HAL
open science

Génération automatique de tests pour des propriétés de sécurité

Hervé Marchand, Jérémy Dubreil, Thierry Jéron

To cite this version:

Hervé Marchand, Jérémy Dubreil, Thierry Jéron. Génération automatique de tests pour des propriétés de sécurité. 4ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information, Jun 2009, Luchon, France. pp.157-174. inria-00420454

HAL Id: inria-00420454

https://inria.hal.science/inria-00420454

Submitted on 29 Sep 2009

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.

Génération automatique de tests pour des propriétés de sécurité

Hervé Marchand*
Jérémy Dubreil*
Thierry Jéron*

Résumé : Dans cet article, nous nous intéressons à l'utilisation conjointe de techniques de génération automatique de tests et de synthèse de contrôleurs pour tester des propriétés de sécurité. Nous suivons une approche orientée modèle, c'est-à-dire que nous supposons disposer d'une spécification du comportement du système. Sur celle-ci nous étudions la satisfaction de certaines classes de propriétés de confidentialité et d'intégrité. Nous proposons une méthode qui consiste à calculer automatiquement un modèle du contrôle d'accès idéal assurant ces deux types de propriétés de sécurité. Nous montrons ensuite comment en dériver des cas de tests qui, exécutés sur l'implémentation, permettent de détecter la non-conformité de l'implémentation et sous certaines hypothèses, la violation des propriétés de sécurité, et une éventuelle mauvaise implémentation du contrôle d'accès mis en place.

Mots Clés : méthodes formelles, confidentialité, intégrité, test de conformité, synthèse de contrôleurs

Introduction

Les méthodes automatiques d'analyse et de validation de propriétés de sécurité connaissent un développement important depuis quelques années. L'engouement pour de telles recherches suit naturellement l'explosion du nombre de services proposés tant sur les systèmes mobiles et embarqués que sur le réseau Internet. En effet, ces derniers sont ouverts par nature et donc vulnérables aux attaques de pirates informatiques. Pour de tels services, comme par exemple les bases de données d'informations médicales, la corruption de données et la fuite d'informations peuvent s'avérer catastrophiques. Des notions de validation et de certification du niveau de sécurité des applications critiques sont donc cruciales dans ce domaine. Dans ce contexte, on observe un intérêt croissant pour la vérification formelle [BAF05a, Low99, BAF05b] le test et la supervision de propriétés de sécurité [Sch00, LBW05, DFG+06, LG07, DJM08]. Afin d'en automatiser l'analyse, les propriétés de sécurité sont souvent classés en trois catégories : les propriétés de disponibilité (un utilisateur a toujours la possibilité de réaliser ce qui est autorisé par la politique de sécurité), d'intégrité (une action illégale ne peut être réalisée par un utilisateur) et de confidentialité (un utilisateur non autorisé ne peut acquérir d'information confidentielle, voir [BL73, FG00] et plus récemment [Maz04, AČZ06]). Une politique de sécurité correspond alors à un ensemble de telles propriétés. Dans le cadre de cet article, nous nous focalisons sur certaines classes de propriétés de confidentialité et d'intégrité.

[1] INRIA, centre Rennes - Bretagne Atlantique

Nous supposons ici que l'on dispose d'une spécification du système dont la sémantique comportementale est modélisée par un système de transitions fini G sur un alphabet Σ. Le système fournit un service aux utilisateurs (et en particulier aux attaquants potentiels) via une interface $\Sigma_a \subseteq \Sigma$. Intuitivement Σ_a définit l'ensemble des entrées et des sorties possibles. Nous supposons qu'un attaquant connaît parfaitement le modèle G du système, ainsi que son interface Σ_a. On souhaite que ce système satisfasse certaines propriétés de confidentialité et d'intégrité. Les propriétés d'intégrité considérées ici sont des propriétés de sûreté définies par un langage rationnel sur Σ. Pour décrire les propriétés de confidentialité, nous adoptons le formalisme de [BBB+07] et modélisons l'ensemble des informations confidentielles par un *secret* φ défini par un langage rationnel sur Σ. Le secret est conservé ou *opaque*, si l'attaquant ne sait pas déduire des traces observées à l'interface Σ_a et de sa connaissance de G que l'exécution courante du système appartient à φ. La notion d'opacité ne permet pas de capturer toutes les propriétés de confidentialité, mais s'avère suffisamment générale pour modéliser des propriétés d'anonymat [SS96] et certains types de non-interférence [FG00] (c.f. [BKMR08] pour plus de détails). Néanmoins, dans la suite de cet article, par abus de langage, nous parlerons indistinctement de propriétés de confidentialité et d'opacité.

La première étape de notre approche consiste en l'analyse des propriétés de sécurité sur le modèle G et leur renforcement. Si ce modèle G permet la fuite d'informations ou viole une propriété d'intégrité, une solution est de composer le système avec un moniteur en charge de la détection ou de la prédiction de ces fuites d'informations (resp. de la violation de l'intégrité du système) [DJM07]. Une solution plus puissante exposée dans [DDM08, DDM09], est de construire un contrôleur assurant ces propriétés de sécurité (c.f. section 3 pour plus de détails).

La dernière étape de notre approche consiste à détecter par tests sur l'implémentation du système si le modèle G du système a été correctement implémenté et satisfait la politique de sécurité. Les tests réalisés sont de type *tests de conformité*, c'est-à-dire que la spécification (son modèle G) est connue mais on ne connaît de l'implémentation que les interfaces de communication avec le testeur. A travers ces interfaces, le testeur interagit avec l'implémentation par des cas de tests qui consistent à stimuler l'implémentation et à observer ses réponses. Le but étant de détecter des différences observables, appelées non-conformités, entre l'implémentation et son modèle [Tre99].

Dans le cadre de cet article, on suppose que l'on dispose d'une implémentation $\mathcal{I}$ de G à laquelle a été adjoint un contrôle d'accès $\mathcal{I}_{\mathcal{AC}}$ ayant pour but d'assurer la politique de sécurité. Afin de valider cette implémentation $\mathcal{I} \times \mathcal{I}_{\mathcal{AC}}$ (avec Σ_a comme interface de communication), nous adaptons les techniques de tests de conformité en engendrant automatiquement des cas de tests qui cherchent non seulement à prouver la non-conformité de l'implémentation par rapport à son modèle, mais aussi à invalider des propriétés de sécurité (en étendant les travaux de [CJMR07] qui se restreignent à des propriétés de sûreté observables[1]), et permettent également de tester la validité du contrôle d'accès $\mathcal{I}_{\mathcal{AC}}$ par rapport au contrôle calculé sur le modèle G. Pour résumer, notre méthodologie est la suivante : la première étape consiste à calculer sur le modèle G du système un modèle du contrôle d'accès idéal assurant la politique de sécurité sur G. La deuxième étape est un algorithme de génération automatique de tests à partir de G, d'une (ou plusieurs) propriété

[1] Dans [CJMR07], les propriétés de sûreté portent uniquement sur le comportement observable du système et pas sur le comportement interne comme nous le supposons dans cet article.

de sécurité et du contrôle d'accès idéal C associé. La dernière étape consiste à exécuter les cas de tests sur l'implémentation, et sous certaines hypothèses liant les comportements internes de $\mathcal{I}$ à G, à détecter les situations suivantes :

- mauvaise implémentation du contrôle d'accès dans l'implémentation contrôlée ;
- violation de la propriété de sécurité par l'implémentation contrôlée ;
- violation de la conformité de l'implémentation par rapport à sa spécification.

Les testeurs sont engendrés en adoptant dans un premier temps le point de vue d'un utilisateur (i.e. l'exécution du cas de tests se fait via l'interface Σ_a), puis en généralisant l'approche pour une interface du testeur différente de Σ_a.

La structure du document est la suivante : la section 1 pose les notations. En section 2, nous présentons les notions de confidentialité et d'intégrité considérées et montrons brièvement comment vérifier ces deux types de propriétés de sécurité sur un modèle. La section 3 décrit les techniques permettant de calculer automatiquement des contrôles d'accès permettant d'assurer ces propriétés sur le modèle. Finalement, la section 4 est dédiée à la présentation de la méthode de génération automatique de tests pour ce type de propriétés.

1 Modèles & notations

Dans cette section, nous fixons le vocabulaire, les notations et définissons les opérations utilisées par la suite. Soit Σ un alphabet fini d'actions. Une *séquence* (ou *trajectoire*) est une suite finie d'actions de Σ, la séquence vide étant dénotée par ϵ. La concaténation de deux séquences s et t est notée $s.t$. L'ensemble des séquences de Σ est noté Σ^*. Un sous-ensemble de Σ^* est appelé un *langage* sur Σ. Soit L un langage sur Σ, L est *clos par suffixe* si $L.\Sigma^* = L$. La *clôture par préfixe* de L est définie par $\overline{L} = \{s \in \Sigma^* \mid \exists t \in \Sigma^* \text{ t.q. } s.t \in L\}$. L est *clos par préfixe* si $L = \overline{L}$.

Nous considérons ici des spécifications de systèmes dont la sémantique comportementale peut être modélisée par des systèmes de transitions étiquetés finis, en abrégé LTS (pour Labelled Transition System) :

Définition 1 (Système de transitions) *Un* système de transitions étiqueté (LTS) *est un quadruplet* $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$ *où* Σ *est un alphabet fini,* Q_G *est un ensemble fini d'états,* $q_G^0 \subseteq Q_G$ *est l'état initial,* $\rightarrow_G \subseteq Q_G \times \Sigma \times Q_G$ *est la relation de transition.*

Notations. Soit G un LTS, nous utiliserons les notations suivantes :

- $q \xrightarrow{\sigma}_G q'$ lorsque $(q, \sigma, q') \in \rightarrow_G$ et on note $q \xrightarrow{\sigma}_G$ sil existe un état $q' \in Q_G$, t.q. $q \xrightarrow{\sigma}_G q'$. Nous étendons naturellement ces notations par $q \xrightarrow{s}_G q'$ pour $s \in \Sigma^*$ et par $q \xrightarrow{s}_G$ sil existe $q' \in Q_G$, t.q. $q \xrightarrow{s}_G q'$.
- $\Sigma(q) \triangleq \{\sigma \in \Sigma \mid q \xrightarrow{\sigma}_{\mathrm{G}}\}$ dénote l'ensemble des événements admissibles dans un état q de G. G est *complet* si $\forall q \in Q_{\mathrm{G}}, \Sigma(q) = \Sigma$.
- $\Delta_{\mathrm{G}}(q, s) \triangleq \{q' \in Q_{\mathrm{G}} \mid q \xrightarrow{s}_{\mathrm{G}} q'\}$ dénote l'ensemble des états accessibles depuis q par la séquence s. Cette notation s'étend à un ensemble de langages et d'états comme suit : pour $L \subseteq \Sigma^*$, $\Delta_{\mathrm{G}}(q, L) \triangleq \{q' \in Q_{\mathrm{G}} \mid q \xrightarrow{s}_{\mathrm{G}} q' \text{ pour } s \in L\}$, et pour $Q' \subseteq Q_{\mathrm{G}}$, $\Delta_{\mathrm{G}}(Q', L) = \bigcup_{q \in Q'} \Delta_{\mathrm{G}}(q, L)$.
- Un ensemble d'états $F \subseteq Q$ est dit *stable* pour $\rightarrow_{\mathrm{G}}$ si $\forall q \in F,\ \forall \sigma \in \Sigma, \Delta_G(q, \sigma) \in F$.

– $L(G) = \{s \in \Sigma^*, q_G^0 \xrightarrow{s}_G\}$ dénote l'ensemble des trajectoires d'un système. Pour un sous-ensemble d'états $F_G \subseteq Q_G$. on notera alors $L_{F_G}(G) = \{s \in \Sigma^* \mid \exists q \in F_G, q_G^0 \xrightarrow{s}_G q\}$ l'ensemble des trajectoires acceptées dans un état de F_G.

L'opération de composition parallèle permet de définir une composition de deux LTS se synchronisant sur leurs actions communes :

Définition 2 (Composition parallèle) *Soit $G^i = (Q^i, \Sigma^i, \rightarrow_{G^i}, q^0_{G^i})$, $i \in \{1,2\}$, deux LTS. La* composition parallèle *entre G^1 et G^2 est le LTS $G^1 \times G^2 = (Q^1 \times Q^2, \Sigma^1 \cup \Sigma^2, \rightarrow_{G^1 \times G^2}, (q^0_{G^1}, q^0_{G^2}))$ et $\rightarrow_{G^1 \times G^2}$ est la plus petite relation de $(Q^1 \times Q^2) \times (\Sigma^1 \cup \Sigma^2) \times (Q^1 \times Q^2)$ satisfaisant*

$$(q_1, q_2) \xrightarrow{\sigma}_{G^1 \times G^2} \begin{cases} (q_1', q_2') \ si\ \sigma \in \Sigma^1 \cap \Sigma^2 \ \wedge\ q_1 \xrightarrow{\sigma}_{G^1} q_1' \ \wedge\ q_2 \xrightarrow{a}_{G^2} q_2' \\ (q_1', q_2) \ si\ \sigma \in \Sigma^1 \setminus \Sigma^2 \wedge\ q_1 \xrightarrow{\sigma}_{G^1} q_1' \\ (q_1, q_2') \ si\ \sigma \in \Sigma^2 \setminus \Sigma^1 \ \wedge\ q_2 \xrightarrow{\sigma}_{G^2} q_2' \end{cases}$$

Clairement, si $\Sigma^1 = \Sigma^2$, alors $L(G^1 \times G^2) = L(G^1) \cap L(G^2)$ et pour des sous-ensembles d'états accepteurs $F_i \subseteq Q^i, i = 1,2$, nous avons également $L_{F_1 \times F_2}(G^1 \times G^2) = L_{F_1}(G^1) \cap L_{F_2}(G^2)$.

Définition 3 (Complétion) *Étant donné un LTS $G = (Q, \Sigma, \rightarrow, q^0)$, un nouvel état q_{new} et un sous-alphabet $\Sigma' \subseteq \Sigma$, la (Σ', q_{new})-complétion de G est un LTS $Comp^{q_{new}}_{\Sigma'}(G) = (Q \cup \{q_{new}\}, \Sigma, \rightarrow', q^0)$, où*

$$\rightarrow' = \rightarrow \cup \{q \xrightarrow{a} q_{new} \mid q \in Q \cup \{q_{new}\}, a \in \Sigma' \ t.q.\ \neg(q \xrightarrow{a})\}$$

Comportement observable : dans la suite nous étudierons la capacité d'un utilisateur à déduire de l'information sur le système en n'observant qu'une partie des événements $\Sigma_a \subseteq \Sigma$, ce qui nous amène à définir la notion de projection. Par la suite, des éléments de Σ_a^* seront notés μ, μ'. Pour le sous alphabet $\Sigma_a \subseteq \Sigma$, la projection naturelle $\Pi_{\Sigma_a} : \Sigma^* \rightarrow \Sigma_a^*$ est définie en conservant seulement les événements de Σ_a. Formellement :

$$\begin{array}{rccl} \Pi_{\Sigma_a} : & \Sigma^* & \rightarrow & \Sigma_a^* \\ & \varepsilon & \mapsto & \varepsilon \\ & s.\sigma & \mapsto & \begin{cases} \Pi_{\Sigma_a}(s).\sigma \text{ si } \sigma \in \Sigma_a \\ \Pi_{\Sigma_a}(s) \text{ sinon} \end{cases} \end{array}$$

Cette projection est étendue à un langage $L \subseteq \Sigma^*$ de la manière suivante : $\Pi_{\Sigma_a}(L) = \{\mu \in \Sigma_a^* \mid \exists s \in L,\ \mu = \Pi_{\Sigma_a}(s)\}$. Pour un système G et un ensemble d'événements observables Σ_a, l'ensemble des *traces observables* de G est défini par $\mathcal{T}_{\Sigma_a}(G) = \Pi_{\Sigma_a}(L(G))$. Inversement, pour un ensemble de traces observables $T \subseteq \Sigma_a^*$, la projection inverse est définie par $\Pi^{-1}_{\Sigma_a}(T) = \{s \in \Sigma^* \mid \Pi_{\Sigma_a}(s) \in T\}$, ce qui permet de définir, pour une trace μ observée sur G, l'ensemble des séquences possibles de G compatibles avec μ :

$$[\![\mu]\!]_{\Sigma_a} \quad \triangleq \quad \Pi^{-1}_{\Sigma_a}(\mu) \cap L(G)$$

Un LTS G est *déterministe* si pour tout $q \in Q_G$, pour tout $\sigma \in \Sigma$, $q \xrightarrow{\sigma}_G q'$ et $q \xrightarrow{\sigma}_G q''$ implique $q' = q''$. Dans la construction d'automates pour tester des propriétés de sécurité, nous aurons besoin de construire, à partir d'un LTS G sur Σ, un LTS déterministe $Det_a(G)$ sur Σ_a et de mêmes traces que G, i.e. tel que $L(Det_a(G)) = \mathcal{T}_{\Sigma_a}(G)$. Cette construction est définie comme suit :

Définition 4 (Déterminisation) *Soit $G = (Q_G, \Sigma, \rightarrow_G, q_G^0)$ un LTS et Σ_a l'ensemble des événements observables. Le* déterminisé de G *par rapport à Σ_a est un LTS $Det_a(G) = (\mathcal{X}, \Sigma_a, \rightarrow_d, X^0)$ où $\mathcal{X} = 2^{Q_G}$ (l'ensemble des parties de Q_G appelées* macro-états*), $X^0 = \Delta_G(\{q_G^0\}, (\Sigma \setminus \Sigma_a)^*)$ et $\rightarrow_d = \{(X, \sigma, \Delta_G(X, \sigma.(\Sigma \setminus \Sigma_a)^*) \mid X \in \mathcal{X}$ et $\sigma \in \Sigma_a\}$.*

Le macro-état X' cible d'une transition $X \xrightarrow{\sigma}_d X'$ est composé des états q' de G qui sont cibles de séquences de transitions $q \xrightarrow{\sigma.s} q'$ avec $s \in (\Sigma \setminus \Sigma_a)$ et $q \in X$. On peut déduire de la définition de $\rightarrow_d$ que $\Delta_{Det_{\Sigma_a}(G)}(X^0, \mu) = \{\Delta_G(q_G^0, [\![\mu]\!]_{\Sigma_a})\}$, ce qui signifie qu'un macro-état atteint à partir de X^0 par μ dans $Det_{\Sigma_a}(G)$ est composé de l'ensemble des états qui sont atteints à partir de q_G^0 par une trajectoire de $[\![\mu]\!]_{\Sigma_a}$ dans G.

2 Propriétés de sécurité

Dans cette section, nous formalisons deux types de propriétés de sécurité : des propriétés de confidentialité (quelque chose de secret ne peut être révélé à un attaquant) et des propriétés d'intégrité (un attaquant ne doit pas être capable de faire évoluer le système dans de mauvaises configurations). Nous montrons également brièvement comment vérifier de telles propriétés sur la spécification du système G.

2.1 Propriétés d'intégrité

Dans cet article, nous considérons comme classe de propriétés d'intégrité, celles pouvant s'exprimer comme une propriété de sûreté sur les trajectoires. De façon habituelle, c'est la négation de cette propriété de sûreté qui se modélise par un observateur qui reconnait les trajectoires violant l'intégrité.

Définition 5 (Intégrité) *La négation d'une propriété d'intégrité Ψ est donnée par le langage accepté par un LTS déterministe et complet $\bar{\Psi} = (Q_{\bar{\Psi}}, \Sigma, \rightarrow_{\bar{\psi}}, q_{\bar{\Psi}}^0)$, avec $F_{\bar{\Psi}}$ un ensemble d'états accepteurs stable.*
On note $L_{\bar{\Psi}} = L_{F_{\bar{\Psi}}}(\bar{\Psi})$.

Les séquences de G qui appartiennent à $L_{\bar{\Psi}}$ sont donc les séquences qui violent la propriété d'intégrité Ψ [2]. Ainsi on dira que la propriété d'intégrité Ψ est violée par le système G si il existe au moins une séquence du système qui est reconnue par $\bar{\Psi}$

Définition 6 (Satisfaction d'une propriété d'intégrité) *Considérons un système G et une propriété d'intégrité Ψ, Ψ est satisfaite par G si $L_{\bar{\Psi}} \cap L(G) = \emptyset$*

Si Ψ est violée par G, on peut montrer que le langage $L(G) \setminus L_{\bar{\Psi}}$ est le sous-langage maximal de $L(G)$ tel que Ψ soit satisfaite.

2.2 Propriétés de confidentialité

Soit un LTS G sur Σ et un sous-alphabet $\Sigma_a \subseteq \Sigma$. L'alphabet Σ_a définit l'interface de communication permettant à un utilisateur d'interagir avec G. Nous formalisons d'abord la notion de secret :

[2] Notons que l'on suppose la stabilité de l'ensemble d'états accepteurs $F_{\bar{\Psi}}$, car, dès qu'une propriété d'intégrité a été violée, elle l'est pour toujours.

Définition 7 (Secret) *Un* secret φ *est donné par le langage accepté par un LTS complet et déterministe* $\varphi = (Q_\varphi, \Sigma, \rightarrow, q^0_\varphi)$*, muni d'un ensemble d'états accepteurs* $F_\varphi \subseteq Q_\varphi$*. On note* $L_\varphi = L_{F_\varphi}(\varphi)$*.*

Dans le cadre que nous considérons, un attaquant $\mathcal{A}$ est un utilisateur d'un service qui cherche à inférer des informations confidentielles modélisées par L_φ. Nous supposons que *l'attaquant connaît entièrement le modèle* G *du service* et l'interface $\Sigma_a \subseteq \Sigma$ à travers laquelle il observe le système. Si l'exécution courante du système est $s \in L(G)$, l'utilisateur ne doit pas être capable d'inférer à partir de $\Pi_{\Sigma_a}(s)$ et de sa connaissance de G que $s \in L_\varphi$.

Exemple 1 *Soit* G *le LTS de la Figure 1 avec* $\Sigma = \{h, p, a, b\}$*,* $\Sigma_a = \{a, b\}$*. Nous*

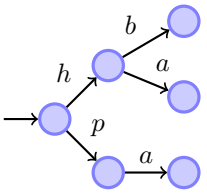


Fig. 1: Un exemple de fuite d'informations

considérons comme secrète l'occurrence de l'événement inobservable h *(i.e.* $L_\varphi = \Sigma^* h \Sigma^*$*). Il est facile de voir que l'occurrence de* h *se déduit de l'observation de l'événement* b*. Un tel système n'est donc pas sûr vis à vis de ce secret.* ◇

Informellement, on dira qu'un secret φ est opaque pour un modèle G et une interface Σ_a si pour toute trajectoire du système, l'attaquant $\mathcal{A}$ ne peut pas inférer de ses observations via Σ_a et de sa connaissance de G si φ est satisfaite par cette trajectoire [AČZ06, BKMR06, BBB+07]. Formellement,

Définition 8 (Opacité) *Considérons un système* G *et un secret* φ*. On dit que* φ *est opaque sur* G *relativement à* Σ_a *si*

$$\forall s \in L(G), [\![\Pi_{\Sigma_a}(s)]\!]_{\Sigma_a} \not\subseteq L_\varphi \tag{1}$$

De façon équivalente, φ est opaque sur G relativement à Σ_a si et seulement si $\forall \mu \in \mathcal{T}_{\Sigma_a}(G),\ [\![\mu]\!]_{\Sigma_a} \not\subseteq L_\varphi$.

Vérification de l'opacité. La proposition suivante donne une condition nécessaire et suffisante de l'opacité d'un secret :

Proposition 1 *Etant donnés un système* G *et un secret* φ *t.q.* $L_\varphi \subseteq \Sigma^*$*,* φ *est opaque relativement à* G *et* Σ_a *si et seulement si* $L_F(Det_a(G \times \varphi)) = \emptyset$*, avec* $F = 2^{Q_G \times F_\varphi}$*.*

En effet $L_F(Det_a(G \times \varphi))$ est exactement l'ensemble des traces μ dont toutes les trajectoires compatibles satisfont φ. Vérifier l'opacité d'un secret φ consiste donc à vérifier que cet ensemble est vide, ce qui s'opère en vérifiant que l'ensemble d'états F n'est pas atteignable dans $Det_a(G \times \varphi)$.

Remarque 1 *Considérons deux LTS G_1 et G_2 sur Σ, un secret φ. Si φ est opaque relativement à $L(G_1)$ et $L(G_2)$, alors il est opaque relativement à $L(G_1) \cup L(G_2)$, mais pas nécessairement relativement à $L(G_1) \cap L(G_2)$. De plus, étant donnés trois LTS G_1, G_2 et G_3 sur Σ tels que $L(G_1) \subseteq L(G_2) \subseteq L(G_3)$, φ peut être opaque relativement à $L(G_2)$ mais non-opaque relativement à $L(G_1)$ ou à $L(G_3)$.* $\diamond$

Quand φ n'est pas opaque relativement à $L(G)$ et Σ_a, il est possible de restreindre le comportement de G de manière à rendre φ opaque :

Proposition 2 *[BBB+07] Considérons un système G et un secret φ, il existe un plus grand sous-langage de $L(G)$ clos par préfixe, noté $\textsc{Op}^{\uparrow}(L(G), L_\varphi, \Sigma_a)$, tel que φ soit opaque relativement à $\textsc{Op}^{\uparrow}(L(G), L_\varphi, \Sigma_a)$ et Σ_a. Ce langage est donné par la formule suivante :*

$$\textsc{Op}^{\uparrow}(L(G), L_\varphi, \Sigma_a) = L(G) \setminus ((L(G) \setminus \Pi_{\Sigma_a}^{-1}(\Pi_{\Sigma_a}(L(G) \setminus L_\varphi))).\Sigma^*) \tag{2}$$

Intuitivement, le langage $\Pi_{\Sigma_a}^{-1}(\Pi_{\Sigma_a}(L(G) \setminus L_\varphi))$ contient l'ensemble des séquences du système qui ne révèlent pas le secret φ, tandis qu'une séquence de $L(G) \setminus \Pi_{\Sigma_a}^{-1}(\Pi_{\Sigma_a}(L(G) \setminus L_\varphi))$ révèle le secret φ (ces séquences sont étendues par Σ^* dans la mesure où, dès lors que φ a été révélé, il l'est pour toujours). Dans la suite de cet article, on notera $\textsc{Op}^{\uparrow}(G, \varphi, \Sigma_a)$ le LTS qui génère le langage $\textsc{Op}^{\uparrow}(L(G), L_\varphi, \Sigma_a)$. On verra dans la section 3 qu'un contrôleur implémentable doit restreindre G à un sous-langage de $\textsc{Op}^{\uparrow}(L(G), L_\varphi, \Sigma_a)$.

3 Synthèse Automatique d'un contrôle d'accès

Dans cette section, nous présentons des techniques basées sur la théorie du contrôle introduite par [RW89], permettant de restreindre le comportement du système de manière à assurer des propriétés de confidentialité ou d'intégrité. Nous exhiberons les conditions selon lesquelles il est possible de calculer automatiquement de tels contrôleurs. Mais dans un premier temps, nous rappelons brièvement la théorie du contrôle des systèmes à événements discrets de Ramadge & Wonham.

3.1 Bref aperçu de la théorie du contrôle

Etant donné un langage clos par préfixe $K \subseteq L(G) \subseteq \Sigma^*$ spécifiant un sous-comportement attendu de G, l'objectif du contrôle est de restreindre le comportement du système à un sous-langage de K en composant le système avec un moniteur (ou *contrôleur*), modélisé par un LTS $C = (Q_C, \Sigma_m, \rightarrow_C, q_C^0)$ qui observe un sous-ensemble Σ_m des événements de Σ et contrôle un sous-ensemble Σ_c des événements de Σ (les autres événements sont appelés incontrôlables). Le contrôleur C agit sur l'évolution d'un système modélisé par un LTS G en permettant (ou en interdisant) l'occurrence d'actions contrôlables en fonction du comportement observable passé du système. Pour qu'un contrôleur soit valide, le comportement du système contrôlé $L(G \times C)$ doit vérifier les propriétés suivantes :

Définition 9 (Contrôlabilité, observabilité) *Un langage clos par préfixe $K \subseteq L(G)$ est*

- contrôlable *relativement à $L(G)$ et Σ_c si $K.(\Sigma \setminus \Sigma_c) \cap L(G) \subseteq K$*
- observable *relativement à $L(G)$ et Σ_m si, pour toutes les séquences $s, s' \in K$ t.q. $\Pi_{\Sigma_m}(s) = \Pi_{\Sigma_m}(s')$ et pour chaque action contrôlable $\sigma \in \Sigma_c$, $(s\sigma \in L \wedge s'\sigma \in K) \Rightarrow s\sigma \in K$.*

K est donc contrôlable par rapport à Σ_c et $L(G)$ s'il n'est pas nécessaire d'empêcher l'occurrence d'événements incontrôlables pour restreindre le comportement de G à celui de K. Notons que l'union d'un nombre arbitraire de langages contrôlables est encore un langage contrôlable. La notion d'observabilité signifie que le contrôle est consistant avec l'observation. En d'autres termes, si l'occurrence d'un événement contrôlable σ doit être interdite après une séquence s, alors l'occurrence de σ doit également être interdite après toute séquence observationnellement équivalente à s, donc appartenant à $[\![\Pi_{\Sigma_m}(s)]\!]_{\Sigma_m}$. Sous l'hypothèse $\Sigma_c \subseteq \Sigma_m$, l'observabilité est stable par union arbitraire de langages, il existe donc un plus grand sous-langage $K^{\uparrow}$ de $L(G)$ à la fois contrôlable, observable et inclus dans le comportement attendu K du système G. Ce langage est le plus grand sous-langage de $K(\subseteq L(G))$ qui puisse être forcé par contrôle. Si ce langage est non vide, il existe alors un contrôleur C, dit *maximal* qui, couplé avec le système, réduit le comportement du système à ce langage, i.e. tel que $L(G \times C) = K^{\uparrow}$ [CL99].

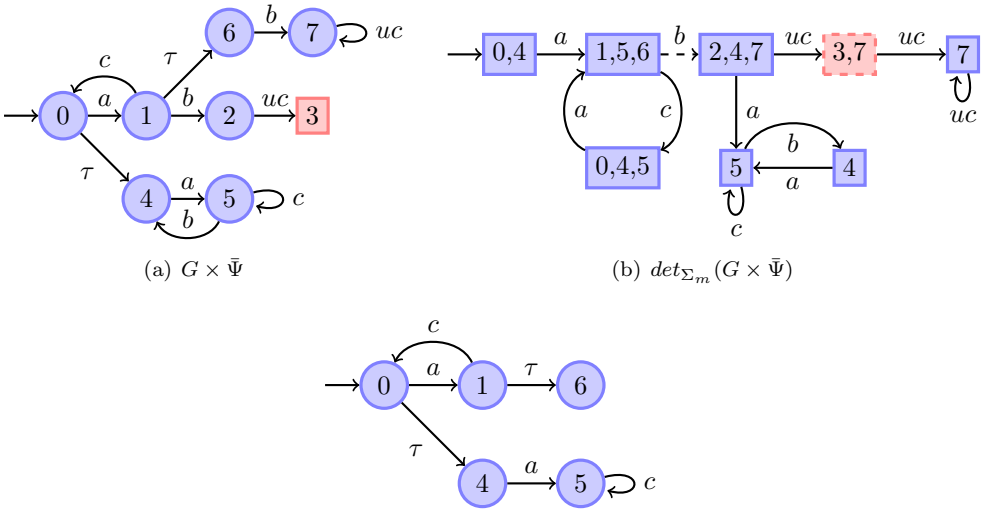
3.2 Synthèse de contrôleurs d'accès

Par la suite, nous supposons qu'un attaquant connaît parfaitement le modèle G du système et l'interface d'observation du contrôleur Σ_m, et est capable de réaliser tous les calculs nécessaires à la synthèse d'un contrôleur. Dans la suite de cet article, *nous supposerons toujours que $\Sigma_c \subseteq \Sigma_m$ (i.e. les événements contrôlables sont tous observables par le contrôleur).* Nous décrivons maintenant la méthodologie permettant de calculer des contrôles d'accès empêchant soit des fuites d'information, soit la violation d'une propriété d'intégrité. Notons que dans le cas où une politique de sécurité est donnée par un ensemble de propriétés de ces deux types, il faut d'abord calculer un contrôle d'accès pour assurer l'intégrité, puis le raffiner en considérant simultanément toutes les propriétés de confidentialité.

3.2.1 Assurer une propriété d'intégrité

Considérons une spécification d'un système G et une propriété d'intégrité Ψ, modélisée par la négation d'une propriété de sûreté $\bar{\Psi}$, l'objectif est de calculer automatiquement (quand il existe) le contrôleur $C = (Q_C, \Sigma_m, \rightarrow_C, q_C^0)$ maximal, tel que Ψ soit satisfaite par $G \times C$. Pour une telle propriété, la solution s'exprime simplement dans la théorie du contrôle développée par Ramadge & Wonham. En effet, le contrôleur est simplement obtenu en calculant par itération de point fixe le plus grand sous-langage contrôlable et observable du langage $K = L(G) \setminus L_{\bar{\Psi}}$. L'exemple suivant illustre le calcul d'un tel contrôleur.

Exemple 2 *La Figure 2 illustre le calcul du système contrôlé permettant d'assurer une propriété d'intégrité. La Figure 2 (a) décrit le produit $G \times \bar{\Psi}$, l'état 3 étant accepteur. Ainsi, toute séquence qui arrive dans l'état 3 est une séquence qui viole Ψ. On suppose ici que $\Sigma_m = \{a, b, c, uc\}$ et que $\Sigma_c = \{a, b, c\}$. Les décisions du contrôleur sont prises en fonction du comportement observé du système modélisé par le LTS de la Figure 2(b)). Si le contrôleur observe une séquence de $a.(c.a)^*.b.uc$, alors il sait que le système est soit dans l'état 7, soit dans l'état 3 (qui correspond à un état mauvais). Pour éviter d'arriver dans l'état* $\boxed{3,7}$*, le contrôleur doit donc empêcher l'occurrence de "uc". Or, celui-ci est incontrôlable. Par conséquent, le seul moyen pour le contrôleur d'éviter d'aller dans cet état est d'empêcher l'occurrence de "b" dans l'état* $\boxed{1,5,6}$*. Le LTS ainsi obtenu (en ne*



(a) $G \times \bar{\Psi}$

(b) $det_{\Sigma_m}(G \times \bar{\Psi})$

(c) Le système contrôlé $G \times C$

Fig. 2: Exemple de contrôle pour une propriété d'intégrité.

conservant que la partie atteignable) est le contrôleur maximal assurant la propriété Ψ. Le comportement du système contrôlé est donné par la figure 2(c). ◇

3.2.2 Assurer une propriété de confidentialité

Considérons maintenant une spécification d'un système G et un secret φ, l'objectif est de calculer automatiquement (quand il existe) le contrôleur $C = (Q_C, \Sigma_m, \rightarrow_C, q_C^0)$ maximal, tel que φ soit opaque relativement à $L(G \times C)$ et Σ_a. Sa caractérisation exacte est présentée dans [DDM08, DDM09]. Nous nous contentons ici d'en illustrer la construction sur un exemple avant de donner les conditions précises de son existence :

Exemple 3 *Le système que nous cherchons à contrôler est modélisé par le LTS G décrit par la figure 3. Nous supposons que $\Sigma_a = \{a, b, d, e\}$, $\Sigma_m = \{a, c_1, c_2, b, d, e, \}$, et que l'ensemble des événements incontrôlables est $\{a, d, h, \tau\}$. Le secret est donné par le langage $\Sigma^*.h.\Sigma^*$. En observant d, l'attaquant sait que h s'est produit dans le système. Il y a donc une fuite d'information et c'est la seule présente dans le système. Le contrôle doit donc interdire l'occurrence de l'événement c_1 (d et h sont incontrôlables), ce qui résulte en le LTS de la figure 4(a). Toutefois, en contrôlant de cette manière le système, le secret est maintenant révélé à l'attaquant dès qu'il observe l'évènement b ce qui nous amène à couper l'événement c_2. Nous obtenons alors le LTS de la Figure 4(b). On peut vérifier que le secret est maintenant opaque relativement à ce LTS et que celui-ci est maximal.* ◇

Le théorème suivant résume les conditions suffisantes sous lesquelles il est possible de calculer un contrôleur maximal fini (i.e. pour lequel le comportement du système contrôlé est rationnel). Les algorithmes sont détaillés dans [DDM08, DDM09].

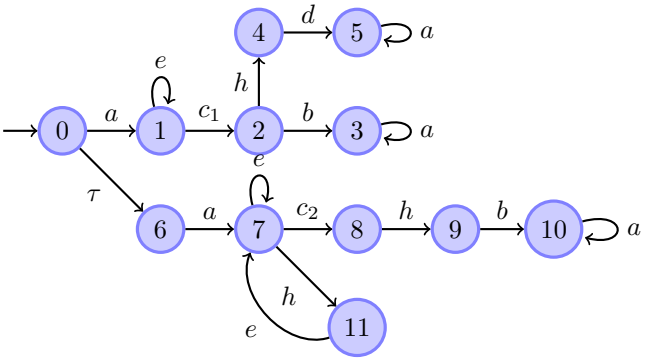
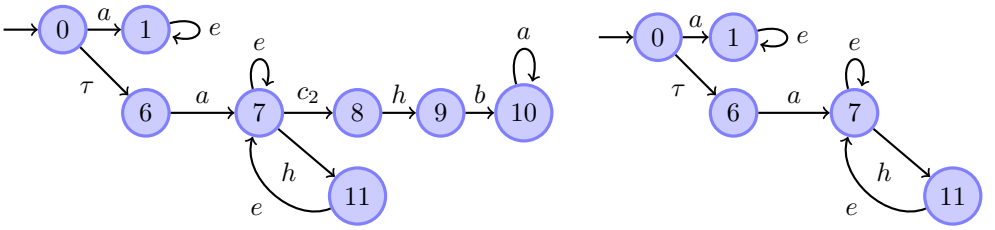


Fig. 3: Modèle de système à contrôler pour assurer l'opacité.



(a) étape 1 (b) étape 2

Fig. 4: Contrôle pour assurer l'opacité.

Théorème 1 *[DDM08, DDM09] Considérons un système G et un secret φ. Sous l'hypothèse $\Sigma_c \subseteq \Sigma_m$, il existe un contrôle d'accès maximal C tel que φ soit opaque relativement à $L(G \times C)$ et Σ_a si*

- *φ est opaque relativement à $L(G) \cap \Sigma_{uc}^*$ et Σ_a*
- *Σ_m et Σ_a sont comparables i.e.*

  (1) *$\Sigma_m \subseteq \Sigma_a$, ou*

  (2) *$\Sigma_a \subseteq \Sigma_m$.*

La condition (1) signifie que les événements observables du contrôleur sont observables par l'attaquant. D'un point de vue algorithmique, le système contrôlé est obtenu en calculant d'abord le plus grand sous-langage opaque $Op^{\uparrow} = Op^{\uparrow}(L(G), L_{\varphi}, \Sigma_a)$, puis le plus grand sous-langage observable et contrôlable de $Op^{\uparrow}$ [DDM08]. La condition (2) signifie que tous les événements de l'attaquant sont visibles par le contrôleur, certains d'entre eux pouvant être non-contrôlables. C'est le cas, par exemple, d'un firewall sur des services internet, où le contrôleur peut filtrer certaines requêtes envoyées par l'attaquant au système, tandis que les sorties de celui-ci ne peuvent pas être interdites par le contrôleur. Sous cette condition, le calcul du contrôleur est plus compliqué et le développement d'algorithmes spécifiques, sortant du cadre habituel du contrôle, a été nécessaire [DDM09].

4 Génération automatique de tests pour des propriétés de sécurité

Dans cette section, nous nous proposons de tester si le comportement d'une implémentation du système est correcte vis à vis du modèle de sa spécification, des propriétés de sécurité et du contrôle d'accès "idéal" calculé automatiquement (voir section précédente). Pour cela, nous adaptons la théorie du test de conformité dont le but est d'établir si une implémentation "*boîte noire*" est conforme à sa spécification.

Le test de conformité est une technique de validation de systèmes réactifs. Cette technique suppose l'existence d'une spécification formelle G des comportements du système ainsi que d'une implémentation $\mathcal{I}$ de ce système, un programme exécutable. Une hypothèse fondamentale pour formaliser le test de conformité consiste à supposer que l'implémentation se comporte comme un modèle inconnu. Nous assimilerons ici l'implémentation avec son modèle que nous noterons aussi I, et supposerons qu'il s'agit d'un LTS. Le but est de valider l'implémentation $\mathcal{I}$ par rapport à sa spécification G. Dans le test de conformité, l'implémentation est une boîte noire : seules ses interfaces sont connues et accessibles. A travers ces interfaces, un *testeur* interagit avec l'implémentation par des *cas de tests* qui décrivent des *stimuli* de l'implémentation (e.g. des envois de messages, des appels de fonctions ou de méthodes), et des *observations* (messages reçus, retours d'appels de fonctions, de méthodes, etc). Le but de ces expériences est de détecter des divergences observables, appelées *non-conformités*, entre implémentation et spécification. La dichotomie entre contrôle (entrées du système) et observation (sorties du système) amène à partitionner l'alphabet Σ_o des événements observables par le testeur en $\Sigma_o = \Sigma_? \cup \Sigma_!$, avec $\Sigma_?$ l'ensemble des événements d'entrées et $\Sigma_!$ l'ensemble des événements de sorties.

Dans le cadre du test de propriétés de sécurité, nous supposons dans un premier temps que le testeur se place du point de vue de l'attaquant (nous reviendrons sur cette hypothèse dans la remarque 3). On pose donc $\Sigma_a = \Sigma_? \cup \Sigma_!$.

On considère une implémentation $\mathcal{I}$ du système et un modèle de spécification G ayant tous deux Σ_a comme interface observable, une propriété de sécurité (soit une propriété de confidentialité avec φ comme secret, soit une propriété d'intégrité Ψ). On suppose qu'un contrôle d'accès "idéal" C et d'interface Σ_m a été calculé sur le modèle G, et permet d'assurer cette propriété sur le modèle produit $C \times G$ (c.f. section 3). On suppose également qu'un contrôle d'accès "réel" $\mathcal{I}_{\mathcal{AC}}$ de même interface Σ_m a été implémenté et connecté à $\mathcal{I}$ (on a alors $L(\mathcal{I} \times \mathcal{I}_{\mathcal{AC}}) \subseteq L(\mathcal{I})$). Comme expliqué plus haut, nous assimilons $\mathcal{I}_{\mathcal{AC}}$ et $\mathcal{I}$ à des modèles inconnus, ici des LTS. Le but du test est alors de vérifier que l'implémentation du contrôle d'accès, qui est censée réduire le comportement de l'implémentation $\mathcal{I}$ pour assurer la propriété de sécurité, est conforme à sa spécification C calculée pour la spécification G du système. L'architecture de test est représentée en Figure. 5.

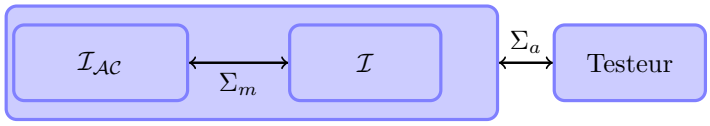


Fig. 5: Architecture de Test.

Nous proposons un algorithme de génération de tests prenant en entrée la spécification G du système, une propriété de confidentialité φ ou une propriété d'intégrité Ψ et son contrôle d'accès correspondant C. Il produit en sortie un cas de tests qui, quand

il est exécuté sur l'implémentation essaie simultanément d'invalider le contrôle d'accès implémenté, et de guider l'implémentation pour tenter de violer la propriété de sécurité. L'exécution d'un cas de tests peut détecter une ou plusieurs des situations suivantes :

- invalidation du contrôle (le moins restrictif) de l'implémentation contrôlée ;
- violation de la propriété de sécurité par l'implémentation contrôlée ;
- violation de conformité entre l'implémentation et la spécification. La relation de conformité entre l'implémentation boîte noire $\mathcal{I}$ et sa spécification G que nous avons choisie est $\leq_{io}$, une version de IOCO [Tre99] pour laquelle le blocage n'est pas pris en compte. $\leq_{io}$ est formellement définie par :

$$\mathcal{I} \leq_{io} G \triangleq \mathcal{T}_{\Sigma_a}(G).\Sigma_! \cap \mathcal{T}_{\Sigma_a}(\mathcal{I}) \subseteq \mathcal{T}_{\Sigma_a}(G) \tag{3}$$

  Informellement : après toute trace commune à la spécification G et l'implémentation I, toutes les sorties de l'implémentation doivent être acceptées par la spécification.

La relation de conformité a été formalisée par [Tre96] et la méthodologie du test de conformité relativement à IOCO ou à $\leq_{io}$ a été automatisée, depuis la génération de cas de tests à partir d'une spécification, jusqu'à l'exécution de ceux-ci sur une implémentation et l'obtention des verdicts [BFdV+99, Jé02].

4.1 Calcul du testeur canonique

Dans cette section, nous nous concentrons sur la génération automatique de tests pour des propriétés de confidentialité (la méthodologie pour des propriétés d'intégrité est similaire). Notons d'abord que ce type de propriété est défini par rapport aux comportements internes du système. Or le test de conformité "boîte-noire" n'observe que les comportements observables du système. Observer une trace de l'implémentation ne permet donc pas a priori de déduire quoique ce soit sur l'ensemble inconnu des trajectoires internes de celle-ci compatibles avec la trace observée. Nous avons donc besoin d'hypothèses supplémentaires reliant les comportements internes de l'implémentation à ceux de la spécification, qui eux sont connus. Par la suite, nous supposerons que :

Hypothèse 1 : $\mathcal{I} \leq_{io} G \Rightarrow L(\mathcal{I}) = L(G)$

Cette hypothèse signifie que le comportement (interne) de l'implémentation correspond à celui de la spécification tant qu'aucune non-conformité n'a été détectée par la campagne de tests. Même si cette hypothèse semble relativement restrictive, elle reflète bien le fait que l'attaquant connaît parfaitement le système qu'il cherche à attaquer. Elle est de plus nécessaire si l'on veut tester le comportement du contrôle d'accès. En effet, si $L(\mathcal{I})$ était différent de $L(G)$, alors le contrôle d'accès que l'on ajoute sur $\mathcal{I}$ serait différent de celui de G. L'opacité n'étant pas préservée par inclusion (c.f. remarque 1), même si on supposait $L(\mathcal{I}) \subseteq L(G)$, il pourrait exister dans $\mathcal{I}$ des fuites d'information non présentes dans G et réciproquement, ce qui implique un mécanisme de contrôle différent.

Remarque 2 *Si le but de la campagne de tests est simplement de découvrir si une fuite d'information est possible sur l'implémentation, alors l'hypothèse peut être relâchée et devient l'hypothèse 2 décrite ci-dessous.*

Hypothèse 2 : $\forall \mu \in T_{\Sigma_a}(\mathcal{I}) \cap T_{\Sigma_a}(G), \Pi_{\Sigma_a}^{-1}(\mu) \cap L(\mathcal{I}) \subseteq L(G)$

Cette nouvelle hypothèse signifie que dès qu'une trace observée (entrées/sorties) est acceptée par l'implémentation, alors toutes les séquences de l'implémentation compatibles avec cette observation sont aussi des séquences de la spécification. En particulier, cela implique que si le secret est révélé dans G après une trace μ, alors il sera sûrement révélé dans $\mathcal{I}$.

Par la suite on supposera que l'hypothèse 1 est respectée.

Comme décrit précédemment, nous allons chercher à tester une propriété de confidentialité et le contrôle d'accès qui lui correspond. Le testeur que nous considérons sera donc dérivé du modèle de spécification $G = (Q_{\mathrm{G}}, \Sigma, \rightarrow_{\mathrm{G}}, q^0_{\mathrm{G}})$, du secret spécifié par $\varphi = (Q_\varphi, \Sigma, \rightarrow, q^0_\varphi)$ muni de $F_\varphi \subseteq Q_\varphi$, et du modèle controlé $G_C = G \times C = (Q_{\mathrm{C}}, \Sigma, \rightarrow_{\mathrm{C}}, q^0_{\mathrm{C}})$ calculé précédemment. Notons que G_C décrit une propriété de sûreté, le plus grand langage observable et contrôlable inclus dans $L(G)$ qui garantit que le secret φ n'est jamais révélé. Le LTS $Comp^{V_{AC}}_{\Sigma}(G_C)$ muni de l'état accepteur V_{AC} est donc un observateur reconnaissant la négation de cette propriété.

Pour construire le testeur, nous considérons dans un premier temps le LTS suivant :

$$G^C_\varphi = (Q^C_\varphi, \Sigma, \rightarrow^C_\varphi, q^C_{0_\varphi}) = G \times \varphi \times Comp^{V_{AC}}_{\Sigma}(G_C)$$

avec les ensembles d'états accepteurs suivants :

- $F = Q_G \times F_\varphi \times (Q_{\mathrm{C}} \cup \{V_{AC}\})$, qui accepte les trajectoires de G satisfaisant le secret φ ;
- $F_{AC} = Q_G \times Q_\varphi \times \{V_{AC}\}$, qui accepte les trajectoires violant le contrôle d'accès.

G^C_φ possède les propriétés suivantes : comme φ et $Comp^{V_{AC}}_{\Sigma}(G_C)$ sont des LTS complets, $L(G^C_\varphi) = L(G)$. De plus, $L_F(G^C_\varphi) = L(G) \cap L_\varphi \subseteq L_\varphi$ tandis que $(L(G^C_\varphi) \backslash L_F(G^C_\varphi)) \cap L_\varphi = \emptyset$ et $L_{F_{AC}}(G^C_\varphi) \cap L(G_C) = \emptyset$. Le rôle de F est donc de reconnaître les trajectoires de G qui révèlent le secret, alors que le rôle de F_{AC} est de reconnaître les trajectoires de G qui violent le contrôle d'accès.

Exemple 4 *Basé sur l'exemple 3, le LTS G^C_φ est représenté par la figure 6 avec comme convention que les états rectangulaires appartiennent à F_{AC} et également à F si ils sont en pointillé.* ◇

Un testeur canonique permettant de tester la conformité d'une implémentation relativement à son modèle est généralement défini par $Test(G) = Comp^{Fail}_{\Sigma_!}(Det_a(G))$ [CJMR07]. Comme notre but est également de tester l'opacité du secret φ et la validité du contrôle d'accès, le testeur canonique est ici construit à partir de G^C_φ de la manière suivante :

$$Test(G, \varphi) = (X, \Sigma_a, \rightarrow_t, X_o) = Comp^{Fail}_{\Sigma_!}(Det_a(G^C_\varphi))$$

$Test(G, \varphi)$ peut être vu comme un cas de tests qui "raffine" le testeur canonique $Test(G)$ dans la mesure où il permet non seulement de détecter la non-conformité, mais également les fuites d'informations relatives à φ ainsi qu'une implémentation incorrecte du contrôle d'accès devant assurer l'opacité du secret φ. Toutefois, à cause de l'architecture de tests et, en particulier le fait que le testeur n'observe que partiellement le système via l'interface Σ_a, les verdicts du testeur ne sont pas donnés relativement aux trajectoires du système, mais relativement aux traces observées, correspondant à l'ensemble des trajectoires de même observation.

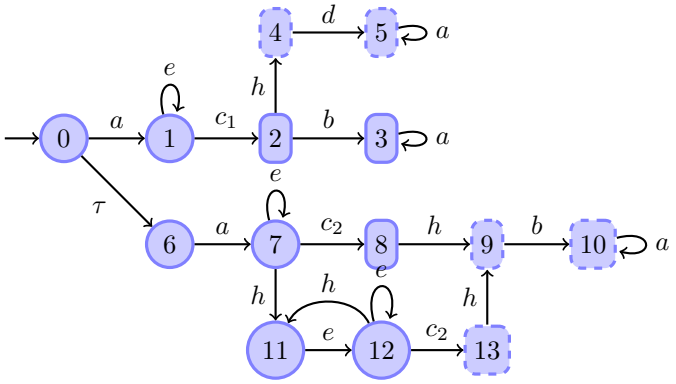


Fig. 6: G^C_φ.

En se basant sur les différents ensembles d'états accepteurs, $\{Fail\}$, F et F_{AC}, on peut attacher à $Test(G,\varphi))$ les différents verdicts suivants : $\forall \mu \in \mathcal{T}_{\Sigma_a}(\mathcal{I}\times\mathcal{I}_{AC})\cap L(Test(G,\varphi))$,

$$\mathcal{O}_{Test(G,\varphi)}(\mu) = \begin{cases} NotConf & \text{si} \quad \Delta_{Test(S,\varphi)}(X_0,\mu) = \{Fail\} \\ Violate_{AC} & \text{si} \quad \Delta_{Test(S,\varphi)}(X_0,\mu) \subseteq F_{AC} \ \wedge \Delta_{Test(S,\varphi)}(X_0,\mu) \not\subseteq F \\ Leak & \text{si} \quad \Delta_{Test(S,\varphi)}(X_0,\mu) \subseteq F \end{cases}$$

Quand ce testeur est exécuté en parallèle avec l'implémentation $\mathcal{I}\times\mathcal{I}_{\mathcal{AC}}$, après une séquence observée d'entrées/sorties μ, l'interprétation des verdicts est la suivante :

- Si $\mathcal{O}_{Test(G,\varphi)}(\mu) = NotConf$, alors par définition de $Comp^{Fail}_{\Sigma_!}(Det_{\Sigma_a}(G^C_\varphi))$, $\exists \mu' \in \Sigma^*, a \in \Sigma_!, \mu = \mu'.a$ avec $\mu' \in L(Det_{\Sigma_a}(G^C_\varphi)) = \mathcal{T}_{\Sigma_a}(G^C_\varphi) = \mathcal{T}_{\Sigma_a}(S)$. Donc $\mu \in \mathcal{T}_{\Sigma_a}(G).\Sigma_!$, et $\mu \in \mathcal{T}_{\Sigma_a}(\mathcal{I}\times\mathcal{I}_{AC}) \subseteq \mathcal{T}_{\Sigma_a}(\mathcal{I})$ mais $\mu \notin L(Det_{\Sigma_a}(G^C_\varphi)) = \mathcal{T}_{\Sigma_a}(G)$. Ainsi par définition de $\leq_{io}$, $\neg(\mathcal{I} \leq_{io} G)$ et la campagne de tests peut être stoppée.
- Si $\mathcal{O}_{Test(G,\varphi)}(\mu) = Violate_{AC}$, alors $[\![\mu]\!]_{\Sigma_a} \subseteq L_{F_{AC}}(G^C_\varphi)$. Mais comme $L_{F_{AC}}(G^C_\varphi) \cap L(G_C) = \emptyset$, on obtient $[\![\mu]\!]_{\Sigma_a} \cap L(G_C) = \emptyset$. Comme de plus, $[\![\mu]\!]_{\Sigma_a} \not\subseteq L_{F_{AC}}(G^C_\varphi)$, il existe $s \in [\![\mu]\!]_{\Sigma_a} \setminus L_\varphi$. On peut donc en conclure que le contrôle d'accès n'est pas bien implémenté, mais que le secret n'a pas encore été révélé.
- Si $\mathcal{O}_{Test(S,\varphi)}(\mu) = Leak$, alors $[\![\mu]\!]_a \subseteq L_F(G^C_\varphi)$ et $[\![\mu]\!]_a \subseteq L_\varphi$, ce qui implique une fuite d'informations.

Exemple 5 *En reprenant l'exemple 4, le testeur $Test(G,\varphi)$ est donné par le LTS représenté par la figure 7. Pour cet exemple, on suppose que $\Sigma_a = \{a,b,d,e\}$ avec comme partitionnement entrées/sorties $\Sigma_! = \{b,e\}$ et $\Sigma_? = \{a,d\}$.*

Le verdict $Violate_{AC}$ est émis si en exécutant le testeur en parallèle avec l'implémentation, l'état $\boxed{3,10}$ est atteint, signifiant que le contrôleur d'accès n'est pas bien implémenté. Si la séquence a.d est observée, le secret est révélé à l'attaquant et le testeur émet le verdict Leak (l'état $\boxed{5}$ appartenant à F). Le verdict Fail est émis si l'état Fail est atteint (une non-conformité ayant été détectée : la sortie e n'aurait pas dû être admissible dans les états $\boxed{0,6}$ ou $\boxed{3,10}$). ◇

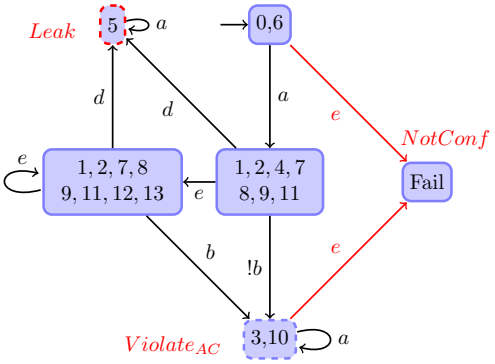


Fig. 7: $Test(G, \varphi)$.

Remarque 3 *Dans certains cas, il est possible que l'interface entre le testeur et l'implémentation soit différente de celle entre l'utilisateur et l'implémentation. La manière de calculer le testeur est alors un peu différente. Supposons que l'interface de test soit donnée par* $\Sigma'_m \neq \Sigma_a$.

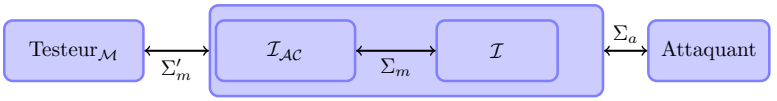


Fig. 8: Architecture de test (II).

La principale différence concerne le calcul des verdicts Leak. En effet, le secret φ *est révélé à l'attaquant par une séquence* $s \in L(G)$ *si et seulement si* $\Pi_a(s) \in L_F(Det_a(G \times \varphi))$. *Nous sommes intéressés par tester la propriété suivante : "Le secret* φ *a été révélé à l'attaquant", qui correspond au langage :* $\Pi_a^{-1}(L_F(Det_a(G \times \varphi))) \cdot \Sigma^*$. *Ce langage peut être reconnu par un LTS* Ω, *muni d'un ensemble d'états accepteurs* F_Ω *t.q. :*

$$\mathcal{L}_{F_\Omega}(\Omega) = \Pi_a^{-1}(L_F(Det_a(G \times \varphi))) \cdot \Sigma^*. \tag{4}$$

Par la suite, le calcul du testeur $Tester_{\mathcal{M}}$ *est obtenu en remplaçant* φ *par* Ω *dans l'algorithme de génération de tests que nous venons de décrire et en prenant comme interface d'observation* Σ'_m. $\diamond$.

4.2 Sélection de cas de test

Le but premier de la campagne de tests est de tester la violation de la propriété de confidentialité (ce qui se traduit par l'émission du verdict *Leak*). Si l'implémentation ne peut plus mener le testeur à générer ce verdict, alors l'exécution du cas de tests ne pourra plus détecter la violation de la propriété et il peut donc être intéressant d'arrêter le test (cela se traduira par l'émission d'un verdict spécial *Inconclusive*). Cette opération consiste à supprimer dans $Test(S, \varphi)$ l'ensemble des sous-graphes qui ne mènent plus à *Leak*. Suivant qu'un tel sous-graphe est atteint par une entrée ou une sortie, la manière d'élaguer le graphe est différente.

- Le sous-graphe est atteint par une *entrée*. Dans ce cas, la transition correspondante est simplement enlevée. Intuitivement, c'est le testeur qui choisit les entrées à envoyer à l'implémentation, celui-ci peut donc décider de ne pas stimuler l'implémentation avec cette entrée si il sait que la détection de la violation de la propriété ne pourra plus se faire.
- Le sous-graphe est atteint par une *sortie* (qui ne mène pas directement à $Fail$). Dans cas, la transition correspondante est gardée mais redirigée vers un état puits $Inconc$, qui signifie que le verdict $Leak$ ne peut plus être émis (mais la non-conformité n'a pas été détectée). On émet alors le verdict $Inconclusive$, i.e. $\forall \mu \in \mathcal{T}_{\Sigma_a}(\mathcal{I} \times \mathcal{I}_{AC}) \cap L(Test(G, \varphi))$,

$$\mathcal{O}_{Test(G,\varphi)}(\mu) = Inconclusive \text{ si } \Delta_{Test(S,\varphi)}(X_0, \mu) \subseteq \{Inconc\}.$$

Exemple 6 *Le résultat de cette opération appliquée sur le testeur $Test(G, \varphi)$ de la figure 7 est donné par le LTS représenté par la figure 9. Après l'émission d'un a, si l'on reçoit en sortie un événement b, alors le test peut s'arrêter, comme φ ne peut plus être révélé à l'attaquant.* ◇

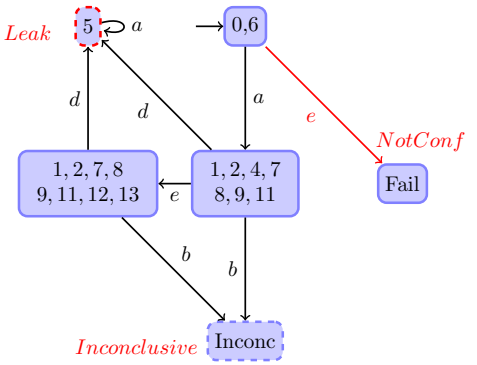


Fig. 9: Le testeur $Test(G, \varphi)$.

5 Conclusion

Dans cet article, nous nous sommes intéressés à la génération automatique de tests pour des propriétés de sécurité sur des systèmes ouverts partiellement observables. Notre approche est orientée modèle : nous supposons que l'on dispose d'une spécification du système modélisée par un système de transitions fini ainsi que d'une modélisation sous forme de langages rationnel des propriétés de confidentialité et d'intégrité. Notre méthode consiste à calculer dans un premier temps un modèle du contrôle d'accès idéal assurant ces deux types de propriétés de sécurité. La deuxième étape de notre approche consiste à calculer automatiquement un testeur dont le but est de générer des cas de tests qui, exécutés sur l'implémentation du système, permettent de détecter la non-conformité de l'implémentation et sous certaines hypothèses, la violation des propriétés de sécurité, ainsi qu'une éventuelle mauvaise implémentation du contrôle d'accès mis en place pour assurer ces propriétés de sécurité.

Ce travail ouvre de nombreuses perspectives. On peut envisager notamment d'étendre ces résultats à des modèles plus expressifs tels que les systèmes avec des données à domaines non bornés afin d'appréhender les questions de confidentialité de données. Le point délicat devient alors le calcul du contrôleur et du testeur qui nécessitent de passer par des approximations. Dans le même ordre d'idées, nous avons supposé ici que l'attaquant connaissait parfaitement le système (hypothèse 1) et cette hypothèse joue un rôle essentiel pour la validité des verdicts. On peut rechercher des relations moins fortes entre une implémentation et son modèle abstrait afin d'assurer que les résultats des tests soient préservés sur l'implémentation.

Références

[AČZ06] 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.

[BAF05a] B. Blanchet, M. Abadi, 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.

[BAF05b] B. Blanchet, M. Abadi, 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.

[BBB+07] E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17 :425–446, 2007. extended version of a Wodes'06 paper.

[BFdV+99] A. Belinfante, J. Feenstra, R. de Vries, J. Tretmans, N. Goga, L. Feijs, and S. Mauw. Formal test automation : a simple experiment. In International Workshop on the Testing of Communicating Systems (IWTCS'99), pages 179–196, 1999.

[BKMR06] Jeremy Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. In Theo Dimitrakos, Fabio Martinelli, Peter Y. A. Ryan, and Steve A. Schneider, editors, *Revised Selected Papers of the 3rd International Workshop on Formal Aspects in Security and Trust (FAST'05)*, volume 3866 of *Lecture Notes in Computer Science*, pages 81–95, Newcastle upon Tyne, UK, 2006. Springer.

[BKMR08] Jeremy W Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6) :421–435, May 2008.

[BL73] David D.E. Bell and Leonard J. La Padula. Secure computer system : Mathematical foundations. Technical Report 2547, MITRE, March 1973.

[CJMR07] C. Constant, T. Jéron, H. Marchand, and V. Rusu. Integrating formal verification and conformance testing for reactive systems. *IEEE Transactions on Software Engineering*, 33(8) :558–574, August 2007.

[CL99] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.

[DDM08] J. Dubreil, Ph. Darondeau, and H. Marchand. Opacity enforcing control synthesis. In *Workshop on Discrete Event Systems, WODES'08*, Gothenburg, Sweden, March 2008.

[DDM09] J. Dubreil, Ph. Darondeau, and H. Marchand. Supervisory control for opacity. Research Report 1921, IRISA, 2009.

[DFG+06] 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.

[DJM07] J. Dubreil, T. Jéron, and H. Marchand. Construction de moniteurs pour la surveillance de propriétés de sécurité. In *6ème Colloque Francophone sur la Modélisation des Systèmes Réactifs*, Lyon, France, October 2007.

[DJM08] J. Dubreil, T. Jéron, and H. Marchand. Monitoring information flow by diagnosis techniques. Technical Report 1901, IRISA, August 2008.

[FG00] R. Focardi and R. Gorrieri. Classification of security properties : Information flow. In *Foundations of Security Analysis and Design, LNCS No 2171*, pages 331–396, 2000.

[Jé02] T. Jéron. Tgv : théorie, principes et algorithmes. *Techniques et Sciences Informatiques, numéro spécial Test de Logiciels*, 21(9) :1265–1294, 2002.

[LBW05] J. Ligatti, L. Bauer, and D. Walker. Edit automata : enforcement mechanisms for run-time security policies. *Int. J. Inf. Sec.*, 4(1-2) :2–16, 2005.

[LG07] G. Le Guernic. Information flow testing - the third path towards confidentiality guarantee. In *Advances in Computer Science, ASIAN 2007. Computer and Network Security, LNCS No 4846*, pages 33–47, 2007.

[Low99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3) :89–146, 1999.

[Maz04] Laurent Mazaré. Using unification for opacity properties. In *Proceedings of the 4th IFIP WG1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 165–176, Barcelona (Spain), 2004.

[RW89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE ; Special issue on Dynamics of Discrete Event Systems*, 77(1) :81–98, 1989.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1) :30–50, 2000.

[SS96] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *Computer Security SORICS 96, LNCS No 1146*, pages 198–218, 1996.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3) :103–120, 1996.

[Tre99] J. Tretmans. Testing concurrent systems : A formal approach. In *Concurrency Theory (CONCUR'99)*, number 1664 in LNCS, pages 46–65, 1999.