République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaia
Faculté des Sciences Exactes

Département d'Informatique

ECOLE DOCTORALE RESEAUX ET SYSTEMES DISTRIBUES

Mémoire de Magister

En Informatique

Option : Réseaux et Systèmes Distribués

Thème

Elaboration d'un cadre formel pour le renforcement de politiques de sécurité dans les programmes

---

Présenté par

M^r^ AISSANI Sofiane

Devant le jury composé de :

Président	KERKAR Moussa	Professeur	Université de Bejaia, Algérie
Rapporteur	ADI Kamel	Professeur	Université du Québec en Outaouais, Canada
Rapporteur	MEJRI Mohamed	Professeur	Université de Laval, Canada
Examinateur	BEGHDAD Rachid	M.C.A	Université de Bejaia, Algérie
Examinateur	ALIOUAT Makhlouf	M.C.A	Université de Sétif, Algérie

Promotion 2007-2008

Remerciements

A l'issue de ce travail je remercie, en premier lieu, le bon dieu de m'avoir donné la force et le courage de le mener à terme.

J'adresse adresse mes sincères remerciements à mes directeurs de thèse, le Professeur Kamel ADI et le professeur Mohamad MEJRI pour nous m'avoir encadré pour la réalisation de ce travail.

Mes profonds remerciements vont à Lamia HAMZA qui a toujours été présente quand j'avais besoin d'elle.

Mes sincères remerciement s'adressent à Mr KERKAR Moussa d'avoir accepté de présider le jury de ma soutenance, et M^{r} BEGHDAD Rachid et M^{r} ALIOUAT Makhlouf de m'avoir fait l'honneur d'être dans ce jury.

Je remercie également tous ceux qui ont contribué à notre formation, en particulier monsieur TARI Kamel, pour tous les efforts qu'il fourni pour notre école doctorale.

Et que tous ceux qui ont contribué de prêt ou de loin à la réalisation de ce travaille trouve ici l'expression de notre sincère gratitude.

Dédicaces

A mes deux êtres les plus chers sur t erre : mes parents, pour leur amour, confiance et sacrifice sans limite.

A mes frères et sœurs : Loulou, Didine, Sassah et sa famille,

A tous mes cousins et cousines, en particulier Malosse et Moumi.

A mon club d'échecs (Cavalier Fou d'Aokas) : Lkhier Topalov, Bilal Karpov, Saadi Fisher, Mustapha Kasparov et à tous les autres adhérents.

A la famille du CSP d'Aokas : Hicham, Samia, Aouicha, Sabrina, Nadir, Nadia, Nacir, Mme Hammadi et Mme MERROUL, Ferhat, et tous les autres (personnel et adhérents).

A toute la promotion se Resyd5 en particulier Said, Samia, Rbiha, Zahia et Badrina.

A tous les étudiants de l'école doctorale ReSyD. Et tous mes étudiants du département informatique.

A tous les amis de la composante connexe (ils se reconnaitront), en particulier mon ex Binôme Nabil MEDDOUR.

A tous les autres amis : Linda M, Riad H, Mebrouk L, Ghanou A, Hassane M, Faycel A et à tous mes anciens amis de l'USTHB, la liste est tellement longue que je ne peux citer tous le monde.

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

Depuis la nuit des temps, les humains avaient des secrets qu'ils voulaient garder pour eux mêmes, pour des raisons personnelles ou autres. L'un des exemples les plus connus est : Lorsque Jules César envoyait des messages à ses généraux, il ne faisait pas confiance à ses messagers. Il remplaçait donc tous les "A" contenus dans ses messages par des "D", les "B" par des "E", et ainsi de suite pour tout l'alphabet. Seule la personne connaissant la règle du "décalage par trois" pouvait déchiffrer ses messages.

De nos jours, l'informatique et la sécurité sont indissociables et sont à la confluence de diverses disciplines scientifiques comme les mathématiques, la physique et la logique . . . etc. Nous, informaticiens, vivons dans un contexte hostile, nous sommes harcelés par les pirates et les utilisateurs malicieux, qui ont été aidé par les mutations profondes et rapides de l'environnement de l'information et les proliférations des nouvelles technologies. Cette mutation a donné naissance à plusieurs technologies, parmi elles, on s'intéresse à celle des codes mobiles.

On entend par code mobile un ensemble d'instructions écrites dans un langage de programmation qui s'exécute sur une ou plusieurs machines hôtes auxquelles il est lié temporairement (le temps de l'exécution), il va de la simple applet à des agents logiciels intelligents.

Les principaux avantages des codes mobiles sont : l'optimisation de la bande passante et la flexibilité, comme d'habitude, une plus grande flexibilité est livrée avec un coût qui est une plus grande vulnérabilité aux intrusions, ce qui justifie la citation qui dit qu'un code mobile est utile, mais il est potentiellement hostile.

Au milieu des années 80, Basit et Amdjad de Lahore, Pakistan, s'aperçurent que des personnes pirataient leurs logiciels. Ils réagirent en écrivant le premier virus informatique, un programme plaçant sa propre réplique et un message de copyright dans chaque disquette copiée par leurs clients, de cette simple histoire a commencé toute une contre-culture du virus. Aujourd'hui les nouveaux virus peuvent balayer la planète en quelques heures et les craintes qu'ils suscitent sont amplifiées par les medias. Il suffit alors d'un simple clic sur la souris ou d'être tous simplement connecté au réseau mondial pour être menacé d'une possible infection virale.

Le présent projet entre dans le domaine de la sécurité dans les programmes, en particulier les codes mobiles. Nous soulignons trois objectifs principaux pour ce projet : Le premier consiste à classer les aspects de la sécurité reliés aux codes mobiles, ensuite d'étudier différentes méthodes et approches utilisées pour les combattre. Le second objectif consiste à effectuer une étude comparative entre les différentes techniques étudiées. Le troisième objectif consiste proposer une nouvelle solution pour combattre les codes mobiles.

Le rapport est structuré en quatre chapitres : le premier est un état de l'art sur le domaine de la sécurité informatique, commençant par un historique du phénomène de la piraterie, puis introduit les notions les plus utilisées dans le domaine et se termine par la présentation de quelques techniques formelles utilisées dans le domaine.

Le second chapitre tente de classer les différents aspects reliés à la sécurité des codes mobiles, puis de présenter quelques techniques utilisées pour la protection contre les codes mobiles malicieux, et se termine en dressant une liste de critères de comparaisons et des tableaux de comparaison entre les techniques.

Le troisième chapitre présente une étude de cas, la méthode élue est une méthode formelle se basant sur l'algèbre de processus BPA, puis on étudie la possibilité d'utiliser un autre type d'équivalence plus intéressant que celui utilisé par les créateurs de la méthode.

Le dernier chapitre s'engage à introduire *l'unification modulo une théorie équationnelle* qui utilise l'équivalence des environnements, ensuite *définir une logique* qui nous permettra d'exprimer les propriétés de sécurité d'une manière plus naturelle, puis, nous *proposons une idée pour l'optimisation* des calculs, et avant de finir on va adapter la méthode du troisième chapitre aux nouvelles notions, en énonçons quelques *propositions que nous validerons par des démonstration formelles*. Une fois ceci réalisé, nous entamons la tache

laborieuse d'évaluation de la nouvelle méthode, d'abord par des exemples, ensuite par un tableau comparatif.

Pour finir, La conclusion générale synthétise ce que nous avons étudié dans le cadre de ce projet, et met l'accent sur tous les apports de notre travail, ensuite, présente une liste de perspectives et axes de recherches qui permettront l'amélioration du travail réalisé.

CHAPITRE I

Etat de l'art sur la sécurité informatique

1. Introduction

Alors que l'ère de l'information devient une réalité pour un nombre croissant d'individus dans le monde, les technologies qui la sous tendent se font sans cesse plus sophistiquées et utiles. Les opportunités sont immenses. Cela représente un gigantesque bond en avant qui décuple leurs capacités à communiquer et à créer, à s'exprimer et à se faire entendre; paradoxalement ces avancées technologiques ont donné naissance à leur plus grand ennemi, le piratage, qui a grandi au même temps qu'elles. C'est dans le but de neutraliser cet ennemi qu'on consacre de plus en plus d'efforts pour le développement de la sécurité informatique.

On peut considérer deux aspects de la sécurité informatique, la sécurité physique et la sécurité logique ; le premier type concerne les locaux où est placé le matériel informatique utilisé pour la gestion des données, tels que les serveurs de télécommunications, serveurs de base de données, ordinateurs, etc.….

Mais ce qui nous intéresse dans ce travail, c'est le second type, c'est pour cela que nous consacrons ce chapitre à son étude ; nous étudierons d'abord les notions de base de la sécurité informatique, puis quelques menaces existantes et les mécanismes qui ont été proposés comme solutions, puis nous présenterons les méthodes formelles les plus utilisées dans ce domaine et nous terminerons par une conclusion.

2. Historique et motivation du piratage informatique

Le programme Elk Cloner est le premier à s'être répandu hors d'un laboratoire en 1982, avant même la formalisation des virus [1]. L'objectif de son inventeur était juste de tester ses capacités, ceci veut dire que ça relevait juste d'une curiosité ludique.

En 1983 Fred Cohen développa, à des fins de recherche, l'un des premiers programmes possédant la capacité d'infecter d'autres programmes et de se reproduire [2].

En 1988 le premier ver ayant une conception adaptée pour Internet fit son apparition. Ce verre nommé MORRIS du nom de son concepteur causa de gros dégâts. Sa vitesse de propagation importante suscita l'intérêt des pirates informatiques. Des travaux ont montré qu'il est possible d'infecter plus de trois cents mille machines en moins de quinze minutes [3]. Cette vitesse, bien que théorique, a été approchée par le ver CODE RED, lancé à des fins commerciales à l'encontre des serveurs Web de Microsoft en 2001 [4].

En 2005 le premier virus sur téléphones mobiles a été détecté, COMMWARRIOR. Il utilise la technique de stéganographie et la technologie MMS pour se propager. [5]

Selon l'enquête annuelle de Symantec sur la menace informatique plus 1.6 millions programmes malveillants ont été détectés au cours de l'année 2009. [5]

3. Notions de base de la sécurité informatique

Les notions les plus utilisées dans la sécurité informatique sont :

Identification : l'obtention de l'identité d'une personne ou d'une entité.

Authentification : vérification qu'une personne ou une entité correspond bien à son identité déclarée.

Autorisation : vérification qu'une personne ou une entité possède les droits nécessaires pour accéder ou modifier une ressource.

Disponibilité : capacité d'une ressource à être accessible.

Intégrité : propriété d'une information qui n'a pas été modifiée où altérée lors d'un échange (par exemple sur un réseau) ou pendant sa conservation.

Confidentialité : propriété d'une information qui n'est ni disponible, ni divulguée aux personnes ou entités non autorisées.

Maintenant que nous connaissons toutes ces notions, nous pouvons définir la sécurité informatique comme étant la discipline qui nous permet d'identifier et d'authentifier les entités qui possèdent les droits d'accès nécessaires aux ressources qui doivent être disponibles et intègres, et peuvent même contenir des données confidentielles.

4. Les menaces et solutions

Les différentes menaces peuvent être énumérées comme suit [6] :

Spoofing identity (usurpation d'identité) un utilisateur non accrédité usurpe l'identité d'un utilisateur valide de l'application.

Tampering with data (altération des données) un utilisateur détruit ou modifie les informations sans autorisation.

Repudiability (répudiation) la possibilité qu'un utilisateur puisse nier avoir effectué telle ou telle opération.

Information disclosure (divulgation d'information) des données confidentielles sont rendues visibles à des utilisateurs non accrédités.

Denial of service (déni de service) l'application ou la ressource est rendue indisponible pour l'entité qui veut l'utiliser.

Elevation of privilege (Elévation de privilège) un utilisateur dispose un niveau d'accès à l'application supérieur à celui qui devrait lui être accessible.

Pour se prémunir de ces menaces, différentes techniques liées à la mise en œuvre de la sécurité sont envisageables :

Authentification identification des applications clientes. Les mécanismes d'authentification permettent de se prémunir contre l'usurpation d'identité (signature).

Sécurité des communications il faut faire en sorte que les messages circulant sur le réseau restent privés (chiffrement) et non altérés (signature du message).

Audit enregistrement des actions de l'application cliente, qu'elles soient autorisées ou non, pour garantir la non répudiation.

Autorisation définition des droits d'un client authentifié vis-à-vis d'une application ou d'une ressource.

5. Mécanismes de sécurité

Les différents mécanismes de sécurité sont repartis en deux niveaux :

Premier niveau : A ce niveau de la sécurité, ou l'accès à une ressource est permis ou bien il ne l'est pas.

Second niveau : (contrôle d'accès) Concerne les autorisations, ce qui veut dire : qui a le droit de faire Quoi ?et sur quelle ressource ?

5.1. Premier niveau

5.1.1. L'authentification

Authentifier quelqu'un signifie s'assurer de son identité, cette authentification n'est possible que si les deux protagonistes partagent un secret commun, ce secret peut prendre divers formes, et comme conséquence on distingue deux modes d'authentification :

5.1.1.1. Authentification directe

Lorsque l'entité A qui veut s'authentifier auprès de l'entité B, s'adresse directement à elle et lui divulgue le secret qu'ils partagent.

5.1.1.2. Authentification indirecte

Lorsque les deux entités ne partagent aucune relation de confiance, dans ce cas il est nécessaire d'utiliser un agent intermédiaire qui va vérifier l'identité de l'entité qui veut s'authentifier. Les deux mécanismes les plus connus qui illustrent ce type d'authentification sont

A. **Les certificats X509 :** Par définition, un certificat est un document établissant un lien entre une clef et un sujet ou un privilège. Cette liaison dépend du contexte et du type

de certificat. Un certificat peut servir à établir un ou plusieurs faits dont la confirmation de l'identité d'une personne, de l'identification d'une société.

L'ITU (International Télécommunication Union) et l'ISO (International Organisation for Standardisation) ont publié la première version du standard X.509 qui a été adoptée par l'IETF (International Enginering Task Force). A l'origine conçue en 1988, le certificat X.509 est passé par trois versions principales pour arriver à sa forme actuelle, comme ceci est illustré par la figure :

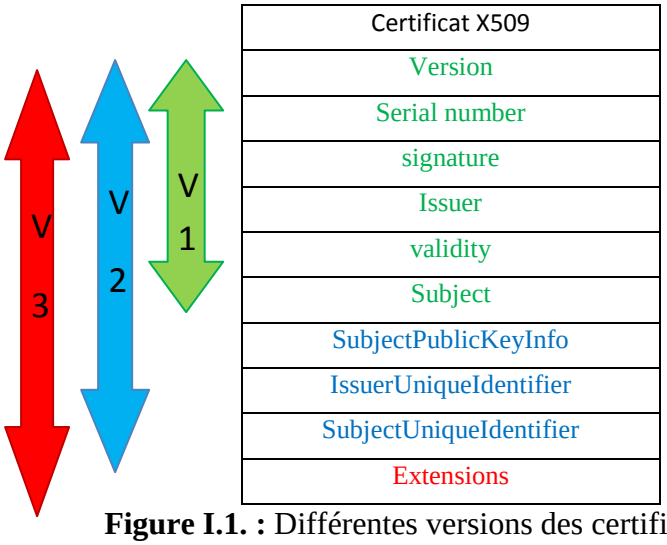

Figure I.1. : Différentes versions des certificats X509

Un certificat X.509v3 contient les données énumérées dans la figure. Ce format regroupe un certain nombre de champs « standards » et de champs dits « d'extension », qui permettent de personnaliser les certificats en fonction de leur usage. Néanmoins, une partie des extensions est « normalisée » par X.509v3 et notamment celles permettant de spécifier un certain nombre d'informations en fonction de l'usage prévu d'un certificat. Notons que certains de ces champs sont facultatifs (extensions).

B. Le ticket Kerberos : Kerberos, en français Cerbère, qui est dans la mythologie grecque le monstrueux chien à trois têtes gardien des enfers, c'est le nom choisi pour le mécanisme d'authentification d'un projet appelé Athena [7], la version 5 du protocole est un standard.

Kerberos est un protocole qui permet de vérifier l'identité de plusieurs entités (un utilisateur, un serveur, . . . etc.), sur un réseau quelconque, n'offrant aucune sécurité particulière. Son principe de fonctionnement est le suivant :

L'utilisateur s'authentifie (avec un mot de passe ou une carte à puce) vis-à-vis du service KDC (Key Distribution Center). Le client localise le KDC le plus proche en faisant appel au service de nommage (DNS, Domain Name Service). Le KDC délivre un Ticket Granting Ticket (TGT). Le TGT est un jeton qui doit, pour pouvoir accéder à d'autres ressources, être présenté au Ticket Granting Service (TGS) qui lui, délivre des tickets d'accès (Session Tickets) aux services réseau. Ces tickets contiennent des données chiffrées incluant le mot de passe qui confirme l'identité de l'utilisateur vis-à-vis du service réseau sollicité. L'utilisateur peut alors utiliser la ressource conformément aux droits qui lui sont attribués. Le processus d'authentification est ainsi entièrement masqué à l'utilisateur

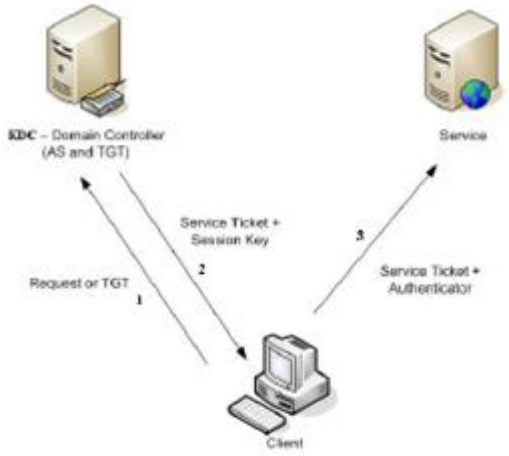

Figure I.2 : Principe de fonctionnement de Kerberos

5.1.2. Sécurité des communications

5.1.2.1. La cryptographie (chiffrement)

La cryptographie possède ses origines dans l'ancien grec « Kryptos » signifiant « cacher » et de « Graphein » signifiant « écrire ».C'est la science de la dissimulation de messages de sorte que seuls certains initiés disposant d'une donnée secrète, appelée « clef », soient en mesure de prendre connaissance de leur contenu.

On distingue trois grands types de chiffrement : chiffrement à clef secrète, chiffrement à clef publique, chiffrement non inversible.

A. Le chiffrement à clef secrète : Les algorithmes de chiffrement à clef secrète (ou symétriques ou encore conventionnels) sont ceux pour lesquels émetteur et destinataire partagent une même clef secrète, autrement dit, les clefs de chiffrement et de déchiffrement sont identiques.

Si on note $M' = c(M, C)$ pour dire M' est le message M crypté avec la clef C, et $M = d(M', C)$ pour dire que le décryptage du message M' avec la clef C donne le message M. La propriété principale de la cryptographie symétrique est la suivante : $d(c(M, C), C) = M$.

Comme exemples d'algorithmes de e cryptographie asymétrique, on peut citer IDEA (International Data Encryption Algorithm), DES (Data Encryption Standard).

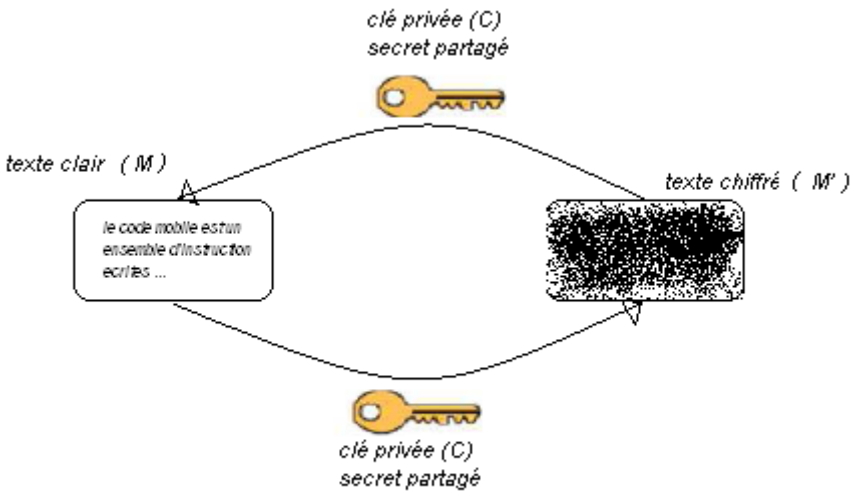

Figure I.3 : Principe de la cryptographie symétrique

B. Chiffrement à clef publique : La cryptographie à clef publique (ou asymétrique) : inventée en 1976 par Difie et Hellman, mais l'algorithme le plus connu de chiffrement asymétrique est ce lui de Rivest [8](RSA) évite le partage d'un secret entre les deux protagonistes, ce qui veut dire qu'il y a deux clefs différentes (publique et privée). Les clés publique et privée sont utilisées conjointement aux règles respectives de chiffrement et de déchiffrement. La clé publique est distribuée (diffusée) par le

destinataire à tout émetteur souhaitant lui communiquer des informations sensibles. Le destinataire conserve toutefois le secret de sa clé privée.

Si on note $M' = c(M, Cp)$ pour dire que le cryptage du message M avec la clef publique Cp donne le message M', alors le déchiffrement de M' se fait par une autre clef Cr différente de Cp, $M = d(M', Cr)$.

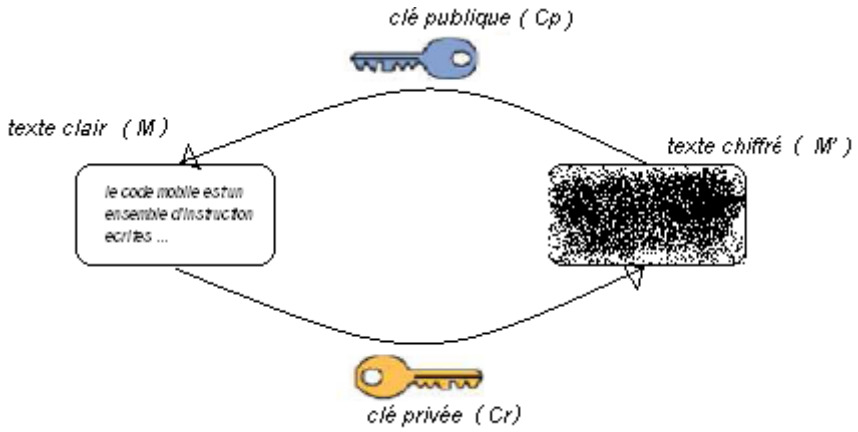

Figure I.4: Principe de la cryptographie asymétrique

C. **La signature numérique** (cryptographie non inversible) : c'est une petite portion de données chiffrées attachée à un message, elle est utilisée pour s'assurer qu'un message est authentique et intact. Son rôle est d'obtenir les mêmes fonctionnalités que la signature que l'on appose au bas d'un texte à support papier, il faut que chacun puisse vérifier une signature mais que personne ne puisse l'imiter. Elle garantit alors à la fois l'intégrité du message reçu et l'identité de son auteur. L'exemple le plus connu est l'algorithme MD5 (Message Digest 5).

Certains algorithmes de chiffrement à clef publique fournissent immédiatement une solution : il suffit à l'auteur de chiffrer un condensé (obtenu par l'emploi d'une fonction de hachage) du message avec sa propre clef secrète pour produire une signature ; chacun pourra alors la vérifier en utilisant la clef publique correspondante.

La fonction de Hachage permet d'obtenir une « emprunte digitale » à partir d'un message M, elle est notée $e(M)$. Cette fonction doit satisfaire les conditions suivantes :

- Soit un message M, $e(M)$ est facilement calculable.

- Il est impossible de trouver M à partir d'une emprunte $e(M)$.

Soit un message M, il est pratiquement impossible de trouver un autre message M' différent de M tel que : $e(M) = e(M')$.

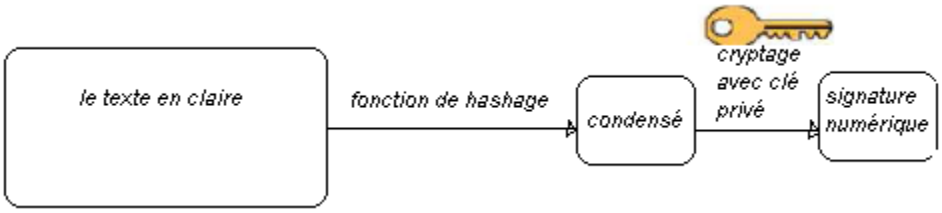

Figure I.5 : Création d'une signature numérique

5.2. Second niveau de la sécurité (Le contrôle d'accès)

En général les systèmes de contrôle d'accès peuvent être représentés en termes de sujets, d'objets, d'opérations et de règles dont les définitions sont les suivantes :

- ✓ Un sujet peut être une application, un humain . . . etc. il interagit avec des données informatiques.
- ✓ Un objet est la donnée avec laquelle le sujet veut interagir (la ressource).
- ✓ Une opération représente ce que veut faire le sujet sur l'objet (lecture, exécution. . .).
- ✓ Une règle définit une relation entre un sujet, un objet et une opération. (Un ensemble de règles représente une politique de contrôle d'accès).

Parmi les mécanismes de contrôle d'accès les plus connus, on peut citer RBAC, XACML :

5.2.1. RBAC (role based access control)

Le standard RBAC est développé par le NIST (National Institute of Standards and Technology) Dans ce modèle, les permissions d'accès pour les opérations sont basées sur les rôles que jouent les utilisateurs dans l'organisation concernée. Tous les sujets ayant reçu l'autorisation de jouer un rôle hérite alors des permissions associées à ce rôle. Chaque rôle définit un ensemble de droits d'accès et l'usage des ressources est limité aux individus autorisés à endosser l'un des rôles autorisant cet accès précis.[9]

RBAC se compose de quatre composants de base :

1. USERS : un ensemble d'utilisateurs.

2. ROLES : représente la fonction ou la responsabilité de l'utilisateur.

3. PERMISSIONS : approuver ou refuser certains modes d'accès à des objets.

4. SESSIONN : représente chaque session via laquelle un utilisateur est assigné à un ou plusieurs rôles.

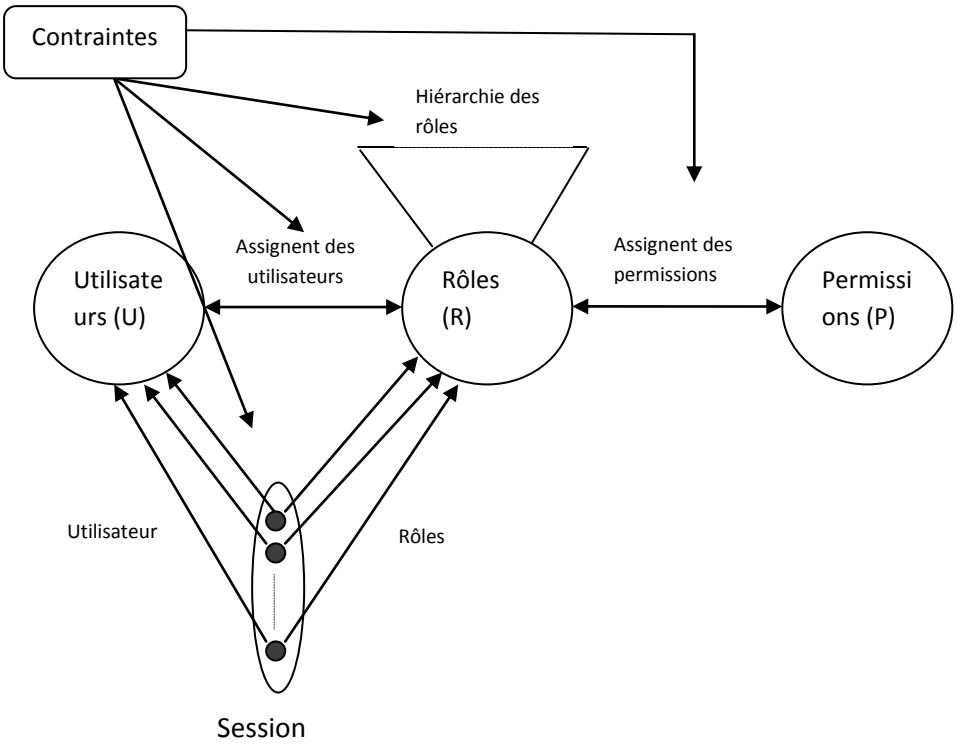

Figure I.6 : Modèle hiérarchique RBAC avec contraintes [9]

A partir du principe de base de RBAC naquirent plusieurs autres méthodes (Time RBAC, et GeneralizedTRBAC [10], WorkflowRBAC [11], AccessControlXmlRoles [12]).

5.2.2. XACML (eXtensible Access Control Markup Language)

XACML [13,14] est un puissant langage défini par OASIS (Organization for the Advancement of Structured Information Standards), dans le but de standardiser les décisions

de contrôle d'accès aux documents XML. Il spécifie la politique et les décisions d'autorisations, lorsqu'il reçoit une requête il détermine si l'accès à la ressource est autorisé totalement, partiellement ou refusé.

XACML permet la gestion des règles d'accès et assure une politique de sécurité pérenne et cohérente, des autorisations sont exprimées par des règles et des politiques, où les éléments de politique contiennent un ou plusieurs éléments de règle. Des éléments de politique et de règle peuvent être organisés de façon hiérarchique, ils peuvent hériter des éléments qui sont placés plus haut dans la hiérarchie.

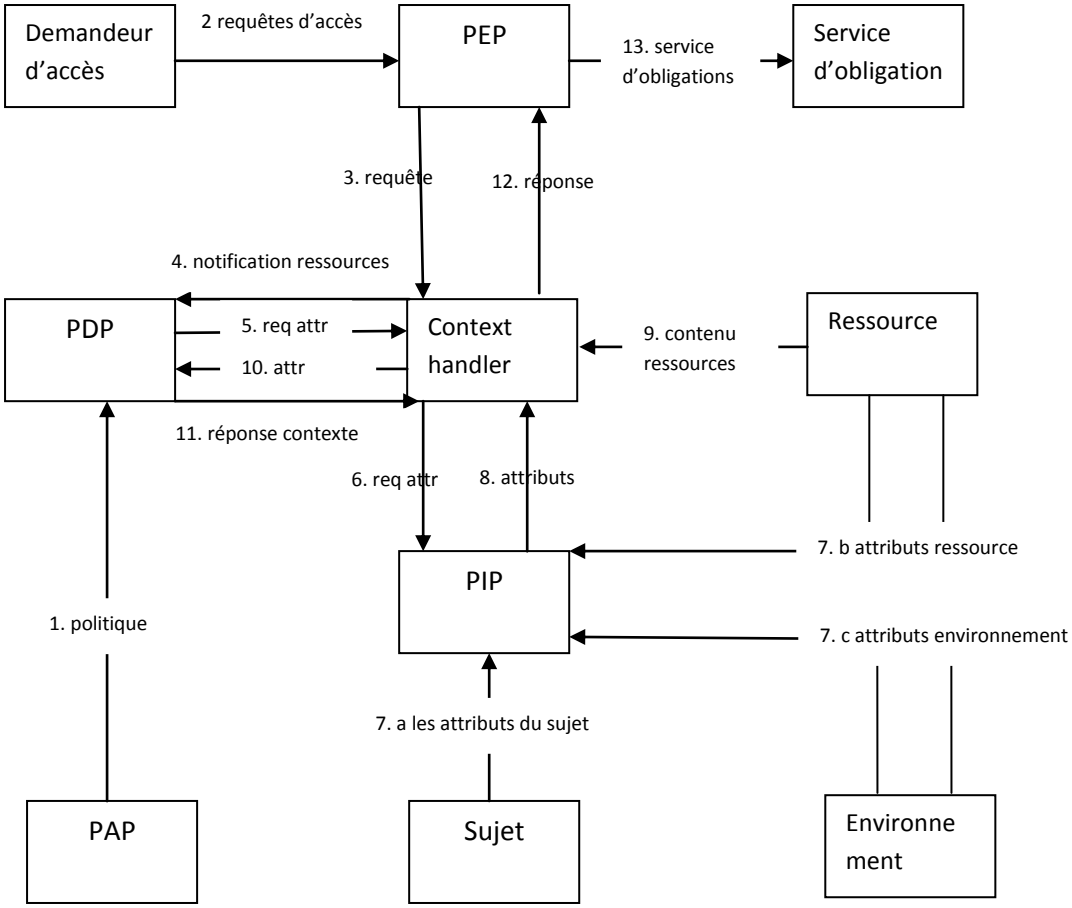

Figure I.7 : Diagramme de fonctionnement XACML [13]

La norme donne une définition de ces concepts qu'on récapitule comme suit :

- ✓ **Le module de Policy Enforcement Point (PEP) :** impose la décision d'accès prise par le point de décision.

- ✓ **Le module de Policy Decision Point (PDP)** : reçoit une demande d'accès et agit l'un sur l'autre avec PAP qui encapsule l'information requise pour identifier les politiques applicables. Alors il évalue la demande contre les politiques applicables et renvoie la décision d'autorisation au module de PEP.
- ✓ **Le module de Policy Administration Point (PAP)** : recherche les politiques applicables à une demande donnée d'accès et les renvoie au module de PDP.
- ✓ **Le module de Policy Information Point (PIP)** : fournit des valeurs d'attributs au sujet, de la ressource, et de l'action (la fonction à exécuter).
- ✓ **Context Handler :** traduit les demandes d'accès dans un format indigène (format quelconque) en format canonique (format de base de XACML).
- ✓ **Environnement** : il fournit un ensemble d'attributs qui sont appropriés à la prise par décision d'autorisation et qui sont indépendants d'un sujet particulier, ressource, et action.

6. Quelques méthodes formelles utilisées dans la sécurité informatique

Durant ces dernières années, la complexité des systèmes informatiques a augmenté considérablement, ce qui rend la détection des problèmes dans ces systèmes de plus en plus couteuse et difficile, devant ces faits, l'utilisation des méthodes formelles est incontournable pour la spécification et l'analyse des systèmes informatiques dans le but d'augmenter le niveau de sécurité et de fiabilité. On regroupe sous l'appellation de méthodes formelles les mathématiques appliquées à la modélisation et à l'analyse des systèmes d'information. L'idée associée à l'utilisation des méthodes formelles est de modéliser le système que l'on veut étudier. Le niveau d'abstraction adopté pour modéliser le système doit dépendre des propriétés qu'on veut étudier.

Plusieurs méthodes formelles ont été adaptées ou développées dans le but de décrire mathématiquement, sans ambigüité des systèmes informatiques.

6.1. Algèbres de processus

Les algèbres de processus sont des langages formels permettant de modéliser le fonctionnement des processus complexes en termes d'actions qu'ils peuvent réaliser. Une algèbre est un langage ayant des règles syntaxiques et sémantiques.

La première algèbre de processus fut développée par Milner en 1980 et fut publiée dans [15], sous l'appellation de Calculus of communicating systems, soit CCS.

6.1.1. Etude cas (Algèbre BPA)

L'ensemble des algèbres de processus est très vaste, notre choix est porté sur l'algèbre BPA (Basic Process Algebra) car nous aurons besoin de cette algèbre durant notre travail. Elle a été introduite dans [16,17], ici on définit sa syntaxe et sa sémantique :

6.1.1.1. Syntaxe

Soit « a » une action prise dans un ensemble fini d'action $\sum$, la syntaxe de BPA est défini par la grammaire BNF suivante :

P ::=	a	(action élémentaire)
	P1.P2	(composition séquentielle)
	P1+P2	(composition alternative)

Tableau. I.1 : Syntaxe de BPA

La première version de BPA contient deux operateurs de base qui permettent de composer des processus élémentaire pour obtenir des processus complexes.

L'operateur « . » représente la composition séquentielle, P.Q est un processus qui exécute dans un premier temps P, et après la terminaison de celui-ci il exécute Q ;

L'operateur « + » représente la composition alternative, P+Q est un processus qui exécute soit le processus P soit le processus Q.

6.1.1.2. Sémantique de BPA

Nous distinguons deux types de sémantiques ; la sémantique axiomatique qui sert comme un système de réécriture des processus, et la sémantique opérationnelle qui sert à décrire l'évolution des différents processus.

A. Sémantique axiomatique

Le tableau suivant représente les différents axiomes de l'algèbre BPA :

P+Q = Q+P	A1
(P+Q)+R = P+ (Q+R)	A2
P+P = P	A3
(P+Q).R = P.R + Q.R	A4
(P.Q).R = P. (Q.R)	A5

Tableau I.2: sémantique axiomatique de BPA

- ✓ A1 exprime la commutativité de l'operateur « + » ; le fait de choisir entre P et Q ou bien entre Q et P signifie la même chose.
- ✓ A2 exprime l'associativité de l'operateur « + » ; le fait de choisir entre P et Q ensuite entre le résultat et R, ou bien entre Q et R ensuite entre P est le résultat signifie la même chose.
- ✓ A3 exprime l'idempotence de la somme ; le fait de choisir entre P et P revient à choisir P.
- ✓ A4 exprime la distributivité à droite de l'operateur « . » par rapport à l'operateur «+ » ; Les deux processus (P+Q).R et P.R et P.R+Q.R représentent un choix entre P et Q, suivi par le processus R.
- ✓ A5 exprime l'associativité de l'operateur « . » ; Les deux processus (P.Q).R et P.(Q.R) représentent le sous processus P suivi du sous processus Q suivi du sous processus R.

B. Sémantiques opérationnelle

Soit le processus 0 représentant le processus nul (un processus qui ne peut pas évoluer), la sémantique opérationnelle est donnée par le tableau suivant :

(Act) $\dfrac{\boxdot}{a \xrightarrow{a} 0}$	$(R_{.0})$ $\dfrac{P1 \xrightarrow{a} 0}{P1.P2 \xrightarrow{a} P2}$
(R_{+l}) $\dfrac{P1 \xrightarrow{a} P'}{P1 + P2 \xrightarrow{a} P'}$	$(R_{.})$ $\dfrac{P1 \xrightarrow{a} P'}{P1.P2 \xrightarrow{a} P'.P2}$
(R_{+r}) $\dfrac{P2 \xrightarrow{a} P'}{P1 + P2 \xrightarrow{a} P'}$	

Tableau I.3: Sémantique opérationnelle de BPA

- ✓ La règle (Act) indique qu'un processus formé d'une action atomique (a) peut évoluer en exécutant cette action et se terminer ;
- ✓ L règle (R_{+l}) veut dire que si nous avons un processus $P1 + P2$ et si le sous processus $P1$ peut évoluer en exécutant une action (a) et devenir le sous processus P', alors $P1 + P2$peut exécuter la même action et devenir P' ;
- ✓ La règle (R_{+r}) veut dire que si nous avons un processus $P1 + P2$et si le sous processus $P2$ peut évoluer en exécutant une action (a) et devenir un sous processus P', alors $P1 + P2$peut exécuter la même action et devenir P' ;
- ✓ La règle $(R_{.0})$ indique que si nous avons un processus $P1.P2$ et si le processus $P1$ peut évoluer en exécutant une action et se terminer, alors le processus $P1.P2$ peut évoluer en exécutant la même action et devenir $P2$;
- ✓ La règle $R_{.}$ signifie que la composition séquentielle de deux processus $P1$ et $P2$ ne peut évoluer que si le processus $P1$ évolue.

Il y a beaucoup de travaux qui sont faits avec les algèbres de processus pour modéliser les systèmes informatiques, et générer des solutions automatiques pour les différents problèmes, comme ceci est les cas dans [18] ou encore dans [19] que nous allons détailler plus tard.

6.2. Automates à états fini

Un automate à états finis M sur un alphabet A est un quintuple $M = (X, Q, q_0, F, \delta)$:

X : L'alphabet d'entrée de l'automate M (cet alphabet est non vide).

Q : Un ensemble fini non vide appelé ensemble d'états de l'automate M.

q_0 : Un élément de l'ensemble Q appelé état initial.

F : Un sous-ensemble de Q appelé ensemble des états finaux ou terminaux.

Transition δ: $Q \times A \rightarrow Q$

$$(p, c) \rightarrow q = \delta(p, c)$$

q = δ (p, c) veut dire qu'en lisant le caractère c, l'automate passe alors de l'état p à l'état q.

6.2.1. Représentation d'automate

6.2.1.1. Graphiquement

L'automate est représenté par un graphe où :

- ✓ Un état se représente par un cercle ;
- ✓ Un état initial se représente par un cercle avec une flèche entrante provenant de l'extérieur ;
- ✓ Un état terminal se représente par un double cerclage.

Exemple

Soient l'alphabet X ={a,b,c} et l'ensembles des états Q ={q_0, q_1, q_2, q_3, q_4}

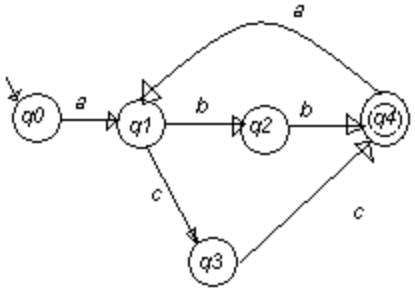

Figure I.8 : Exemple d'un AEF

L'état initial est q_0, et il n y a qu'un seul état final, c'est q_4.

6.2.1.2. Table (Matrice de transitions)

L'automate est représenté par un tableau où :

- ✓ La première ligne contient les lettres de l'alphabet ;
- ✓ La première colonne contient les différents états de l'automate ;
- ✓ Les cases contiennent le résultat de l'application de la fonction de transition.

Exemple

Le graphe de l'exemple précédent peut être représenté avec la matrice suivante, ou q_4 représente l'état final :

	a	b	C
→q_0	q_1	/	q_3
q_1	/	q_2	/
q_2	/	q_4	/
q_3	/	/	q_4
q_4	q_1	/	/

Tableau I.4 : Exemple d'un AEF

Les automates sont un mécanisme qui s'adapte très facilement aux notions informatiques, ce qui facilite énormément la modélisation avec ce formalisme. Un exemple de l'utilisation des AEF en sécurité informatique est le travail des auteurs de [20] qui sera détaillé dans le chapitre suivant.

6.3. La théorie des jeux

La théorie des jeux fut inventée par Von Neuman et Morgenstern en 1944 [21], l'application de la théorie des jeux était essentiellement centrée sur l'économie, mais récemment elle est de plus en plus appliquée au domaine de la sécurité informatique pour l'aide à la prise de décision.

C’est une discipline mathématique qui étudie les situations où le sort de chaque joueur dépend non seulement ses propres décisions mais aussi des décisions prises par d’autres décideurs.

6.3.1. Le jeu

C’est toute interaction entre plusieurs joueurs ayant des intérêts partiellement ou totalement opposés, où chacun est en possession d’un ensemble d’actions parmi les lesquelles il fait son choix et dans un cadre défini à l’avance (les règles du jeu), qui permet de déterminer qui peut faire quoi et quand, une fois que les joueurs font un choix ils reçoivent un gain [22].

6.3.2. Définition formelle

Un jeu sous forme normale peut être représenté sous la forme suivante [23]:

$JN =< M, \{X_i\}_{i\in N,} \{f_i\}_{i\in N}$ tel que :

$M = \{1,2 \dots, n\}\ n \in N, n \geq 2$ est l’ensemble des protagonistes appelés joueurs, un joueur quelconque est désigné par l’indice i ; $i \in M$.

$X_i \subset |R^{ni}, i \in N$ X_i : désigne l’ensemble des stratégies du i^{ème} joueur. Si cet ensemble est infini on dit que le jeu est infini.

$f_i: X = X_1 \times \dots \times X_N \rightarrow |R,\ \forall\ i \in N$, la fonction gain du i^{ème} joueur.

Un jeu est dit à information parfaite si chaque joueur au moment de choisir sa stratégie a une connaissance parfaite de l’ensemble des décisions prise antérieurement par les autres joueurs, sinon il est dit à information imparfaite.

Un jeu est dit à information complète si chaque joueur connait lors de la prise de décision : l’ensemble des joueurs, l’ensemble de ses propres stratégies, l’ensemble des stratégies des autres joueurs ainsi que l’ensemble de leurs fonctions objectifs.

Les problèmes reliés à la sécurité informatique sont similaires à la modélisation formelle des jeux. En effet il y a un attaquant (un nœud malicieux), est un défenseur (le nœud sain attaqué par exemple), l’attaquant peut gagner des privilèges ou des informations satisfaisant ses objectifs, quant au défenseur il encoure des dommages comme la perte

d’information. Ce qui nous donne un jeu à deux joueurs, infini, à informations imparfaites et incomplètes. L’application de cette méthode est très bien illustrée par Shmatikov et al dans [24] pour modéliser des attaques de types Denis de services.

7. Conclusion

Ce chapitre a été consacré aux généralités sur la sécurité informatique, nous avons d’abord défini les termes les plus importants relatifs au domaine, puis nous avons présenté les différentes menaces et les outils qui peuvent être utilisés pour contrer ces menaces. Ensuite, nous avons étudié quelques méthodes formelles utilisées dans la sécurité informatique, ces méthodes sont particulièrement intéressantes pour la spécification et l’analyse des systèmes informatiques complexes.

Dans le second chapitre nous verrons quels sont les méthodes adaptées pour la sécurité des codes mobiles et quelles sont les nouvelles méthodes créées. Et nous effectuerons une étude comparative entre les différentes techniques.

CHAPITRE II

Etude des approches de la sécurité dans les codes mobiles

1. Introduction

Récemment, la recherche en systèmes répartis a vu l'émergence d'une nouvelle technologie qui est celle des codes mobiles. On entend par code mobile un ensemble d'instructions écrites dans un langage de programmation qui s'exécute sur une ou plusieurs machines hôtes aux quelles il est lié temporairement (le temps de l'exécution), il va de la simple applet à des agents logiciels intelligents.

Les avantages principaux des codes mobiles sont : l'optimisation de la bande passante et la flexibilité. Comme d'habitude, une plus grande flexibilité est livrée avec un coût qui est une plus grande vulnérabilité aux intrusions, ce qui a amené les chercheurs à travailler sur le problème de la sécurité des codes mobiles.

2. Exemples de codes malicieux

Un code malicieux est un programme qui a comme but de nuire, cette section donne quelques exemples des codes malicieux :

Virus : Un virus est un logiciel capable de s'installer sur un ordinateur à l'insu de son utilisateur. Le terme « virus » est réservé aux logiciels qui se comportent avec un but malveillant. Créé en 1982, il s'incruste dans un hôte (généralement, un fichier exécutable). Lorsque l'hôte est lancé, le virus commence son travail.

Chevaux de Troie : Un cheval de Troie est un logiciel qui se présente sous un jour honnête, utile, ou agréable, et qui une fois installé sur un ordinateur y effectue des actions cachées et pernicieuses.

Bombe logique : une bombe logique est un code malicieux ajouté à une application et qui sera provoqué par une occurrence spécifique, comme une condition logique, une heure spécifique, ou un autre événement prévisible.

Les compte-gouttes : c'est un programme utilisé pour installer des virus sur des ordinateurs. Dans certain cas, le compte-gouttes n'est pas infecté par un autre code malicieux, et c'est pourquoi, il n'est pas détecté par les programmes scanneurs de virus. Un compte-gouttes peut aussi se connecter à Internet et télécharger des mises à jours pour programmes virusés qui résident dans les systèmes infectés.

Enregistreur de frappe : c'est un code malicieux utilisé pour enregistrer ce qui est tapé sur le clavier, il peut par exemple enregistrer les mots de passe des différents utilisateurs.

Vers: Un ver est une variété de virus qui se propage par le réseau. En fait, alors qu'il y a cinq ou six ans les virus n'étaient pas des vers (ils ne se propageaient pas par le réseau) et les vers n'étaient pas des virus (ils ne se reproduisaient pas), aujourd'hui la confusion entre les deux catégories est presque totale.

3. Taxonomie des attaques

Axel Bürkle et all ont considéré trois types d'attaques dans [25] :

- ✓ La protection des hôtes qui exécutent un code mobile malicieux ;
- ✓ La protection d'un code mobile qui s'exécute sur un hôte malicieux ;
- ✓ La protection d'un hôte attaqué par un hôte malicieux (ce cas ne concerne pas notre thème qui est la sécurité dans les programmes).

4. La protection des codes exécutés sur un hôte malicieux

Ce problème est relativement difficile car l'environnement possède un contrôle total sur le code mobile, autrement dit, la protection par l'hôte n'est pas possible. Les attaques possibles rassemblées par les auteurs de [25] sont :

Accéder aux données du code : comme la lecture de données confidentielles comme les clefs ou la modification des données collectées en cour de route pour les comparateurs de prix par exemple.

Accéder au code de l'agent : la lecture ou la modification des chemins ou des algorithmes de routage de l'agent ce qui modifiera le comportement de l'agent au bénéfice de la plateforme malicieuse ou pour nuire à d'autres plateformes.

On peut distinguer essentiellement deux axes pour contrer ces attaques:

4.1. Protection des données

La protection des données est intéressante du fait que les agents mobiles peuvent rencontrés sur leur chemin des hôtes qui vont essayer de leur faire un lavage de cerveau.

Intégrité des données [26, 27, 28] : ce sont des méthodes qui s'assurent que les données récoltées par les code mobiles sont correctes, on peut imaginer le cas d'un comparateur de prix par exemple.

Confidentialité des données [29] : il s'agit d'une méthode qui utilise le mécanisme de la cryptographie pour garantir la confidentialité des informations récoltées ou transmises par le code mobile.

4.2. Protection de l'exécution du code

On doit toujours s'assurer que le code fait ce qu'il est sensé faire, ceci veut dire qu'il s'exécute normalement. Il existe aussi des applications qui doivent demeurer secrète, donc les hôtes doivent ne pas connaitre la fonction qu'ils exécutent.

Intégrité de l'exécution [30, 31,32] : il est évident que le code mobile doit faire ce qu'il est sensé faire donc son intégrité doit être vérifiée sinon les résultats obtenus seront faussés, les

auteurs de [30] propose une méthode qui s'assure de l'intégrité du chemin suivi par le code mobile.

Confidentialité de l'exécution [33]: il est parfois nécessaire d'assurer la sécurité en gardant secrètes les méthodes utilisées, donc l'exécution d'un programme doit rester confidentielle.

5. Protection de l'hôte d'un code mobile malicieux

Notre travail se base essentiellement sur ce problème qui a été beaucoup étudié ces dernières années, Les codes mobiles générés par des hôtes malicieux (intrus) peuvent attaquer l'environnement ou ils s'exécutent par les attaques suivantes :

Denis de service: le code utilise les ressources de l'hôte d'une manière excessive comme sa mémoire, CPU ou la bande passante, ceci dans le but de rendre ses services inaccessibles.

L'accès non autorisé aux données de l'hôte : le code essaye d'accéder aux données confidentielles.

Mascarade : l'agent se comporte comme un autre agent qui possède plus de droits d'accès.

Les attaques complexes : ou il y'a coopération avec d'autres agents, comme ceci est illustré par les auteurs de [23].

Il est important de noter que les solutions basées sur des mécanismes d'authentification sont insuffisants, car dès lors que la confiance n'est basée que sur l'origine du code et non pas sa sémantique, on ne peut prédire le comportement du code. Par conséquent la sécurité des ressources doit reposer sur des mécanismes d'auto-défense.

Les techniques de protections des hôtes ont évolué en en deux directions :

- ✓ Renforcer les mécanismes d'authentification et de contrôle d'accès et les adapter aux codes mobiles.
- ✓ Analyse de la sémantique du code mobile ; Consiste à effectuer des vérifications sur la structure du code ou son comportement.

5.1. Renforcer les mécanismes d'authentification et de contrôle

5.1.1. Confinement (sandboxing)

La traduction littérale du terme sandbox en français donne le terme « bac à sable », c'est une métaphore qui fait référence à un environnement restreint ou confinement.

Le confinement est une technique logicielle utilisée pour protéger les hôtes qui exécutent des codes mobiles malicieux. Dans un environnement d'exécution, le code local est exécuté avec toutes les permissions et il a accès à toutes les ressources, quant aux codes mobiles, ils sont exécutés dans un environnement restreint appelé sandbox. Les restrictions concernent certaines opérations comme l'ouverture des fichiers systèmes, ouverture de connexion réseau ou invocations de certains programmes. Ce mécanisme assure le respect d'une certaine politique de sécurité, la politique doit spécifier les règles et les restrictions que le code mobile doit respecter. Ce principe est illustré par la figure suivante :

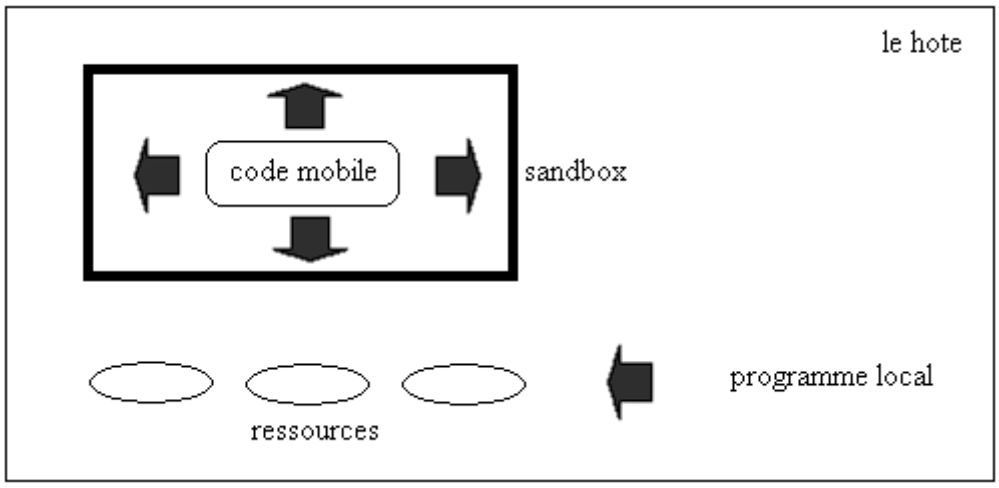

Figure II.1: principe de fonctionnement du confinement

Cette technique est utilisée dans les premières versions de la JDK. [34, 35, 36], Cependant elle a récemment été adaptée dans [37] ou les auteurs ont implémenté un environnement virtuel pour protéger à la fois la plate forme et le code mobile, cet environnement utilise à la fois le principe du confinement et la cryptographie.

L'inconvénient majeur de cette méthode est que les programmes qui peuvent s'exécuter dans des environnements aussi restreint sont rarement utiles. Donc il faudrait trouver un équilibre entre les restrictions et l'utilité des programmes à exécuter.

5.1.2. Contrôle d'accès

C'est comme le confinement avec une granularité plus précise, en effet chaque niveau d'accès peut être comparé à du sandboxing avec des restrictions différentes, ce principe est illustré par la figure suivante :

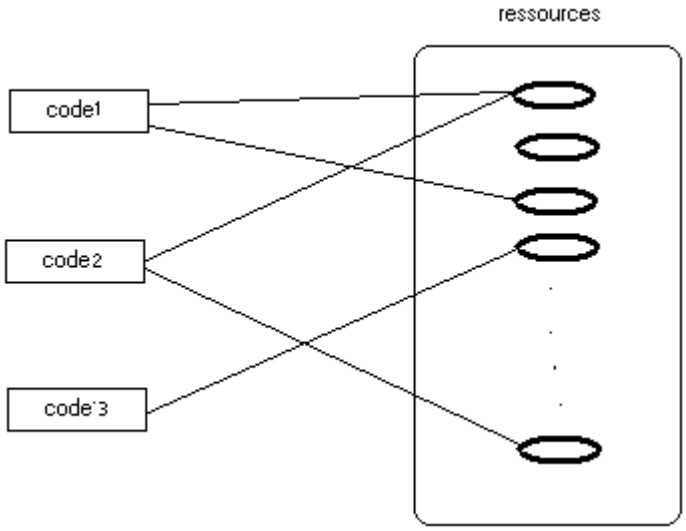

Figure II.2: Principe de fonctionnement du contrôle d'accès

Le contrôle d'accès est meilleur que la signature du code et le sandboxing, cependant son inconvénient majeur est sa gourmandise en temps d'exécution.

5.1.3. Signature numérique

La signature du code est le processus par lequel le code est signé digitalement par le producteur du code pour assurer l'authenticité et l'intégrité du code pour le consommateur.

Ce modèle est initialement introduit par Microsoft avec le frame work Active-X [38], puis par JDK avec la notion d'applets signés (si l'applet n'a pas de signature le code sera exécuté dans une sandbox comme vu dans la première version JDK)

Cet axe a été amélioré ultérieurement par les chercheurs, ceci est illustré par le travail fait par les auteurs de [39] ou ils proposent une plateforme ou l'idée principale est que la

signature numérique ne doit pas prouver uniquement l'origine du code ou son intégrité, mais aussi sa conformité avec le contrat qu'ils négocient avant.

Le grand inconvénient de cette approche est l'aspect rudimentaire du contrôle d'accès (si le code est signé il aura tous les droits, s'il ne l'est pas, il n'aura rien comme droit).

5.2. Analyse du code mobile

L'analyse du code mobile consiste à effectuer des vérifications sur la structure du code ou son comportement et de modifier son état en conséquence ; cette vérification se fait par rapport à une politique de sécurité donnée.

On distingue trois types d'analyse qui sont : Analyse statique, l'analyse dynamique et l'analyse hybride (les méthodes de réécriture) :

5.2.1. Analyse statique

L'analyse statique consiste à déduire mécaniquement des propriétés sur le comportement d'un programme sans l'exécuter, la méthode la plus connue qui illustre ce type d'analyse est la méthode PCC.

5.2.1.1. PCC (Proof-Carrying Code)

PCC [40,41] est une méthode formelle très intéressante ou l'idée principale est que le producteur du code doit démontrer au consommateur que ce code vérifie une politique de sureté négociée auparavant, ceci se fait en utilisant la logique du premier ordre. En d'autres termes, le principe de cette méthode se base sur la formule suivante :

$C \in sain \ \ ssi \ \ \exists P. A \vdash P: \{Pré\} C \{Post\}$ tel que :

- ✓ C : le code mobile à exécuter ;
- ✓ $sain$: l'ensemble des codes sains ;
- ✓ P : la preuve ;
- ✓ A : les axiomes ;
- ✓ $Pré$: les conditions qui doivent être vérifiées avant l'exécution du code ;
- ✓ $Post$: les conditions qui doivent être vérifiées après l'exécution du code.

Le principe de cette méthode est illustré par la figure suivante :

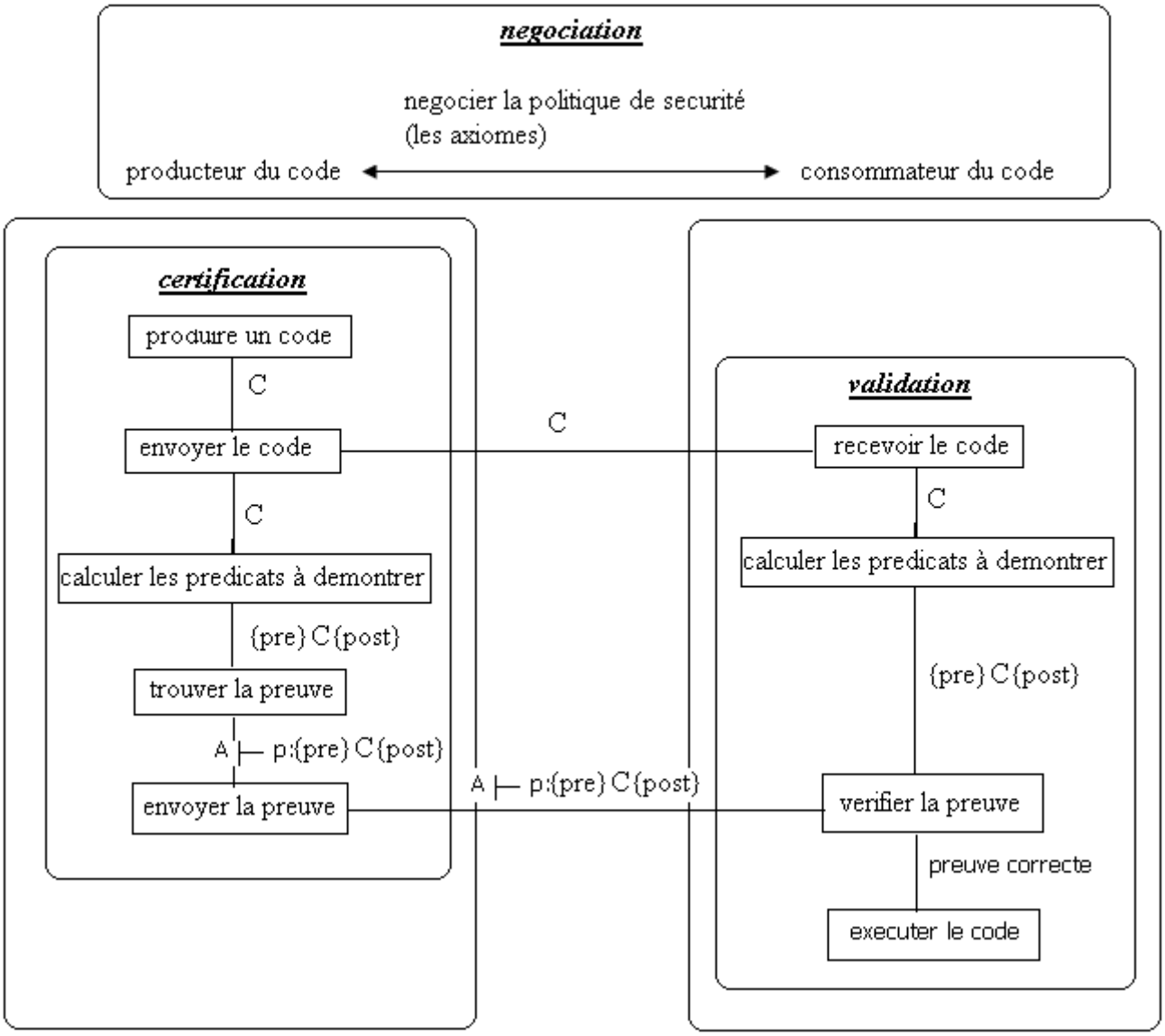

Figure II.3: principe de fonctionnement de PCC

Nous pouvons remarquer que PCC est partagé en trois grandes étapes :

La négociation : le producteur et le consommateur fixent la politique de sûreté. (L'ensemble des axiomes qu'on utilisera dans la seconde étape).

La certification : à cette étape, le producteur compile le programme (calcule des prédicats à démontrer) et génère une preuve que le programme respecte la politique de sûreté négocié à l'étape précédente (démontrer les prédicats en utilisant les axiomes).

La validation : le consommateur vérifie la preuve (recalcule les prédicats, vérifie que ce sont les mêmes et vérifie la preuve).

Vu l'importance de cette technique, beaucoup d'autres méthodes se basant sur elle, ont étés proposées, par exemple Appel et Felty [42] ont utilisé une logique d'un haut dégrée, Bernard et Lee ont utilisé la logique temporelle [43].

En plus de la difficulté de définir la politique de sécurité, l'autre inconvénient de PCC c'est que ce n'est pas toujours évident de démontrer les prédicats ; dans le but de répondre à cette lacune, il y a eu la naissance de iPCC.

5.2.1.2. iPCC

interactive PCC[45] est l'une des nombreuses méthodes dérivées de PCC, c'est une méthode probabiliste, en effet les auteurs démontrent qu'on peut écrire les prédicats sous forme de conjonctions, puis le producteur du code démontre les prédicats conjonction par conjonction, et le consommateur du code vérifie les sous preuves au fur et à mesure, comme ceci est illustré par la figure II.4.

Dans [47] et [48], les auteurs ont proposé une approche qui se base sur la même idée que PCC, il s'agit d'une technique pour produire un code certifié, dans ce cas le certificat n'est pas une preuve mais plutôt un type annoté qui renferme une approximation statique du comportement dynamique du programme, pour ce faire les auteurs utilisent le Langage assembleur annoté TAL (Typed Assembly Language)

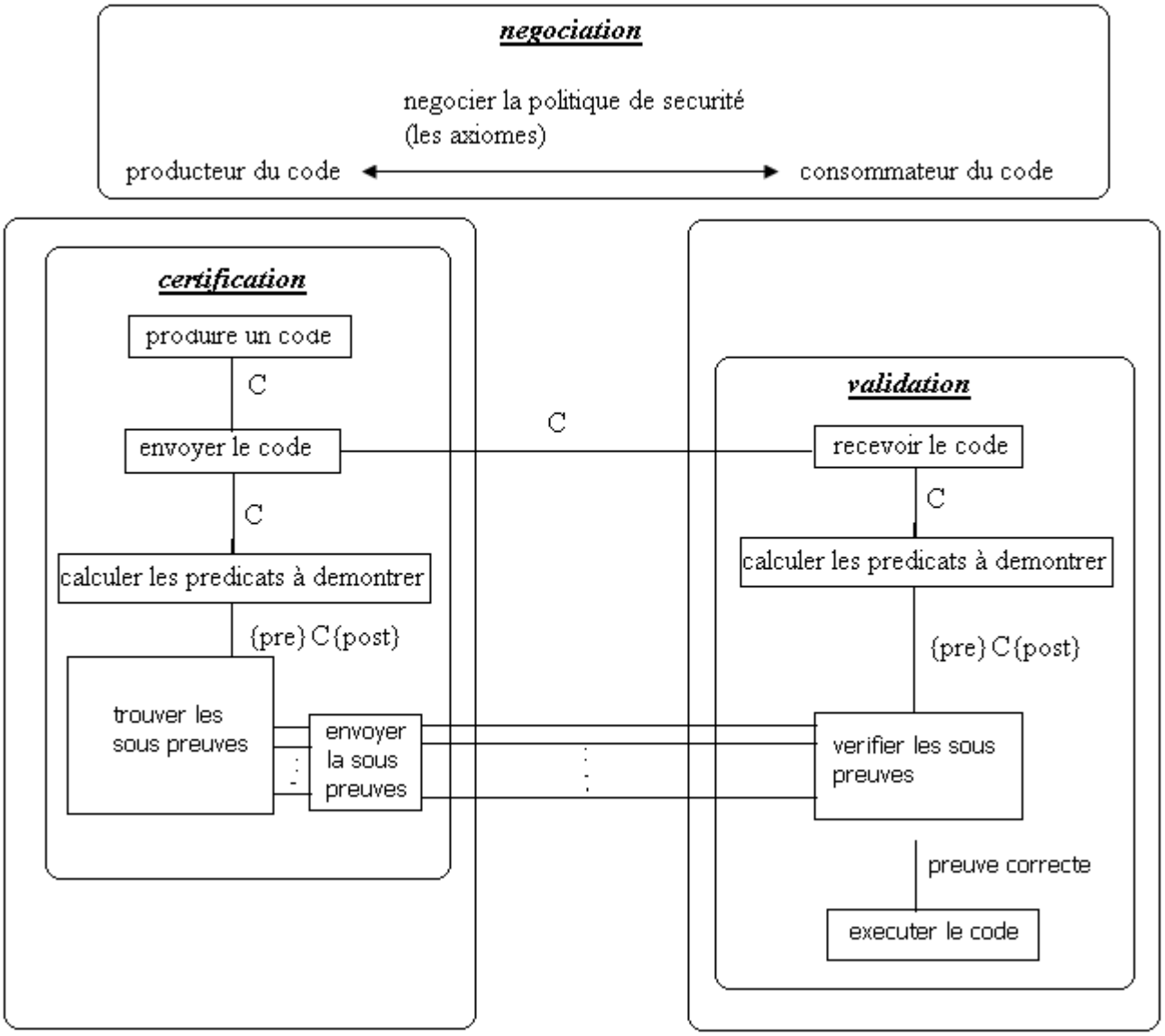

Figure II.4 : principe de fonctionnement de iPCC

5.2.2. Analyse dynamique

L'analyse dynamique consiste à laisser les programmes s'exécuter et surveiller leur comportement à l'aide des moniteurs de surveillance (monitoring), contrairement à l'analyse statique le moniteur peut avoir accès aux valeurs réelles des variables d'un programme.

Le moniteur peut fournir des informations pour l'utilisateur au fur et à mesure que le programme s'exécute, soit effectuer des analyses sur le comportement et fournir des

informations après l'exécution du programme en question. Dans le premier cas le moniteur peut être automatique ou interactif.

Il existe trois facteurs clés dans tous les moniteurs :

- ✓ **Volume des informations :** une grande quantité des informations est disponible pour un moniteur, nous devons faire un choix judicieux à quel type d'informations, nous nous intéressons. Sinon, il existe un grand risque de consommer beaucoup de ressources, ce qui peut nuire à l'exécution du programme surveillé. Les types d'information à surveiller dépendent de la politique de sécurité.
- ✓ **Intrusion :** un moniteur influence l'exécution du programme cible, même par le partage des ressources uniquement. Ce changement s'appelle intrusion, il en existe deux catégories :

- **Passive :** interception de tous les appels de fonctions systèmes par le moniteur du programme cible.
- **Active :** l'environnement du programme cible est instrumenté pour faciliter la surveillance. Deux cas sont possibles :
    a. Le code source peut être modifié manuellement ou automatiquement de sorte qu'il génère les informations désirées.
    b. Une machine virtuelle peut être utilisée pour interpréter le langage de programmation et générer les informations nécessaires.

- ✓ **Accès :** un moniteur doit avoir accès à certains types d'informations déterminées dynamiquement à l'exécution du code cible. Ceci est difficile dans un moniteur actif puisque le partage d'un espace mémoire est généralement interdit par les systèmes d'exploitation. De plus, ceci peut causer une perte de performances additionnelles. En effet, le moniteur peut changer des données stockées en mémoire et par conséquent peut modifier les résultats attendus.

Ces trois problèmes existent dans les trois différentes approches surveillance dynamique des codes :

- Interception des différentes fonctions au niveau du noyau ;
- Encapsulation des différents systèmes APIs ;
- Insertion des tests dynamiques : instrumentation.

5.2.2.1. Interception des différentes fonctions au niveau du noyau

Le principe de fonctionnement de cette technique est le suivant : à chaque fois que le programme veut exécuter une action qui sollicite le noyau. Le moniteur intercepte l'appel. Il vérifie si l'exécution de l'action ne viole pas une propriété de sécurité, si c'est le cas alors il acheminera l'appel pour le noyau. Sinon, il arrêtera le programme. Ce principe est illustré par la figure

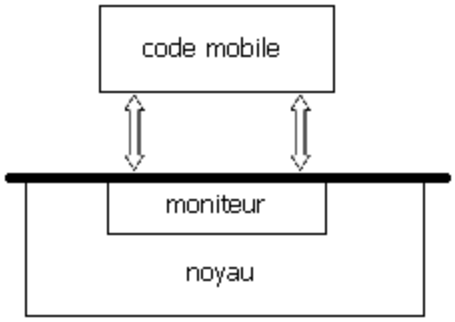

Figure II.5 : Interception au niveau du noyau

L'avantage majeur de cette technique est sa fiabilité car à chaque fois que le noyau est concerné il y a une vérification, cependant à cause de la même raison le programme s'alourdit considérablement, et c'est son plus grand inconvénient.

5.2.2.2. Encapsulation des différents systèmes

L'architecture de la technique d'encapsulation. Il s'agit de modifier les fonctions systèmes jugées critiques (relativement à la politique de sécurité). En effet, une autre version de chaque fonction système est implémentée. Dans la nouvelle version des fonctions en question, on inclut les tests nécessaires qui permettent de vérifier qu'on ne viole pas la politique de sécurité. C'est l'origine du terme encapsulation. Intuitivement, on encapsule la version originale de chaque fonction à l'intérieur d'une autre fonction comme ceci est illustré par la figure II.6. Cette dernière inclut une série de tests. Il va sans dire que les tests diffèrent d'une fonction à une autre dépendamment de la politique de sécurité. L'or d'un appel a une fonction système, si les tests réussissent, on achemine l'appel vers la version originale de la fonction en question, sinon le programme est arrêté. Un exemple de cette approche est présenté dans [49] :

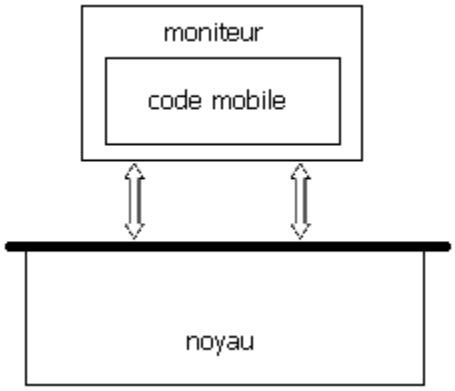

Figure II.6 : Principe de l'encapsulation

Le problème avec cette technique c'est la difficulté de programmer une autre fonction pour chaque fonction système, et de trouver les tests adéquats pour chacune d'elles.

5.2.2.3. Instrumentation

La technique de surveillance dynamique consiste à ajouter du code dans un programme. Le code ajouté est sous forme de tests qui assure le respect de la politique de sureté. La figure présente l'architecture générale de cette technique, cette fois, le moniteur est inclus dans le programme. Dernièrement beaucoup de travaux de recherche ont été réalisé en rapport à cette technique, toutes fois la majorité de ces travaux sont des techniques ad hoc. Un exemple de cette technique est SASI « Security automata SFI Implementation » [50].

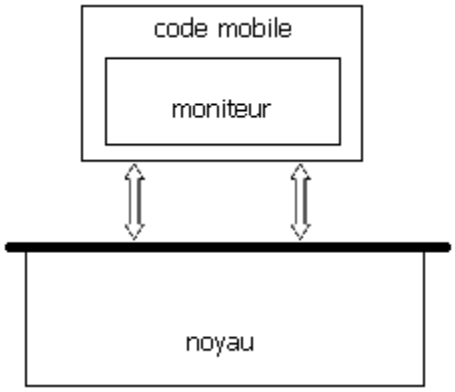

Figure II.7 : Instrumentation

L'analyse statique se limite à la vérification des propriétés de sécurité de base (mémoire et flot de contrôle). Pour vérifier des propriétés de sécurité de haut niveau, nous avons souvent besoin des valeurs des variables. C'est pourquoi il faut utiliser les techniques d'analyse dynamique. Par contre ces dernières causent souvent une perte de performance

considérable. Ces pertes sont principalement causées par les tests que nous ajoutons à l'intérieur du code.

5.2.3. Les méthodes hybrides (de réécriture des programmes)

Les méthodes hybrides sont très intéressantes du fait qu'elles essayent de joindre, à la fois, les avantages des méthodes statiques et dynamiques.

On les appelle les méthodes de réécritures car on prend le programme initial et le transforme pour obtenir un autre programme ou il y a des conditions supplémentaires à des endroits critiques, pour pouvoir utiliser les valeurs des variables lors de l'exécution.

5.2.3.1. Réécriture des programmes avec les Automates à états finis

Un AEF est défini par un quintuple $(Q, \Sigma, \delta, i, F)$ qui représentent respectivement : l'ensemble des états, l'alphabet en entrée, la fonction de transition, l'état initial et l'ensemble des états finaux.

La fonction δ est définie comme suit : $\delta: Q \times \Sigma \rightarrow \mathrm{P(Q)}$

$$(\mathrm{q}, \mathrm{a}) \rightarrow \{\mathrm{q}'\}$$

Le principe de cette méthode est qu'à partir d'une propriété de sécurité ϕ et d'un programme Pg, on réussisse à obtenir un autre programme vérifiant cette propriété.

Les auteurs de [20] représentent respectivement le programme et la propriété de sécurité par des AEF comme suit : $A_{Pg}(Q_1, \Sigma, \delta_1, i_1, F_1)$ et $A_{\phi}(Q_2, \Sigma, \delta_2, i_2, F_2)$.

Puis ils ont définit l'operateur d'injection qui permet de calculer le nouvel automate qui respecte la propriété de sécurité, cet operateur est défini comme suit :

$$A_{Pg,\phi} = A_{Pg} \odot_{end}^{enf} A_{\phi}$$

Tel que : $A_{Pg,\phi} = (Q_{12}, \Sigma, \delta_{12}, < i_1, i_2 >, F_{12})$

- Avec Q_{12} est le plus petit ensemble qui vérifie les conditions suivantes :
- ✓ $< i_1, i_2 > \in Q_{12}$
- ✓ Si $< s_1, s_2 > \in Q_{12}$ $et\ s'_1 = \delta_1(s_1, a_1)$ et $s'_2 = \delta_2(s_2, a_2)$ $Alors$ $< s'_1, s'_2 > \in Q_{12}$
- δ_{12} est la fonction de transition qui est décrite comme suit :

- ✓ $\delta_{12} : Q_{12} \times \Sigma \rightarrow Q_{12}$
- ✓ $\delta_{12}\left(< s_i, s'_i >, enf(a_i, a'_i)\right) =< s_i, s'_i >$

  Où $s_j = \delta_1(s_i, a_i)$ et $s'_j = \delta_2(s'_i, a'_i)$

- ✓ $enf\big((p_1, a_1), (p_2, a_2)\big) = (h(p_1, a_2) \wedge h(p_2, a_1) \wedge p_1 \wedge p_2, Max(a_1, a_2))$ *avec :*

  p_1, p_2 sont une conjonction de deux prédicats où le premier est défini avec les actions du programme (*exemple : pr_i= ¬read(x), send(x) …)* et le second est défini avec les variables du programme (c_i= *x<=5, x= vrai …)* donc quelque soit $i : P_i = pr_i \wedge c_i$.

  Max est une fonction qui retourne l'action la plus forte, selon un ordre donné entre les actions *(exemple : send(x)> write(x) > lire(x) …)*

  h est une fonction qui produit un nouveau prédicat, en appliquant les arguments du prédicat sur ceux de l'action. Elle donne vrai ou faux selon le fait que l'action satisfasse le prédicat est décidable avec l'analyse statique, sinon elle donne une condition dynamique qui dépend des valeurs des variables *(exemples : h(¬read(x), read(0))= faux, h (¬write(0), write(x))= x <> 0)* .

- F_{12} est l'ensemble des états finaux, qui est défini par la fonction *end* comme suit :

  $end: Q_{12} \times \Sigma \times \delta_{12} \times< i_1, i_2 >\rightarrow F_{12}$

  La fonction end nous permet de décider quels sont les états finaux, cette fonction traverse l'automate obtenu et assigne à l'ensemble des états finaux (F_{12}) :

  1. tous les états qui ont une transition sortante évaluée à *faux* (toutes ces transitions sont supprimées de l'automate résultat) ;
  2. les derniers états des chemins qui n'ont pas d'évaluation à *faux.*

Une fois l'automate respectant la politique de sécurité obtenu, celui-ci sera transformé en un programme, Les auteurs de cet article ont démontré que le programme trouvé est complet et juste.

5.2.3.2. Méthode algébrique

Dans cette méthode, la politique de sécurité ϕ et le programme P sont représentés sous forme de processus, en utilisant une algèbre de processus, les auteurs de [19] ont proposé une approche algébrique permettant de générer un autre programme P_0 qui respecte la politique ϕ qui se comporte comme P (en respectant l'équivalence par trace) et lorsque P essaye d'exécuter une action qui viole la politique de sécurité. L'approche proposée transforme le problème de trouver P_0 à la résolution d'un système d'équations linéaires sous une algèbre précise (BPA). Nous allons détailler d'avantage cette méthode dans l'étude de cas.

6. Etude comparative

Chacune de ces méthodes peut convenir à un domaine particulier, selon ses avantages et ses inconvénients, par exemple les applications temps réels ne peuvent se permettre d'utiliser des méthodes trop complexes, les applications dans les domaines critiques ne peuvent utiliser celles qui sont probabiliste. Cette section nous permet de tirer les avantages et les inconvénients des différentes méthodes.

6.1. Les critères de comparaison

Complexité pour le producteur (le consommateur): peut se définir comme étant le temps, la capacité de calcul nécessaire et la difficulté de réalisation de la tache du producteur du code (respectivement le consommateur du code).

Rigidité : ce critère sert à savoir si une méthode donnée peut modifier un code pour qu'il soit accepté, ou rejette directement les codes qui ne sont pas compatibles avec la politique.

Disponibilité d'outils automatique : porte sur la disponibilité d'outils nécessaires pour l'exécution des différentes taches de la méthode.

Sureté: pour savoir si la méthode est probabiliste ou bien sure à 100%.

Extensibilité : ce critère permet de savoir si la modification de la politique ou des programmes est difficile à prendre en considération par la méthode.

Les codes exécutés : pour connaitre l'importance et l'utilité des codes exécutés avec cette méthode.

Fausses alertes et interventions : pour savoir si la méthode empêche l'exécution des codes non malicieux.

Accès aux valeurs des variables : ce critère détermine si la méthode peut accéder aux valeurs réelles des variables.

Nombre de messages échangés : le nombre de messages échangés entre le producteur du code et le consommateur du code.

6.2. Tableaux de comparaison

6.2.1. Les méthodes d'authentification et de contrôle d'accès

Méthodes / Critères	Sandboxing	Signature numérique	Contrôle d'accès
Complexité pour producteur	Aucune	simple	Simple
Complexité pour consommateur	Simple	simple	complexe
Rigidité	Rigide	souple	Souple
Disponibilité d'outils automatique	Oui	Oui	Oui
Sureté	Oui	Non	Non
Les codes exécutés	Souvent des codes inutiles	Tous les codes	Tous les codes
Fausses alertes et interventions	Oui	Non	Non

Tableau II.1 : comparaison des méthodes d'authentification et de contrôle d'accès

6.2.2. Les méthodes par analyse du code

Méthodes / Critères	PCC	iPCC	Algèbre BPA	Les AEF	Analyse dynamique (monitoring)
Complexité pour producteur	complexe	complexe	Complexe	complexe	Aucune
Complexité pour consommateur	Simple	simple	Complexe	simple	Complexe
Rigidité	rigide	rigide	Souple	souple	Souple
Disponibilité d'outils automatique	Outils semi-automatiques (Coq[52])	Outils semi-automatiques (Coq…)	Indisponibilité d'outils	Indisponibilité	Oui
sureté	Oui, si la politique est sure	probabiliste	Oui, si la politique est sure	Oui, si la politique est sure	Moyen
Les codes exécutés	Les codes compatibles avec la politique	Les codes compatibles avec la politique	Tous les codes	Tous les codes	Tous les codes
Fausses alertes et interventions	Non	non	Non	non	Oui
Accès aux valeurs des variables	Non	Non	Non	Oui	Oui
Nombre de messages échangés	2	N	1	1	1

Tableau II.2 : comparaison des méthodes par analyse du code.

6.3. Discussion

On remarque que les mécanismes d'authentification et de contrôle sont assez simples, coté producteur du code ou coté consommateur, à part la méthode de contrôle d'accès qui est

assez complexe coté consommateur. Comme ces méthodes sont anciennes, il est normal de trouver plusieurs outils implémentés.

L'inconvénient majeur de ces méthodes, est le fait qu'elles basent leur décision d'exécuter un code ou de ne pas le faire sur son origine et pas sur le code lui-même. Car ce n'est pas parce qu'on fait confiance à un hôte qu'il ne peut pas nous envoyer volontairement ou involontairement un code malicieux. De ce fait, nous jugeons que les méthodes basée sur l'analyse du code sont meilleurs que celles basées sur son origine.

La méthode formelle PCC et ses différentes dérivées sont très intéressantes, car c'était la première méthode formelle, donc on était certain que la méthode ne permet que l'exécution des codes respectant une certaine politique. El plus, la vérification de la preuve par le consommateur est assez simple, car il suffit de vérifier qu'on utilise l'un des axiomes dans chaque étape de la démonstration.

La rigidité de cette méthode est l'un de ses rares inconvénients, en effet, si on n'arrive pas à démontrer le moindre prédicat, le code sera rejeté, et la méthode ne permet pas la modification du code pour le rendre compatible avec la politique.

Pour régler ce problème, les auteurs de [19] ont proposé une autre méthode formelle qui avait les qualités de PCC et qui réglait la lacune de la rigidité, en effet, cette méthode permet l'obtention d'un code respectant une politique donnée à partir d'un code qui ne la respecte pas, de ce fait, on peut dire qu'elle permet l'exécution de tous les codes.

Cette souplesse est offerte avec un cout, qui est une plus grande complexité pour le consommateur du code par rapport à l'autre méthode formelle (PCC).

Contrairement à ces méthodes qu'on peut qualifier de préventives, le monitoring est une méthode détective qui est très souple, car elle permet l'exécution de tous les codes, et elle est sensée intervenir quand le code essaye d'exécuter une action illégale.

Le problème de cette méthode est qu'elle possède un grand taux d'erreur, inadmissible dans certains domaines.

En gros, on peut remarquer que les méthodes basées sur l'analyse du code sont meilleures que celles d'authentification et de contrôle, car des lors qu'on base notre décision d'exécuter un code que sur son origine, on ne peut être sur de son intégrité.

7. Conclusion

Ce chapitre nous a permis de faire le tour des différentes méthodes utilisées pour protéger un hôte qui exécute un code malicieux, nous avons remarqué l'inadaptabilité des méthodes d'authentification et de contrôle d'accès à assurer qu'un code est sain.

Tous les critères sur lesquels nous avons basé notre comparaison sont très importants, cependant, nous jugeons qu'il y a deux critères qui sortent du lot : la sureté et l'ensemble des codes exécutés, en effet, la sureté est le but même de cette étude, on ne doit exécuter un programme que s'il est sure, mais ceci ne doit pas nous empêcher de modifier un programme pour pouvoir exécuter au moins quelques unes de ses fonctions.

Pour ces raisons nous pouvons dire que les méthodes de réécriture sont plus intéressantes que les autres, donc nous consacrons le prochain chapitre à l'étude de l'une de ces méthodes : la méthode algébrique.

CHAPITRE III

Étude de cas : la méthode algébrique de réécriture avec l'algèbre BPA

1. Introduction

Comme nous l'avons noté dans le chapitre précédent, les méthodes de réécriture sont les plus intéressantes, car elles nous offrent la possibilité de modifier un programme pour le rendre compatible avec une politique de sécurité donnée, c'est pour cette raison que nous choisissons la méthode de réécriture avec l'algèbre BPA, proposée par MEJRI et all dans [19].

Ce chapitre est subdivisé en trois parties : La première sert à expliquer le fonctionnement de la méthode proposée avec l'algèbre de processus $BPA^*_{0,1}$. La seconde partie sert à montrer qu'avec l'enrichissement de l'algèbre avec des actions conditionnelles, la méthode peut s'appliquer à de vrais programmes. Et la troisième étudie l'applicabilité d'une autre équivalence sur l'approche (équivalence basée sur la bissimulation).

2. Utilisation de l'algèbre $BPA^*_{0,1}$

2.1. Spécification du langage $BPA^*_{0,1}$

Cette algèbre a été introduite initialement dans [16,17] comme c'est montré dans le premier chapitre, les auteurs de [19] l'ont adapté comme suit :

2.1.1. Syntaxe

Soit a une action prise dans un ensemble fini d'action Σ , la syntaxe de $BPA^*_{0,1}$ est défini par la grammaire BNF suivante :

$$P := 0 \mid 1 \mid a \mid P.P' \mid P + P' \mid P * P'$$

a : Une action élémentaire prise dans un ensemble fini d'actions Σ ;

1 : Signifie que le processus a fini toutes les actions qu'il peut exécuter ;

0 : Un processus en inter blocage ;

$P + P'$: Est un choix entre les deux processus P et P' ;

$P.P'$: Signifie l'exécution des deux processus P et P'en séquentiel ;

$P * P'$: Est un processus qui se comporte comme $P.(P * P') + P'$. C'est une version binaire de l'operateur étoile de Kleene [53].

2.1.2. Sémantique de $BPA^*_{0,1}$

La sémantique opérationnelle est définie par la relation de transition $\rightarrow \in P \times \Sigma \times P$ donnée par le tableau :

R^a $\quad \dfrac{\boxdot}{a \xrightarrow{a} 1}$	R^1 $\quad \dfrac{\boxdot}{1 \downarrow}$
R_l $\quad \dfrac{P \downarrow \quad Q \xrightarrow{a} Q'}{P.Q \xrightarrow{a} Q'}$	R_r $\quad \dfrac{P \xrightarrow{a} P'}{P.Q \xrightarrow{a} P'.Q}$
R_l^+ $\quad \dfrac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$	R_r^+ $\quad \dfrac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$
R_l^* $\quad \dfrac{P \xrightarrow{a} P'}{P * Q \xrightarrow{a} P'.(P * Q)}$	R_r^* $\quad \dfrac{P \xrightarrow{a} P'}{P * Q \xrightarrow{a} P'.(P * Q)}$

Tableau III.1 : Sémantique opérationnelle de $BPA^*_{0,1}$

2.2. Définitions et Propriétés

Définition 1

Pour introduire l'équivalence basée sur la trace, on définit l'operateur $\twoheadrightarrow$ comme suit :

$$\frac{\boxdot}{p \twoheadrightarrow^{\varepsilon} p}$$

$$\frac{p \twoheadrightarrow^{\tau} p' \qquad p' \twoheadrightarrow^{a} p''}{p \twoheadrightarrow^{\tau.a} p''}$$

Définition 2 ($\subseteq_T$)

Soient p et q deux processus, on dit que $p \subseteq_T q$ si pour toute action $\tau \in \Sigma^*$, on a :

$Si \quad p \twoheadrightarrow^{\tau} p' \quad alors \quad q \twoheadrightarrow^{a} q'$

Définition 3 (équivalence par trace)

On dit que deux processus p et q sont équivalents par trace si $p \subseteq_T q$ et $q \subseteq_T p$

Propriétés

Les auteurs de [19] ont démontré les propriétés suivantes de l'équivalence par trace:

$(B1)$	$p + (q + r) \sim (p + q) + r$	$(B6)$	$p + p \sim p$
$(B2)$	$p.(q.r) \sim (p.q).r$	$(B7)$	$p.1 \sim p \sim 1.p$
$(B3)$	$p + q \sim q + p$	$(B8)$	$p.0 \sim p$
$(B4)$	$(p + q).r \sim p.r + q.r$	$(B9)$	$p + 0 \sim p$
$(B5)$	$p.(q + r) \sim p.q + p.r$	$(B10)$	$p * q \sim q + pp * q$

Définition 4 (PGFC)

Soient P et Q deux processus, le PGFC (Plus Grand Facteur Commun) des deux processus P et Q, noté $P \sqcap Q$, est un processus R tel que :

- ✓ $R \subseteq P$
- ✓ $R \subseteq Q$
- ✓ $Pour\ tout\ R' tel\ que\ R' \subseteq P\ et\ R' \subseteq Q\ alors\ R' \subseteq R$

Propriétés

Les auteurs de [19] ont proposé les propriétés suivantes :

$$P \sqcap 0 \sim 0$$

$$P \sqcap P \sim P$$

$$P \sqcap Q \sim Q \sqcap P$$

$$a.P \sqcap a.Q \sim a.(P \sqcap Q)$$

$$a.P \sqcap b.Q \sim 0$$

$$P \sqcap (Q + R) \sim P \sqcap Q + P \sqcap R$$

$$P \sim P' \rightarrow P \sqcap Q \sim P' \sqcap Q$$

Définition 5 (dérivatives)

Les auteurs de [19] ont adapté la définition des dérivatives de Brzozowski défini dans [54] comme suit :

Intuitivement :

$$\partial_a(p) = \{p' \quad tq: \quad p \xrightarrow{a} p'\}$$

Formellement : La dérivative d'un processus p par rapport à une action a noté $\partial_a(p)$ est définie comme suit :

$$\partial_a(0) = 0$$

$$\partial_a(a) = 1$$

$$\partial_a(b) = 0$$

$$\partial_a(p * q) = \partial_a(p).p * q + \partial_a(q)$$

$$\partial_a(p + q) = \partial_a(p) + \partial_a(q)$$

$$\partial_a(p.q) = \partial_a(p).q + o(p)\partial_a(q)$$

Où la fonction *o* sert à déterminer si un processus *P* peut être réduit au processus **1**, autrement dit, s'il termine immédiatement sans avoir à exécuter une quelconque action, Cette fonction prend un processus *P* comme argument et retourne la valeur **1** si *P* termine (si $P\downarrow$) ou la valeur **0** sinon. Elle définie comme suit :

$$o(0) = 0$$

$$o(1) = 1$$

$$o(a) = 0$$

$$o(p * q) = o(q)$$

$$o(p.q) = o(p) \times o(q)$$

$$o(p + q) = o(p) + o(q)$$

Définition 6 (les débuts)

La fonction δ représente l'ensemble des débuts d'un processus, elle est définie comme suit :

Intuitivement :

$$\delta(p) = \{a \quad tq: \quad p \xrightarrow{a} p'\}$$

Formellement :

$$\delta(0) = 0$$

$$\delta(1) = 0$$

$$\delta(a) = a$$

$$\delta(p * q) = \delta(p) \cup \delta(q)$$

$$\delta(p + q) = \delta(p) \cup \delta(q)$$

$$\delta(p.q) = \delta(p) \cup o(p) \otimes \delta(q)$$

Tel que la fonction $\otimes$ est définie comme suit :

$$\otimes : \{0,1\} \times 2^{\Sigma} \to 2^{\Sigma}$$

$$(0, S) \longmapsto \emptyset$$

$$(1, S) \longmapsto S$$

2.3. Résolution du problème

En utilisant la définition de l'équivalence par trace, les auteurs de [19] ont déduit que :

$$P \backsim o(P) + \sum_{a \in \delta(P)} a.\ \partial_a(P)$$

Intuitivement, cette formule signifie que tous processus non terminé est équivalent à la somme de toutes les actions de Σ concaténées à la dérivative du processus par rapport à l'action.

Et d'après les propriétés du PGCF on déduit que :

$$P \sqcap Q \backsim o(P) \times o(Q) + \sum_{a \in \delta(P) \cap \delta(Q)} a.(\partial_a(P) \sqcap \partial_a(Q))$$

Intuitivement, cette formule signifie que l'intersection de deux processus non terminés est équivalent à la somme de toutes les actions appartenant à l'intersection des deux ensembles de débuts, concaténées à l'intersection des dérivatives des processus par rapport à l'action.

Algorithme

La résolution du problème revient a trouver le PGCF des processus P et , les auteurs de cette méthode propose l'algorithme suivant :

Algorithme 1 : calcul de $P \sqcap \phi$ dans $BPA^*_{0,1}$

---

1 :

$$E \leftarrow \{P \sqcap \phi \backsim o(P) \times o(\phi) + \sum_{a \in \delta(P) \cap \delta(\phi)} a.(\partial_a(P) \sqcap \partial_a(\phi))\}$$

2 : Tant qu'il existe $P_i \sqcap \phi_i$ dans le coté droit d'une équation et qui n'apparaît dans le coté gauche de toutes les autres équations **Faire :**

3 :

$$E \leftarrow \{P_i \sqcap \phi_i \sim o(P_i) \times o(\phi_i) + \sum_{a \in \delta(P_i) \cap \delta(\phi_i)} a.(\partial_a(P_i) \sqcap \partial_a(\phi_i))\}$$

Fin tant que ;

4 : Retourner la solution du système linéaire E ;

Fin.

Le système d'équations linéaires générés par cet algorithme est de la forme : $AX + B = X$ ou A est une matrice de taille $n \times n$, B un vecteur de taille n et X un vecteur de variables de taille n. La résolution du système est donnée par la formule $X = X * B$

3. Utilisation de l'algèbre $CBPA^*_{0,1}$

Dans la seconde partie de l'article, on trouve une algèbre plus riche avec le rajout des variables et la possibilité d'exécution des actions conditionnelles. L'algèbre enrichie est nommée $CBPA^*_{0,1}$.

3.1. Définitions

- **Un terme :** soit c'est une variable, une constante (fonction d'arité 0) ou bien une fonction de termes $(f(t_1, \dots, t_n))$.
- L'ensemble des termes est noté $T(F, X)$, F est l'ensemble des fonctions, X l'ensemble des variables.
- Deux termes a et b de $T(F, X)$ sont unifiables s'il existe une substitution σ tel que $a\sigma = b\sigma$.
- **mgu :** (most general unifier) est la substitution la plus générale entre deux termes pour laquelle $a\sigma = b\sigma$.
- Un terme est dit fermé s'il ne contient pas de variable.
- Un processus est dit fermé si toutes ses actions sont des termes fermés.
- L'ensemble des actions fermés est noté $T(F)$.
- Si a est une action fermée alors on note $\bar{a}$ l'ensemble $T(F) \setminus \{a\}$.

3.2. Spécification $\boldsymbol{CBPA^{*}_{0,1}}$

C'est la version étendue de $BPA^{*}_{0,1}$. en incluant des variables et des conditions, pour des raisons de simplicité les auteurs de l'article ont considéré que a est un dans $T_1(F,X)$, ou $T_1(F,X)$ est un sous ensemble des termes de $T(F,X)$ qui contient au maximum une fonction d'arité supérieur à 0.

Syntaxe

Soit B un ensemble d'expressions booléennes et $[\![\_]\!]_B$ une fonction défini de B vers {0,1} , la syntaxe est défini par la grammaire BNF suivante :

$$P ::= 0 \mid 1 \mid c \rhd a \mid P.P' \mid P + P' \mid P * P'$$

$c \rhd a$: signifie que si la condition 'c' est vraie, alors on exécute l'action 'a'.

3.2.1. Sémantique

La sémantique opérationnelle des processus fermés est la suivante :

R^{a} $\quad \dfrac{\boxdot}{c \rhd a \xrightarrow{a} 1}\ [\![c]\!]_B = 1$	R^{1} $\quad \dfrac{\boxdot}{1 \downarrow}$
$R^{*}_{r\downarrow}$ $\quad \dfrac{Q \downarrow}{(P * Q) \downarrow}$	$R_{\downarrow}$ $\quad \dfrac{P \downarrow \quad Q \downarrow}{(P.Q) \downarrow}$
R_{l} $\quad \dfrac{P \downarrow \quad Q \xrightarrow{a} Q'}{P.Q \xrightarrow{a} Q'}$	R_{r} $\quad \dfrac{P \xrightarrow{a} P'}{P.Q \xrightarrow{a} P'.Q}$

$R_{l\downarrow}^{+}$ $\dfrac{P\downarrow}{(P+Q)\downarrow}$	$R_{r\downarrow}^{+}$ $\dfrac{Q\downarrow}{(P+Q)\downarrow}$
R_{l}^{+} $\dfrac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'}$	R_{r}^{+} $\dfrac{Q \xrightarrow{a} Q'}{P+Q \xrightarrow{a} Q'}$
R_{l}^{*} $\dfrac{P \xrightarrow{a} P'}{P*Q \xrightarrow{a} P'.(P*Q)}$	R_{r}^{*} $\dfrac{P \xrightarrow{a} P'}{P*Q \xrightarrow{a} P'.(P*Q)}$

Tableau III.2: sémantique opérationnelle de $CBPA_{0,1}^{*}$

Si un processus n'est pas fermé il est considéré comme suit : $P(X) =^{Def} \sum_{\sigma\in\Gamma} P\sigma$

Les axiomes reliés aux expressions booléennes sont les suivants :

$\neg\top = \bot$	$\top \wedge a = \top$
$\bot \wedge a = \bot$	$a \wedge (a' \wedge a'') = (a \wedge a') \wedge a''$
$a \wedge a' = a' \wedge a$	$\neg\neg a = a$
$\bot \rhd a = 0$	$a \wedge a = a$
$\neg\bot = \top$	$a \vee a' = \neg(\neg a \wedge \neg a')$

3.3. Résolution du problème (Calcule de $P \sqcap Q$):

3.3.1. Cas simple : *Si* $P = c_p \rhd a_p$ *et* $Q = c_q \rhd a_q$

$$P \sqcap Q = \begin{cases} \emptyset & si\ mgu(a_p, a_q)\ n'existe\ pas \\ \{c_p\sigma \wedge c_q\sigma \rhd a_p\sigma\} & si\ \sigma = mgu(a_p, a_q)\ existe \end{cases}$$

Pour des raisons de simplicité, on note le $mgu(a_p, a_q)$ par $\sigma a_p a_q$ et on définit $c\sigma + a_p a_q$ comme suit :

$$c\sigma + a_p a_q = \begin{cases} \perp & si\ \ \sigma a_p a_q\ \ n'existe\ pas \\ c\sigma a_p a_q & si\ \ \sigma a_p a_q\ \ existe \end{cases}$$

Donc on obtient :

$$P \sqcap Q = c_p\sigma + a_p a_q \wedge c_q\sigma + a_p a_q \rhd a_p\sigma + a_p a_q$$

3.3.2. Cas général

Soient :

$$\textbf{\textit{Si}} \quad P = c_p^1 \rhd a_p^1 \dots \dots c_p^n \rhd a_p^n \quad et \quad Q = c_q^1 \rhd a_q^1 \dots \dots c_q^n \rhd a_q^n$$

$$P \sqcap Q = ((c_p^1 \wedge c_q^1) \rhd a_p^1)\sigma_1 +$$

$$((c_p^1 \wedge c_q^1) \rhd a_p^1)\sigma_2\sigma_1 . (c_p^2 \wedge c_q^2) \rhd a_p^2)\sigma_2\sigma_1 +$$

$$\vdots$$

$$\left((c_p^1 \wedge c_q^1) \rhd a_p^1\right)(\sigma_n \dots \sigma_1) \dots \dots \left((c_p^n \wedge c_q^n) \rhd a_p^n\right)(\sigma_n \dots \sigma_1)$$

3.3 Exemple : $P = r(x,z).s(2) \quad et \quad Q = (y > 1) \rhd r(y,3).s(y)$

On a $\sigma_1 = \{x \mapsto y, 3 \mapsto z\}\ et\ \sigma_2 = \{2 \mapsto y\}$

$$P \sqcap Q = (x > 1) \rhd r(x,3) + (2 > 1) \rhd r(2,3).s(2)$$

A partir de l'exemple précédent il parait que c'est dur de trouver une formule simple pour calculer $P \sqcap Q$, il parait aussi que la taille de $P \sqcap Q$ est largement plus grande que la taille de P et Q, mais on remarque que :

$$P \sqcap Q = (x > 1 \wedge z = 3) \rhd r(x,z).(x = 2) \rhd s(2)$$

L'idée derrière l'obtention de cette forme simplifiée de $P \sqcap Q$, est de ne pas propager toutes les substitutions obtenues du mgu, on propage uniquement une partie de ces substitutions, la partie ou on substitue des variables.

Pour ce faire on considère que $\sigma = \sigma_\alpha \cup \sigma_\approx$ avec $\sigma_\approx = \{x \mapsto t/t \ est\ une\ variable\}$ $\sigma_\alpha = \sigma - \sigma_\approx$.

$\lceil\sigma_\alpha\rceil$ est considérée comme une condition additionnelle extraite de σ_α comme suit :

$\lceil\emptyset\rceil = \top$

$\lceil x \to t\rceil = (x = t) \wedge \lceil\sigma'\rceil$

En utilisant cette observation on calcule $P \sqcap Q$ comme suit :

$$P \sqcap Q = (c_p \wedge c_q \wedge \lceil\sigma_\alpha\rceil) \rhd a_p\sigma_\approx.(P_1\sigma_\approx \sqcap Q_1\sigma_\approx)$$

Définition ∇

On définit un nouvel operateur dans l'ensemble Σ comme suit :

$$\alpha_1 \nabla \alpha_2 = \begin{cases} \emptyset & si\ mgu(\alpha_1, \alpha_2)\ \ n'existe\ pas \\ \{((c_1 \wedge c_2 \wedge \lceil\sigma_\alpha\rceil) \rhd \alpha_1\sigma_\approx, \sigma_\approx)\} & si\ \ \sigma = si\ mgu(\alpha_1, \alpha_2)\ existe \end{cases}$$

Avec $\alpha_1 = c_1 \rhd a_1 \quad et \quad \alpha_2 = c_2 \rhd a_2$

Cet operateur, sert trouver les substitutions (s'elles existent) qui rendent deux actions équivalentes, et rajoute les substitutions de constantes comme condition supplémentaire ; on généralise sa définition aux ensembles d'actions conditionnelles comme suit :

$$S_1 \nabla S_2 = \bigcup_{\alpha_1 \in S_1,\ \alpha_2 \in S_2} \alpha_1 \nabla \alpha_2$$

Donc la formule finale pour calculer $P \sqcap Q$ est la suivante :

$$P \sqcap \mathrm{Q} \sim o(P) \times o(Q) + \sum_{\substack{(\alpha_1,\ \alpha_2) \in \delta(P) \times \delta(\mathrm{Q}) \\ (\alpha,\sigma) \in \alpha_1 \nabla \alpha_2}} \alpha.(\partial_{\alpha_1}(\mathrm{P})\sigma \sqcap \partial_{\alpha_2}(\mathrm{Q})\sigma)$$

Intuitivement, la formule signifie que l'intersection entre deux processus est équivalente à la somme des actions appartenant aux débuts des deux processus, et dont il existe une substitution qui les rend équivalentes, concaténées à l'intersection des dérivatives par rapports aux actions et avec la même substitution.

3.3.2. Algorithme

Et l'algorithme qui permet de générer le système d'équations est le suivant :

Algorithme2 : calcul de $P \sqcap \mathrm{Q}$ dans $CBPA^{*}_{0,1}$

1 :

$$E \leftarrow \{P \sqcap \mathrm{Q} \sim o(P) \times o(Q) + \sum_{\substack{(\alpha_1,\ \alpha_2) \in \delta(P) \times \delta(\mathrm{Q}) \\ (\alpha,\sigma) \in \alpha_1 \nabla \alpha_2}} \alpha.(\partial_{\alpha_1}(\mathrm{P})\sigma \sqcap \partial_{\alpha_2}(\mathrm{Q})\sigma)\}$$

2 : **Tant qu'il** existe $P_i \sqcap \mathrm{Q_i}$ dans le coté droit d'une équation et qui n'apparaît dans le coté gauche de toutes les autres équations **Faire :**

3 :

$$E \leftarrow E \cup \{P_i \sqcap \mathrm{Q_i} \sim o(P_i) \times o(\mathrm{Q_i}) + \sum_{\substack{(\alpha_1,\ \alpha_2) \in \delta(P_i) \times \delta(\mathrm{Q_i}) \\ (\alpha,\sigma) \in \alpha_1 \nabla \alpha_2}} \alpha.\left(\partial_{\alpha_1}(P_i)\sigma \sqcap \partial_{\alpha_2}(\mathrm{Q_i})\sigma\right)\}$$

Fin tant que ;

4 : Retourner la solution du système linéaire E

Fin.

Cette méthode est très intéressante car elle fournit un outil formel pour renforcer la sécurité dans les programmes, cependant les auteurs de la méthode étudiée ont utilisée l'équivalence par trace, cette équivalence est très significative du fait qu'elle se base sur les traces des processus pour dire si deux processus sont équivalents ou non, cependant, *il se peut que deux processus agissent différemment même s'ils ont les mêmes traces possibles.*

4. Applicabilité de l'équivalence par bissimulation

En étudiant cette méthode, l'idée d'utiliser un autre type d'équivalence nous est venue à l'esprit, et pour entamer notre étude, nous avons besoin de définir la relation de bissimulation :

4.1. Définitions

4.1.1. Définition de la bissimulation forte

Étant donné un système de transition d'états étiqueté $(S, \Sigma, \rightarrow)$, (dans notre cas ces états sont des processus).

Une relation binaire (R) de S sur S (c.à.d $R \subseteq S \times S$) est une relation de bissimulation ssi : Pour chaque paire d'élément P, Q dans S tq (P, Q) dans R, pour tout a dans Σ, les propriétés sont suivantes sont vérifiées :

1. $P \xrightarrow{a} P'$ $alors$ $\exists Q' \in S$ $tel\ que$ $Q \xrightarrow{a} Q'$ et $(P', Q') \in R$
2. $Q \xrightarrow{a} Q'$ $alors$ $\exists P' \in S$ $tel\ que$ $P \xrightarrow{a} P'$ et $(P', Q') \in R$

4.1.2. Définition de bissimilarité forte

Étant donnés deux processus p et q dans S, p est fortement bissimilaire à q, noté $p \sim q$, s'il existe une bissimulation forte R telle que (p, q)soit dans R.

Remarque : La relation de bisimilarité $\sim$ est une relation d'équivalence. (Facilement démontrable).

La bissimulation forte "$\sim$" fait trop de discriminations qui peut, dans certaines situations, devenir un inconvénient majeur. Par exemple, les deux processus $a.0$ et $a.\tau.0$ ne sont pas équivalents même si l'action τ reflète l'exécution d'une action interne inobservable de l'extérieur. Pour abstraire les actions invisibles, il faudrait baser la comparaison seulement sur les actions visibles, la notion de bissimulation faible a été introduite comme suit :

4.1.3. Définition de la bissimulation faible

Nous introduisons l'operateur $\Rightarrow$ de manière similaire à sa définition dans l'Algèbre CCS. Voici sa sémantique opérationnelle :

$$\frac{P \overset{\varepsilon}{\Rightarrow} P \qquad P \xrightarrow{\tau} P' \qquad P' \overset{\varepsilon}{\Rightarrow} Q}{P' \overset{\varepsilon}{\Rightarrow} Q}$$

$$\frac{P \overset{\varepsilon}{\Rightarrow} P' \qquad P' \xrightarrow{\gamma} Q' \qquad Q' \overset{\varepsilon}{\Rightarrow} Q}{P \overset{\gamma}{\Rightarrow} Q}$$

Une relation binaire de S sur S (c.à.d $R \subseteq S \times S$) est une relation de bisimulation faible ssi :

Pour chaque paire d'élément P, Q dans S tq (P, Q) dans R, pour tout a dans Σ, les propriétés suivantes sont vérifiées :

1. $P \xrightarrow{a} P' \; alors \; \exists Q' \in S \;\; tel\, que \;\; Q \xrightarrow{\hat{a}} Q' \;\; et \;\; (P', Q') \in R$
2. $Q \xrightarrow{a} Q' \; alors \; \exists P' \in S \;\; tel\, que \;\; P \xrightarrow{\hat{a}} P' \;\; et \;\; (P', Q') \in R$

Avec $\hat{a} = a \;\; si \; a \neq \tau \;\; et \;\; \hat{a} = \varepsilon$

Remarque : epsilon dénote une absence d'action.

4.1.4. Définition de bissimilarité faible

Étant donnés deux processus p et q dans S, p est faiblement bissimilaire à q, noté $p \sim\sim q$, s'il existe une bisimulation faible R telle que (p, q) soit dans R.

4.2. Etude du problème

Maintenant que nous savons ce qu'est la bissimulation considérons l'exemple suivant :

Soient : $P = a.(b + c)$ et $Q = a.b + a.c$ qui sont équivalents par trace car leur traces possibles sont dans l'ensemble $\{a, ab, ac\}$, cependant ces deux processus ne sont pas bissimilaires, ceci se voit par le fait que pour le processus Q, la première action exécutée, qui est forcement a, détermine la seconde action à pouvoir être exécutée, ce qui n'est pas le cas pour le processus P. Selon les définitions (1 et 3), pour que P et Q soient bissimilaires, il faudrait que le processus $b.0 + c.0$ soit en relation de bissimulation avec le processus $b.0$ ou le processus $c.0$; ces processus ne sont mêmes pas équivalents par trace, donc ils ne peuvent être bissimilaires.

Pour cette raison, nous *étudions l'applicabilité de l'équivalence par bissimulation dans la résolution algébrique* déjà étudiée :

Pour pouvoir appliquer l'algorithme proposé, il faudrait que toutes les propriétés (de B1 à B10) présentées dans la première section de ce chapitre soient correctes avec l'équivalence par bissimulation :

Nous avons relevé deux propriétés qui ne sont pas vérifiées (B5, et B8)

Contre exemple pour B5

Soient $x = c, \;\; y = a.P, \;\; z = b.P$

$V = c.(a.P + b.P)$ est il équivalent par bissimulation au processus $W = c.a.P + c.b.P$

Pour le processus W la première action exécutée, qui est forcement c, détermine la seconde action à pouvoir être exécutée, ce qui n'est pas le cas pour le processus V. Selon la définition de la bissimulation , pour que V et W soient bissimilaires, il faudrait que le processus $a.P + b.P$ soit en relation de bissimulation avec le processus $a.P$ et le processus $b.P$, étant donné que ces processus ne sont même pas équivalents par trace, ils ne peuvent être équivalents par bisssimulation.

Résolution du problème de B5

Définition de la simulation : c'est une relation entre processus associant des processus qui se comportent de la même façon au sens qu'un processus simule l'autre.

Intuitivement, un système simule l'autre s'il peut imiter toutes ses actions.

Formellement, c'est une relation binaire de S sur S (c.à.d $R \subseteq S \times S$), telle que pour chaque paire d'éléments P, Q dans S, si $(P, Q) \in R$ alors pour toute action a,

1. $P \xrightarrow{a} P'$ $alors$ $\exists Q' \in S$ $tel\ que$ $Q \xrightarrow{\hat{a}} Q'$ et $(P', Q') \in R$

On dit que Q simule P et on note $P \leq Q$

Le processus $(a.P + b.P)$ simule à la fois le processus $a.P$ et le processus $b.P$ car

$a.P \leq a.P + b.P$ $\quad et \quad$ $b.P \leq a.P + b.P$

donc la propriété B5 devient comme suit : $x.y + x.z \leq x.(y + z)$

Contre exemple pour B8

Soit $X = a$; les processus $a.0$ et a sont ils équivalents ? En exécutant l'action 'a' on obtiendra respectivement les deux processus 0 et 1, pour que cette propriété reste correcte il y'a deux solutions possibles :

Résolution du problème de B8

Solution 1 : enlever le processus bloqué de la syntaxe et le considérer comme un processus terminé, ceci peut se faire car un processus en inter blocage ne peut pas se libéré car il n'y a

pas la notion d'envoi et de réception de message ; donc on peut considérer que ce processus aura le même effet que le processus qui se termine.

Solution 2 : Supposer que les deux processus 0 et 1 sont bissimilaires, ceci est logique car les deux processus ne peuvent exécuter aucune action donc on ne peut trouver d'action 'a' pour laquelle les dérivées de 0 et de 1 par rapport à a ne soient pas équivalentes par trace.

NB : Nous optons pour seconde solution, car la première nous empêchera d'enrichir l'algèbre avec les opérations de communications qui sont indispensables si on veut appliquer cette méthode aux programmes réels.

4.3. Formalisation du nouveau problème

Soit un programme P et une politique de sécurité ϕ ; le but de ce travail est de générer un autre programme P_0 qui respecte la politique de sécurité et qui est dans P (P simule P_0 noté $P_0 \leq P$). Ce processus P doit vérifier les propriétés suivantes :

Justesse : Pour que P_0 soit correcte il faudrait que :

$P_0 \leq P$

$P_0 \leq \phi$

Complétude : Pour que P_0 soit complet il faudrait que :

S'il existe un autre processus P_1 tel que $P_1 \leq P$ et $P_1 \leq \phi$ alors $P_1 \leq P_0$.

4.4. Étude du nouveau problème

Définition (PGFCB) : Plus Grand Facteur Commun Bissimilaire

Soient P et Q deux processus, le PGFCB des deux processus P et Q, noté $P \sqcap_B Q$, est un processus R tel que :

- ✓ $R \leq P$;
- ✓ $R \leq Q$;
- ✓ Pour tout R' tq $R' \leq P \quad et \quad R' \leq Q \ \ alors \ \ R' \leq R$.

Donc la résolution de la problématique revient à trouver PGFCB de la politique de sécurité ϕ et du programme P.

Pour poursuivre la même démarche que dans l'article étudié, et pour pouvoir utiliser le même algorithme, il faudrait que les propriétés (P1 à P7 présentées dans la page 48) présentées dans la première section soient correctes

Considérons les propositions qui sont directement déduites à partir des définitions présentées :

Propositions

Soient P, Q et R trois processus :

- ✓ $P\ bissimilaire\ Q \quad ssi \quad P \leq Q\ \ et\ \ Q \leq P$ $\quad (Pr1)$
- ✓ $P \leq P$ $\quad (Pr2)$
- ✓ $P \leq Q\ \ ou\ P \leq R \quad ssi \quad P \leq Q + R$ $\quad (Pr3)$
- ✓ $P \leq Q \quad ssi \quad P \sqcap_B Q \leq Q \sqcap_B R$ $\quad (Pr4)$
- ✓ $P \leq Q\ \ et\ \ R \leq Q \quad alors\ \ P + R \leq Q$ $\quad (Pr5)$

Considérons la propriété P6 suivante : $P \sqcap (Q + R) \sim P \sqcap Q + P \sqcap R$.

Pour que $P \sqcap (Q + R)$ soit équivalent par bissimulation à $P \sqcap Q + P \sqcap R$ il faudrait que

$P \sqcap Q + P \sqcap R$ simule $P \sqcap (Q + R)$ et $P \sqcap (Q + R)$ simule $P \sqcap Q + P \sqcap R$

Démontrons que $P \sqcap Q + P \sqcap R \leq P \sqcap (Q + R)$

Nous savons que :

$\begin{cases} R \leq R \\ Q \leq Q \end{cases}$ $(Pr2)$ Alors

$\begin{cases} R \leq Q + R \\ Q \leq Q + R \end{cases}$ $(Pr3)$ Donc

$\begin{cases} P \sqcap R \leq P \sqcap (Q + R) \\ P \sqcap Q \leq P \sqcap (Q + R) \end{cases}$ $(Pr4)$

Donc on utilisant la proposition $(Pr5)$ on aura : $P \sqcap Q + P \sqcap R \leq P \sqcap (Q + R)$

Démontrons par l'absurde que l'autre cas n'est pas vrai :

On suppose que $P \sqcap (Q + R) \ \leq P \sqcap Q + P \sqcap R$

Soit $P \sqcap (Q + R) \ \leq P \sqcap Q$ ou bien $P \sqcap (Q + R) \ \leq P \sqcap R$ selon $(Pr3)$

Cas 1 : $P \sqcap (Q + R) \leq P \sqcap Q$ donc avec $(Pr4)$ $Q + R \leq Q$ (impossible)

Cas2 : $P \sqcap (Q + R) \leq P \sqcap R$ donc avec (Pr4) $Q + R \leq Q$ (impossible)

Il y'a une contradiction dans les deux cas, donc la supposition est incorrecte. *La propriété (P6) n'est pas vérifiée*, on ne peut avoir qu'une relation de simulation entre les deux formules.

5. Conclusion

Nous avons choisi l'une des méthodes les plus intéressantes qui traitent le problème de la sécurité des codes mobiles, nous avons détaillé cette méthode de réécriture proposée par MEJRI et all [19]. Ensuite, nous avons mis l'accent sur les limites de l'équivalence par trace qui est utilisée par cette méthode pour comparer les processus, et nous avons rappelé la définition d'un autre type d'équivalence qui est l'équivalence par bissimulation, puis nous avons prouvé que cette équivalence ne peut s'appliquer directement à cette méthode.

Dans le prochain chapitre nous enrichissons d'avantage les aspects de cette méthode, puis, nous validerons la méthode par des démonstrations formelles.

CHAPITRE IV

Approche algébrique pour le renforcement des politiques de sécurité sur des programmes

1. Introduction

Les auteurs de l'article [19] ont proposé une approche algébrique par réécriture de code pour le renforcement de la sécurité dans les programmes. L'avantage majeur de cette approche est qu'elle ramène le problème de renforcement des politiques de sécurité à un problème de résolution d'un système d'équations algébriques. Par ailleurs, l'approche algébrique offre plusieurs autres avantages en réduisant considérablement la complexité des systèmes à analyser, contrairement aux autres approches (automates à état finis, etc.). De plus, les algèbres de processus offrent un grand potentiel en termes d'implémentation, car l'écart entre la spécification et l'implémentation se trouve considérablement réduit. Cependant, l'approche proposée dans [19] est encore trop abstraite pour espérer son application sur un langage réel comme C++. Dans ce chapitre, nous nous proposons, donc, de redéfinir la technique en y apportant des améliorations à plusieurs niveaux permettant de faciliter l'application de cette méthode sur un langage réel. En effet, nous nous proposons de spécifier les propriétés par un formalisme logique de haut niveau. Par ailleurs, nous nous proposons d'introduire l'unification modulo une théorie équationnelle afin de prendre en considération les actions équivalentes du programme. L'équivalence des actions étant ramenée à leurs effets sur l'environnement. Finalement, nous optimisons le calcul du pgfc en caractérisant les

actions à effet nul sur la propriété de sécurité considérée. À la fin du chapitre, effectuons une évaluation de notre méthode.

Notre rappelons donc que notre objectif de recherche est de concevoir une technique qui, étant donné un processus P et une propriété de sécurité ϕ, de générer un processus P' qui :

- satisfait la propriété de sécurité ϕ ;
- est conforme à la spécification du processus P, dans le sens où toutes les traces d'exécution possibles satisfaisant ϕ que le processus P peut exécuter sont aussi des traces d'exécution possibles du processus P' ;
- n'introduit aucune nouvelle trace que le processus P ne peut exécuter.

Plus précisément, le processus P' à générer doit respecter les deux propriétés cruciales suivantes :

1. **Correction** :
   $\cdot P' \sqsubseteq P$: Toutes les traces du processus P' sont des traces du processus P.
   $\cdot P' \vDash \phi$: Toutes les traces du processus P' satisfont la propriété de sécurité ϕ.

2. **Complétude** : pour tout processus P'', si $P'' \sqsubseteq P$ et $P'' \vDash \phi$, alors $P'' \sqsubseteq P'$.

Intuitivement, le processus P' est le processus le moins restrictif qui satisfait la propriété ϕ et qui est conforme au processus P. La relation $\sqsubseteq$ capture la notion de « processus plus restrictif » et sera définit ultérieurement.

Ce chapitre est organisé en trois parties. La première partie présente des définitions de base. La deuxième partie présente l'algèbre CBPAE pour représenter les processus et une logique temporelle pour les formules de sécurité, ensuite, énonce les théorèmes de correction et de complétude. La troisième partie est consacrée à la validation de la méthode par des preuves et des exemples, et enfin, on effectue l'évaluation de la méthode proposée.
Dans ce qui suit, nous présentons les principales définitions formelles utilisées dans cette section.

2. Définitions

2.1. Définition (environnement)

Un environnement est une fonction partielle qui associe à des identificateurs (variables du programme) des valeurs calculables. Par exemple $E = [x \to 3, y \to true]$ est un environnement dans lequel x est associé à la valeur 3 et y à la valeur $true$. Nous définissions l'opérateur ‡ sur les environnements comme suit : $E \ddagger [x \to v]$ désigne l'environnement E augmenté de l'association $x \to v$. Si la variable x appartient déjà à E alors sa valeur est mise à jour.

Nous notons $\tilde{E}$ l'ensemble des environnements. Nous définissons le domaine de E, noté $Dom(E)$ comme étant l'ensemble des variables dans E i.e. :

$$Dom(E) = \{x(x \to v) \in E\}$$

Dans le reste de ce chapitre, nous ne considérons pas l'effet de bord des actions, mais leurs effets sur l'environnement. Soit E un environnement et a une action alors l'effet de a dans E noté $Eff(a, E)$ est un ensemble de couples (variable, valeur). Si E est un environnement d'exécution alors après l'exécution de l'action a, l'environnement d'exécution devient $E \ddagger Eff(a, E)$. Nous considérons que deux actions a et a' sont équivalentes si elles génèrent le même effet sur l'environnement, i.e. : $Eff(a, E) = Eff(a', E)$.

2.2. Définition (Algèbre des termes)

Un terme défini sur un ensemble de symboles F et un ensemble de variables X est soit :

- une variable $x \in X$,
- une constante $c \in F$ où c est une fonction d'arité zéro,
- ou une expression de la forme $f(t_1, \dots, t_n)$ où $f \in F$, n est l'arité de f et $t_1, \dots, t_n$ sont des termes définis sur F et X.

Nous désignons par $T(F, X)$ l'algèbre de termes définie sur F et X. Un terme de $T(F)$ est dit clos (ne contient pas de variables)

2.3. Définition (Unification)

Deux termes a et b appartenant à $T(F,X)$ sont unifiables s'il existe une substitution $\sigma: X \to T(F,X)$ telle que $\sigma(a) = \sigma(b)$. L'unificateur le plus général de a et b est noté $mgu(a,b)$ est une substitution σ telles que :

$$\begin{cases} \sigma(a) = \sigma(b) \\ \forall \sigma' : \sigma'(a) = \sigma'(b), \exists \sigma'' : \sigma' = \sigma \, o \, \sigma'' \end{cases}$$

2.4. Définition (Unification modulo une théorie équationnelle)

Soit E une théorie équationnelle $a =_E b$ ssi $a_{\downarrow} = b_{\downarrow}$ dans E, où $a_{\downarrow}$ désigne la forme normale (canonique) de a dans E. Deux termes a et b appartenant à $T(F,X)$ sont unifiables modulo une théorie équationnelle s'il existe une substitution $\sigma: X \to T(F,X)$ telle que $\sigma(a) =_E \sigma(b)$. L'unificateur le plus général de a et b est noté $mgu_E(a,b)$ est une substitution σ telles que :

$$\begin{cases} \sigma(a) =_E \sigma(b) \\ \nexists \sigma, \sigma' : \sigma'(a) = \sigma'(b) \; et \; \sigma = \sigma' \, o \, \sigma'' \end{cases}$$

La théorie équationnelle que nous définissons dans cette section contient deux équations : la première consiste à considérer chaque terme différent d'un terme donné comme étant son inverse, et la seconde équation considère que deux termes qui possèdent le même effet sur l'environnement sont équivalents. Donc, notre théorie équationnelle, qui concerne toute la suite du chapitre, est la suivante :

$$\text{E=} \begin{cases} \neg f(t_1, \dots, t_n) = g(t'_1, \dots, t'_n) \; si \; f \neq g \\ f(t_1, \dots, t_n) = g(t'_1, \dots, t'_n) \; si \; Eff(f) = Eff(g) \quad et \quad \begin{matrix} \exists \sigma \;\; \sigma(t_1) =_E \sigma(t'_1) \\ \vdots \\ \exists \sigma \;\; \sigma(t_n) =_E \sigma(t'_n) \end{matrix} \end{cases}$$

3. Renforcement des propriétés de sûreté

Dans cette section, nous décrivons l'approche algébrique qui nous permet de renforcer les propriétés de sûreté sur un programme manipulant des variables et instruction conditionnelles. Tout d'abords, nous décrirons l'algèbre de processus adoptée pour spécifier les programmes. Plus précisément, nous décrirons la syntaxe et la sémantique opérationnelle de ce langage de spécification. Puis, nous suivrons les mêmes étapes pour présenter la logique qui nous permettra de spécifier les propriétés de sécurité. Ensuite, nous décrivons la démarche adoptée, afin de générer le programme sure.

3.1. Langage de spécification CBPAE*01

Soit Σ un ensemble d'actions. Soit c une expression booléenne dans **B** et $[\![\_]\!]_B$ une fonction définie de **B** vers $\{0, 1\}$. La syntaxe de CBPAE$^*_{01}$ est présentée par la grammaire qui suit. Dans cette syntaxe, l'exécution d'une action $a \in \Sigma$ est conditionnée par une expression booléenne ($c \rhd a$). Dans ce qui suit, on désigne par $\top, \bot$, vrai et faux respectivement. On va également utiliser a comme abréviation de l'expression $\top \rhd a$ et 0 comme abréviation de $\bot \rhd a$. La syntaxe de CBPAE$^*_{01}$ est décrite par la grammaire qui suit :

$$P := 0 \mid 1 \mid c \rhd a \mid P.P' \mid P + P' \mid P * P'$$

- 0 désigne un processus en état de blocage.
- 1 désigne un processus qui a terminé normalement son exécution (c.à.d. qui n'a plus d'actions à exécuter).
- a désigne le processus qui peut exécuter l'action a.
- $P + P'$ représente le choix entre deux processus P ou P'.
- $P.P'$ représente la composition séquentielle de deux processus P et P'.
- $P * P'$ représente le processus qui se comporte selon le processus $P.P * P' + P'$. Il s'agit de la version binaire de l'opérateur étoile de Kleene [53]. Pour des soucis de simplicité, nous désignons par P^ω le processus $P * 0$.

Dans ce qui suit, nous notons $\mathcal{P}$ l'ensemble des processus décrit avec CBPAE$^*_{01}$.

La sémantique opérationnelle

La sémantique opérationnelle de CBPAE$^{*}_{01}$ est définie par la relation de transition décrite dans la table IV.2.

R^{a} $\quad \dfrac{\boxdot}{c \rhd a \xrightarrow{a} 1}\ [\![c]\!]_{B} = 1$	R^{1} $\quad \dfrac{\boxdot}{1 \downarrow}$
$R^{*}_{r\downarrow}$ $\quad \dfrac{Q \downarrow}{(P * Q) \downarrow}$	$R_{\downarrow}$ $\quad \dfrac{P \downarrow \quad Q \downarrow}{(P . Q) \downarrow}$
R_{l} $\quad \dfrac{P \downarrow \quad Q \xrightarrow{a} Q'}{P . Q \xrightarrow{a} Q'}$	R_{r} $\quad \dfrac{P \xrightarrow{a} P'}{P . Q \xrightarrow{a} P' . Q}$
$R^{+}_{l\downarrow}$ $\quad \dfrac{P \downarrow}{(P + Q) \downarrow}$	$R^{+}_{r\downarrow}$ $\quad \dfrac{Q \downarrow}{(P + Q) \downarrow}$
R^{+}_{l} $\quad \dfrac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$	R^{+}_{r} $\quad \dfrac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$
R^{*}_{l} $\quad \dfrac{P \xrightarrow{a} P'}{P * Q \xrightarrow{a} P' . (P * Q)}$	R^{*}_{r} $\quad \dfrac{P \xrightarrow{a} P'}{P * Q \xrightarrow{a} P' . (P * Q)}$

Tableau IV.2 : Sémantique opérationnelle de CBPAE$^{*}_{01}$

Les règles sémantiques sont interprétées comme suit :

- La règle R^{a} spécifie le fait que le processus constitué de l'action atomique *a*, peut évoluer uniquement en exécutant cette action qui est conditionnelle à la satisfaction de

la condition booléenne qui lui est associée et représenté par le test $[\![c]\!]_B = 1$ où c est une condition booléenne.

- Les règles (R^+) spécifient le fait que si l'un des processus P ou Q évolue respectivement vers un processus P' ou Q' alors le processus $P + Q$ évolue également vers l'un des processus P' ou Q'.

- Les règles (R_l) et (R_r) traduisent le fait qu'afin de permettre à la composition séquentielle d'évoluer, il faut que le premier processus de la composition puisse évoluer, ou bien que le premier processus termine et que le second puisse évoluer.

- Les règles (R_l^*) et (R_{rl}^*) spécifient les comportements répétitifs. Le processus $P * Q$ a le choix entre l'exécution du processus P et l'exécution du processus Q. S'il choisit d'exécuter le processus P et qu'il termine l'exécution de ce processus, il se retrouve encore une fois devant le choix d'exécuter le processus P ou le processus Q. Cette situation est réitérée jusqu'à ce qu'il décide d'exécuter le processus Q.

- Les règles $R_{r\downarrow}^*$, $R_{\downarrow}$, $R_{l\downarrow}^+$, $R_{r\downarrow}^+$ spécifient les cas de terminaison des processus comme suit :
  - Pour que le processus $P * Q$ termine, il faut que le processus Q termine.
  - Pour que le processus $P.Q$ termine, il faut que les processus P et Q terminent.

  Pour que le processus $P + Q$ termine, il faut que le processus P termine ou que le processus Q termine.

3.2. Langage logique pour la spécification des politiques de sécurité (ℵ):

La majorité des travaux existants dans la littérature utilisent des langages logiques comme langage formel pour spécifier les politiques de sécurité. En effet, les opérateurs tels que la négation et la conjonction n'ont pas leurs équivalents directs dans les algèbres de processus. En outre, les logiques permettent d'exprimer les propriétés de sécurité d'une manière plus naturelle. Dans le cadre de cette recherche, nous avons constaté le besoin de définir une logique permettant de décrire les politiques de sécurité qui seront utilisées lors du renforcement. Toutefois, nous gardons le même objectif qui, rappelons-le, consiste à

développer un cadre formel pour le renforcement de politiques de sécurité dans des programmes.

Par ailleurs, l'utilisation de la logique comme langage de spécification de politiques de sécurité a engendré l'introduction d'un nouvel opérateur pour le calcul du plus grand facteur commun (noté $\sqcap_{\aleph}$) entre un processus et une formule logique de telle sorte que $P' = P \sqcap_{\aleph} \phi$ respecte les propriétés suivantes :

$$\begin{cases} P' \vDash \phi & (1) \\ P' \sqsubseteq P & (2) \\ \forall Q : (Q \vDash \phi \wedge Q \sqsubseteq P) \rightarrow Q \sqsubseteq P' & (3) \end{cases}$$

La propriété (1) assure que le processus P' satisfait la politique de sécurité ϕ. La propriété (2) assure que toutes les traces de $P \sqcap_{\aleph} \phi$ sont aussi des traces de P. La propriété (3) assure que P' est le programme le plus général (le plus grand en termes de traces d'exécution) qui satisfait ϕ.

Il existe une variété de logiques qui ont été élaborées afin de pouvoir exprimer les propriétés spécifiques au renforcement de politiques de sécurité. Lesdites logiques sont en général de nature temporelle : ce sont des propriétés qui portent sur l'évolution des processus au cours du temps. Dans la littérature, il existe plusieurs logiques temporelles, parmi lesquelles, nous pouvons citer la logique temporelle linéaire, la logique de Hennessy-Milner ou le μ-calcul modal [56,57]. L'objectif de cette section est de définir une logique pour la spécification de la classe de politique de sûreté. Elle doit répondre aux critères suivants :

- Linéaire et temporelle : nous avons besoin d'une logique qui permet d'exprimer des propriétés qui vérifient un modèle linéaire. L'hypothèse de linéarité revient à dire qu'un processus ne peut être que dans un seul état à un instant donné.
- Bien adapté pour la spécification des propriétés de sûreté. Ces dernières expriment le fait que rien de mauvais n'arrivera durant l'exécution d'un programme.
- Permettant la spécification de propriétés infinies. En effet, il existe des programmes qui s'exécutent indéfiniment et il est important de pouvoir capturer ce genre de comportements.

Dans le but de répondre aux critères énoncés ci-dessus, nous avons défini une logique, notée par $\aleph$, inspirée du formalisme pour les expressions régulière étendues. D'une manière

intuitive, ℵ est une logique linéaire qui exprime la classe de langage régulier mais avec la possibilité d'exprimer des propriétés infinies. Notre choix est motivé par le fait que nous cherchons une logique qui se marie bien avec la syntaxe des algèbres de processus afin de répondre à notre objectif principal qui consiste à renforcer une politique de sécurité sur un programme. Soit $a \in \Sigma$, la syntaxe et la sémantique de ℵ est présentée ci-après.

Syntaxe de ℵ

Nous avons choisis les operateurs les plus utilisés par les logiques temporelles, et qui correspondent à nos besoins pour exprimer les propriétés de sécurité.

$\phi ::= \top$
$\mid 0$
$\mid 1$
$\mid a$
$\mid \neg\phi$
$\mid \phi.\phi'$
$\mid \phi \vee \phi'$
$\mid next(\phi)$
$\mid \cup(\phi, \phi')$

Syntaxe de ℵ

$\top$: true

$0(1)$: Exprime les formules correspondant aux processus bloqués (respectivement terminé).

a : C'est la formule correspondant à une action élémentaire a.

$\neg\phi$: La négation d'une formule

$\phi.\phi'$: Conjonction de deux formules

$\phi \vee \phi'$: Le ou inclusif entre deux formules

$next(\phi)$: A partir de la prochaine action la formule ϕ sera vraie.

$\cup(\phi, \phi')$: ϕ' deviendra vraie quand ϕ s'arrêtera d'être vraie.

Sémantique opérationnelle de$\aleph$

Dans ce qui suit, nous notons $L_{\aleph}$ l'ensemble des formules logiques de $\aleph$. La sémantique d'une formule logique de $\aleph$ correspond au processus CBPAE$^{*}_{01}$ pour lequel la formule est vraie. Formellement la sémantique de $\aleph$ est définie par la fonction sémantique

$[\![\ ]\!] : L_{\aleph} \rightarrow \mathcal{P}\ .\ P \vDash \phi \leftrightarrow P \sqsubseteq [\![\phi]\!]\ .$

$$[\![\top]\!] = \sum_{P \in \mathcal{P}} P$$

$$[\![0]\!] = 0$$

$$[\![1]\!] = 1$$

$$[\![a]\!] = a$$

$$[\![\neg\phi]\!] = \sum_{P \in \mathcal{P}} P \setminus [\![\phi]\!]$$

$$[\![\phi\ .\ \phi']\!] = [\![\phi]\!].[\![\phi']\!]$$

$$[\![\phi \vee \phi']\!] = [\![\phi]\!] + [\![\phi']\!]$$

$$[\![next(\phi)]\!] = \sum_{a \in \Sigma} a\ .\ [\![\phi]\!]$$

$$[\![U(\phi, \phi')]\!] = [\![\phi]\!] * [\![\phi']\!]$$

Sémantique de $\aleph$

$[\![\top]\!] = \sum_{P \in \mathcal{P}} P$ Signifie que $\top$ est vrai pour tous les processus possibles.

$[\![0]\!] = 0, [\![1]\!] = 1, [\![a]\!] = a$: Ces formules satisfont respectivement les processus $0,1,a$.

$[\![\neg\phi]\!] = \sum_{P \in \mathcal{P}} P \setminus [\![\phi]\!]$ Les processus qui satisfont $\neg\phi$ sont tous ceux qui ne satisfont pas ϕ.

$[\![\phi\ .\ \phi']\!] = [\![\phi]\!].[\![\phi']\!]$ La conjonction entre deux formules satisfait la concaténation des deux processus satisfaisant ces deux formules.

$[\![\phi \vee \phi']\!] = [\![\phi]\!] + [\![\phi']\!]$ le processus qui satisfait la disjonction de deux formules est le choix entre les processus qui satisfont ces formules.

$[\![next(\phi)]\!] = \sum_{a \in \Sigma} a . [\![\phi]\!]$ le processus qui satisfait $next(\phi)$ est la concaténation de toute action a de Σ aux processus qui satisfait ϕ.

$[\![U(\phi, \phi')]\!] = [\![\phi]\!] * [\![\phi']\!]$ le processus qui satisfait $U(\phi, \phi')$ est le processus $[\![\phi]\!] * [\![\phi']\!]$.

3.3. Définitions

Définition (Dérivatives d'une formule de $\aleph$)

Nous adaptons la définition des dérivatives de Brzozowski [54] aux formules logiques de $\aleph$, la dérivative d'une formule logique φ par rapport à une action a , noté ${}^{\aleph}\partial_a(\varphi)$, est définie comme suit :

$${}^{\aleph}\partial_a(0) = 0$$

$${}^{\aleph}\partial_a(a) = 1$$

$${}^{\aleph}\partial_a(b) = 0$$

$${}^{\aleph}\partial_a(next(\varphi)) = {}^{\aleph}\partial_a(\varphi)$$

$${}^{\aleph}\partial_a(U(\varphi, \varphi')) = {}^{\aleph}\partial_a(\varphi).U(\varphi, \varphi') \vee {}^{\aleph}\partial_a(\varphi')$$

$${}^{\aleph}\partial_a(\varphi \vee \varphi') = {}^{\aleph}\partial_a(\varphi) \vee {}^{\aleph}\partial_a(\varphi')$$

$${}^{\aleph}\partial_a(\varphi.\varphi') = {}^{\aleph}\partial_a(\varphi).\varphi' \vee {}^{\aleph}o(\varphi){}^{\aleph}\partial_a(\varphi')$$

Où la fonction ${}^{\aleph}o$ sert à déterminer si la formule φ est équivalente à la formule 1 , cette fonction prend comme argument une formule φ et retourne la valeur 1 si $\varphi = 1$ ou la valeur 0 sinon.

Définition (les débuts d'une formule de $\aleph$**)**

Nous adaptons la fonction qui calcule les débuts aux formules logiques φ de $\aleph$**,** cette fonction est noté ${}^{\aleph}\delta(\varphi)$, elle est définie comme suit :

$$^{\aleph}\delta(0) = 0$$

$$^{\aleph}\delta(1) = 0$$

$$^{\aleph}\delta(a) = a$$

$$^{\aleph}\delta\big(next(\varphi)\big) = {}^{\aleph}\delta(\varphi)$$

$$^{\aleph}\delta(U(\varphi, \varphi')) = {}^{\aleph}\delta(\varphi) \cup {}^{\aleph}\delta(\varphi')$$

$$^{\aleph}\delta(\varphi \vee \varphi') = {}^{\aleph}\delta(\varphi) \cup {}^{\aleph}\delta(\varphi')$$

$$^{\aleph}\delta(\varphi.\varphi') = {}^{\aleph}\delta(\varphi) \cup ({}^{\aleph}o(\varphi) \otimes {}^{\aleph}\delta(\varphi'))$$

La fonction $\otimes$ est la même que celle définie par les auteurs de [19], elle a été présentée au chapitre précédent.

4. Optimisation du renforcement

Au cours de notre étude, nous avons remarqué, qu'il y a des dérivatives de formule qui ne changent pas, c'est de la que nous est venu l'idée de l'optimisation. C'est pour cette raison que nous définissons l'ensemble des actions non pertinentes d'une formule logique, cet ensemble contient les actions qui ne changent pas l'état d'une formule, si on calcule ses dérivatives par rapport à elles. On définit la fonction Γ qui calcule les actions non pertinentes de la formule.

Une action a n'a pas d'effet sur l'environnement si $\partial_a(\varphi_i) = \varphi_i$ quelquesoit la sous formule φ_i de φ

$$\begin{cases} \Gamma^0(\varphi) = \emptyset \\ \Gamma^i(\varphi) = \bigcup\limits_{\substack{a \in \varphi^+ \\ {}^{\aleph}\partial_a^i(\varphi) = \varphi}} \{a\} \end{cases}$$

Ou $^{\aleph}\partial_a^i(\varphi)$ est la i^{eme} dérivative par rapport à a, et φ^+est l'ensemble des actions de φ.

Et nous généralisons cette définition comme suit :

$$\Gamma(\varphi) = \bigcup_{i=1}^{\infty} \Gamma^i(\varphi)$$

Définition On adapte la définition de l'operateur ∇ comme suit :

$$\nabla(\alpha, \beta, A) = \begin{cases} (\alpha, \emptyset) & si \quad \beta \in A \\ \emptyset \quad si\ mgu_E(\alpha, \beta) & n'existe\ pas \\ \{((c_\alpha \wedge c_\beta \wedge \lceil \sigma_\alpha \rceil) \rhd \alpha\sigma_\approx, \sigma_\approx)\} & si\ \ \sigma = si\ mgu_E(\alpha, \beta)\ existe \end{cases}$$

Cet operateur, sert trouver les substitutions (s'elles existent) qui rendent deux actions équivalentes, et rajoute les substitutions de constantes comme condition supplémentaire, le A contiendra les actions non pertinentes, donc si notre action α n'est pas pertinente on reste dans le même état et sans substitution.

On généralise la définition de cet operateur aux ensembles d'actions conditionnelles comme suit :

$$\nabla(S_1, S_2, A) = \bigcup_{\alpha \in S_1, \beta \in S_2} \nabla(\alpha, \beta, A)$$

5. Résolution du problème

Les propriétés de l'intersection vue dans le chapitre précédent reste valable, pour notre operateur d'intersection qui s'effectue modulo une théorie équationnelle.

Nous considérons les deux propositions suivantes :

5.1. Proposition1.

Si P est un processus, alors il peut être exprimé comme suit :

$$P \backsim o(P) + \sum_{a \in \delta(P)} a.\ \partial_a(P)$$

Démonstration. Nous allons procéder par induction sur le nombre d'actions par lesquelles un processus P peut débuter.

Soit H le prédicat suivant :

$$\mathrm{H(n)} : P \backsim o(P) + \sum_{a \in \delta(P)} a.\ \partial_a(P) \quad avec \quad |\delta(P)| = n$$

Ce prédicat traduit le fait que cette équivalence est vraie pour les n actions atomiques par lesquelles le processus P commence son exécution.
On veut montrer par induction mathématique que $\mathrm{H}(\mathrm{n})$ est vraie pour tout $\mathrm{n} \geq 0$.

– ***Base :*** $\mathrm{H}(0)$ est vraie. Cela correspond au cas d'un processus qui débute avec zéro actions (c.à.d. aucune action). Les seuls processus définis dans notre algèbre possèdent cette particularité sont les processus qui sont équivalents au processus 0 et les processus qui sont équivalents au processus 1. En effet, on a :

- Pour les processus équivalents au processus 0, $\mathrm{H}(0)$ est vraie car : $0 \backsim o(0) + 0 = 0$
- Pour les processus équivalents au processus 1, $\mathrm{H}(0)$ est vraie car : $0 \backsim o(1) + 0 = 1$

– ***Hypothèse d'induction :*** Soit $\mathrm{n} \geq 0$. Supposons que $\mathrm{H}(\mathrm{n})$ est vraie, nous devons montrer que $\mathrm{H}(\mathrm{n}+1)$ est vraie. À partir de l'hypothèse d'induction, nous avons :

$$P \backsim o(P) + \sum_{a \in \delta(P)} a.\ \partial_a(P) \quad avec \quad |\delta(P)| = n \qquad (I)$$

Soit R un processus appartenant à l'ensemble $\mathcal{P}$, et b une action atomique appartenant à l'ensemble Σ. Afin d'avoir $n+1$ actions par lesquelles le processus P peut débuter, on distingue deux cas de figures :

- ***Cas du processus*** $P' = P + b.R$: À partir des propriétés de $\backsim$, en composant le processus $b.R$ via l'opérateur $+$ aux deux membres de l'équivalence, on obtient :

$$P + b.R \backsim o(P) + \sum_{a \in \delta(P)} a.\ \partial_a(P) + b.R$$

En intégrant le processus $b.R$ à l'intérieur de la somme, on aboutit à :

$$P + b.R \backsim o(P) + \sum_{a \in \delta(P+b.R)} a.\ \partial_a(P + b.R) \qquad (II)$$

Par hypothèse, le processus P possède n débuts $(n > 0)$, donc le processus P ne termine pas tout de suite son exécution, on déduit donc que : $o(P) = 0$.
Aussi, selon la définition de la fonction o, on a : $o(b.R) = o(b) \times o(R) = 0$.

Par conséquent :

$$o(P + b.R) = Max\big(o(P), o(b.R)\big) = 0 = o(P) \qquad (III)$$

Comme $P' = P + b.R$, on déduit de *II* et de *III* que :

$$P' \backsim\ o(P') + \sum_{a \in \delta(P)} a.\ \partial_a(P') \quad avec \quad |\delta(P')| = n + 1$$

Ce qui montre que $H(n+1)$ est vraie.

- ***Cas du processus*** $P' = P * b.R$: En utilisant les définitions des fonctions o et ∂ , on peut développer l'expression suivante :

$o(P * b.R) + \sum_{a \in \delta(P*b.R)} a.\ \partial_a(P * b.R) \quad \sim$

$o(b.R) + \sum_{a \in \delta(p) \cup \delta(b.R)} a.\Big(\partial_a(P).P * b.R + \partial_a(b.R)\Big) \sim$

$0 + \big(\sum_{a \in \delta(p)} a.\partial_a(P).P * b.R\big) + b.R \quad \sim$

$\Big(\sum_{a \in \delta(P)} a.\ \partial_a(P)\Big).P * b.R + b.R$

Par conséquent :

$$o(P * b.R) + \sum_{a \in \delta(P*b.R)} a.\ \partial_a(P * b.R)\ \sim \left(\sum_{a \in \delta(P)} a.\ \partial_a(P)\right).P * b.R + b.R \qquad (IV)$$

En utilisant l'hypothèse d'induction I , on obtient après remplacement dans IV :

$$o(P * b.R) + \sum_{a \in \delta(P*b.R)} a.\ \partial_a(P * b.R)\ \sim P.P * b.R + b.R \qquad (V)$$

À partir de V, selon la propriété $B10$ de la relation d'équivalence $\sim$ qui porte sur l'étoile binaire de Kleene, on déduit que :

$$o(P * b.R) + \sum_{a \in \delta(P*b.R)} a.\ \partial_a(P * b.R)\ \sim P * b.R$$

Comme $P' = P * b.R$, on déduit que :

$$o(p') + \sum_{a \in \delta(P\prime)} a.\ \partial_a(P')\ \sim P' \qquad avec \qquad |\delta(P')|\ = n+1$$

Ce qui montre que $H(n+1)$ est vraie.

– ***Conclusion :*** Par le principe d'induction généralisé, nous avons montré que $H(n)$ est vraie pour tout $n \geq 0$.

5.2. Proposition2. On calcule l'intersection entre le processus P et $[\![\phi]\!]$ (qu'on va noter Φ) par la formule finale suivante :

$$P \sqcap_E \Phi = o(P) \times o(\Phi) + \sum_{\substack{(\alpha,\beta)\in\delta(P)\times^{\aleph}\delta(\Phi) \\ (\mu,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi))}} \mu.\,(\partial_\alpha(\mathrm{P})\sigma \sqcap_E {}^{\aleph}\partial_\beta(\Phi)\sigma)$$

Démonstration. Afin de démontrer cette proposition, on a d'abords besoin de prouver le lemme suivant :

Lemme.1 Étant donné n processus $P_1, \ldots, P_n$, n formules logiques $\Phi_1, \ldots, \Phi_{\mathrm{n}}$ et n actions atomiques distinctes $a_1, \ldots, a_n$, la propriété suivante de $\sqcap_E$ est vérifiée :

$$\sum_{i=1}^{n} a_i P_i \sqcap_E \sum_{i=1}^{n} a_i \Phi_{\mathrm{i}} \sim \sum_{i=1}^{n} a_i (P_i \sqcap_E \Phi_{\mathrm{i}})$$

Démonstration. Nous allons démontrer ce lemme par induction sur le nombre d'actions a_i.
Soit H le prédicat suivant :

$$H(n): \sum_{i=1}^{n} a_i P_i \sqcap_E \sum_{i=1}^{n} a_i \Phi_{\mathrm{i}} \sim \sum_{i=1}^{n} a_i (P_i \sqcap_E \Phi_{\mathrm{i}}) \qquad (VI)$$

– ***Base :*** $H(1)$ est vraie. Ceci correspond au cas suivant :

$a_1 P_1 \sqcap_E a_1 \Phi_1 \sim a_1 (P_1 \sqcap_E \Phi_1)$ Ceci est vrai selon les propriétés de énoncées de $(\sqcap_E)$

– *Hypothèse d'induction :* Soit $n \geq 1$. Supposons que $H(n)$ est vraie, nous devons montrer que $H(n+1)$ est vraie.

En utilisant les propriétés de la somme et de l'intersection $\sqcap_E$, on a :

$\sum_{i=1}^{n+1} a_i P_i \sqcap_E \sum_{i=1}^{n+1} a_i \Phi_i \sim (\sum_{i=1}^{n+1} a_i P_i + a_{n+1} P_{n+1}) \sqcap_E (\sum_{i=1}^{n} a_i \Phi_i + a_{n+1} \Phi_{n+1}) \sim$

$(\sum_{i=1}^{n} a_i P_i \sqcap_E \sum_{i=1}^{n} a_i \Phi_i) +$

$(\sum_{i=1}^{n} a_i P_i \sqcap_E a_{n+1} \Phi_{n+1}) + (a_{n+1} P_{n+1} \sqcap_E \sum_{i=1}^{n} a_i \Phi_i) + (a_{n+1} P_{n+1} \sqcap_E a_{n+1} \Phi_{n+1}) \sim$

$(\sum_{i=1}^{n} a_i P_i \sqcap_E \sum_{i=1}^{n} a_i \Phi_i) + (a_{n+1} P_{n+1} \sqcap_E a_{n+1} \Phi_{n+1}) \quad car \quad \forall i \ 1 \leq i \leq n, a_{n+1} \neq a_i.$

Par conséquent, on déduit que :

$$\sum_{i=1}^{n+1} a_i P_i \sqcap_E \sum_{i=1}^{n+1} a_i \Phi_i \sim (\sum_{i=1}^{n} a_i P_i \sqcap_E \sum_{i=1}^{n} a_i \Phi_i) + a_{n+1} (P_{n+1} \sqcap_E \Phi_{n+1}) \qquad (VII)$$

À partir de l'hypothèse de récurrence VI et de l'équivalence VII, on aboutit à :

$$\sum_{i=1}^{n+1} a_i P_i \sqcap_E \sum_{i=1}^{n+1} a_i \Phi_i \sim \sum_{i=1}^{n} a_i (P_i \sqcap_E \Phi_i) + a_{n+1} (P_{n+1} \sqcap_E \Phi_{n+1}) \sim \sum_{i=1}^{n+1} a_i (P_i \sqcap_E \Phi_i)$$

Ce qui montre que $H(n+1)$ est vraie

– ***Conclusion :*** Par le principe d'induction généralisé, nous avons montré que $H(n)$ est vraie pour tout $n \geq 1$.

Afin de prouver la proposition 2, nous allons également procéder par récurrence sur le nombre d'actions unifiables par lesquelles débutent un processus et une formule.

Soit $H'(n)$le prédicat suivant :

$H'(n)$:

$$P \sqcap_E \Phi = o(P) \times {}^{\aleph} o(\Phi) + \sum_{\substack{(\alpha,\beta) \in \delta(P) \times {}^{\aleph}\delta(\Phi) \\ (\mu,\sigma) \in \nabla(\alpha,\beta,\Gamma(\Phi))}} \mu.(\partial_\alpha(P)\sigma \sqcap_E {}^{\aleph}\partial_\beta(\Phi)\sigma) \ avec \ |\nabla(\alpha,\beta,\Gamma(\Phi))| = n$$

(VIII)

Ce prédicat traduit le fait que cette équivalence est vraie pour les n actions atomiques unifiables par lesquelles le processus P et la formule Φ commencent leur exécution. On veut montrer par induction mathématique que $H'(n)$ est vraie pour tout $n \geq 0$.

– ***Base :*** $H'(0)$ est vraie. Cela correspond au cas $\nabla(\alpha,\beta,\Gamma(\Phi)) = \emptyset$, où il n'existe pas de mgu_E entre les débuts du processus et de la formule. Les cas possibles sont les suivants :

- Le processus est équivalent à 1 et la formule est équivalente à 0, ou bien l'inverse.

Dans ce cas, $H'(0)$ est vraie car :

  - Si $P \sim 1 : 1 \prod_E \Phi \sim o(1) \times {}^{\aleph} o(\Phi) + 0 \sim {}^{\aleph} o(\Phi)$
  - Si $P \sim 0 : 0 \prod_E \Phi \sim o(0) \times {}^{\aleph} o(\Phi) + 0 \sim 0$

- Ni le processus ni la formule ne sont équivalents à 0 ou 1. Dans ce cas, le processus $P \prod_E \Phi \sim 0$, on a :

  $H'(0)$ est vraie car : $P \prod_E \Phi \sim o(P) \times {}^{\aleph} o(\Phi) + 0 \sim 0$

– ***Hypothèse d'induction :*** Soit $n > 0$. Supposons que $H'(n)$ est vraie, nous devons montrer que $H'(n+1)$ est vraie.

Soient R un processus appartenant à l'ensemble $\mathcal{P}$, R' une formule de ${}^{\aleph}$ et b une action atomique appartenant à l'ensemble Σ telle que : $\delta(P) \cap \{b\} = {}^{\aleph}\delta(\Phi) \cap \{b\} = \emptyset$

Il est à noter que P et Φ partagent n actions de débuts unifiables ($n > 0$), donc le processus P et la formule Φ ne terminent pas immédiatement leurs exécutions, on déduit donc que : $o(P) = {}^{\aleph} o(\Phi) = 0$.

Afin d'avoir $n+1$ actions unifiables par lesquelles P et Φ peuvent débuter, on distingue trois cas de figures :

- $P' = P + b.R \; et \; \Phi' = \Phi + b.R'$:

En utilisant les définitions des fonctions : $o, \delta \; et \; \partial$, on peut développer l'expression suivante :

$$o(P + b.R) \times {}^{\aleph} o(\Phi + b.R') + \sum_{\substack{(\alpha,\beta) \in \delta(P') \times {}^{\aleph}\delta(\Phi') \\ (a,\sigma) \in \nabla(\alpha,\beta,\Gamma(\Phi'))}} a.\left(\partial_a(P + b.R)\sigma \prod_E {}^{\aleph} \partial_\beta(\Phi + b.R')\sigma\right) \sim$$

$$0 \times 0 + \sum_{\substack{(\alpha,\beta) \in \delta(P) \cup \{b\} \times {}^{\aleph}\delta(\Phi) \cup \{b\} \\ (a,\sigma) \in \nabla(\alpha,\beta,\Gamma(\Phi + b.R'))}} a.((\partial_a(P)\sigma + \partial_a(b.R) \prod_E ({}^{\aleph} \partial_\beta(\Phi)\sigma + \partial_a(b.R')) \sim$$

$$\sum_{\substack{(\alpha,\beta)\in\delta(P)\cup\{b\}\times{}^{\aleph}\delta(\Phi)\cup\{b\}\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi+b.R'))}} a.((\partial_a(P)\sigma+\partial_a(b.R)\prod_E({}^{\aleph}\partial_\beta(\Phi)\sigma+\partial_a(b.R')) \quad \sim$$

$$\sum_{\substack{(\alpha,\beta)\in\delta(P)\times{}^{\aleph}\delta(\Phi)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi))}} a.(\partial_a(P)\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi)\sigma)+(b.R\prod_E b.R')$$

À partir de l'hypothèse d'induction VIII, on obtient :

$$o(P+b.R)\times{}^{\aleph}o(\Phi+b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P\prime)\times{}^{\aleph}\delta(\Phi\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi\prime))}} a.(\partial_a(P+b.R)\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi+b.R')\sigma)$$

$$\sim(P\prod_E\Phi)+(b.R\prod_E b.R') \qquad (IX)$$

D'une autre part, en utilisant les propriétés de $\prod_E$, nous avons :

$$(P+b.R)\prod_E(\Phi+b.R')\sim(P\prod_E\Phi)+(P\prod_E b.R')+(b.R\prod_E\Phi)+(b.R'\prod_E b.R')$$

$$\sim(P\prod_E\Phi)+(b.R'\prod_E b.R') \qquad (X)$$

Comme $P'=(P+b.R)\ et\ \Phi'=(\Phi+b.R')$, à partir de IX et X, on conclut que :

$$P'\prod_E\Phi'=o(P')\times{}^{\aleph}o(\Phi')+\sum_{\substack{(\alpha,\beta)\in\delta(P\prime)\times{}^{\aleph}\delta(\Phi\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi\prime))}} a.(\partial_\alpha(\mathrm{P}')\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi')\sigma) \quad avec\ |\nabla(\alpha,\beta,\Gamma(\Phi'))|$$

$$=n+1$$

Ce qui signifie que $H'(n+1)$ est vérifié.

- Cas des processus $P'=P*b.R\ \ et\ \ \Phi'=\Phi*b.R'$:

En utilisant les définitions des fonctions $o,\delta\ et\ \partial$, on peut développer l'expression suivante :

$$o(P*b.R)\times{}^{\aleph}o(\Phi*b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P\prime)\times{}^{\aleph}\delta(\Phi\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi\prime))}} a.(\partial_a(P*b.R)\sqcap_E{}^{\aleph}\partial_\beta(\Phi*b.R')) \quad \sim$$

$$0+\sum_{\substack{(\alpha,\beta)\in\delta(P)\cup\{b\}\times{}^{\aleph}\delta(\Phi)\cup\{b\}\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi*b.R'))}} a.((\partial_a(P)\sigma.P'+\partial_a(b.R))\prod_E({}^{\aleph}\partial_\beta(\Phi)\sigma.\Phi'+\partial_a(b.R')))\sim$$

$$\sum_{\substack{(\alpha,\beta)\in\delta(P)\times{}^{\aleph}\delta(\Phi)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi))}} a.(\partial_a(P)\sigma.P*b.R\prod_E{}^{\aleph}\partial_\beta(\Phi)\sigma.\Phi*b.R')+(b.R\prod_E b.R')$$

En utilisant le lemme.1, on obtient :

$$o(P*b.R)\times{}^{\aleph}o(\Phi*b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P*b.R)\times{}^{\aleph}\delta(\Phi*\mathrm{b.R}\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi*\mathrm{b.R}\prime))}} a.(\partial_a(P*b.R)\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi*b.R')\sigma)\sim$$

$$\left(\sum_{a\in\delta(P)}a.\partial_a(P)\sigma\right).P*b.R\prod_E\left(\sum_{a\in{}^{\aleph}\delta(\Phi)}a.{}^{\aleph}\partial_\beta(\Phi)\sigma\right).\Phi*b.R'+(b.R\prod_E b.R')$$

En utilisant la proposition.1, on aboutit à :

$$o(P*b.R)\times^{\aleph}o(\Phi*b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P*b.R)\times^{\aleph}\delta(\Phi*b.R\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi*b.R\prime))}}a.(\partial_a(P*b.R)\prod_E{}^{\aleph}\partial_\beta(\Phi*b.R'))\sim$$
$$(P.P*b.R\prod_E\Phi.\Phi*b.R')+(b.R\prod_E b.R') \qquad (XI)$$

D'une autre part à partir des propriétés de $\prod_E$, on a :

$$(P*b.R\prod_E\Phi*b.R')\sim(P.P*b.R+b.R)\prod_E(\Phi.\Phi*b.R'+b.R')$$
$$\sim(P.P*b.R\prod_E\Phi.\Phi*b.R')+(P.P*b.R\prod_E b.R')$$
$$+(b.R\prod_E\Phi.\Phi*b.R')+(b.R\prod_E b.R')$$
$$\sim(P.P*b.R\prod_E\Phi.\Phi*b.R')+(b.R\prod_E b.R') \qquad (XII)$$

Comme $P'=P*b.R\ et\ \Phi'=\Phi*b.R'$, à partir de XI et XII, on conclut que :

$$P'\prod_E\Phi'=o(P')\times^{\aleph}o(\Phi')+\sum_{\substack{(\alpha,\beta)\in\delta(P\prime)\times^{\aleph}\delta(\Phi\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi\prime))}}a.(\partial_\alpha(\mathrm{P}')\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi')\sigma)\quad avec\ |\nabla(\alpha,\beta,\Gamma(\Phi'))|$$
$$=n+1$$

Ce qui signifie que $H'(n+1)$ est vérifiée.

- Cas des processus $P'=P*b.R\ et\ \Phi'=\Phi+b.R'$:

  En utilisant les définitions des fonctions $o,\delta\ et\ \partial$, on peut développer l'expression suivante :

$$o(P*b.R)\times^{\aleph}o(\Phi+b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P\prime)\times^{\aleph}\delta(\Phi\prime)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi\prime))}}a.(\partial_a(P*b.R)\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi+b.R')\sigma)\quad\sim$$
$$0\times0+\sum_{\substack{(\alpha,\beta)\in\delta(P)\cup\{b\}\times^{\aleph}\delta(\Phi)\cup\{b\}\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi+b.R'))}}a.((\partial_a(P)\sigma.P'+\partial_a(b.R))\prod_E({}^{\aleph}\partial_\beta(\Phi)\sigma.+\partial_a(b.R')))\sim$$
$$\sum_{\substack{(\alpha,\beta)\in\delta(P)\times^{\aleph}\delta(\Phi)\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi))}}a.(\partial_a(P)\sigma.P*b.R\prod_E{}^{\aleph}\partial_\beta(\Phi)\sigma.\Phi)+(b.R\prod_E b.R')$$

En utilisant le lemme.1, on obtient :

$$o(P*b.R)\times^{\aleph}o(\Phi*b.R')+\sum_{\substack{(\alpha,\beta)\in\delta(P)\cup\{b\}\times^{\aleph}\delta(\Phi)\cup\{b\}\\(a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi+b.R'))}}a.(\partial_a(P*b.R)\sigma\prod_E{}^{\aleph}\partial_\beta(\Phi+b.R')\sigma)\sim$$
$$\left(\sum_{a\in\delta(P)}a.\partial_a(P)\right).P*b.R\prod_E\sum_{a\in{}^{\aleph}\delta(\Phi)}({}^{\aleph}\partial_\beta(\Phi))+(b.R\prod_E b.R')$$

En utilisant la proposition.1, on aboutit à :

$$o(P * b.R) \times {}^{\aleph}o(\Phi + b.R') + \sum_{\substack{(\alpha,\beta)\in\delta(P)\cup\{b\}\times{}^{\aleph}\delta(\Phi)\cup\{b\} \\ (a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi+b.R'))}} a.(\partial_a (P * b.R)\sigma\prod_E {}^{\aleph}\partial_\beta(\Phi + b.R')\sigma) \sim$$

$$(P.P * b.R\prod_E\Phi) + (b.R\prod_E b.R') \qquad (XIII)$$

D'une autre part à partir des propriétés de $\prod_E$, on a :

$$\begin{aligned}(P * b.R\prod_E\Phi + b.R') &\sim (P.P * b.R + b.R)\prod_E(\Phi + b.R') \\ &\sim (P.P * b.R\prod_E\Phi) + (P.P * b.R\prod_E b.R') \\ &\quad +(b.R\prod_E\Phi) + (b.R'\prod_E b.R') \\ &\sim (P.P * b.R\prod_E\Phi) + (b.R'\prod_E b.R') \qquad (XIV)\end{aligned}$$

Comme $P' = P * b.R \; et \; \Phi' = \Phi + b.R'$, à partir de XIII et XIV, on conclut que :

$$P'\prod_E\Phi' = o(P') \times {}^{\aleph}o(\Phi') + \sum_{\substack{(\alpha,\beta)\in\delta(P')\times{}^{\aleph}\delta(\Phi') \\ (a,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi'))}} a.(\partial_\alpha(\mathrm{P}')\sigma\prod_E {}^{\aleph}\partial_\beta(\Phi')\sigma) \quad avec \; |\nabla(\alpha,\beta,\Gamma(\Phi'))|$$

$$= n + 1$$

Ce qui signifie que $H'(n+1)$ est vérifié.

– ***Conclusion :*** Par le principe d'induction généralisé, nous avons montré que $H'(n)$ est vraie pour tout $n \geq 0$.

5.3. Algorithme

Algorithme qui calcule $P\prod_E\Phi$

---

1 :

$$E \leftarrow \{P\prod_E\Phi = o(P) \times o(\Phi) + \sum_{\substack{(\alpha,\beta)\in\delta(P)\times{}^{\aleph}\delta(\Phi) \\ (\mu,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi))}} \mu.(\partial_\alpha(\mathrm{P})\sigma\prod_E {}^{\aleph}\partial_\beta(\Phi)\sigma)\}$$

2 : **Tant qu'il** existe $P_i\prod_E\Phi_\mathrm{i}$ dans le coté droit d'une équation et qui n'apparaît dans le coté gauche de toutes les autres équations **Faire :**

3 :

$$E \leftarrow E \cup \{P_i\prod_E\Phi_\mathrm{i} = o(P_i) \times o(\Phi_\mathrm{i}) + \sum_{\substack{(\alpha,\beta)\in\delta(P_i)\times{}^{\aleph}\delta(\Phi_\mathrm{i}) \\ (\mu,\sigma)\in\nabla(\alpha,\beta,\Gamma(\Phi_\mathrm{i}))}} \mu.(\partial_\alpha(P_i)\sigma\prod_E {}^{\aleph}\partial_\beta(\Phi_\mathrm{i})\sigma)\}$$

Fin tant que ;

4 : Retourner la solution du système linéaire E

Fin.

5.4. Exemple

Soit P le programme suivant :

while $(\boldsymbol{true})$***Do***

if $(\boldsymbol{x} > 1)$ ***then*** $\boldsymbol{w}(\boldsymbol{x})$;

if $(\boldsymbol{x} < 5)$ ***then*** $\boldsymbol{r}(\boldsymbol{x})$;

while $(\boldsymbol{x} > 0)$

$\{\boldsymbol{s}(\boldsymbol{x})$;

$\boldsymbol{x} - -;\}$

EndDo

Et ϕ la politique de sécurité qui indique qu'on peut exécuter l'action qu'on désire, mais dès qu'on exécute $r(x)$, on ne pourra plus exécuter $s(x)$; donc ϕ est représenté par la formule logique suivante :

$$\phi = U\big(\neg r(x), r(x)\big).\neg s(x)$$

Le programme P peut être représenté par le processus suivant :

$$P = (\big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).P_1))^{\omega}$$

$$P_1 = \big((x < 5) \rhd r(x) + (x \geq 5) \rhd 1\big).P_2$$

$$P_2 = ((x > 0) \rhd s(x).x - -) * (x \leq 0) \rhd 1$$

Notons que : $o(P) = o(P_1) = o(P_2) = 0$ et

$$\partial_{\alpha}(\mathrm{P}) = \partial_{\alpha}(\big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).P_1)$$

$$= \partial_{\alpha}\big((x > 1) \rhd w(x)\big) + \partial_{\alpha}(x \leq 1) \rhd 1).P_1$$

$$\partial_{\alpha}(P_1) = \partial_{\alpha}\Big(\big((x < 5) \rhd r(x) + (x \geq 5) \rhd 1\big).P_2\Big)$$

$$= \partial_{\alpha}\big((x < 5) \rhd r(x)\big) + \partial_{\alpha}((x \geq 5) \rhd 1)).P_2$$

$$\partial_{\alpha}(P_2) = \partial_{\alpha}\Big(\big((x > 0) \rhd s(x).x - -\big) * (x \leq 0) \rhd 1\Big)$$

$$= \partial_\alpha\Big(\big((x > 0) \rhd s(x).x - -\big)\Big).P_2 + \ \partial_\alpha\big((x \leq 0) \rhd 1\big)$$

$$= \partial_\alpha\big((x > 0) \rhd s(x)\big).x - -.P_2 + \ \partial_\alpha\big((x \leq 0) \rhd 1\big)$$

Et

$$\delta(P) = \{(x > 1) \rhd w(x), (x \leq 1) \rhd 1\}$$

$$\delta(P_1) = \{(x < 5) \rhd r(x), (x \geq 5) \rhd 1\}$$

$$\delta(P_2) = \{(x > 0) \rhd s(x), (x \leq 0) \rhd 1\}$$

La politique de sécurité donnée par la formule logique peut être écrite comme suit :

$$\phi = U(\neg r(x), \phi_1)$$

$$\phi_1 = r(x).\phi_2$$

$$\phi_2 = \neg s(x)$$

Notons aussi que : $o(\phi) = o(\phi_1) = o(\phi_2) = 0$

$$^{\aleph}\partial_a(\phi) = {}^{\aleph}\partial_a(\neg r(x)).\phi \vee {}^{\aleph}\partial_a(\phi_1)$$

$$^{\aleph}\partial_a(\phi_1) = {}^{\aleph}\partial_a\big(r(x)\big).\phi_2$$

$$^{\aleph}\partial_a(\phi_1) = {}^{\aleph}\partial_a(\neg s(x))$$

Nous calculons $P\prod_E \phi$

$$P\prod_E \phi = X_1$$

$$X_1 = \big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).\ \partial_{r(x)}(P)\prod_E \partial_{r(x)}(\phi)$$

$$= \big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).P_1\prod_E \phi$$

$$= \big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).X_2$$

$$X_2 = (x < 5) \rhd r(x).\ \partial_{r(x)}(P_1)\prod_E \partial_{r(x)}(\phi) + \ (x \geq 5) \rhd 1.\ \partial_{r(x)}(P_1)\prod_E \partial_{r(x)}(\phi)$$

$$= (x < 5) \rhd r(x).P_2\prod_E \phi_2 + \ (x \geq 5) \rhd 1.P_2\prod_E \phi$$

$$= (x < 5) \rhd r(x).X_3 + \ (x \geq 5) \rhd 1.X_4$$

$$X_3 = (x \leq 0) \rhd 1.\ \partial_{s(x)}(P_2)\prod_E \partial_{s(x)}(\phi_2)$$

$$= (x \leq 0) \rhd 1.(1\prod_E \phi_2)$$

$$= (x \leq 0) \rhd 1$$

$$X_4 = (x \leq 0) \rhd 1.\ \partial_{r(x)}(P_2)\prod_E \partial_{r(x)}(\phi)$$

$$= (x > 0) \rhd 1.s(x).(x - -.P_2\prod_E \phi) + (x \leq 0) \rhd 1.(1\prod_E \phi)$$

$$= (x > 0) \rhd 1.s(x).(x - -.P_2\prod_E \phi) + \ (x \leq 0) \rhd 1$$

$$= (x > 0) \rhd 1.s(x).X_5 + (x \leq 0) \rhd 1$$

$$X_5 = x - -.(P_2 \sqcap_E \phi)$$

$$= x - -.X_4$$

Nous obtenons le système d'équation suivant :

$$X_1 = \big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).X_2$$

$$X_2 = (x < 5) \rhd r(x).X_3 + \ (x \geq 5) \rhd 1.X_4$$

$$X_3 = (x \leq 0) \rhd 1$$

$$X_4 = (x > 0) \rhd 1.s(x).X_5 + (x \leq 0) \rhd 1$$

$$X_5 = x - -.X_4$$

Donc :

$$X_1 = \big((x > 1) \rhd w(x) + (x \leq 1) \rhd 1\big).X_2$$

$$X_2 = (x < 5) \rhd r(x).X_3 + \ (x \geq 5) \rhd 1.X_4$$

$$X_4 = (x > 0) \rhd 1.s(x).x - -.X_4 + (x \leq 0) \rhd 1$$

$$= \big((x > 0) \rhd 1.s(x).x - -\big) * (x \leq 0) \rhd 1$$

Ce qui donne le programme suivant :

```
while (true)Do
    if (x > 1) then w(x);
    if (x < 5) then r(x);
    if (x >= 5)
        { while (x > 0)
            {s(x);
            x - -;}
        }
EndDo
```

6. Evaluation de la proposition

La tâche d'évaluation de notre approche reste une tâche laborieuse et qui n'est pas très formelle. Ceci est du à l'absence d'une norme standard pour la comparaison et la validation.

En ce qui nous concerne, pour évaluer notre approche nous la comparons avec les autres techniques en utilisant les critères de comparaison présentés dans le second chapitre.

Critères	Complexité Pour producteur	Complexité Pour consommateur	Rigidité	Disponibilité d'outils automatique	Sureté
	Complexe	complexe	souple	Non	Oui, si la politique est sure

Les codes exécutés	Fausses alertes et interventions	Accès aux valeurs	Nombre de messages échangés
Tous les codes	Non	oui	1

Etant donné que notre approche est directement dérivée de celle de BPA, il est normal qu'elle hérite de tous ses avantages, la différence entre les deux c'est leur méthode de comparaison entre les actions, BPA utilise l'équivalence par trace qui est trop stricte, par rapport à notre équivalence qui se base directement sur les modifications de l'action sur l'environnement. Ce qui pourrait nous faire penser à rajouter un autre critère qui serait « la base de comparaison des actions » ou notre approche serait meilleure que celle proposée par les auteurs de [19].

6.1. Implémentation

6.1.1. Présentation de l'environnement

Nous avons choisi d'implémenter notre prototype sur le système FEDORA vu sa stabilité, et pour le fait qu'il soit open source, ce qui pourrait être un avantage majeur pour

assurer une meilleure sécurité. Le Projet Fedora indique que sa mission est de mener la promotion des logiciels libres et open source et le contenu en tant que communauté de collaboration [55]. Le projet surveille les rejets du système d'exploitation sur un cycle de six mois pour intégrer les caractéristiques les plus récemment mises au point Ce système possède un compilateur C intégré, c'est le compilateur gcc, que nous utilisons pour compiler exécuter notre programme.

6.1.2. La grammaire

La grammaire implémentée est représentée par la grammaire BNF suivante :

P :: *instruction OMEGA/ condition COND instruction OMEGA/instruction*instruction OMEGA*

Instruction :: *a(var,var).inst/a(var,cte).inst/r(var).inst/s(var).inst/w(var).inst*

Var :: *a/b/c…/z*

Cte :: *1/2/…../100*

Condition :: *(var=var) ET condition/(var=cte)ET condition/ (var=var) OU condition/(var=cte) OU condition/(var>var) ET condition/(var>cte)ET condition/ (var>var) OU condition/(var>cte) OU condition/(var<var) ET condition/(var<cte)ET condition/ (var<var) OU condition/(var<cte) OU condition/epsilon*

Cette grammaire nous permet de représenter toutes les actions d'un programme, une condition, une boucle et des opérations simples.

6.1.3. Exemple

Notre application ne possède pas d'interface, elle affiche le résultat de l'algorithme sur le terminal de FEDORA directement comme ceci est montré par la figure suivante :

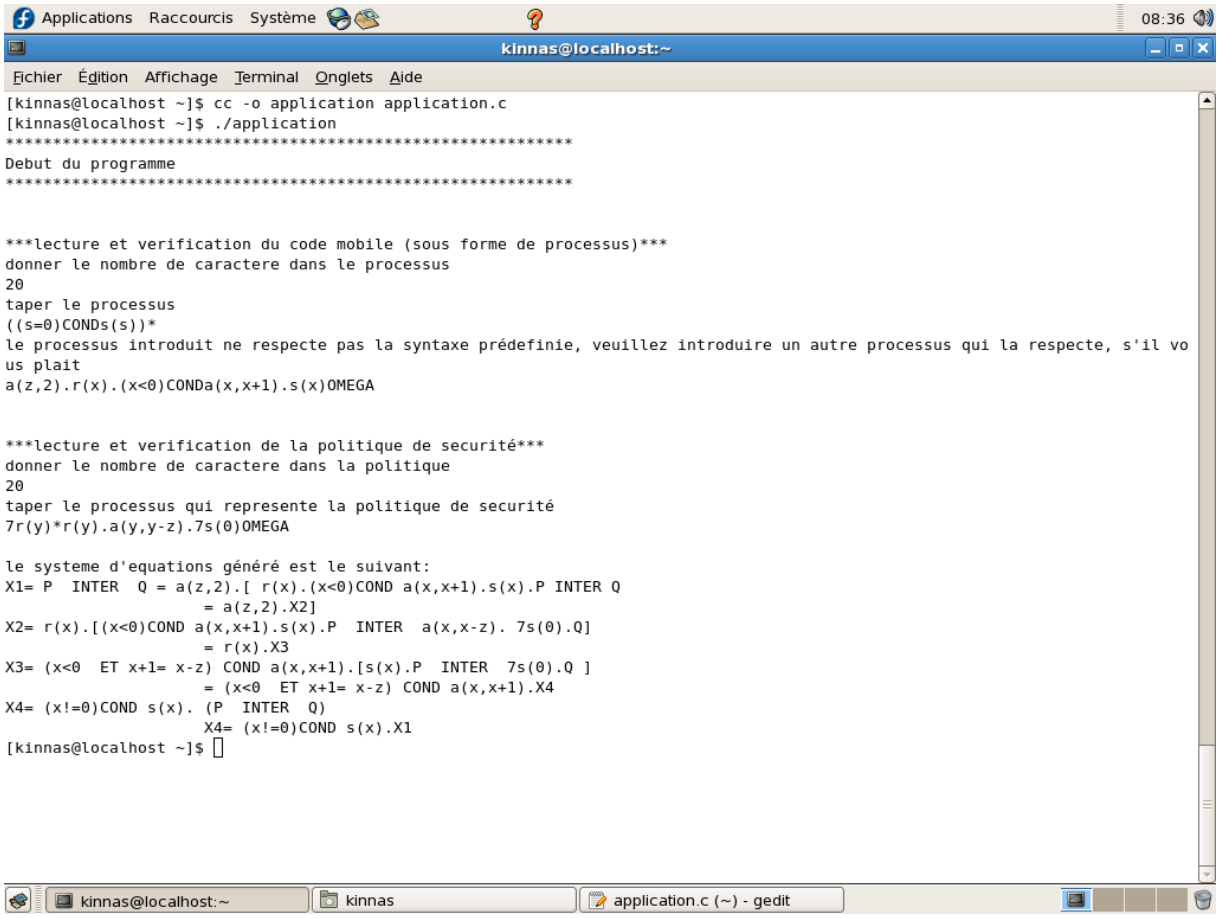

Figure : résultat de l'algorithme

Cet exemple présente deux cas de figure :

Le premier, présente le cas ou le processus introduit ne respecte pas la syntaxe de la grammaire, en effet le processus introduit est le suivant : *((s=0)CONDs(s)*)* ce processus possède une condition à l'intérieur d'une boucle ce qui n'est pas permis par la grammaire. C'est pour cela que le prototype affiche que le processus introduit ne respecte pas la grammaire.

Dans le second cas, on introduit le processus suivant :

P= a(z,2).r(x) .(x<0)COND a(x,x+1).s(x) OMEGA

Et la politique suivante :

*Φ=7r(y)*r(y).a(y,y-z).7s(0) OMEGA*

La version du prototype implémenté ne fait que générer un système d'équations et ne résout pas le système.

7. Conclusion

Dans ce travail nous avons d'abord introduit la notion d'environnement qui ne prend pas en considération l'effet de bord, puis l'unification modulo une théorie équationnelle. Ensuite, Nous avons défini l'Algèbre CBPAE et la logique $\mathcal{L}_a$ qui serviront respectivement à représenter les programmes et spécifier les politiques de sécurité. Puis, nous proposons une méthode d'optimisation et nous adaptons les formules de calcul aux nouvelles notions et nous les démontrons formellement. Et avant de finir, nous présentons le nouvel algorithme avec un exemple. Et nous terminons par une section d'évaluation.

L'enrichissement de l'algèbre est fait dans le but de pouvoir appliquer cette méthode de réécriture aux langages de programmation réels.

Conclusion Générale

Ce travail nous a permis de comprendre quelques notions importantes dans le domaine de la sécurité informatique en général, et la sécurité des codes mobiles en particulier. Nous nous sommes rendus compte que les gens investissent des moyens humains et financiers gigantesques dans ce domaine, pourtant tout reste encore à faire, car malgré tous ce qui a été fait, il n'existe toujours pas de moyen efficace contre toutes les attaques, l'ensemble des utilisateurs des systèmes informatiques, qui ne cesse de s'élargir, se retrouve dans l'obligation de faire des compromis entre la sureté et l'efficacité.

Nous avons étudié plusieurs méthodes qui sont utilisées pour le renforcement de la sécurité des codes mobiles, et nous avons pu apporter un apport personnel qui peut se résumer en comme suit :

- Une étude comparative entre les différentes techniques pour la protection des hôtes qui exécutent des codes mobiles, cette étude nous a permis de connaître les avantages et inconvénients de chaque méthode, ainsi nous pourrons choisir la méthode qui convient le mieux à n'importe quelle domaine d'application.
- Un autre apport réside dans le fait d'avoir démontré qu'on ne peut utiliser directement la bissimilarité dans la méthode algébrique étudiée dans le troisième chapitre, ainsi si nous voulons envisager d'utiliser cette équivalence, nous savons désormais qu'il faudrait changer l'algorithme qui génère le système d'équation.
- La définition d'une théorie équationnelle utilisant la notion d'environnement, désormais l'unification des actions se fera modulo cette théorie équationnelle.

- Proposition d'une astuce d'optimisation, ou on ne prend pas en considération les actions à effet nul sur les politiques de sécurité.
- Un autre apport est l'adaptation des nouvelles formules de calcul que nous avons validées à l'aide de démonstrations formelles.

La tâche d'évaluation de notre approche reste une tâche laborieuse et qui n'est pas très formelle. L'absence d'une norme standard pour la comparaison ne permet pas de tirer des conclusions fiables. Nous suggérons de concevoir un protocole de comparaison qui a pour but au moins d'imposer l'utilisation d'une base de codes mobiles malicieux pour avoir une comparaison plus objective. Ceci pourrait être une première perspective et une direction de recherche pour des travaux à venir.

Une autre perspective serait l'enrichissement de l'environnement. En effet l'environnement considéré dans ce travail n'est constitué que de la mémoire, alors, nous pouvons envisager de l'enrichir avec d'autres notions tel que l'état des ports ou l'état des canaux de communication par exemple, puis l'implémentation d'un bon prototype pour cette méthode.

Nous pourrions aussi penser à l'enrichissement de la méthode de réécriture basée sur les AEF avec la même notion de l'environnement, et d'implémenter les deux méthodes pour pouvoir comparer et trouver quelle est la meilleure technique, car avec notre tableau de comparaison, nous ne pouvons distinguer entre les deux méthodes de réécriture.

Bibliographies

[1] M overton: The journey, so far: trends, graphs and statistics. Dans proceedings of the virus Bulletin, conference, Vienne, Autriche, 2007

[2] S.Qinga et W Wen: A survey and trends on Internet worms, Institute of software, Chinese academy of sciences, pp.334-346, china, 2005.

[3] S. Staniford, V. Paxson et NWeaver: how to awn the Internet in your spare time. Dans proceedings of the 11th USENIX security symposium, pp.149- 167, Berkley, CA, USA, 2002

[4] Caida Project. Caida Analysis of code red, http://www.caida.org/research/security/code-red/.

[5] Explosion de la cybercriminalité en 2008, ZDnet Business et technologies, http://www.zdnet.fr/galerie-image/0,50018840,39501080,00.htm

[6] E.Maiwald: sécurité des réseaux. Campus press 2001

[7] Jacques DEMERJIAN ; "Services d'autorisation et Intégration au protocole d'attribution dynamique des adresses", thèse de doctorat à l'Ecole Nationale Supérieure française des Télécommunications Spécialité : Informatique et Réseaux, Dec 2004

[8] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM, 21(2):pp. 120 126, Fev 1978.

[9] JuHum Kwon, Chang-Joo Moon; Visual modeling and formal specification of constraints of RBAC, Knoledg- Based systems; article de science direct; pp. 321…328, 2006.

[10] James B.D. Joshi, Elisa Bertino, Usman Latif, and Arif ; Ghafoor ; "A Generalized Temporal Role-Based Access Control Model"; IEEE transactions on knowledge and data engineering, Jan 2005

[11] Jacques Wainera, Akhil Kumarb, Paulo Barthelmessc ; "DW-RBAC : A formal security model of delegation and revocation in workflow systems"; science direct ; nov 2005.

[12] H.N. Talantikite, S. Aissani , N. Boudjlida, D. Aissani, and N. Meddour. Contrôle d'accès aux services Web complexes. Actes de la 1ère conférence internationale sur les

Systèmes d'Informations et Intelligence Economique, SIIE'2008,IHE éditions, ISBN 9978-9973-868-19-0, pp. 517-532, Hammamet, Tunisie, Fév 2008

[13] docs.oasis-open.org/xacml/2.0/accesscontrol-xacml-2.0-core-spec-os.pdf.

[14] Simon Godik,Tim Moses ; "eXtensible Access Control Markup Language (XACML) Version 1.0", OASIS Standard, Fev 2003, www.oasisopen.org/committees/xacml/repository/

[15] R. Milner. A calculus on communicating systems. Lecture Notes in computer science, 1980.

[16] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. pp. 82–95.

[17] J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report W 206, Mathematisch Centrum, Amsterdam, 1982.

[18] L.Hamza et K. Adi : Formal technique for discovering complex attacks in computer systems. International Conference on New Software Methodologies Tools and techniques, Italie, 2007.

[19] Mohammad MEJRI et All, Enforcing Security Policies Using Algebraic Approach, 2009

[20] Ould slimane H, Mohammad MEJRI, Kamel ADI "Enforcing security policies by rewriting programs using automata" Proceeding of the conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the fifth SoMeT_06, pp. 195-207.2006

[21] J von newman et O Morgenstern: Theory of games and economic behavior. Princeton University, 1944

[22] S Razika : application de la théorie des jeux dans l'organisation industrielle, mémoire de magister, université de Bejaia 2009.

[23] Santana Torrellas, G. A network security architectural approach for systems integrity using multiagent systems engineering. In International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN).2004

[24] A.Mahimkar et V. Shmatikov :Game based analysis of denial of service prevention protocols. In 18th IEEE computer security Foundations Workshop (CSFW),pp.287-301, Aix-en-Provence, France 2005

[25] Axel Bürkle · Alice Hertel et all , « Evaluating the security of mobile agent platform» Auton Agent Multi-Agent Syst Springer Science+Business Media, pp. 295–311 ; 2009

[26] Mihir Bellare ET Bennet Yee. Forward integrity for secure audit logs. Rapport technique, UC at San Diego, Dept. of Computer Science and Engineering, Nov 1997.

[27] : G. Karjoth, N. Asokan ET C. Gulcu. Protecting the computation results of free roaming agents. In Kurt Rothermel and Fritz Hohl, editors, Proc. of the Second International Workshop, Mobile Agents 98, pp. 195-207, 1998.

[28] : Sergio Loureiro, Refik Molva, and Alain Pannetrat. Secure data collection with updates. In Proceedings of the Workshop on Agents on Electronic Commerce - First Asia Pacific Conference on Inteligent Agent Technology, Hong-Kong, Dec 1999.

[29]: A. Young ET Moti Yung. Sliding encryption : a cryptographic tool for mobile agents. Dans Proc. 4th International workshop fast software encryption 97. Springer-Verlag Lecture Notes in Computer Science No. 1267.1997

[30] M.Warnier, M.A. Oey, R.J. Timmer, B.J. Overeinder, and F.M.T. Brazier: `Enforcing Integrity of Agent Migration Paths by Distribution of Trust', Int. J. of Intelligent Information and Database Systems, Vol. 3, No. 4, pp. 382 - 396

[31] : Yaron Minsky, Robbert van Renesse, Fred B. Schneider, ET Scott D. Stoller. Cryptographic support for fault-tolerant distributed computing. In Proceedings of the Seventh ACM SIGOPS European Workshop, pp.109-114, Connemara, Ireland, Sept 1996.

[32] : Bennet Yee. A sanctuary for mobile agents. Technical Report CS97-537, UC at San Diego, Dept. of Computer Science and Engineering, avr 1997.

[33] : Tomas Sander and Christian Tschudin. Towards mobile cryptography. In Proceeding of the 1998 IEEE Symposium on Security and Privacy, Oakland, California, Mai 1998.

[34] James Gosling, Bill Joy, and Guy Steele. The Java Language Specification. Addison-Wesley, 1996.

[35] Li Gong,"Secure Java class loading," IEEE Internet Computing, pp 56-61, 1998.

[36] M. Hauswirth, C. Kerer, and R. Kurmanowytsch, "A secure execution framework for Java," In Proceedings of the 7th ACM conference on computer and communications security (CCS 2000), pp. 43-52, Athens, Greece, Nov. 2000.

[37] U. Topaloglu , C. Bayrak, Secure mobile agent execution in virtual environment, Springer Auton Agent Multi-Agent Syst 16:pp. 1–12. 2008

[38] Richard Marsden, Microsoft Authenticode for the Small Independent Software Vendor Foundations in Information Assurance, Graduate School of Management, University of Dallas.2006

[39] N. Dragoni et All, Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code, EuroPKI 2007, LNCS 4582, pp. 297–312, 2007

[40] George C. Necula. Proof-carrying code. In Proceedings of the 24th ACM Symposium on Principles of Programming Languages, Paris, France, January 1997.

[41] George C. Necula and Peter Lee. Safe, Untrusted Agents using Proof-Carrying Code. Lecture Notes in Computer Science N. 1419. Springer-Verlag, 1998.

[42] Appel, A.W. and Felty, A.P. A semantic model of types and machine instruction for proof-carrying code. In Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, pp. 243–253.2000

[43] Bernard, A. and Lee, P. Temporal logic for proof-carrying code. In Proceedings of the 18th International Conference on Automated Deduction, Vol. 2392 of Lecture Notes in Computer Science, Springer-Verlag, pp.31–46.2002

[44] Entrust Certificate Services, "The code signing process for Microsoft Authenticode" Authenticode Signing , User Guide Document issue: 2.0, Oct 2009,

[45] YASUYUKI TSUKADA ,Interactive and Probabilistic Proof of Mobile Code Safety∗, Automated Software Engineering, Springer Science + Business Media, Inc. pp. 237–257, Netherlands 2005.

[46] Jeremy Condit et All, Unifying Type Checking and Property Checking for Low-Level Code, Savannah, Georgia, USA. ACM 978-1-60558-379-2/09/01; 2009.

[47] D. Grossman and J. G Morrisett. Scalable certification for typed assembly language. In selected papers from the third International Workshop on types in compilation, springer-Verlag, pp 117-146, 2001

[48] J. G Morrisett , D. Walker, K. Crary and N.Glew. From system f to typed assembly language. ACM trans. Program Lang. Syst.21(3) :pp. 527-568, 1999

[50] F. Schneider and U. Erlingsson. SASI enforcement of security policies: A retrospective.2003

[51] V.Monfort, S.Goudeau; "web services et interopérabilité des SI", édition DUNOD Paris 2004.

[52] Y. Bertot and P. Castéran. Interactive Theorem Proving and Program Development, Coq'Art :the Calculus of Inductive Constructions. Springer-Verlag, 2004.

[53] S.C. Kleene. Representation of events in nerve nets and finite automata. In Automata Studies, Princeton University Press, pp 3–41 1956.

[54] J. A. Brzozowski. Derivatives of regular expressions. J. ACM, 11(4):pp. 481–494, 1964.

[55] http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci1508198,00.html

Résumé

De nos jours, l'informatique et la sécurité sont indissociables et sont à la confluence de diverses disciplines scientifiques. Ce projet se classe dans ce domaine, son principal objectif est l'élaboration d'un cadre formel permettant le renforcement automatique d'un programme par une politique de sécurité. Plus précisément, étant donné un programme P et une politique de sécurité Φ, nous avons élaboré une technique permettant de dériver un programme P' à partir de P et de Φ de telle manière que P' satisfait la politique Φ et reste correct par rapport à P.

Mots Clés : Algèbre de processus, Logique, Politique de sécurité, renforcement automatique, code mobile.

Abstract

Nowadays, information technology and security are inseparable and are at the confluence of various scientific disciplines. This project falls into this area, its main objective is to develop a formal framework for building an automatic program by a security policy. More precisely, given a program P and a security policy Φ, we developed a technique to derive a program P' from P and such that P' satisfies the policy Φ and remains correct with respect to P.

Keywords: Process algebra, logic, security policy, automatic building, mobile code.