to the attacker", which corresponds to the extension-closed language: $\Pi_A^{-1}(L_F(\chi_{\mathcal{M}}^\varphi(G))) \cdot \Sigma^*$. This language can be recognized by an LTS Ω , equipped with a set of final states F_Ω such that:

$$\mathcal{L}_{F_\Omega}(\Omega) = \Pi_A^{-1}(L_F(\chi_{\mathcal{M}}^\varphi(G))) \cdot \Sigma^* \quad (2)$$

Example 3 To illustrate the computation of (2), let us come back to Example 2. The corresponding LTS Ω is shown in Fig. 5:

Figure 5: The LTS Ω computed from $\chi_{\mathcal{A}}^{\varphi}(G)$

5.1 Supervision of Information Flow

Given a system G , an attacker \mathcal{A} observing G via the interface $\Pi_{\mathcal{A}}$ and a secret φ (that we assume to be non-opaque), we describe now a method allowing an administrator \mathcal{M} observing G via the interface $\Pi_{\mathcal{M}}$ to know whether there is an information flow or not. We assume that the monitor in charge of the supervision has a full knowledge of G and knows the observation mask $\Pi_{\mathcal{A}}$.

As mentioned in the introduction of this section, \mathcal{M} does not directly observe φ . Only the trajectories causing an information flow have to be supervised. We consider then the stable property Ω corresponding to the trajectories of G inducing an information flow from G to \mathcal{A} (see (2)).

In order to construct the observer $\mathcal{O}_{\mathcal{M}}^{\Omega}$ in charge of the supervision of Ω (i.e. corresponding to the information leak of φ), we first build $G_{\Omega} = G \times \Omega$ and the sets $F_{G_{\Omega}}$, $I_{G_{\Omega}}$, $P_{G_{\Omega}}$, $N_{G_{\Omega}}$ (as described in Step 2., Section 3.1).

Now, based on the techniques of the section 3.1, one can compute the LTS $\chi_{\mathcal{M}}^{\Omega}(G)$ over $\Sigma_{\mathcal{M}}$ from which we can derive an observer $\mathcal{O}_{\mathcal{M}}^{\Omega}$ with the following verdicts: for $\mu \in \mathcal{T}_{\mathcal{M}}(G)$,

- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = Yes$: \mathcal{M} infers that Ω is satisfied and thus can deduce that \mathcal{A} knows φ ;
- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = No$: \mathcal{M} knows that \mathcal{A} does not know φ but might know it in the future;
- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = Inev$: \mathcal{M} knows that \mathcal{A} will inevitably know φ but does not know it yet;
- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = Inev_Yes$: \mathcal{M} knows that \mathcal{A} already knows or will know φ ;
- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = Never$: \mathcal{M} knows that \mathcal{A} will never know φ .
- $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = ?$ means that \mathcal{M} cannot deduce anything about the knowledge of \mathcal{A} .

Unfortunately, the case $\mathcal{O}_{\mathcal{M}}^{\Omega}(\mu) = ?$ does not imply that the attacker \mathcal{A} does not know φ . As \mathcal{M} and \mathcal{A} observe the system via different interfaces, it might be the case that \mathcal{A} already knows φ and that \mathcal{M} will never infer this information. This corresponds to the non-diagnosability of Ω [8]. This can occur when there exist two arbitrarily long trajectories s and s' corresponding to the same observation μ such that $s \in L_{F_{\Omega}}(\Omega)$ (thus a non-opaque trajectory of φ) and $s' \notin L_{F_{\Omega}}(\Omega)$. In the next section, we will give necessary and sufficient conditions under which this case does not occur.

5.2 Necessary and sufficient conditions for detection/prediction of information flow

Consider the system G as well as the property Ω described in the previous section.

5.2.1 Diagnosability

Intuitively, G is Ω -diagnosable ([11, 8]) if there exists $n \in \mathbb{N}$ such that for any trajectory s of G such that $s \models \Omega$, Ω becomes non-opaque after waiting for at most n observations. This can be formalized as follows

Definition 6 Given a system G , a stable property Ω and an interface $\Pi_{\mathcal{M}}$, G is Ω -diagnosable if,

$$\begin{aligned} \exists n \in \mathbb{N}, \forall s \in L(G) \cap L_{F_{\Omega}}(\Omega) \cap \Sigma^* \cdot \Sigma_{\mathcal{M}}^{-1}, \\ \forall t' \in L(G), t' = s \cdot t \wedge |\Pi_{\mathcal{M}}(t)| \geq n \Rightarrow \llbracket \Pi_{\mathcal{M}}(s \cdot t) \rrbracket_{\mathcal{M}} \subseteq L_{F_{\Omega}}(\Omega) \end{aligned}$$

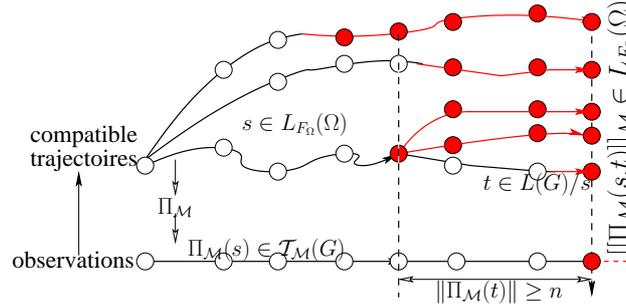


Figure 6: Intuition of the diagnosability property

The Ω -diagnosability property means that whenever a trajectory s of the system satisfies Ω , then whatever the extension t of s , t having at least n observable events w.r.t. $\Pi_{\mathcal{M}}$, all the trajectories compatible with the observation $\Pi_{\mathcal{M}}(s \cdot t)$ satisfy Ω .

In the case of monitoring opacity, this means that when the monitor is observing a trace in $L_{F_{\Omega}}(\Omega)$, a “Yes” answer should be produced by the observer after finitely many observed events. Hence, if there exists $s \in L(G)$ triggered by the system such that φ is non-opaque for \mathcal{A} , then \mathcal{M} will surely know it at most n observed events after the observation of $\Pi_{\mathcal{M}}(s)$.

5.2.2 Predictability

If the system is Ω -diagnosable, then it might be interesting to refine the verdict by predicting the satisfaction of the property strictly before its actual occurrence [7]. Roughly speaking, Ω is predictable if it is always possible to detect the future satisfaction of Ω , strictly before this happens, only based on the observations.

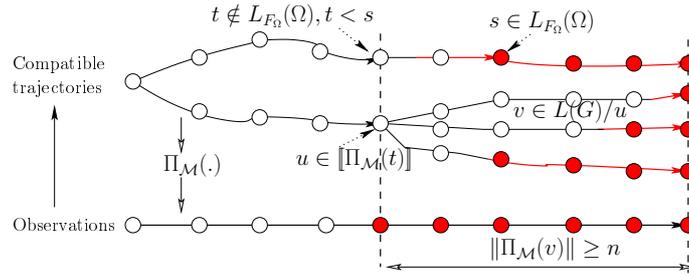
Definition 7 Given a system G , a property Ω and an interface $\Pi_{\mathcal{M}}$, G is Ω -predictable if

$$\begin{aligned} \exists n \in \mathbb{N}, \forall s \in \mathcal{L}(G) \cap L_{F_{\Omega}}(\Omega) \cap \Sigma^* \cdot \Sigma_{\mathcal{M}}^{-1}, \\ \exists t \in (\mathcal{L}(G) \cap \Sigma^* \cdot \Sigma_{\mathcal{M}}^{-1}) \cup \{\epsilon\}, t < s \wedge t \notin L_{F_{\Omega}}(\Omega) \text{ s.t.} \\ \forall u \in \llbracket \Pi_{\mathcal{M}}(t) \rrbracket_{\mathcal{M}}, \forall v \in \mathcal{L}(G)/u, |\Pi_{\mathcal{M}}(v)| \geq n \Rightarrow u \cdot v \in L_{F_{\Omega}}(\Omega) \end{aligned}$$

This property means that for any trajectory s that satisfies Ω , there exists a strict prefix t that does not satisfy Ω , such that any trajectory u compatible with observation $\Pi_{\mathcal{M}}(t)$ will inevitably be extended into a trajectory $u \cdot v$ satisfying Ω ².

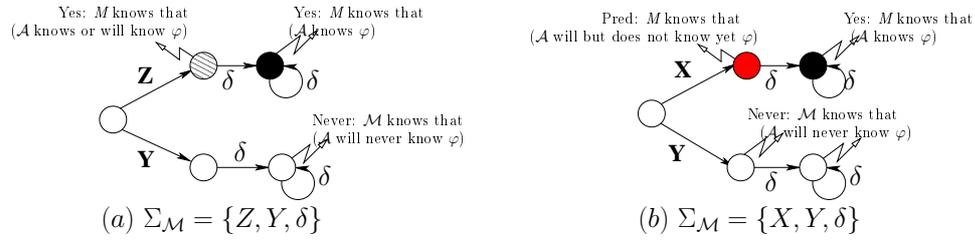
In our setting, this means that \mathcal{M} can always predict that \mathcal{A} will know φ and then the system operator can be warned in time to halt the system or can take counter-measures in order to avoid the secret to be revealed. In other words, if \mathcal{M} observes a trace $\mu \in \mathcal{T}_{\mathcal{M}}(G)$ such that $\mu = \Pi_{\mathcal{M}}(t)$, then \mathcal{M} knows that the secret is not revealed to \mathcal{A} , but will be after at most n observations.

²Note that predictability implies diagnosability [7].

Figure 7: Intuition of the Ω -predictability

Remark 2 *There is an algorithm of polynomial complexity for verifying that a system G is Ω -diagnosable or Ω -predictable. More details can be found in [8, 7].*

Example 4 *To illustrate this section, we still consider the system G and the secret φ defined in Example 2. The property Ω and the set of non-opaque trajectories (i.e. the ones that reveal the secret φ) are given by the LTS described in Fig. 5.*

Figure 8: Observation function $\mathcal{O}_{\mathcal{M}}^{\Omega}$ w.r.t. two different interfaces

Assume that the interface of the monitor \mathcal{M} is reduced to $\Sigma_{\mathcal{M}} = \{Z, Y, \delta\}$. Then, one can show that G is Ω -diagnosable, but not Ω -predictable. The corresponding $\mathcal{O}_{\mathcal{M}}^{\Omega}$ is represented in Figure 8(a). *A contrario*, if the interface of the monitor \mathcal{M} is $\Sigma_{\mathcal{M}} = \{X, Y, \delta\}$, then the system is Ω -predictable. Indeed, after the observation of X , \mathcal{M} knows that all the possible extensions will satisfy Ω and thus that the secret will be revealed (C.f. Figure 8(b)).

6 Construction of Monitors Using Abstractions

Until then, we made the assumption that the attacker \mathcal{A} knows a model perfectly reflecting the behavior of the system. This entails that all the attack scenarios that \mathcal{A} can compute correspond to real attacks. Especially, the opacity of the model implies the opacity of the system. But the methods presented above may be not effective for the kind of models we might be interested in for modelling real systems. For example, our approach relies on reachability analysis and determinization which are in general not possible for infinite systems. Moreover, even for finite LTS, the determinization has an exponential complexity in the number of states which can be intractable for large LTS. It is then realistic to consider an attacker reasoning on a finite state abstraction of the system.

Unfortunately, as we will see later, opacity is not preserved by abstraction. Then abstraction cannot be used to infer that a system is opaque when the abstraction is. Also, reasoning on the abstraction, there can be cases of information flow which are not possible on the system. Then using abstraction to verify opacity is not relevant for accepting or rejecting systems. Nevertheless, we will see how abstractions can help an attacker to infer secret information and the administrator to detect the attacks.

Let S be a system over an alphabet Σ with a set of behaviors $L(S) \subseteq \Sigma^*$. Let \mathcal{A} be an attacker interacting with S via an interface $\Pi_{\mathcal{A}}$ and knowing a finite abstraction $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$ of S such that $L(S) \subseteq L(G)$. Following the techniques described in section 3.1, \mathcal{A} builds the monitor $\chi_{\mathcal{A}}^{\varphi}(G)$ and the function $\mathcal{O}_{\mathcal{A}}^{\varphi}$. Let $s \in L(S)$ be the sequence executed in S and $\mu = \Pi_{\mathcal{A}}(s) \in \mathcal{T}_{\mathcal{A}}(S)$ be the observation of the attacker.

- Suppose that $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = Yes$. This happens if and only if $L(G) \cap \Sigma^* \cdot \Sigma_{\mathcal{A}}^{-1} \cap \Pi_{\mathcal{A}}^{-1}(\mu) \subseteq L_{F_{\varphi}}(\varphi)$. Since $L(S) \subseteq L(G)$, it follows that $L(S) \cap \Sigma^* \cdot \Sigma_{\mathcal{A}}^{-1} \cap \Pi_{\mathcal{A}}^{-1}(\mu) \subseteq L_{F_{\varphi}}(\varphi)$. The attacker \mathcal{A} can then deduce from $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = Yes$ that $s \in L_{F_{\varphi}}(\varphi)$ and there is an information flow.
- Because of the abstraction, the verdict $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = ?_{\mathcal{A}}$ is no more accurate regarding the opacity of S . Indeed, S may be non-opaque for μ , but the uncertainty can come from a sequence $s' \in L(G) \setminus L(S)$ such that $s' \in \llbracket \mu \rrbracket_{\mathcal{A}}$ in G and $s' \notin L_{F_{\varphi}}(\varphi)$. This sequence will somehow hide the non-opacity of S to the attacker.

It is important to note that the abstraction G of S is a parameter of the problem. We assume G to be a part of the attacker's model, beside his observation power $\Pi_{\mathcal{A}}$ and $\llbracket \cdot \rrbracket_{\mathcal{A}}$ as well as his ability to compute $\mathcal{O}_{\mathcal{A}}^{\varphi}$. Indeed, the more precise is the abstraction, the more accurate will be the verdict of the function $\mathcal{O}_{\mathcal{A}}^{\varphi}$, and then the likelihood to infer information. On the other hand, given an abstraction such that $\mathcal{O}_{\mathcal{A}}^{\varphi^{-1}}(Yes) \cap \mathcal{T}_{\mathcal{A}}(S) = \emptyset$, another attacker \mathcal{A}' using a less precise abstraction of S than the abstraction G will not be able to infer any information about φ .

We adopt now the administrator point of view, assuming that \mathcal{A} and \mathcal{M} are using the same abstraction G .

- if for all $\mu \in \mathcal{T}_{\mathcal{A}}(S)$, $Ob_{\mathcal{A}}\varphi(\mu) \neq Yes$, then S is safe as the attacker will not be able to infer any information;
- otherwise, any observation $\mu \in \mathcal{T}_{\mathcal{A}}(S)$ such that $\mathcal{O}_{\mathcal{A}}^{\varphi}(\mu) = Yes$ corresponds to a real attack.

Following the methodology of Section 5, \mathcal{M} can thus build a function $\mathcal{O}_{\mathcal{M}}^{\Omega}$, where Ω is such that $L_{F_{\Omega}}(\Omega) = \{s \in L(G) \mid \mathcal{O}_{\mathcal{A}}^{\varphi}(\Pi_{\mathcal{A}}(s)) = Yes\} \cdot \Sigma^*$. The verdicts given by the function $\mathcal{O}_{\mathcal{M}}^{\Omega}$ are sound: all the alarms the monitor raises correspond to real cases of information flow.

7 Conclusion

In this paper, given a system modeled by a labeled transition system and a secret property modeled by a regular language, we have shown how to characterize cases of confidential information flow. Then we exposed how an administrator can construct a monitor raising an alarm whenever an attack is detected. Further, we provide necessary and sufficient conditions for such information flows to be always detected by the administrator; in a bounded delay in the case of diagnosability or before they occur in the case of predictability. We finally showed how to use abstraction for generating attacks and monitoring them. In this paper, we focused on the detection of information flow. The monitors are passive, they raise an alarm whenever an attack occurred. In parallel, we have investigated in [6] the on-line control of the system by a supervisor in order to avoid the secret to be revealed. This, in some points, extends the approach proposed in [2], in which all the events are controllable, and earlier work done by Schneider on security automata [12], and subsequently extended to edit automata [9].

Future Work: We first plan to extend these results to more expressive models mixing control and data. Also, the attacker is interested in deducing a given secret and the administrator is concerned with what does the attacker knows. Epistemic logic seems to be a good candidate to generalize this approach to more than two participants. Finally, we presented in a simple case how an attacker can infer information knowing only an abstraction of the system. But the initial knowledge plays an important role in our approach. A natural extension would then be to consider an attacker having

an arbitrary initial knowledge of the system and using learning techniques. This attacker will try to acquire a more precise model of the system to likely infer some confidential information.

References

- [1] Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving secrecy under refinement. In *ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming*, pages 107–118. Springer, 2006.
- [2] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. In S. Lafortune, F. Lin, and D. Tilbury, editors, *8th Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, Michigan, USA, July 2006.
- [3] B. Blanchet, Abadi; M., and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 331–340, Chicago, IL, June 2005. IEEE Computer Society.
- [4] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 2008.
- [5] V. Darmaillacq, J.-C. Fernandez, R. Groz, L. Mounier, and J.-L. Richier. Test generation for network security rules. In *TestCom 2006*, volume 3964 of *LNCS*, 2006.
- [6] J. Dubreil, Ph. Darondeau, and H. Marchand. Opacity enforcing control synthesis. In *Workshop on Discrete Event Systems, WODES'08*, Gothenburg, Sweden, March 2008.
- [7] T. Jéron, H. Marchand, S. Genc, and S. Lafortune. Predictive diagnosis for discrete event systems. Technical Report 1834, IRISA, March 2007.
- [8] T. Jéron, H. Marchand, S. Pinchinat, and M-O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems, WODES'06*, Ann-Arbor (MI, USA), July 2006.
- [9] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, February 2005.
- [10] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.
- [11] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 1995.
- [12] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.