Apport du suivi de flux d'information pour la sécurité des systèmes

Valérie Viet Triem Tong

▶ **To cite this version:**

Valérie Viet Triem Tong. Apport du suivi de flux d'information pour la sécurité des systèmes. Cryptographie et sécurité [cs.CR]. Université Rennes 1, 2015. <tel-01342243>

HAL Id: tel-01342243

https://hal.inria.fr/tel-01342243

Submitted on 5 Jul 2016

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

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

ANNÉE 2015

HDR / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour l'obtention de
L'HABILITATION A DIRIGER DES RECHERCHES

Mention : Informatique
présentée par

Valérie Viet Triem Tong

préparée à l'unité de recherche
EPC CIDRE
INRIA - Université de Rennes 1 - CNRS
CentraleSupelec

Apport du suivi de flux d'information pour la sécurité des systèmes

HDR soutenue à Rennes
le 3 décembre 2015

devant le jury composé de :

Christele Faure
Directeur Scientifique à SafeRiver / *Rapporteur*
Jean Goubault-Larrecq
Professeur à l'ENS Cachan / *Rapporteur*
Radu State
Chercheur à l'université du Luxembourg / *Rapporteur*
Mathieu Jaume
Maitre de Conférence à l'université Paris 6 / *Examinateur*
Maryline Laurent
Professeur à Telecom Sud Paris / *Examinatrice*
Ludovic Mé
Professeur à CentraleSupelec / *Examinateur*
Olivier Ridoux
Professeur à l'université de Rennes 1 / *Examinateur*

Remerciements

Ce document commence par ce que j'ai écrit en dernier, mais ce document n'aurait jamais débuté sans l'aide de toutes les personnes à qui je souhaiterais dire : « Merci ».

Tout d'abord, je souhaiterais remercier les professeurs qui ont marqué mon esprit d'étudiante. En particulier, merci à René Cori et Gilles Dowek qui m'ont donné envie de devenir, un jour, un aussi bon professeur qu'eux. J'y travaille.

Ensuite, j'aimerais remercier Thomas Genet pour m'avoir accompagnée dans le monde de la recherche.

Je tiens aussi à remercier ici Ludovic Mé de m'avoir accueillie dans l'équipe SSIR devenue CIDRE.

Merci à tous les étudiants qui m'ont accordé leur confiance pour l'encadrement de leur propre travail. Merci à Rado, Stéphane, Thomas, Laurent, Adrien A., Adrien B., Nicolas, Mourad, Jean François, Guillaume, Eric, Frédéric, Mathieu pour leur participation à ces travaux.

Merci à Mathieu, Eric, Christophe, Ludo, Philippe et Sylvie d'avoir relu ce document.
Enfin je tiens tout particulièrement à remercier tous les membres de l'équipe : pour la science évidemment, mais aussi pour les blagues jamais renouvelées du lundi matin, lundi midi, lundi 15h, mardi matin, ...

Merci à mes parents de m'avoir donner l'opportunité de développer ma curiosité.
Merci à Philippe de m'accepter chaque jour.
Merci à Alexandre et Emilie d'être mes enfants, tout simplement.

Table des matières

1 Curriculum vitæ **3**

2 Activités d'enseignement **7**

3 Activités scientifiques **15**

I Contributions scientifiques **31**

4 Causes de la dissémination de l'information, remèdes possibles **33**

4.1 De la nature des flux d'information . . . 33
4.2 Flux d'information possibles $\rightarrow$ . . . 37
4.2.1 Propriétés de $\rightarrow$ . . . 38
4.3 Flux d'information autorisés $\rightarrow_{/\mathbb{A}}$, politiques de flux . . . 38
4.4 Treillis de classes de sécurité . . . 41
4.5 Mise en œuvre des politiques de flux . . . 41
4.5.1 Travaux de Bell et La Padula . . . 42
4.5.2 Les travaux de Biba . . . 43
4.5.3 Les travaux de D. et P. Denning . . . 43
4.5.4 Les travaux de Myers et Liskov . . . 44
4.5.5 Les travaux menés dans l'équipe CIDRE . . . 45
4.6 Bilan . . . 45

5 Modèle et mise en œuvre d'un moniteur de flux d'information **47**

5.1 Caractérisation de l'information . . . 47
5.2 Politiques de flux mises en œuvre dans Blare . . . 49
5.3 Mise en œuvre . . . 50
5.3.1 Suivre l'information . . . 51
5.3.2 Projeter la politique et la mettre à jour . . . 52
5.3.3 Légalité des flux d'information . . . 52
5.4 Implémentation . . . 53
5.5 Bilan . . . 55

6 Construction et utilisation de politiques de flux d'information à grain fin 57
6.1 Construction de politiques de flux . . . 58
6.1.1 Qui est responsable de la définition d'une politique ? . . . 58
6.1.2 Le point de vue des données ou celui des conteneurs ? . . . 59
6.1.3 Composition de politiques . . . 60
6.2 *Blare Security Policy Language* . . . 62
6.2.1 Implémentations existantes . . . 63
6.3 Utilisations possibles . . . 63
6.4 Automatiser la construction des politiques . . . 65
6.4.1 Bilan des politiques BSPL . . . 68
6.5 Quelques expérimentations . . . 70
6.6 Bilan . . . 71

7 Comprendre et caractériser le comportement des applications 73
7.1 Représenter les flux d'information . . . 73
7.1.1 Opérations sur les Graphes de Flux Système . . . 74
7.1.2 Construction des SFGs . . . 76
7.2 Comprendre une attaque . . . 76
7.3 Extraction de signatures comportementales . . . 79
7.4 Evaluation de l'approche . . . 80
7.4.1 Classification de comportements malveillants . . . 80
7.4.2 Pertinence des signatures comportementales . . . 83
7.5 Limites actuelles de l'approche . . . 85
7.6 Bilan . . . 86

8 Pour conclure 87
8.1 Bilan . . . 87
8.2 Etendre l'approche à d'autres niveaux d'observations . . . 88
8.3 Complétudes des approches - validation des implémentations . . . 88
8.4 Complétude des comportements étudiés . . . 89
8.5 Obfuscation comportementales . . . 90
8.6 Affiner l'observation des tags . . . 90
8.7 Déclassifier l'information . . . 90

Bibliographie 96

Chapitre 1

Curriculum vitæ

Valérie Viet Triem Tong,
née le 13 novembre 1977 à Clichy-la-Garenne dans les Hauts-de-Seine,
de nationalité française.

Coordonnées Professionnelles

✉ CentraleSupelec, avenue de la Boulaie, CS 47601, 35576 Cesson-Sévigné Cedex.
☎ 02.99.84.45.00.
@ valerie.viettriemtong@centralesupelec.fr

Parcours professionnel

DEPUIS OCTOBRE 2006

Professeur associé à CentraleSupelec [1]

Je fais partie de l'équipe CIDRE CNRS/INRIA/Univ. de Rennes 1/CentraleSupelec. Mes travaux de recherche s'intéressent à la sécurité des systèmes et plus particulièrement :
– la protection de l'information ;
– la détection d'intrusion ;
– la caractérisation de maliciels ;
– la sécurité dans les systèmes distribués.

1. SUPELEC est devenue CentraleSupelec le premier janvier 2015

Je suis responsable des projets logiciels des étudiants de 2e année depuis 2009. Je suis aussi responsable du parcours sécurité du Master Recherche en Informatique (MRI) depuis 2011. Ce master est délivré par l'université de Rennes 1 et co-habilité par les principaux établissements d'enseignement supérieur en informatique de Bretagne dont CentraleSupelec.
Depuis novembre 2012, j'exerce mon activité à 80%.

DE FÉVRIER À SEPTEMBRE 2006

Ingénieur de Recherche / Post-doctorante chez Orange (site de Caen)

Etude et proposition d'un cryptosystème biométrique : génération de clés cryptographiques à partir de données biométriques.

DE SEPTEMBRE 2004 À FÉVRIER 2006

Maitre de Conférence à l'université de Rouen

Enseignements en cours magistraux, travaux dirigés, travaux pratiques de la première année de licence à la dernière année de master. Essentiellement programmation, calcul formel, cryptographie.
Responsable du *Master 2 Sécurité des Systèmes Informatiques* (1 an)

DE SEPTEMBRE 2003 À AOUT 2004

Attachée temporaire à l'enseignement et à la recherche à l'université de Rennes 1

Enseignements en travaux dirigés, travaux pratiques de la première année de licence à la première année de master. Essentiellement programmation orientée objet, cryptographie, statistiques, langages web à l'université de Rennes 1 et programmation à l'Ecole Spéciale Militaire de Saint-Cyr Coëtquidan (ESM Saint-Cyr).

DE SEPTEMBRE 2003 À AOUT 2004

Vacataire de l'enseignement supérieur à l'université de Rennes 1

Durant mes trois années de thèse, j'ai enseigné à l'université de Rennes 1 et à l'Institut National des Sciences Appliquées (INSA) de Rennes. Le volume de ces enseignements représentait environ cent soixante heures sur trois ans, principalement programmation fonctionnelle, programmation orientée objet, algorithmique et complexité, compilation.

2000 À 2002

Enseignante en mathématiques

Employée de la société ACADOMIA.

de septembre 1998 à juillet 1999

Tuteur de mathématiques
Tuteur pour les étudiants de DEUG MASS et MIAS à l'université Paris 7.

Formation

Doctorat mention Informatique de l'université de Rennes 1

Automates d'arbres et réécriture pour l'étude de problèmes d'accessibilité
préparé dans le projet Lande de l'IRISA, soutenu publiquement le 19 décembre 2003 devant le jury composé de :

Jean-Pierre Banâtre alors Professeur à l'univ. de Rennes 1, président du jury
Jean Goubault Larrecq alors Professeur à l'ENS Cachan, rapporteur
Pierre Réty alors Maitre de Conférence (HDR) à l'univ. d'Orléans, rapporteur
Thomas Jensen alors Directeur de Recherche au CNRS, directeur de thèse
Thomas Genet alors Maitre de Conférence à l'univ. de Rennes 1, encadrant
Francis Klay alors Ingénieur de Recherche à France Telecom R&D, examinateur.

obtenu avec la mention très honorable.

Diplôme d'Études Approfondies (DEA)
Programmation Sémantique Preuve & Langages obtenu en 2000 à l'université Paris 7 avec la mention Assez Bien

Maitrise de Mathématiques
orientation *logique mathématique* obtenue en 1999 à l'université Paris 7

Licence de Mathématiques
obtenue en 1998 à l'université Paris 7 avec la mention Bien

DEUG Mathématiques, informatique et applications aux sciences
obtenu en 1997 à l'université Paris 7 avec la mention Assez Bien

Baccalauréat série S
orientation *mathématiques* obtenu en 1995 avec la mention Assez Bien

---

Chapitre 2

Activités d'enseignement

J'ai commencé à enseigner en 1999, lorsque j'étais en maîtrise de mathématiques, à la fois comme enseignante de mathématiques employée par la société Acadomia et comme tuteur de mathématiques à l'université Paris 7. Durant ma thèse de septembre 2000 à octobre 2003, j'étais vacataire de l'enseignement supérieur à l'université de Rennes 1. J'ai effectué durant ces trois années cent soixante heures de travaux dirigés et pratiques. J'ai ensuite été *attachée temporaire à l'enseignement et à la recherche* (ATER) à temps plein durant l'année universitaire 2003-2004. J'ai été recrutée comme maître de conférence à l'université de Rouen en octobre 2004. A Rouen, en 2006, j'ai pris pour un semestre la responsabilité du Master Sécurité des Systèmes Informatiques (SSI). Enfin, depuis septembre 2006, je suis professeur associé à CentraleSupelec où j'enseigne aux étudiants des trois années d'études de l'école.

J'ai choisi dans ce document de présenter les enseignements que j'ai donnés en les regroupant par thèmes, en indiquant le volume horaire lorsque j'ai pu retrouver cette donnée. J'ai choisi de détailler uniquement les cours, travaux dirigés ou travaux pratiques qui ont été pour moi les plus marquants parce que j'ai construit leur programme ou leur support.

Enseignements non directement liés à l'informatique

Statistiques

[2003] [IUP Miage à l'université de Rennes 1] [ATER]
Responsable des travaux dirigés et des travaux pratiques. J'ai créé la moitié des sujets de travaux dirigés et l'ensemble des sujets de travaux pratiques pour les étudiants de 2e année d'IUP (Bac+4).

Théorie des jeux

[2009 à 2014] [2A SUPELEC] [Professeur associé à SUPELEC]
Je suis responsable depuis 2009 d'un cours présentant la théorie des jeux. J'ai entière-

ment créé cet enseignement qui comprend dix-huit heures de cours magistral ponctuées d'exercices. Il est construit en deux parties. La première partie présente une classification des jeux et des résultats théoriques associés. Le cours débute par la présentation des jeux statiques en information complète. Après avoir défini ce type de jeux et présenté quelques exemples et notamment le dilemme du prisonnier, je présente comment la théorie des jeux étudie ces jeux : les stratégies dominantes, les équilibres de Nash, le théorème de Nash, les équilibres en stratégies mixtes, les optima de Pareto. Ensuite les étudiants s'exercent sur quelques exemples. Je complexifie peu à peu le type des jeux étudiés : les jeux dynamiques, en information complète, incomplète, les jeux Bayesiens, les jeux répétés un nombre fini ou infini de fois. Pour chaque type de jeux, je répète le même plan de présentation : les jeux, les outils, les méthodes de résolution associées, les résultats théoriques, ainsi que des exemples. A chaque fois, les étudiants font des exercices d'applications directes ce qui leur permet de s'approprier pleinement les notions présentées.
Sophie Pinchinat, professeur à l'université de Rennes 1 clôt cette première partie en présentant les jeux infinis.
La seconde partie du cours présente des utilisations de la théorie des jeux comme les mécanismes d'enchères, la preuve de propriétés en sécurité informatique.
Ce cours est évalué grâce à la présentation d'articles de recherche par les étudiants. Je leur demande toujours de présenter une version épurée du papier qu'ils ont à étudier en privilégiant l'intuition du travail.

Informatique fondamentale

[2011-2014] [3A SUPELEC] [CM][Professeur associé à SUPELEC]
J'ai construit un cours d'informatique fondamentale à destination des étudiants de dernière année avec l'aide de Nicolas Prigent et Frédéric Tronel. Ce cours est optionnel, il est appelé cours de *mineure* en complément des cours de la *majeure* dont le thème est la sécurité informatique. Ce cours est ouvert à tous les étudiants de dernière année du campus (informatique, électronique et automatique). En pratique, il est essentiellement suivi par les informaticiens.
A travers cette série de sept séances de trois heures dont une séance de travaux pratiques, nous nous posons la question de ce qu'il est possible de calculer, de décider avec une machine. Nous nous demandons aussi s'il existe un modèle,une approche, un paradis dans lequel nous pourrions calculer toutes les fonctions que nous imaginons.

Calculer avec une machine de Turing Dans ce cours je présente une première caractérisation des fonctions calculables en étudiant les machines de Turing. Les fonctions calculables seraient donc celles que l'on peut calculer avec une machine de Turing. J'explique le *Busy Beaver game* ou jeu du castor affairé et présente ainsi une fonction que l'on peut comprendre et spécifier et qui malheureusement n'est pas calculable. Le cours se termine par le problème de l'arrêt.

Calculer avec le λ-calcul Après avoir présenté la thèse de Church, j'essaye une autre approche pour mieux cerner ce que sont les fonctions calculables en étudiant le λ-calcul. Je présente la mécanique classique du λ-calcul : les variables libres, liées, l'α-conversion. Je présente un pas de calcul : la β-réduction, puis des pas de calculs. Viennent ensuite les résultats classiques sur les formes normales et la confluence. Le cours s'intéresse ensuite à montrer ce que l'on peut calculer avec le λ-calcul : les entiers de Church, le codage des entiers, des booléens, de la fonction `if-then-else`. Je montre les différentes stratégies de réduction en faisant le lien avec le travail d'un compilateur et les choix faits dans les différents langages de programmation que les étudiants connaissent. Je montre que l'on peut essayer de remédier aux termes qui n'ont pas de forme normale en utilisant le typage. Je ne montre que le λ-calcul simplement typé et explique que les termes qui ont un type dans ce calcul ont nécessairement une forme normale. Hélas, je montre aussi qu'on a perdu un terme essentiel Y qui n'est pas typable dans ce calcul et qui pourtant est bien pratique pour faire une exponentiation par exemple.

Un pas vers la théorie des modèles Je présente un peu de théorie des modèles et tâche de faire comprendre aux étudiants les résultats de complétude et incomplétude de Gödel.

Et si tout n'était qu'algorithmes ? Cette mineure se termine par une intervention de Gilles Dowek directeur de recherches chez INRIA qui présente aux étudiants le lien entre preuve et programme. Ce dernier cours est ouvert aux étudiants L3 de l'ENS Rennes.

Algorithmique et complexité

[2001-2002] [Maitrise d'informatique] [50h de TP][doctorante, vacataire de l'université de Rennes 1] Ces TP proposaient d'étudier la complexité de différents algorithmes de manipulations des grands nombres. J'ai mis en place les sujets de TP et assuré leur encadrement.

Calcul symbolique

[2004-2006] [DEUG MIAS 2e année] [48h de TD et 36 h de TP][MCF à l'université de Rouen] Ces séances proposaient de découvrir les algorithmes permettant de manipuler et calculer avec des grands nombres, des polynômes, des matrices. Le langage support de ce cours était le langage OCaml.

Architecture des systèmes informatiques

[2013-2014] [L1 INFO] [22h de TP] Ce cours est un cours de découverte les grands principes de l'informatique pour des non-spécialistes. Une première partie voit le bit comme unité de stockage et va jusqu'au principe de fonctionnement d'un disque dur. La seconde partie voit le bit comme unité d'information et présente un peu de théorie

de l'information. L'objet des TP est de faire de l'expérimentation. Tout d'abord sur le fonctionnement des systèmes de fichiers, puis sur un peu de théorie de l'information (distribution, quantité d'information, entropie etc) enfin il y a une partie de recherche documentaire sur une "objet informatique" grand public comme les dispositifs de stockage ou d'impression. J'assure les séances de travaux pratiques pour un groupe (environ une vingtaine d'étudiants).

Apprentissage de la programmation

Programmation impérative

[2004-2006] [DEUG MIAS 1ère année] [22h de TD et 30 h de TP][MCF à l'université de Rouen] Découverte de la programmation, présentation des structures de données (tableaux essentiellement) et des structures de contrôle. Le langage support était Pascal.
[2009-2014] [1A SUPELEC] [TD-TP][Professeur associé à SUPELEC]
Bases de la programmation impérative en utilisant le langage Java. Présentation des structures de données, des structures de contrôle. Le nombre d'heures de travaux pratiques encadrés varie selon les années et représente une quinzaine d'heures.

Programmation fonctionnelle

[2000-2002] [DEUG 1 SM et STPI] [30h de TD et environ 50h de TP][vacataire à l'université de Rennes 1]
Bases de la programmation fonctionnelle en utilisant Mathématica.

[2003-2004] [ESM Saint-Cyr Coëtquidan] [TP][ATER à l'université de Rennes1]
Bases de la programmation fonctionnelle en utilisant Mathématica.

[2004-2006] [DEUG 2] [CM][MCF à l'université de Rouen]
J'ai construit un cours magistral de programmation fonctionnelle à destination des étudiants de 2e année de DEUG MIAS à l'université de Rouen. J'ai aussi participé à la construction des travaux dirigés et pratiques. Ce cours présentait les structures de données, les types construits, les fonctions récursives. Durant ce module les étudiants n'avaient pas le droit d'écrire une boucle `for` ou une boucle `while`, ce qui les obligeait à n'utiliser que des fonctions récursives. Pour les étudiants, ce cours venait après une année d'étude durant laquelle ils avaient étudié le langage Pascal et la programmation impérative.

Programmation orientée objet

[2003-2004] [DESS CCI] [TP][ATER à l'université de Rennes 1]
Familiarisation avec les principes de la programmation orientée objet avec le langage

Eiffel dans l'environnement Studio pour des étudiants en fin d'études de divers horizons souhaitant acquérir une bonne culture informatique.

[2009-2014] [1A SUPELEC] [TD-TP][Professeur associé à SUPELEC]
Bases de la programmation orientée objet en utilisant le langage Java. Etude des structures de données arborescentes. Ce qui représente jusqu'à douze heures de travaux pratiques.

[2 ans entre 2012 et 2014] [2A étrangers SUPELEC] [TD-TP][Professeur associé à SUPELEC]
Bases de la programmation impérative puis objet en utilisant le langage Java. Présentation des structures de données, structures de contrôle, objet. Ce cours est une remise à niveau en informatique destinée aux étudiants étrangers intégrant Supelec en 2e année.

[2013-2014] [FC enseignants de classes préparatoires][CM-TP][Professeur associé à SUPELEC]
Cette formation présente les bases de la programmation objet en utilisant le langage Python. Cette formation dure trois jours, elle a été jouée trois fois à ce jour. J'ai assuré deux fois la première journée qui présente le langage et les concepts de la programmation orientée objet. J'ai monté le support de la seconde journée (travaux pratiques) et ai assuré son enseignement trois fois.

Securité informatique

Cryptographie

[2004-2006] [M2 SSI][12h TD 18h TP][MCF à l'université de Rouen]
Principaux algorithmes de chiffrement, chaine de confiance, boite à outils cryptographiques.

[2004-2006] [MIUP2 et M2GMI][2×16h TP][MCF à l'université de Rouen]
Travaux pratiques sur l'utilisation des outils cryptographiques et la stéganographie.

[2008-2013] [3A SUPELEC et M2 MRI][4h30 CM][Professeur associé à SUPELEC]
Vérification de protocoles cryptographiques

[2011-2014] [3A SUPELEC][3h TP][Professeur associé à SUPELEC]
Approche pratique de la cryptanalyse.

[2011-2014] [M2 SSI et M2 IR][2×6h TP][Professeur associé à SUPELEC]
Approche pratique de la cryptanalyse.

Virus

[2008-2010] [Master 2 MRI][3h CM][Professeur associé à SUPELEC]
Introduction à la virologie et aux techniques anti-virales.

Conception d'applications sécurisées

[2010-2014] [3A SUPELEC][4h30 CM et 6h TP][Professeur associé à SUPELEC]
Je présente une série de cours et travaux pratiques sur les méthodes et outils qui permettent de concevoir des logiciels sûrs. Je présente tout d'abord la méthode B, les assistants à la preuve Coq et Isabelle. Depuis janvier 2015, je propose aussi un challenge de programmation aux étudiants. Il s'agit pour eux d'être capables de programmer une simple fonction en Java manipulant deux listes. En pratique, les étudiants ont du mal à produire un code qui réponde réellement à la spécification et sont alors conscients de l'apport que peuvent avoir les méthodes formelles dans le développement logiciel. Ce challenge est repris d'un sujet de Thomas Genet, maitre de conférence à l'université de Rennes 1.

Détection d'intrusion

[2013-2014] [Master 2 MRI][3h CM][Professeur associé à SUPELEC]
Je présente aux étudiants de master Recherche en Informatique pourquoi les méthodes traditionnelles de protection des données reposant sur le contrôle d'accès sont insuffisantes. Après avoir expliqué une petite attaque par délégation, les étudiants comprennent qu'il est nécessaire de savoir contrôler comment les informations circulent sur un système. Je montre que, bien sûr, cette idée existait dès les premiers travaux sur la sécurité de Denning, Biba, Bell et La Padula mais que leur travaux sont difficiles à mettre en place et administrer dans les systèmes actuels. Je présente ensuite des approches plus récentes comme le *tainting*. Je montre aussi que le suivi de flux d'information peut se décliner à plusieurs niveaux d'observation qui permettent d'assurer des propriétés de sécurité différentes.

[2013-2014] [Master 2 SRI][6h CM][Professeur associé à SUPELEC]
Je présente une introduction à la détection d'intrusion ainsi que des travaux sur le contrôle des flux d'information.

[2013-2014] [FC Mastere Cyber-Sécurité][3h CM][Professeur associé à SUPELEC]
Je présente une introduction à la détection d'intrusion ainsi que des travaux sur le contrôle des flux d'information.

Encadrement de projets

Projet professionnel

A l'université de Rouen j'ai encadré le projet de fin de M2 de deux groupes en 2006 ce qui représentait deux fois vingt-quatre heures de travaux pratiques.

Projet logiciel

A SUPELEC, les étudiants de 2e année doivent réaliser un projet logiciel. Ce projet est encadré par les enseignants. Chaque année depuis 2007 je participe à ce projet à titre d'enseignante-encadrante. J'ai supervisé des projets variés comme de nombreuses réalisations de petits jeux vidéos, des applications Android, des sites Internet, un projet robot pour le concours de France de robotique ou des projets plus exotiques comme un projet visant à permettre la détection de moutons couchés sur le dos [1], un projet *entreprise* sur le thème de l'équitation *low-cost*. Dans le cadre de ces projets, il est demandé aux étudiants de spécifier et concevoir des applications complètes avec une interface graphique et un peu de difficulté algorithmique. Ces projets sont réalisés par des étudiants du cursus ingénieur de Supelec qui sont des étudiants généralistes. Ces étudiants ont *a minima* suivi la formation générale de l'école en informatique. Depuis 2009, je suis responsable de ces projets. Je propose aux étudiants de choisir eux même leur sujet. Ils viennent discuter de leur choix et motivations avec moi avant que leur projet soit validé. Si besoin le projet est redimensionné. Ce changement a, je pense, permis d'augmenter l'intérêt et l'implication des étudiants. Il faut admettre que les sujets qu'ils proposent ne sont pas toujours très ambitieux mais certains aboutissent tout de même à de jolies réalisations.

Projet long

A SUPELEC, les étudiants de 2e année peuvent choisir de réaliser un projet de plus grande envergure que le projet logiciel et se déroule durant toute l'année en remplacement du projet logiciel, du projet de conception et de deux enseignements d'options. En 2014–2015, j'ai encadré avec Gilles Vaucher un projet long dont l'objectif était la réalisation d'un logiciel d'aide à la démonstration. Ce logiciel devait offrir une interface utilisateur facile d'utilisation et devait permettre de transformer la démonstration produite en une ébauche de preuve formelle pour Coq. Les étudiants à l'origine de ce projet étaient très motivés, ils ont su rattraper un important de bagage théorique et ont réalisé un bon projet.

En 2015–2016, j'encadre avec Pascal Cottret un projet visant la réalisation d'un potager urbain.

1. Les moutons couchés sur le dos meurent rapidement, il est important pour l'éleveur de les détecter rapidement

Projet de fin d'études 3A SUPELEC

A SUPELEC, les étudiants de troisième année doivent réaliser un projet de fin d'études qui se déroule sur cinq mois (de novembre à mars) et durant lesquels un groupe de deux à quatre étudiants travaillent à l'école pendant 225 heures. En 2011-2012, j'ai encadré avec Frédéric Tronel un projet consistant à étudier des attaques informatiques et à montrer qu'il aurait été possible de les détecter en utilisant le moniteur d'information Blare.

Contrat d'étude industrielle (CEI)

Le projet de fin d'étude des étudiants de 3A peut prendre la forme d'un contrat d'étude industrielle. Les étudiants doivent alors réaliser un projet pour le compte d'une entreprise. Les contraintes scientifiques, techniques et le volume horaire sont les mêmes que celles du projet de fin d'étude à SUPELEC décrit précédemment. Ce projet est encadré par un ou plusieurs enseignants de l'école. Dans ce cadre, j'ai suivi le travail de deux groupes de projets de deux étudiants à chaque fois. Ces deux CEI étaient conclus avec la même équipe de la même entreprise (CASSIDIAN EADS), le premier en 2009-2010, le second en 2010-2011. Ces deux projets avaient pour but d'étudier la fuite d'informations sensibles.

Dans le premier cas, il s'agissait de mettre en évidence les informations sensibles qui peuvent fuiter dans des documents de bureautique de type Word. Quelles sont les informations qui sont contenues dans un tel document, dont l'utilisateur n'a pas conscience et qui peuvent être envoyées par ce biais à une personne hors de l'entreprise. Par exemple, lorsque un utilisateur inclut une image dans un document et que cette image est rognée pour éviter de montrer des objets, des visages, l'utilisateur pense avoir enlevé les informations sensibles mais en réalité, l'image est incluse entière dans l'archive doc et est donc accessible dans sa globalité.

Dans la seconde étude, CASSIDIAN a demandé aux étudiants de mettre en évidence les informations sensibles d'une grande organisation ou entreprise que l'on peut trouver sur les sites web de ces organisations. Dans ce cadre, les étudiants ont reconstruit une partie de l'organigramme d'une entreprise en utilisant son site web et un peu d'ingénierie sociale. Cette étude était co-encadrée par Guillaume Piolle, professeur assitant à SUPELEC.

Chapitre 3

Activités scientifiques

Résumé des activités de recherche

Les activités de recherche que j'ai menées depuis mon arrivée à SUPELEC en 2006 concernent la sécurité des systèmes informatiques. J'ai développé, mené ou encadré des travaux sur la sécurité des systèmes distribués : mécanismes de certification et systèmes de réputation, sur la sécurité des systèmes de vote électronique, sur l'analyse de logs. J'ai eu l'opportunité de creuser un sujet en particulier, celui de la protection de l'information par suivi de flux d'information. Le problème qui m'intéresse est d'étudier comment il est possible de protéger l'accès à l'information. Traditionnellement, les données stockées sur un ordinateur sont protégées par un ensemble de règles de contrôle d'accès. Les différents systèmes de contrôle d'accès s'appuient sur des ensembles de sujets et des ensembles d'objets. Les sujets sont les entités actives (comme des processus) qui peuvent demander à accéder à des données contenues dans les objets qui sont des entités passives (comme des fichiers). Les différents mécanismes de contrôles d'accès s'attachent ensuite à définir les circonstances dans lesquelles les sujets peuvent accéder (lire, modifier, exécuter) à ces objets. Le but premier du contrôle d'accès est de protéger les données contenues dans les objets. Par exemple, pour empêcher Bob d'accéder aux photos d'Alice, il est possible de définir des règles de contrôle d'accès spécifiant que les processus appartenant à l'utilisateur Bob n'ont pas le droit de lire les fichiers d'Alice. Hélas, le contrôle d'accès est impuissant à protéger les données une fois que celles ci sont sorties de leur conteneur d'origine. Si Alice laisse Charlie lire ses données, elle lui laisse l'opportunité de recopier ses photos dans d'autres fichiers qui eux pourront être lus par Bob.

Lorsque je suis arrivée dans l'équipe CIDRe, autrefois SSIR, Guillaume Hiet était en deuxième année de thèse et poursuivait un travail initié par Jacob Zimmermann lors de sa thèse dans l'équipe. Ce travail était la réalisation de la première version de Blare, un moniteur de flux d'information qui permettait d'assurer une politique de sécurité en suivant comment les contenus se déplaçaient dans un système d'information de type Unix. Jacob Zimmerman avait réalisé la première version de Blare, montrant ainsi la

faisabilité de l'approche. Guillaume Hiet avait repris la modélisation formelle de ce moniteur et cherchait à prouver que ce moniteur était à la fois fiable et pertinent.
A la demande de Ludovic Mé, je me suis intéressée à ce travail et force est de constater que le sujet m'intéresse toujours. Entre septembre 2006 et septembre 2014, j'ai exploré avec l'aide d'autres chercheurs et doctorants les questions suivantes :

Peut-on formaliser le travail réalisé par Blare ? Blare était à l'origine un outil de détection d'intrusion voulant détecter les attaques dites par délégation. Autrement dit les attaques qui se basent sur une coopération d'agents dans le but de contourner le contrôle d'accès en utilisant ses propres faiblesses comme celles décrites précédemment. Il s'est avéré que Blare permettait bien plus que cela et il était nécessaire de pouvoir modéliser exactement l'approche suivie par l'outil et les propriétés de sécurité qui en découlaient. Je pense qu'aujourd'hui nous avons obtenu un modèle de Blare simple à comprendre et qui permet de formaliser efficacement les bonnes propriétés de l'approche. Ce résultat a été obtenu par raffinements successifs avec Guillaume Hiet (alors doctorant), Laurent George (alors stagiaire de M2), Christophe Hauser (alors doctorant), Stéphane Geller (alors doctorant) et Fréderic Tronel (professeur associé). J'étais encadrante des travaux de thèse de Guillaume Hiet et Stéphane Geller et j'ai encadré le stage de Laurent George. Christophe Hauser était encadré par Frédéric Tronel.

Comment exprimer formellement la différence entre des politiques de contrôle d'accès et des politiques de flux d'information ? J'ai travaillé avec Mathieu Jaume, maître de conférence HDR à l'université Paris6 à la définition d'une propriété exprimant la cohérence entre la mise en œuvre d'une politique de contrôle d'accès et la politique de flux qu'elle induit.

Comment spécifier des politiques de flux d'information ? Les travaux de thèse de Jacob Zimmermann et ceux de Guillaume Hiet avaient montré que les politiques de sécurité qui peuvent être assurées/surveillées par Blare sont différentes des politiques de contrôles d'accès. Guillaume Hiet et Jacob Zimmermann avaient proposé de spécifier une politique de flux pour un système surveillé par Blare en interprétant les droits d'accès déjà en vigueur sur ce système. Durant son stage de M2, Laurent George a été un peu plus loin en proposant de spécifier une politique de flux en interprétant une politique ORBAC [KBB+03]. Ces deux travaux consistaient en fait à traduire des politiques d'accès en des politiques de flux ce qui était un premier pas mais qui ne permettait pas de spécifier des propriétés de sécurité qui peuvent être mises en œuvre par une politique de flux d'information mais qui ne sont pas exprimables par des politiques de contrôle d'accès.

J'ai proposé à Stéphane Geller en 2009 d'étudier comment spécifier simplement et formellement des politiques de contrôle de flux d'information. Au cours de son travail de thèse[1], Stéphane a proposé un langage permettant de spécifier

1. malheureusement non soutenu

une politique sécurité basée sur les flux d'information observables par Blare. Ce langage se nomme BSPL pour *Blare Security Policy Langage*.

Ces politiques sont elles fiables ? pertinentes ? utilisables ? Durant son stage de L3, Thomas Saliou a implémenté un outil (*BSPL policy manager*) permettant de composer et appliquer des politiques spécifiées en BSPL. Avec Radoniaina Andriatsimandefitra (alors doctorant), j'ai proposé une méthode semi-automatique permettant de construire des politiques en BSPL pour des applications Android. Nous avons montré que cette approche permettait de protéger des applications et de détecter des versions malveillantes.

Peut-on utiliser Blare pour caractériser et reconnaître des comporte--ments malveillants ? Ce sujet a été exploré par Radoniaina Andriatsimandefitra dans ses travaux de thèse. Il a proposé une structure de données permettant de donner une représentation compacte sous forme de graphes des logs de Blare. Cette représentation permet de calculer ce que nous avons appelé des signatures comportementales de malware (ou maliciels en français). Ces graphes constituent des signatures puisqu'ils décrivent exactement un ensemble de caractéristiques propres à un malware. Nous avons qualifié ces signatures de comportementales puisqu'elles décrivent le comportement malveillant d'une application dans l'environnement système.

Peut-on forcer l'exécution de comportements malveillants ? Pour pouvoir ensuite les monitorer avec Blare et les analyser ? Cette question s'est posée très rapidement après la définition des signatures comportementales. Si l'on pense que les signatures comportementales sous la forme de graphes de flux système sont prometteuses, alors il faut pouvoir créer automatiquement ces signatures en grand nombre. En pratique cela implique qu'on doit être capable de déclencher l'exécution d'un comportement malveillant pour pouvoir ensuite l'analyser. Ce n'est pas un problème trivial car évidemment les auteurs de malware font tout pour se cacher de l'analyse dynamique. Adrien Brunelat stagiaire de M2, Jean-François Lalande maitre de conférence à l'INSA de Bourges en délégation INRIA dans l'équipe CIDRE et moi avons commencé à étudier ce problème.

Quid de la complétude de l'observation des flux de Blare ? D'un point de vue purement théorique, Blare est un outil qui ne devrait pas avoir de faux négatifs, c'est-à-dire que si un flux d'information illégal se produit il est forcément intercepté par Blare. Il ne devrait donc pas y avoir d'attaques causant des flux illégaux qui restent non détectées. En pratique, Blare est développé pour Linux et pour Android dans une branche de développement distincte et a été développé par trois doctorants, un chercheur, un stagiaire et un ingénieur expert, chacun ayant tour à tour trouvé des défauts dans l'implémentation mettant en danger les résultats obtenus précédemment. Avec Frédéric Tronel, Guillaume Piolle, Mathieu Jaume, nous avons proposé à Laurent Georget d'étudier ce problème dans sa thèse de doctorat. Laurent a débuté dans son stage de M2 en 2014 une formalisation des appels système Linux. Il poursuit ce travail en thèse depuis septembre 2014. Son objectif est d'étudier formellement les flux d'information qui sont rendus possibles

par un système d'exploitation de type Linux.

Encadrement scientifique

Entre septembre 2006 et septembre 2015, j'ai encadré les travaux des étudiants suivants :

Encadrement de stagiaire de L3 :
[juin et juillet 2013] [stage L3 ENS]
Thomas Saliou. Implémentation d'un *BSLP Policy Manager* pour Android. Tests en détection d'intrusion. Taux d'encadrement 100%.

Encadrement de stagiaire de M1 :
[juin et juillet 2010] [stage L3 université de Rennes 1]
Christopher Humphries. Implémentation d'une interface graphique pour un moteur d'analyse de logs en logique temporelle. Taux d'encadrement 100%.

[juin et juillet 2010] [stage 4A INSA]
Marco Teamboueon. Instrumentation d'un interpréteur BPEL pour le suivi de flux d'information dans les orchestrations de web services. Stage encadré avec Eric Totel. Taux d'encadrement 50%.

Encadrement de stagiaire de M2 :
[février à juillet 2009] [stage Master 2 MRI de l'université de Rennes 1]
Laurent George. Configuration d'un outil de détection d'intrusion par la politique de sécurité. Stage encadré avec Eric Totel. Taux d'encadrement 50%.

[février à septembre 2011] [stage Master 2 SSI de l'université de Rouen]
Radoniaina Andriatsimandefitra. Adaptation de Blare à Android. Taux d'encadrement 100%.

[février à juillet 2012] [stage Master 2 MRI de l'université de Rennes 1]
Paul Lajoie-Mazenc. Système de réputation anonyme. Stage encadré avec Emmanuelle Anceaume, Gilles Guette, Nicolas Prigent. Taux d'encadrement 30%.

[février à juillet 2014] [stage Master 2 MRI de l'université de Rennes 1]
Adrien Brunelat. Déclenchement d'activités malveillantes. Stage encadré avec Jean-François Lalande. Taux d'encadrement 50%.

[février à juillet 2014] [stage Master 2 MRI de l'université de Rennes 1]
Laurent Georget. Formalisation de la sémantique des appels système en termes de flux d'information dans un système d'exploitation de type Unix. Stage encadré avec

Mathieu Jaume, Guillaume Piolle et Frédéric Tronel. Taux d'encadrement 25%.

[septembre 2014 à janvier 2015] [stage Licenciatura en Ciencias de la Computacion Universidad de Buenos Aires]
Julián Sackmann. Formalisation de l'apport de la notion d'équité dans la résolution des conflits. Stage encadré avec Guillaume Piolle. Taux d'encadrement 35%.

[février à juillet 2015] [stage Master 2 MRI de l'université de Rennes 1]
Adrien Abraham. Analyse statique de codes malveillants dans le but d'en automatiser le déclenchement. Stage encadré avec Jean-François Lalande s'inscrivant dans le cadre du projet Kharon CominLabs. Taux d'encadrement 50%.

[avril à septembre 2015] [stage Master 2 MRI de l'université de Rennes 1]
Nicolas Kiss. Rétroingénierie de codes malveillants dans le but de produire une bibliothèque bien documentée de malware Android. Stage encadré avec Jean François Lalande s'inscrivant lui dans le cadre du projet Kharon CominLabs. Taux d'encadrement 50%.

Encadrement de doctorants :
[2005-2008]

Guillaume Hiet. *Détection d'intrusions paramétrée par la politique de sécurité grâce au contrôle collaboratif des flux d'informations au sein du système d'exploitation et des applications : mise en œuvre sous Linux pour les programmes Java.*
Thèse sous la direction de Ludovic Mé, soutenue en 2008. Taux d'encadrement 50%.

[2006-2009]
François Lesueur. *Autorité de certification distribuée pour des réseaux pair-à-pair structurés : modèle, mise en oeuvre et exemples d'applications.*
Thèse sous la direction de Ludovic Mé, soutenue en 2009. Taux d'encadrement 50%.

[2006-2010] [co-tutelle avec Sup'Com Tunis]
Mehdi Talbi. *Spécification et vérification automatique des propriétés des protocoles de vote électronique en utilisant la logique ADM.*
Thèse sous la direction de Christophe Bidan et soutenue en 2010. Taux d'encadrement 50%.

[2009-2013]
Thomas Demongeot. *Politique de contrôle de flux d'information définie par les utilisateurs pour les orchestrations de services. Mise en œuvre dans un orchestrateur BPEL.*
Thèse co-encadrée avec Eric Totel et sous la direction de Yves le Traon et soutenue en 2013. Taux d'encadrement 40%.

[2009-Abandon en 2014]
Stéphane Geller. Thèse proposant un langage de spécification des politiques de flux d'information.
Abandonnée lors de la phase de rédaction.
Thèse sous la direction de Ludovic Mé. Taux d'encadrement 100%.

[2011-2014]
Radoniaina Andriatsimandefitra. Caractérisation et détection de malware Android basées sur les flux d'information.
Thèse sous la direction de Ludovic Mé et soutenue en 2014. Taux d'encadrement 100%.

[2012- 2015]
Paul Lajoie-Mazenc. Thèse portant sur un système de réputation respectant l'anonymat des agents.
Thèse sous la direction d'Emmanuelle Anceaume. Taux d'encadrement 25%.
Soutenue en 2015.

[2014 ∼]
Laurent Georget. Thèse portant sur l'étude formelle des flux d'information rendus effectivement possibles par les appels système sous Linux.
Thèse sous la co-direction de Mathieu Jaume et moi-même, co-encadrée avec Guillaume Piolle et Frédéric Tronel. Taux d'encadrement 25%.
Soutenance en 2017.

[2015 ∼]
Mourad Leslous. Thèse portant sur le déclenchement automatique de comportements malveillants protégés.
Thèse sous la co-direction de Thomas Genet et moi-même, co-encadrée avec Jean-François Lalande. Taux d'encadrement 50%.
Soutenance en 2018.

Coopération scientifique

Depuis 2006 j'ai participé à trois projets de recherche financés par l'ANR : Polux, Save et Lise. En 2014, j'ai déposé avec l'équipe Celtique de l'IRISA un projet LABEX CominLabs, ce projet a débuté en 2015.

POLUX : POLicy Unified eXpression

Le projet POLUX [2] était un projet ANR de l'appel ANR-06-SETIN-012. Le but de ce projet était la définition d'une méthode formelle pour la construction d'une politique de

2. `http://www.rennes.supelec.fr/polux/`

sécurité et son application de manière uniforme à l'ensemble des technologies et composants de sécurité. Ce projet regroupait des chercheurs et enseignants chercheurs de SUPELEC, GET/ENST Bretagne, LACL, Université Paris-Est Créteil, GRIL, Université Sherbrooke, Québec.

SAVE : Sécurité et Audit du Vote Electronique

Le projet SAVE [3] était un projet ANR de l'appel ANR-06-TCOM-0029. Le projet SAVE visait à étudier de manière approfondie la sécurité des systèmes de vote électronique à distance. Le résultat principal de ce projet a été le développement d'un système de vote électronique qui garantit à la fois le secret du vote et la sincérité du scrutin. Ce projet regroupait des chercheurs de Cryptolog, d'Orange, de l'ENS ULM, de l'INT et de SUPELEC.

LISE : Liability Issues in Software Engineering

Le projet LISE était un projet ANR de l'appel SeSur (ANR-07-SESU-007). Le projet LISE visait à proposer des méthodes et des outils pour d'une part définir de manière précise et non ambigüe les responsabilités contractuelles en matière de logiciels et d'autre part d'établir ces responsabilités en cas d'incident Ce projet était un projet multi-disciplinaire qui regroupait des juristes de l'université de Caen Basse Normandie, de l'université de Versailles Saint-Quentin-en-Yvelines et des informaticiens chercheurs à l'INRIA, enseignants chercheurs de l'IMAG, VERIMAG et SUPELEC.

Kharon

Ce projet a débuté en 2015. Il vise à mettre en place une plateforme d'analyse de sécurité des applications mobiles en utilisant à la fois de l'analyse statique et de l'analyse dynamique par suivi de flux d'information. Il implique des chercheurs de SUPELEC et de l'IRISA (équipe Celtique). Il débute avec le recrutement de Radoniaina Andriatsimandefitra comme ingénieur expert, en charge, pour commencer, du développement de la plateforme et d'Adrien Abraham comme stagiaire de MRI chargé d'étudier comment automatiser le déclenchement des exécutions des malware.

Participation à la communauté scientifique

Présentations scientifiques invitées

[Journées sur la Sécurité des Systèmes Informatiques][Rouen][2010]]
Présentation invitée sur le sujet Détection d'intrusion par suivi de flux d'information

[Technicolor][Rennes][2011]
Présentation invitée sur le sujet Détection d'intrusion par suivi de flux d'information

3. `http://www.projet-save.fr/`

[Séminaire IPR][ENS Rennes][2013]
Présentation invitée sur l'apport de la théorie de Jeux en sécurité informatique.

[Ecole Polytechnique de Milan][2014]
Présentation invitée sur la caractérisation d'activités malveillantes par suivi de flux d'information.

[Conférence sur la Sécurité des Systèmes d'Information (CISSI) Kénitra, Maroc][2015]
Présentation invitée sur la caractérisation d'activités malveillantes par suivi de flux d'information.

Participation à un jury de thèse autre que les doctorants encadrés

Bien que n'étant pas encore titulaire de l'habilitation à diriger des recherches, j'ai participé au jury de thèse de Lionel Habib en tant que rapporteur en juin 2011. Thèse de doctorat de l'université Paris 6 sur le sujet *Formalisations et comparaisons de politiques et de systèmes de sécurité* sous la direction de Mathieu Jaume et Thérèse Hardin.

J'ai aussi été examinatrice dans le jury de thèse d'Aurélien Thierry. Thèse de doctorat de l'université de Lorraine sur le sujet *Désassemblage et détection de logiciels malveillants auto-modifiants* sous la direction de Jean Yves Marion. Mars 2015.

Groupe de Travail

[LabOSSec] Membre du groupe de travail du projet *Fourniture d'un référentiel permettant la labelisation d'outils d'analyse de code* mené par SafeRiver pour l'ANSSI.

Production scientifique

Pour finir, cette section énumère l'ensemble de mes publications. Les résultats seront détaillés et expliqués dans la partie suivante. En résumé, et d'un point de vue purement quantitatif, une thèse de doctorat, deux brevets, huit articles de journaux internationaux, un article de journal francophone, trente actes de conférences internationales, neuf actes de conférences francophones, deux posters dans des conférences internationales (avec actes) et cinq rapports de recherches. Certaines de ces publications sont estampillées à l'aide du symbole $\diamond$. Une telle estampille signifie que le travail qui est décrit dans l'article est une étape dans les travaux de recherche qui seront présentés dans la suite du document.

Thèse de doctorat

Automates d'arbres et réécriture pour l'étude de problèmes d'accessibilité
Viet Triem Tong V.

Doctorat de l'université de Rennes 1, (2003)

Brevets

Procédé d'extraction d'un secret à partir de données biométriques
inventeurs Valérie Viet Triem Tong, Hervé Sibert et Marc Girault
dépôt 06 55967 auprès de l'INPI.

Procédé de génération d'un aléa à partir d'une donnée biométrique
inventeurs Valérie Viet Triem Tong, Hervé Sibert et Jean Jacques Schwartzmann
dépôt 07 53940 auprès de l'INPI

Articles de journaux internationaux

Information Flow Policies vs Malware - Final Battle -
Andriatsimandefitra R., Saliou T., Viet Triem Tong V.
JIAS. International Journal of Information Assurance and Security (2015)

Extending Signatures of Reputation
E. Anceaume, G. Guette, P. Lajoie-Mazenc, T. Sirvent, and V. Viet Triem Tong
IFIP AICT, IFIP International Federation for Information Processing (2014)

User Data on Androïd Smartphone Must be Protected
Andriatsimandefitra R., Viet Triem Tong V., Mé L.
ERCIM News, 90 (2012)

User Data Confidentiality in an Orchestration of Web Services
Demongeot T., Totel E., Viet Triem Tong V., Le Traon Y.
JIAS. International Journal of Information Assurance and Security (2012)

Liability issues in software engineering : the use of formal methods to reduce legal uncertainties
Le Métayer D., Maarek M., Mazza E., Potet M.-L., Frénot S., Viet Triem Tong V., Craipeau N., Hardouin R.
Communications of the ACM (2011)

⋄ *Specifying and enforcing a fine-grained information flow policy : model and experiments*
Viet Triem Tong V., Clark A., Mé L.
JoWUA. Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications (2010)

Policy-based intrusion detection in web applications by monitoring Java information flows
Hiet G., Viet Triem Tong V., Mé L., Morin B.
IJICS. International Journal of Information and Computer Security. (2008)

Reachability analysis of term rewriting systems
G Feuillade, T Genet, Viet Triem Tong V.,
JAR. Journal of Automated Reasoning (2004)

Articles de journaux francophones

Spécification et mécanisme de détection de flots d'information illégaux
Jaume M., Viet Triem Tong V., Hiet G.
TSI. Technique et Science Informatiques (2012)

Actes de conférences internationales

GroddDroid : a Gorilla for Triggering Malicious Behaviors
Abraham A, Andriatsimandefitra R., Brunelat A., Lalande J-F, Viet Triem Tong V.
MALCON. International Conference on Malicious and Unwanted Software. Porto Rico (2015)

Detection and Identification of Android Malware Based on Information Flow Monitoring
Andriatsimandefitra R.,Viet Triem Tong V.
CSCloud. IEEE International Conference on Cyber Security and Cloud Computing. Etats-Unis (2015)

An Activity Diagram Extraction and Visualization Toolset Designed for the Linux Codebase
Georget L.,Tronel F.,Viet Triem Tong V.
VISSOFT. IEEE Working Conference on Software Visualization. Allemagne (2015)

Privacy-Preserving Reputation Mechanism : A Usable Solution Handling Negative Ratings
Lajoie-Mazenc P.,Anceaume E., Guette G., Sirvent T., Viet Triem Tong V.
IFIPTM. International Conference on Trust Management. Allemagne (2015)

Towards a Semantics for System Calls in terms of Information Flow
Georget L., Tronel F., Piolle G., Viet Triem Tong V, Jaume M.
ICONS. International Conference on Systems. Espagne (2015)

◇ *Capturing Android Malware Behaviour using System Flow Graph*
Andriatsimandefitra R., Viet Triem Tong V.
NSS. International Conference on Network and System Security. Chine (2014)

A Privacy Preserving Distributed Reputation Mechanism
Anceaume E., Guette G., Lajoie-Mazenc P., Prigent N., Viet Triem Tong V. ICC. International Conference on Communications. Hongrie (2013)

Diagnosing intrusions in Android operating system using system flow graph
Andriatsimandefitra R., Viet Triem Tong V., Mé L.
WISG. Workshop Interdisciplinaire sur la Sécurité Globale. France (2013)

◇ *Information Flow Policies vs Malware.*
Andriatsimandefitra R., Saliou T., Viet Triem Tong V.
IAS. Information assurance and security. Tunisie (2013)

◇ *BSPL : A Language to Specify and Compose Fine-grained Information Flow Policies*
Geller S., Viet Triem Tong V., Mé L.
SECUREWARE. International Conference on Emerging Security Information, Systems and Technologies. Espagne (2013)

Secure states versus Secure executions : From access control to flow control
Jaume M., Andriatsimandefitra R., Viet Triem Tong V., Mé L.
ICISS. International Conference on Information Systems Security. Inde (2013)

User Defined Control Flow Policy for Web Service Orchestration
Demongeot T., Totel E., Viet Triem Tong V.
C&ESAR. Computer & Electronics Security Applications Rendez-vous. France(2012)

Designing information flow policies for Android's operating system
Andriatsimandefitra R., Geller S., Viet Triem Tong V.
ICC. International Conference on Communications. Canada (2012)

Preventing data leakage in service orchestration
Demongeot T., Totel E., Viet Triem Tong V., Le Traon Y.
IAS. International Conference on Information Assurance and Security. Malaisie (2011)

◇ *Flow based interpretation of access control : Detection of illegal information flows*
Jaume M., Viet Triem Tong V., Mé L.
ICISS. International Conference on Information Systems Security. Inde (2011)

From SSIR to CIDre : a New Security Research Group in Rennes
Anceaume E., Bidan C., Gambs S., Hiet G., Hurfin M., Mé L., Piolle G., Prigent N., Totel E., Tronel F. et al
SysSec Workshop, Pays-Bas (2011)

Information Flow Control for Intrusion Detection derived from MAC Policy
Geller S., Hauser C., Tronel F., Viet Triem Tong V.
ICC. International Conference on Communications. Japon (2011)

Liability in Software Engineering Overview of the LISE Approach and Illustration on a Case Study
Le Métayer D., Maarek M., Mazza E., Potet M.-L., Frénot S., Viet Triem Tong V., Craipeau N., Hardouin R., Alleaune C., Benabou V.-L. et al
ICSE. ACM/IEEE International Conf. on Software Engineering. Afrique du Sud (2010)

An Efficient Distributed PKI for Structured P2P Networks
Lesueur F., Mé L., Viet Triem Tong V.
P2P. IEEE International Conference on Peer-to-Peer Computing. États-Unis (2009)

Specification of Anonymity as a Secrecy Property in the ADM Logic - Homomorphic-based Voting Protocols
Talbi M., Viet Triem Tong V., Bouhoula A.
ARES. International Conference on Availability, Reliability and Security. Japon (2009)

Policy-Based Intrusion Detection in Web Applications by Monitoring Java Information Flows
Hiet G., Viet Triem Tong V., Mé L., Morin B.
CRISIS. International Conference on Risks and Security of Internet and Systems. Tunisie (2008)

Specification of Electronic Voting Protocol Properties Using ADM Logic : FOO Case Study
Talbi M., Morin B., Viet Triem Tong V., Bouhoula A., Mejri M.
ICICS, International Conference on Information and Communication Systems. Royaume-Uni (2008)

A Sybilproof Distributed Identity Management for P2P Networks
Lesueur F., Mé L., Viet Triem Tong V.
ISCC. IEEE Symposium on Computers and Communication. Maroc (2008)

A Sybil-Resistant Admission Control Coupling SybilGuard with Distributed Certification
Lesueur F., Mé L., Viet Triem Tong V.
COPS. International Workshop on Collaborative Peer-to-Peer System. Italie (2008)

A distributed certification system for structured P2P networks
Lesueur F., Mé L., Viet Triem Tong V.
AIMS. International Conference on Autonomous Infrastructure, Management and Security. Allemagne (2008)

Detecting and Excluding Misbehaving Nodes in a P2P Network
Lesueur F., Mé L., Viet Triem Tong V.
I2CS. International Conference on Innovative Internet Community Systems. France (2008)

Monitoring both OS and program level information flows to detect intrusions against network servers
Hiet G., Mé L., Morin B., Viet Triem Tong V.
MONAM. IEEE Workshop on Monitoring, Attack Detection and Mitigation. France (2007)

Biometric fuzzy extractors made practical : a proposal based on FingerCodes
Viet Triem Tong V., Sibert H., Lecoeur J., Girault M.
ICB. International conference on Biometrics. République de Corée (2007)

Verification of copy-protection cryptographic protocol using approximations of term rewriting systems
Genet T.,Tang-Talpin Y.M., Viet Triem Tong V.
WITS. Workshop on Issues in the Theory of Security. Pologne (2003).

Reachability analysis of term rewriting systems with Timbuk
Genet T. , Viet Triem Tong V.
LPAR. Conference on Logic for Programming, Artificial Intelligence, and Reasoning. Cuba (2001)

Actes de conférences francophones

*Mise en œuvre de politiques de protection des flux d'information dans l'environnement Android*Viet Triem Tong V., Andriatsimandefitra R., Geller S., Boche S., Tronel F.,

Hauser C.
C&ESAR,Computer & Electronics Security Applications Rendez-vous. France(2011)

Définition des responsabilités pour les dysfonctionnements de logiciels : cadre contractuel et outils de mise en oeuvre
Steer S., Craipeau N., Le Métayer D., Maarek M., Potet M-L, Viet Triem TongV.
Conférence Droit, sciences et techniques, quelles responsabilités ? France (2011)

Contrôle d'accès versus Contrôle de flots
Jaume M., Viet Triem Tong V., Mé L.
AFADL. Journées Francophones sur les Approches Formelles dans l'Assistance au développement des logiciels. France (2010)

Protection des données utilisateurs dans une orchestration de Web-Services
Demongeot T., Totel E., Viet Triem Tong V., Le Traon Y.
SARSSI. Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information. France (2010)

Spécification Formelle des propriétés des Protocoles de Vote au moyen de la Logique ADM
Talbi M., Morin B., Viet Triem Tong V., Bouhoula A., Mejri M.
SARSSI. Actes de la Conférence sur la sécurité des Architectures Réseaux et des Systèmes d'Information. France (2008)

Détection fiable et pertinente de flux d'information illégaux
Hiet G., Mé L., Zimmermann J., Bidan C., Morin B., Viet Triem Tong V.
SARSSI. Conference on Security and Network Architectures. France (2007)

Gestion distribuée d'identités résistante à la Sybil attack pour un réseau pair-à-pair
Lesueur F., Mé L., Viet Triem Tong V.
SARSSI. Conference on Security and Network Architectures. France (2007)

Contrôle d'accès distribué à un réseau pair-à-pair
Lesueur F., Mé L., Viet Triem Tong V.
SARSSI. Conference on Security and Network Architectures. France (2007)

FingerKey, un cryptosystème biométrique pour l'authentification Viet Triem Tong V., Sibert H., Lecoeur J., Girault M.
SARSSI. Conférence sur la Sécurité et Architectures Réseaux. France (2007)

Posters dans des conférences Internationales (avec actes)

Poster Abstract : Automatic Triggering of Android Malware.
Abraham A., Andriatsimandefitra R., Kiss N., Lalande J-F and Viet Triem Tong V.
DIMVA. International Conference on Detection of Intrusions and Malware & Vulnerability Assessment. Italie (2015).

Poster Abstract : Highlighting Easily How Malicious Applications Corrupt Android Devices
Andriatsimandefitra R., Viet Triem Tong V.
RAID. Poster and short paper. Recent Advanced in Intrusion Detection. Suède (2014).

Blare Tools : A Policy-Based Intrusion Detection System Automatically Set by the Security Policy
George L., Viet Triem Tong V., Mé L.
RAID. Poster and short paper. Recent Advanced in Intrusion Detection. France (2009).

Rapports de recherche

A Privacy Preserving Distributed Reputation Mechanism
Anceaume E., Guette G., Lajoie Mazenc P., Prigent N., Viet Triem Tong V.
Rapport de recherche.

Information Flow Policies vs Malware Andriatsimandefitra R., Viet Triem Tong V., Saliou T.
Rapport de recherche (2013)

Liability in Software Engineering : Overview of the LISE Approach and Illustration on a Case Study
Alleaune C., Benabou V.-L., Beras D., Bidan C., Craipeau N., Frénot S., Goessler G., Hardouin R., Le Clainche J., Le Métayer D. et al
Rapport de recherche (2009)

Proving negative conjectures on equational theories using induction and abstract interpretation
Genet T. , Viet Triem Tong V.
INRIA. Rapport de recherche (2002)

Reachability Analysis of Term Rewriting Systems with timbuk (extended version) Genet T. , Viet Triem Tong V.
INRIA. Rapport de recherche (2001)

Première partie

Contributions scientifiques

Chapitre 4

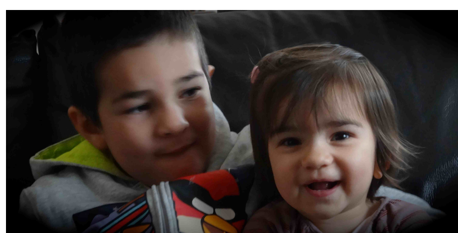
Causes de la dissémination de l'information, remèdes possibles

Ce chapitre décrit le contexte des travaux de recherche présentés dans ce document.
Lors de son écriture, nous nous sommes rendus compte que des notions simples telles que *l'information*, *un flux d'information*, *une politique de flux d'information* n'étaient finalement pas suffisamment bien définies dans la littérature. Nous nous sommes attachés à y remédier.
Plus précisément, nous définissons ici les flux d'informations rendus possibles par les opérations du système (relation $\rightarrow$) et les flux d'information autorisés par une politique de flux (relation $\rightarrow_{/\mathbb{A}}$). Nous reprenons les travaux de Denning [Den76b] pour rappeler que l'utilisation des classes de sécurité induit la structure particulière des politiques de flux d'information. Autrement dit que la relation $\rightarrow_{/\mathbb{A}}$ est nécessairement une relation d'ordre sur l'ensemble des classes de sécurité. Nous étendons l'étude de Sandhu [San93a], pour montrer que tous les modèles de politiques de flux portés à notre connaissance satisfont effectivement ces critères. Enfin nous expliquerons comment s'insèrent les travaux de recherche que nous menons.

4.1 De la nature des flux d'information

Information, données, contenus et représentation informatique. Les systèmes informatiques manipulent des encodages de ce que nous appelons des données. Une donnée est elle-même une représentation de phénomènes physiques ou logiques aussi variés que de la musique, des valeurs monétaires, le nombre π, un numéro de compte bancaire, l'utilisateur d'un ordinateur. Une donnée peut être parfois communiquée par la parole, reproduite sur le papier, encodée sur un système informatique.

Par exemple, une photographie était jusque dans les années quatre-vingts la représentation d'une image sur du papier. Aujourd'hui, une *photo* est plus souvent un fichier informatique contenant une longue suite de valeurs numériques n'ayant de sens que pour un autre fichier informatique contenant lui même une autre suite de valeurs numériques encodant un algorithme d'affichage de notre premier fichier. La figure 4.1 est à la fois une photographie telle que l'entendait le sens commun avant l'avènement de l'ordinateur et sa représentation informatique. De façon générale, il convient de distinguer une donnée de sa représentation faite pour un ordinateur. Cette représentation passe par un encodage qui consiste à associer une donnée à une configuration de bits.
Dans ce travail nous considérerons les données au travers de leur représentation informatique, c'est-à-dire un ensemble de bits. Nous utiliserons le terme données aussi bien que contenus ou informations. Nous utiliserons parfois le terme information au singulier comme le font les anglo-saxons pour qui *l'information* est une quantité indénombrable comme de l'eau, du sucre, du sable.



(a) pour un être humain


```
O;vYC^O\340\365\377^@\246}\2637\345\326\200:^[;\326S
70\234\346\200:\250.^ZM\263\357\332^T\355b\337&\356
44^?7\353\373\272\322\237_\227\352^E\211\357Z\311Um
P/\362\311\347&\346Y^]^VHd^_\353<\263\376\263\376yc
7o\303<\363@^O\267l\356e^[`]\314\333cm\305\367&|\27
'n\377^@R\277\2731\307\376\346O>o^?\322\200
3(\315"<\313^[F\362n\205\2232M^_\356\344\2168\377^@
\345\247\231\346g\244\276\203\277Nh^C1\345\222Gu^N
4\255*t\371\376\200W0\313\344\244S#g\314y^Uc\371\32
35l\261\344f\371\3227\376Q\311^O\374\363\377^@$^CN[
^@Y\313^U_/j\270m\337/E^?\373g\374\273\372\320^Ek\2
7@^X\215"\357!\3265fm\333[\370|\274^_\371i\326o\347\
^?\273\377^@"\2001\357,\244h\334\303"\371\261\252,\2
```


(b) pour un ordinateur

FIGURE 4.1 – La même photo

Les conteneurs d'information. Pour que les représentations des données persistent dans un système informatique, il faut qu'elles soit stockées. Il faut aussi que ce stockage soit organisé de manière à assurer leur disponibilité, faciliter leur lecture ou leur modification. Sur un ordinateur, le stockage est assuré par des entités physiques ou logiques. Les entités physiques sont les supports matériels tels qu'un disque dur, une clé USB, une carte SD, etc. Les entités logiques sont par exemple des machines virtuelles, des fichiers, des variables, des tableaux, des listes, des processus, des plages mémoire. Hélas, ces entités logiques n'ont pas de réalité physique et sont elles mêmes des représentations d'autres données sous la forme d'ensembles de bits. Elles sont donc aussi des données pour un autre niveau d'observation.

Pour résumer, lorsque nous manipulerons des objets tels que `Var` $x = 4$ ou bien encore `/Users/Val/.ssh/id_rsa.pub`, nous nous rappellerons qu'ils sont des conteneurs d'information, soit une variable de nom x et un fichier dont le chemin d'accès est `/Users/Val/.ssh/id_rsa.pub`, et que ces conteneurs contiennent des données, des in-

formations qui sont le cinquième entier naturel, et la valeur de la clé publique RSA d'un utilisateur de nom Val.

Flux d'information Les actions d'un programme, les actions d'un processus ou bien encore les actions d'un utilisateur induisent des flux d'information. Autrement dit, ces actions provoquent des modifications sur les contenus de différents conteneurs du système.

Lorsqu'un processus exécute le petit extrait de programme décrit par l'algorithme 2, il y provoque des flux d'information directs entre les variables x et y. Ce premier exemple manipule en effet trois conteneurs d'information qui sont les variables x, y et a. Avant l'exécution de cette instruction, la pile d'exécution associe trois valeurs à ces trois variables. Après l'exécution de l'instruction qui constitue ce programme les valeurs des variables a et y n'ont pas changé, il n'y a pas eu de flux d'information vers ces deux variables. En revanche, à la fin de l'exécution de ces instructions, la valeur associée à la variable x est modifiée et devient deux fois la valeur associée à la variable y ou un plus deux fois la valeur associée à la variable y selon la valeur associée à la variable a. Nous voyons qu'après l'exécution de ces instructions, la valeur associée à x dépend directement de celle associée à y. Nous voyons aussi que si nous connaissons à la fois la nouvelle valeur de x et la valeur de y alors nous sommes en mesure de décider si la valeur de a est l'entier 0 ou non, en d'autres termes, la valeur de a a influencé la valeur de x. Il y a un flux d'information direct de y vers x, et un flux d'information indirect de a vers x.

Algorithm 1: `Quelques instructions simples provoquant déjà des flux d'information`

```
begin
    if a==0
    then
        x := 2 × y
    else
        x :=2 × y +1
```

De manière similaire, lorsqu'un utilisateur demande la copie d'un fichier f_1 contenant simplement la chaîne de caractères *"chapeau pointu"* à la fin d'un second fichier f_2 contenant lui même la chaîne de caractères *"turlututu"* en utilisant la commande `cat f1 >> f2`, il provoque un flux d'information du fichier f_1 vers le processus exécutant `cat` puis dans un second temps de ce même processus vers le fichier f_2.

La différence entre ces deux premiers exemples est simplement le niveau d'observation. Dans le premier exemple, nous observons comment les informations circulent au niveau applicatif et dans le second exemple nous observons comment les informations peuvent circuler au niveau du système d'exploitation. Dans ces deux exemples, nous remarquons déjà que nous retrouvons des conteneurs (les variables, les fichiers, les processus), des informations (la valeur de x, la valeur de y, 0, *turlututu*, *chapeau pointu*) et que des opé-

rations provoquent la modification des associations entre conteneurs et contenus. Dans la suite de ce document, nous nous intéresserons principalement aux flux d'informations au niveau du système d'exploitation. A ce niveau, il existe des flux explicites, comme par exemple les flux d'information provoqués l'exemple ci-dessus.Il existe des moyens autres que les flux explicites pour influencer un contenu d'un conteneur, ce sont des flux dits implicites. A ce niveau d'observation, ces flux passent par des canaux de communications cachés. Un exemple de tel canal caché est l'existence ou non d'un fichier particulier, ce qui permet d'exposer un bit d'information à un autre processus, comme dans l'exemple ci-dessous.
Imaginons, par exemple, un processus exécutant le code Python ci dessous, dans un répertoire contenant aussi deux autres fichiers `A` et `B`.

```
import os as os

f = open("A")
premiereLigne = f.readline()

if premiereLigne == "123" :
    os.remove("B")
```

Ce processus est responsable d'un seul flux d'information explicite : de `A` vers lui même qui a lieu lors de la lecture du fichier `A`. Néanmoins, si nous avons la connaissance de ce code et si de plus nous sommes en mesure de vérifier si le fichier `B` est encore présent après l'exécution du programme alors nous sommes en mesure de décider si la première ligne de `A` était égal à `123` ou non. Là encore, la valeur de `A` a influencé la présence ou non présence de `B`. Il y a un flux implicite de `A` vers `B`.
Dans ce travail, nous nous sommes concentrés sur les flux d'information au niveau système d'exploitation. Pour ce niveau d'observation, nous n'avons pris en compte que les flux d'information explicites. Nous avons écarté tous les canaux cachés permettant de transmettre implicitement de l'information comme dans l'exemple ci-dessus.

Protéger les données en contrôlant les flux d'information. Protéger les données sur un système informatique revient dès lors à être capable de contrôler comment l'information se dissémine dans ce système et comment/ par qui ces informations peuvent être accédées et modifiées. Ces problèmes ont intéressé les utilisateurs dès les premiers systèmes connectés ou multi-utilisateurs.
Dans ce document, nous nous sommes attaché à donner une vue synthétique des travaux de recherche ayant pour but de définir une politique de flux d'information et de la mettre en œuvre sur un système, que ce système soit un programme, un système d'exploitation, un système multi-utilisateurs. Pour cela, nous revenons tout d'abord sur la définition d'un flux d'information entre deux conteneurs d'information. Nous montrons que l'introduction des classes de sécurité induit la structure particulière des politiques de flux d'information. Enfin, nous montrons comment il est possible de mettre en œuvre des politiques de flux.

4.2 Flux d'information possibles →

Nous proposons des définitions informelles de la notion d'information et de conteneurs d'information. Nous définissons une relation → caractérisant les flux d'information possibles entre les conteneurs d'informations à cause des opérations existantes sur le système considéré. Nous proposons ensuite d'étudier les qualités intrinsèques de ces objets afin de comprendre comment ils ont été utilisés dans la littérature.

Une information
est la représentation informatique d'une réalité. Une information peut aussi être appelée contenu (par opposition à conteneur), ou bien donnée. Une information est dite sensible lorsqu'elle sera considérée comme telle par au moins un utilisateur.

Dans un système informatique, une information est toujours liée au support qui délimite sa représentation.

Un conteneur d'information
est un support informatique physique ou logique assurant le stockage d'une information.

Ces notions d'information et de conteneur d'information sont intimement liées au niveau où on se place pour observer le système. Par exemple, une machine virtuelle peut être vue comme un conteneur d'information si l'on se place au niveau système, cette machine virtuelle contient un système d'exploitation, héberge des services, stocke les données d'utilisateurs. Néanmoins, cette même machine virtuelle peut être vue comme un contenu si nous nous se plaçons au niveau d'un *data center* pour qui la granularité d'observation n'a pas besoin d'être plus fine. Dans ce travail, nous distinguons essentiellement les niveaux applicatif et système. Nous disons que le niveau applicatif est le niveau interne à une application et que le niveau système est le niveau système d'exploitation. Au niveau applicatif, les informations sont les valeurs manipulées par l'application comme les valeurs entières, flottantes, booléennes, les chaines de caractères, et les conteneurs sont les variables, les constantes et les structures de données comme les tableaux, les listes, les objets. Au niveau système d'exploitation, les informations sont les contenus des fichiers, les codes des programmes, les conteneurs sont les fichiers, les processus, les sockets.

Il conviendrait aussi de définir le niveau matériel, le niveau réseau, le niveau cloud.

Un flux d'information
depuis un conteneur d'information source c_s vers un conteneur destination c_d que nous notons $c_s \rightarrow c_d$ est la conséquence d'une action impliquant c_s, effectuée par une entité externe et modifiant les informations contenues par le conteneur d'information c_d.

Les opérations engendrant des flux diffèrent selon le niveau d'observation. Au niveau *applicatif*, ces opérations sont incluses dans le jeu d'instructions qui compose le langage de programmation dans lequel l'application a été écrite. Ainsi l'instruction d'affectation (comme `a := b`) est une opération qui engendre un flux (ici de `b` vers `a`). Au niveau

système, les opérations engendrant des flux d'information sont un sous ensemble des appels systèmes. Les appels systèmes sont des demandes envoyées par les processus au noyau pour réclamer un service que seul le noyau peut fournir. Ces demandes peuvent être par exemple écrire un fichier sur le disque dur, envoyer un paquet sur le réseau.
Cette définition reprend une partie de celle de Dorothy Denning dans [Den76b]. Plus précisément, nous reprenons l'idée qu'un flux d'information $a \to b$ est caractérisé par le fait que la valeur de l'objet a affecte la valeur de l'objet b. Nous rendons ici explicite la notion d'information contenue dans un conteneur, il est important de noter par ailleurs que nous définissons pour l'instant un flux possibles entre des conteneurs.
Nous avons ici défini une relation $\to$ qui caractérise les *flux rendus possibles par les opérations mises en œuvre dans l'environnement.*

4.2.1 Propriétés de $\to$

La relation de flux entre des conteneurs d'information est telle que deux conteneurs d'information s, d appartiennent à cette relation ($s \to d$) si et seulement si il existe une opération capable d'influencer le contenu de d en fonction du contenu de s. Nous allons montrer que cette relation est un pré-ordre. C'est à dire que $\to$ est une relation transitive et réflexive et que ces qualités sont uniquement liées aux opérations engendrant les flux.

La relation de flux $\to$ est une relation réflexive. Trivialement, l'information qui est stockée dans un conteneur d'information peut y rester.

La relation de flux $\to$ est une relation transitive. S'il existe une opération o_1 engendrant un flux d'information de $c_1 \to c_2$ et s'il existe une opération o_2 engendrant un flux $c_2 \to c_3$ alors la séquence des opérations o_1 ; o_2 engendre un flux de $c_1 \to c_3$.

Ces deux propriétés résultent uniquement du fait qu'il est possible de ne rien faire sur un système informatique (réflexivité) et qu'il est possible de séquencer des actions sur un système informatique (transitivité).

La relation de flux $\to$ n'est pas une relation anti-symétrique. Il suffit pour s'en convaincre de considérer le petit exemple suivant :
Prenons f_1 et f_2 deux fichiers et considérons la séquence de commandes suivante `cp` f_1 f_2 suivie de `cp` f_2 f_1 la première commande engendre un flux $f_1 \to f_2$ puis la seconde engendre un flux $f_2 \to f_1$: ces flux sont donc possibles. Pour autant, f_1 et f_2 peuvent être des fichiers différents.

4.3 Flux d'information autorisés $\to_{/\mathbb{A}}$, politiques de flux

Pour contrôler la dissémination de l'information, Denning, Biba, Bell et La Padula ont, les premiers, introduit la notion de classes de sécurité qui regroupent les informations selon leur sensibilité. Par exemple, une entreprise fabriquant des avions militaires pourrait identifier que les informations relatives à la qualité des matériaux utilisés dans la

construction des avions sont des données sensibles. Elle pourrait aussi identifier que les informations relatives au système de navigation sont aussi des données sensibles de même que les données relatives à la paie de ses employés. Ces trois catégories de données sont des catégories différentes et on ne voudrait pas qu'elles soient accessibles à tous. En particulier, un même employé pourrait être autorisé à accéder à une catégorie mais pas à l'autre.
Nous montrons maintenant que l'utilisation des classes de sécurité est une partition des l'ensemble des informations. Nous montrons ensuite que cette partition induit une partition sur l'ensemble des conteneurs. Nous montrons que la relation de flux d'informations sur cette seconde partition est une relation modulo et qu'elle est une relation d'ordre. Nous faisons le lien entre une politique de flux d'information et une relation de flux sur des classes de sécurité. Ainsi, nous aurons finalement montré que l'utilisation des classes de sécurité implique une forme particulière des politiques de flux d'information.

Les classes de sécurité forment une partition de l'ensemble des informations. Les classes de sécurité sont une manière de regrouper les différentes catégories d'information sensibles. Pour être simplement administrable, un système utilisant des classes de sécurité n'en utilisera qu'un nombre fini et n'utilisera qu'un seul nom pour regrouper des informations de sensibilité similaire. De manière générale, les classes de sécurité forment une partition de l'ensemble des informations : chaque donnée a une classe de sécurité et une donnée ne peut appartenir qu'à une seule classe de sécurité. Les classes de sécurité peuvent donc être vues comme des classes d'équivalences sur l'ensemble des informations.

Les classes de sécurité induisent aussi une partition de l'ensemble des conteneurs d'informations. A tout instant, un conteneur est lié à la classe de sécurité des informations qu'il contient ce qui implique que chaque conteneur a une et une seule classe. Par exemple, si un fichier contient une donnée dont la classe de sécurité est la classe A, ce fichier est lui même lié à A tant que son contenu n'est pas modifié.

La relation de flux modulo $\rightarrow_{/\mathbb{A}}$ La relation de flux $\rightarrow$ s'étend naturellement à ces classes d'équivalences. La relation de flux modulo définit les flux entre les classes de sécurité. Elle indique qu'un flux entre deux classes de sécurité est un flux *autorisé* par une politique de sécurité.

$\rightarrow_{/\mathbb{A}}$, Flux d'information autorisés entre classes de sécurité
Un flux d'information depuis une classe de sécurité A_s vers une seconde classe de sécurité A_d que nous notons $A_s \rightarrow_{/\mathbb{A}} A_d$ exprime que tout flux d'information $s \rightarrow d$ depuis tout conteneur s de classe A_s vers tout conteneur d de classe A_d est autorisé.

Nous allons montrer que, contrairement à la relation de flux $\rightarrow$ entre des conteneurs d'information, la relation de flux $\rightarrow_{/\mathbb{A}}$ entre des classes de sécurité est réflexive, transitive et anti-symétrique. Pour cela nous reprenons exactement la démonstration de D. Denning dans [Den76b].

La relation $\rightarrow_{/\mathbb{A}}$ est une relation réflexive. L'argument avancé par Denning est que, pour tout objet a d'une classe A, les flux engendrés par l'opération $a := a$ doivent être trivialement sécurisés et que si la relation de flux qu'elle induit c'est à dire $A \rightarrow_{/\mathbb{A}} A$ ne l'était pas alors on aurait trouvé immédiatement une incohérence.

La relation $\rightarrow_{/\mathbb{A}}$ est une relation transitive. Prenons a, b, c trois conteneurs d'information de classes de sécurité respectives A, B, C. Si $A \rightarrow_{/\mathbb{A}} B$ alors une opération o_1 transférant une valeur x depuis un objet a vers b est autorisée. De la même manière, si $B \rightarrow_{/\mathbb{A}} C$ toute opération o_2 transférant une valeur x de b vers c est elle aussi autorisée. Si $A \not\rightarrow_{/\mathbb{A}} C$ cela signifierait que la séquence des opérations o_1 puis o_2 n'est pas autorisée ce qui serait contradictoire avec l'hypothèse faite par Denning qu'une séquence d'opérations est autorisée si et seulement si chacune des opérations est autorisée.

Anti-symétrie de $\rightarrow_{/\mathbb{A}}$. Reprenons notre exemple sur le transfert de valeurs entre deux fichiers f_1 et f_2 et considérons à nouveau la séquence de commandes suivante `cp` f_1 f_2 e puis `cp` f_2 f_1 la première commande engendre toujours un flux $f_1 \rightarrow f_2$ puis la seconde engendre un flux $f_2 \rightarrow f_1$ ces flux sont toujours possibles si il n'y pas de contraintes sur les opérations qui engendrent le flux. Supposons que f_1 et f_2 soient deux conteneurs d'information de classes respectives A_1 et A_2. Si ces flux sont autorisés alors les catégories d'information que peuvent contenir les objets de la classe A_1 et les objets de la classe A_2 sont la même catégorie. Si l'on admet qu'il n'est pas nécessaire d'utiliser deux classes différentes pour la même catégorie d'information on a nécessairement $A_1 = A_2$.

Nous avons montré en section 4.2.1 que la relation $\rightarrow$ des flux possibles à cause des opérations existantes sur un système était un pré-ordre sur l'ensemble des conteneurs d'information. Nous venons de montrer que la relation $\rightarrow_{\mathbb{A}}$ est en revanche un ordre sur $\mathbb{A}$. En anglais, cette relation est appelée (par Denning) la relation *can_flow* qui sous-entend l'idée que les flux entre les classes A et B peuvent avoir lieu. Comme le remarque R. Sandhu dans [San93a], il eut été plus approprié de nommer cette relation *may_flow* puisque Denning entend par là que les flux d'information sont ceux permis par la politique de sécurité. Nous pensons que la littérature existante sur le sujet n'est pas assez précise sur les qualités des objets manipulés.

Remarquons enfin que définir un ensemble de classes de sécurité munis d'une relation de flux autorisés $\rightarrow_{/\mathbb{A}}$ revient à définir une politique de flux d'information.

> Une politique de flux d'information
> est une paire $(\mathbb{A}, \rightarrow_{/\mathbb{A}})$ où
> – $\mathbb{A}$ est un ensemble de classes de sécurité,
> – $\rightarrow_{/\mathbb{A}}$ est une relation de flux d'information décrivant les flux autorisés.
> et tels que $\rightarrow_{/\mathbb{A}}$ est un ordre sur $\mathbb{A}$.

4.4 Treillis de classes de sécurité

Nous venons de montrer que pour définir une politique de flux d'information en se basant sur des classes de sécurité, il faut nécessairement définir un ensemble ordonné (de classes de sécurité). La grande majorité des travaux existant considèrent que ces classes de sécurité forment un treillis. Nous montrons ici pourquoi cette structure est intéressante dans ce contexte. Un treillis est un ensemble partiellement ordonné comme $(\mathbb{A}, \rightarrow_{/\mathbb{A}})$ tel que tout sous-ensemble de $\mathbb{A}$ admet une borne supérieure. Autrement dit, pour que $(\mathbb{A}, \rightarrow_{/\mathbb{A}})$ soit un treillis, il faut et il suffit que pour tout ensemble $\{A_1, \ldots A_n\}$ de classes de $\mathbb{A}$, il existe une classe $B \in \mathbb{A}$ telle que pour toutes classes A_i de $\{A_1, \ldots A_n\}$, $A_i \rightarrow_{/\mathbb{A}} B$ et pour toute autre classe $C \in \mathbb{A}$ vérifiant $A_i \rightarrow_{/\mathbb{A}} C$ on ait $B \rightarrow_{/\mathbb{A}} C$.

Montrons pourquoi la structure en treillis est intéressante dans notre contexte.

Considérons un ensemble de conteneurs $s_1, \ldots s_n$ de classes respectives $A_1, \ldots A_n$. La borne supérieure de $\{A_1, \ldots A_n\}$ est la plus petite classe B vérifiant $A_i \rightarrow B$ pour tout $1 \leq i \leq n$. Si $s_1, \ldots s_n$ sont les variables d'un programme, c'est par exemple la classe d'un conteneur d'information d définie par $d = s_1 + \ldots + s_n$. Si $s_1, \ldots s_n$ sont des fichiers, c'est la classe d'un processus ayant accédé en lecture à l'ensemble de ces fichiers.
Denning a introduit dans [Den76b] l'opérateur $\oplus$ permettant la *jointure* de deux classes de sécurité et permettant de calculer la borne supérieure.
Denning a montré en 1976 dans [Den76a] qu'une politique de flux d'information reposant sur un ensemble de classes de sécurité muni d'une relation de flux $\rightarrow$ était soit un treillis, soit pouvait être complété en un treillis par l'ajout de nouvelles classes de sécurité qui ne changeaient pas les flux entre les classes originales.
Une fois que l'on a été capable de spécifier une politique de flux d'information, il est intéressant de s'interroger sur la manière donc celle ci peut être mise en œuvre.

4.5 Mise en œuvre des politiques de flux

Les modèles de sécurité utilisant les classes de sécurité ont pour but de contraindre les flux d'information possibles par les opérations du système en ajoutant des mécanismes de contrôle. Sans mécanismes supplémentaires, et en réitérant le raisonnement précédent, il n'y a qu'une seule classe de sécurité dès lors qu'une opération réalisant la copie entre les conteneur d'information existe. Pour assurer la séparation des informations en classes de sécurité, il faut être capable de contraindre, contrôler les opérations du système afin de s'assurer que les flux d'information rendus possibles par les opérations du système protégé sont effectivement des flux autorisés. Autrement dit, mettre en œuvre une politique de flux consiste à s'assurer que les opérations entrainant des flux sont contraintes de sorte que l'on puisse vérifier que $\rightarrow \subseteq \rightarrow_{/\mathbb{A}}$.

Il existe beaucoup de travaux qui ont cherché à contrôler l'accès à information en contrôlant l'accès aux conteneurs d'information. Nous pensons que ces travaux ne peuvent

pas suffire à contrôle la dissémination de l'information puisque le contrôle d'accès ne permet pas de contrôler l'accès à l'information une fois que celle ci n'est plus dans son conteneur d'origine. Nous avons montré avec Mathieu Jaume et Ludovic Mé dans [JVTTM11, JVTTH12, JVTTM10] que les flux réellement possibles dans un système protégé par un mécanisme de contrôle d'accès ne sont pas tous des flux autorisés par la politique de flux déduite des règles de contrôle d'accès. Dès maintenant, nous excluont tous les travaux s'intéressant au contrôle d'accès sans contrôle de flux.
Les premiers travaux s'intéressant au contrôle des flux d'information datent du milieu des années 1970 et ont été menés par Denning, Denning, Bell, Biba et La Padula. Ces premiers travaux avaient pour but de proposer des modèles de fonctionnement de systèmes informatiques dans lesquels il ne peut pas y avoir de flux d'information non autorisé. Si l'ensemble des travaux menés ont pour but d'imposer le respect d'une politique de flux, nous distinguerons deux approches. L'approche statique vise à démontrer que l'objet sous étude n'est jamais responsable d'un flux d'information illégal. L'approche dynamique empêche l'exécution de flux illégaux au moment où ceux ci se produisent. Force est de constater que les approches statiques se prêtent plus à l'étude de programmes et les approches dynamiques sont souvent réservées aux environnements plus grands, plus ouverts, moins maîtrisés, comme les systèmes d'exploitation.
Dans ce qui suit, nous présentons différents modèles/approches et montrons qu'effectivement ils entrent bien dans le cadre défini précédemment, à savoir que les politiques de flux sont des treillis ou des ensembles ordonnés complétables en un treillis. Nous ne détaillons que les travaux dont nous pensons qu'ils apportent une dimension intéressante sur la construction ou la mise en œuvre d'une politique de flux.

4.5.1 Travaux de Bell et La Padula

Trois ans avant la publication de l'article de Dorothy Denning [Den76b], Leonard J. LaPadula and D. Elliott Bell proposaient un modèle visant à contrôler la circulation de l'information dans les systèmes informatiques. Il y a eu beaucoup de versions de ce modèle et nous présentons ici une version épurée comme l'a fait R. Sandhu dans [San93b]. Ce modèle définit des règles de sécurité qui s'ajoutent au contrôle d'accès discrétionnaire et qui permet d'assurer que les flux d'information qui peuvent avoir lieu sur le système sont uniquement des flux d'information autorisés. C'est une approche dynamique.
Plus précisément le modèle de Bell et La Padula s'appuie sur des labels de sécurité qui sont attachés aux objets et aux utilisateurs. Les labels des objets ou des utilisateurs sont fixes, ils dénotent la sensibilité maximale de l'information que peut contenir l'objet ou l'utilisateur. Dans la suite, nous reprenons les notations usuelles de la littérature et notons $\lambda(o)$ (resp. $\lambda(s)$) le label de sécurité d'un objet o (resp. d'un objet s).
Les opérations entrainant des flux comme *read* et *write* sont autorisées lorsqu'elles sont possibles d'après les règles de contrôle d'accès en vigueur sur le système et si elles respectent les règles suivantes :

Propriété simple : Un sujet s peut lire un objet o si et seulement si $\lambda(s) \geq \lambda(o)$

Propriété $\star$ Un sujet s peut écrire dans un objet o si et seulement si $\lambda(o) \geq \lambda(s)$

Nous pouvons relire ces propriétés en termes de flux d'information. La propriété simple concerne les flux d'information des objets vers les sujets, nous pouvons écrire de manière équivalente à la propriété simple que $o \to s$ si et seulement si $\lambda(s) \geq \lambda(o)$ et symétriquement, la propriété $\star$ concerne les flux d'information des objets vers les sujets et est équivalente à $s \to o$ si et seulement si $\lambda(o) \geq \lambda(s)$. En toute généralité et en négligeant la notion d'objet *vs* sujet, nous pouvons écrire que $a \to b$ si et seulement si $\lambda(a) \leq \lambda(b)$. Nous retrouvons donc exactement le cadre défini par Denning, à ceci près que les classes de sécurité sont fixées pour chaque objet et sont définies à travers la fonction λ. Tout naturellement, nous retrouvons les résultats précédents, à savoir que l'ensemble des classes de sécurité est un treillis ou peut être complété en un treillis.

Bell et La Padula ont montré dans [BL73] que si un système est dans un état initial sécurisé et si chaque exécution respecte les propriétés simples et $\star$ alors ce système ne peut atteindre que des états sécurisés où tous les flux d'information ayant eu lieu respectent la politique de sécurité.

Ces travaux permettent d'assurer la confidentialité. En étudiant de près les propriétés simple et $\star$ nous nous apercevons rapidement qu'une information contenue initialement dans un objet o ne peut être disséminée que dans des objets ou des processus dont la classe de sécurité est supérieure à $\lambda(o)$. Autrement dit, plus une information est manipulée par le système plus son accès se restreint, elle est donc de plus en plus confidentielle.

4.5.2 Les travaux de Biba

Les travaux de Kenneth J. Biba en 1977, sont basés sur un modèle dual de celui de Bell et La Padula et vise à assurer l'intégrité des données. Il reprend les mêmes notions, les mêmes propriétés simple et $\star$ mais basées sur un ordre dual. La mise en œuvre proposée est encore une approche dynamique.

4.5.3 Les travaux de D. et P. Denning

Dans [DD77], Dorothy et Peter Denning ont proposé un mécanisme de certification au niveau applicatif permettant de montrer qu'un programme n'est pas responsable de flux définis comme illégaux en regard d'une politique de sécurité définie par un treillis de classes de sécurité. Ce travail est une approche statique c'est-à-dire que l'analyse du programme se fait au moment de la compilation, permettant ainsi d'étudier toutes les exécutions possibles d'un programme. Pour décider si un programme n'entraine pas de flux illégaux, Dorothy et Peter Denning s'appuient sur une variable `certified` manipulée par le compilateur uniquement et initialisée à `true`. Chaque instruction du programme est étendue de sorte que si l'instruction induit un flux interdit la variable `certified` devient `false`. Ces travaux sont précurseurs de beaucoup d'autres dont la finalité est de garantir qu'un programme n'est pas responsable de flux d'information illégaux en regard de la politique définie au préalable. Ces approches se déclinent pour différents paradigmes de programmation.

4.5.4 Les travaux de Myers et Liskov

En 1997, A. Myers et B. Liskov ont proposé dans [ML97] un modèle un peu différent visant là encore à contrôler de la diffusion de l'information. La politique de flux sous jacente à ce modèle se définit au travers de l'ensemble des labels de sécurité. Ces labels eux-mêmes repose sur la notion de *principaux* qui sont les entités *ayant autorité* comme les utilisateurs du système. Un principal peut être propriétaire d'une information, dans ce cas il lui appartient de définir parmi les autres *principals* ceux qui sont des lecteurs autorisés pour cette information. La politique de sécurité liée à une information est l'ensemble des lecteurs autorisés pour cette information. Ces lecteurs sont définis par l'ensemble des propriétaires d'autres informations ayant permis le calcul menant à cette information. Plus formellement, chaque conteneur d'information admet un label de sécurité qui dénote la politique liée au contenu de cette variable ou de cette expression. Les labels sont de la forme générale suivante :

$$L = \{\mathbf{s_i} \triangleright s_{i_\alpha}, ..., s_{i_\beta}; ...; \mathbf{s_j} \triangleright s_{j_\alpha}, ..., s_{j_\beta}\}$$

signifiant que les principaux $\mathbf{s_i} \ldots \mathbf{s_j}$ sont les propriétaires des contenus ayant servi au calcul du contenu portant le label L. Chacun de ces propriétaires ayant respectivement autorisé les $principals s_{i_\alpha}, ..., s_{i_\beta}, \ldots s_{j_\alpha}, ..., s_{j_\beta}$. Les seuls lecteurs réellement autorisés à lire le contenu des variables portant un tel label sont ceux autorisés par tous les propriétaires.

Chaque canal d'entrée de l'information porte un label de sécurité et le label des autres conteneurs d'information se calcule par analyse statique du code grâce à des règles définies pour chaque opération responsable d'un flux.

Le modèle de Myers et Liskov présenté dans [ML97] a été implémenté au niveau applicatif comme une extension du langage Java (nommée JFlow puis Jif) et présenté dans [Mye99a]. Dans Jif, les politiques sont vérifiées à la compilation. Les opérations responsables des flux sont incluses dans le jeu d'instructions du programme et des règles de propagation des labels sont définies pour chacune de ces opérations. Un programme bien typé dans ce genre d'approche est un programme dont aucune exécution n'est responsable de flux illégal, un flux illégal étant un flux créant un label de sécurité pour un conteneur tel qu'aucun *principal* n'est lecteur autorisé de l'information contenue.

Ce modèle est le premier à expliciter la notion de propriétaire d'une information en opposition à la notion de propriétaire d'un conteneur d'information, il apporte ainsi une idée nouvelle qui distingue ce travail des précédents, c'est la notion de politique/ modèle décentralisé. Plus précisément, dans ce modèle c'est le propriétaire d'une information qui définit la politique de cette information. Il n'explicite pas cependant la notion d'information.

Ce modèle s'appuie sur un ensemble de classes de sécurité représenté par les labels. Ces classes sont ordonnées comme attendu. Un flux est entre deux conteneurs s et d autorisé si le label de s est inférieur au label de d après le flux.

4.5.5 Les travaux menés dans l'équipe CIDRE

Le modèle de Blare qui sous tend les travaux présentés dans la suite document a été obtenu par raffinements successifs avec Jacob Zimmerman, Christophe Bidan, Ludovic Mé, Guillaume Hiet, Laurent George, Mathieu Jaume, Frédéric Tronel, Christophe Hauser, et Stéphane Geller. Ce modèle est un modèle décentralisé, c'est-à-dire qu'il offre comme proposé par Myers et Liskov la possibilité aux propriétaires de données de définir la politique liée à leurs données mais aussi à leurs conteneurs. Ce modèle explicite la notion d'information. Ce modèle a été mis en œuvre dynamiquement au niveau des systèmes d'exploitation Linux, Android et au niveau applicatif dans la machine virtuelle Java. Il existe aussi une mise en œuvre pour les systèmes distribués. Ces différentes mises en œuvre sont uniquement faites sous la forme de détecteur de flux illégaux. En effet, les différentes implémentations lèvent des alertes lorsqu'elles observent un flux d'information illégal mais ne l'empêchent pas.
Ce modèle est détaillé dans le chapitre suivant mais nous pouvons déjà dire qu'il entre effectivement dans le cadre proposé ici puisqu'il se base sur un ensemble de classes de sécurité qui sont un sous-ensemble de l'ensemble des parties d'un ensemble. Les classes de sécurité utilisées par Blare sont donc ordonnées et complétables en un treillis.

4.6 Bilan

Ce chapitre a présenté les différents étapes dans les travaux de recherche qui se sont intéressés à la maitrise des flux d'information. Nous nous sommes attaché à donner une vue synthétique et unifiée des différentes approches. Nous avons montré que ce n'est pas un hasard si ces travaux se sont basés sur des treillis de classes de sécurité. Nous n'avons pas développé toutes les mises en œuvre et nous avons privilégié la présentation des modèles. Dans la suite de ce document, nous détaillerons le travail que nous avons fait autour du modèle de Blare qui est un moniteur de flux d'information permettant de mettre en œuvre une politique de flux d'information à grain fin.

Chapitre 5

Modèle et mise en œuvre d'un moniteur de flux d'information

Ce chapitre détaille le modèle de Blare qui sous tend les travaux de recherche présentés dans ce document. Nous revenons tout d'abord sur les motivations qui nous ont conduit à expliciter les informations et les différencier de leur conteneur. Nous proposons un modèle de politique de flux d'information à grain fin. Nous expliquons enfin le fonctionnement et l'implémentation de moniteurs de flux en charge de faire respecter de telles politiques. Enfin nous donnons les limites de telles approches.

Un moniteur de flux d'information est un logiciel capable de déduire dynamiquement les flux d'information ayant lieu sur le système qu'il surveille. Pour cela, il peut se baser sur des observations du système. Il peut aussi se placer en coupure des opérations du système engendrant des flux d'information, c'est-à-dire réaliser ses propres opérations au moment où le système réalise exactement le flux (par exemple lors d'un appel système). Dans l'idéal, on aimerait qu'un tel moniteur soit correct, c'est-à-dire qu'il ne déduise que des flux d'information ayant réellement lieu. On aimerait aussi qu'il soit complet, c'est-à-dire qu'il soit capable de déduire tous les flux d'information explicites ayant réellement lieu.
Un moniteur de flux d'information peut servir de point central à la mise en œuvre d'une politique de flux d'information, il peut aussi servir à comprendre le comportement des applications dans leur environnement. Nous avons exploré ces deux idées, elles feront l'objet des deux chapitres suivants.

5.1 Caractérisation de l'information

Nous pensons que l'information est une notion continue c'est-à-dire qu'il n'y a pas de frontière précise entre une information et une autre. Par exemple, considérons une valeur

flottante comme 2.0, et rangeons là dans un conteneur d'information comme une variable a. Le flux engendré par l'opération $b := a/1000.0$ modifie le contenu de la variable b, ce nouveau contenu dépend de la valeur de a, il y a eu un flux $a \rightarrow b$. Le contenu de b n'est pas exactement l'information étudiée (2.0), pourtant il a été produit directement à partir de cette information et nous pourrions la retrouver à partir de la connaissance de la nouvelle valeur du contenu de la variable b.
Pire encore, supposons maintenant que nous nous intéressions aux contenus de deux fichiers f_1 et f_2. L'opération qui consiste à calculer l'empreinte MD5 de la concaténation de ces deux fichiers réalise plusieurs flux dont le résultat est la production d'une information (le résultat rendu par la fonction de hachage MD5) qui dépend précisément des contenus des deux fichiers f_1 et f_2 mais qui est cependant très différente et dont la connaissance ne permet pas de retrouver les contenus de deux fichiers f_1 et f_2.
Comme nous l'avons vu dans le chapitre précédent, une première manière de caractériser l'information a été d'attribuer une classe de sécurité à chaque information. Ces classes de sécurité sont définies par un expert, un utilisateur particulier du système à qui il appartient aussi de définir les flux autorisés entre ces différentes classes. C'est l'approche suivie par Denning, Denning, Biba, Bell et La Padula à partir du milieu des années 1970. Myers et Liskov apportent un élément supplémentaire pour caractériser l'information en explicitant la notion de propriétaires de l'information. Ces propriétaires sont des éléments particuliers du système comme les utilisateurs, des entités actives du système. C'est à eux qu'appartient de définir vers quels lecteurs leurs informations sont autorisées à se propager. Cette idée est reprise dans le système des labels Asbestos [EKV+05] dans lequel chaque processus peut définir la politique de flux de ses propres données. Ces labels ont été mis en œuvre dans Histar [ZBWKM06a] qui est un système d'exploitation proche d'Unix mais modifié pour permettre d'assurer les politiques de flux exprimées via ces labels.
Dans le modèle de Blare, nous avons discrétisé l'information en attribuant un identifiant à chaque information reconnue comme sensible. En remarquant qu'une information n'existe dans un système informatique que pour trois raisons :

- soit elle était présente à la création du système (comme beaucoup de données permettant le fonctionnement du système d'exploitation) ;
- soit elle a été introduite/créée par une application s'exécutant sur le système ;
- soit elle a été produite lors d'un flux d'information à partir d'autres informations pré-existantes.

Nous avons conclu qu'il devait être possible d'identifier précisément chaque information en analysant et identifiant les informations sensibles existant à l'initialisation, en surveillant les canaux d'entrée de l'information dans le système et enfin en calculant automatiquement un identifiant pour des informations calculées à partir d'autres.
Notre approche revient à définir une classe de sécurité pour chaque information sensible et une dernière classe de sécurité permettant de reconnaître les informations non sensibles. Dans ce modèle, les codes exécutés par les processus sont distingués des autres informations. Plus précisément, nous utilisons les ensembles disjoints $\mathcal{I}$ et $\mathcal{X}$ pour caractériser de l'information. $\mathcal{I}$ dénote les informations sensibles qui ne sont pas des informations exécutées. Une information identifiée par un élément de $\mathcal{I}$ peut être une valeur

numérique, l'encodage d'une photo, un document texte, un code source, du bytecode etc. $\mathcal{X}$ dénote du code sensible effectivement exécuté par des processus. L'ensemble $\mathcal{X}$ n'aura de sens qu'au niveau d'observation système.

$\mathcal{I}$ et $\mathcal{X}$ identifient les informations que l'on pourrait considérer comme atomiques c'est-à-dire les informations identifiées comme sensibles à l'initialisation ou lors de leur entrée dans le système. Un contenu produit à partir d'informations atomiques sera identifié par l'ensemble des identifiants des informations ayant servi à son calcul. Dans ce modèle, chaque donnée est donc associée à un ensemble, éventuellement vide, d'identifiants. Chaque conteneur est à tout moment lié aux identifiants de l'information qu'il contient.

Reprenons les deux exemples proposés précédemment. Si nous considérons que la valeur flottante 2.0 est une information sensible, nous allons lui attribuer un identifiant $i \in \mathcal{I}$. Si cette valeur est stockée dans une variable a, nous allons attacher l'identifiant i à a. Le flux engendré par l'exécution de l'instruction $b := a/1000.0$ modifie le contenu de la variable b qui doit maintenant lui aussi être caractérisé par l'identifiant i, explicitant ainsi que ce contenu est l'information identifiée par i, ou a été produit à l'aide de calculs impliquant cette information et éventuellement des informations non sensibles (donc non marquées).

De même, si nous considérons que les fichiers f_1 et f_2 contiennent des informations sensibles, nous pouvons attacher deux identifiants i_1 et i_2 de $\mathcal{I}$ aux contenus respectifs de ces deux fichiers. L'empreinte MD5 de la concaténation de ces deux fichiers est une information qui est caractérisée par l'ensemble de ces deux informations sensibles i_1 et i_2.

5.2 Politiques de flux mises en œuvre dans Blare

Les politiques de flux mise en œuvre dans Blare sont des politiques qui définissent pour chaque conteneur d'information l'ensemble des mélanges d'informations que ce conteneur peut contenir. Ces politiques de flux très précises sont dites à grain fin. Ces politiques ne s'intéressent pas aux opérations qui engendrent des flux et ne statuent que sur la légalité du résultat de ces opérations (le résultat des flux d'information).

Sur un système informatique, beaucoup de ressources sont réutilisées pour manipuler différentes informations, un exemple assez parlant est l'éditeur de texte qui peut être utilisé pour lire/écrire des données confidentielles, personnelles, publiques, etc. Nous pensons que la mise en œuvre de politiques de flux ne doit pas limiter l'utilisation de ce type de ressources en restreignant leur usage.

Les politiques Blare, permettent de spécifier qu'un même éditeur de texte peut manipuler différentes informations du moment que celles ci ne sont pas mélangées entre elles. Un médecin soignant un patient A et un patient B peut donc utiliser deux processus exécutant l'éditeur de texte pour ouvrir simultanément les dossiers de ces deux patients tant que ceux ci ne sont pas mélangés.

Une politique de flux se traduit donc par une fonction qui à chaque conteneur c associe l'ensemble des mélanges d'informations que celui-ci est autorisé à contenir.

Une politique de flux peut se définir à l'aide d'une fonction
$\mathbb{P}_{\mathcal{C}} : \mathcal{C} \rightarrow \mathscr{P}(\mathscr{P}(\mathcal{I} \cup \mathcal{X}))$

Autrement dit, la politique attachée à un conteneur c sera un élément de la forme $\{\{i_{1_1}, i_{1_2}, \ldots, i_{1_n}\}, \ldots \{i_{j_1}, i_{j_2}, \ldots i_{j_m}\}\}$ signifiant que n'importe quel contenu calculé à partir d'un de ces ensembles a le droit de se trouver dans c. Une politique de flux d'information repose donc sur un ensemble de classes de sécurité qui sont l'ensemble des parties de $\mathcal{I} \cup \mathcal{X}$ [1]. Cet ensemble est ordonné par l'inclusion.
Définir une politique comme nous venons de le faire, revient à s'interroger sur les résultats possibles d'un flux d'information et permet donc de s'abstraire de toutes les opérations engendrant un flux d'information possible dans le système. Cela implique qu'une telle définition est totalement indépendante du système dans lequel elle est mise en œuvre.
Une telle définition prend le point de vue des conteneurs, nous verrons dans la suite qu'il peut être intéressant de prendre le point de vue dual, le point de vue des informations. c'est-à-dire de définir, pour une information donnée, avec quelles autres informations celle ci peut être mélangée et dans quels conteneurs.
Reprenons et enrichissons l'exemple de l'éditeur de texte. Supposons que nous ayons précisément identifié les données du patient A ainsi que celles du patient B et celles du médecin par exemple à l'aide des identifiants respectifs i_A, i_B et i_M. La politique du processus p exécutant un éditeur de texte pourrait être :
$\mathbb{P}_{\mathcal{C}}(p) = \{\{i_A, i_M\}, \{i_B, i_M\}\}$ spécifiant ainsi que ce conteneur peut contenir de l'information issue des informations du patient A et du médecin *ou bien* de l'information issue des informations du patient B et du médecin.

Nous venons de motiver la forme générique d'une politique de flux dans le modèle Blare. Il nous reste à montrer comment une telle politique peut être mise en œuvre. Dans le chapitre suivant, nous nous demanderons comment de telles politiques peuvent être précisément définies et nous nous demanderons si de telles politiques sont effectivement utilisables.

5.3 Mise en œuvre

Une fois les informations sensibles d'un système identifiées, nous supposons pour l'instant que nous savons comment définir et mettre à jour une politique de flux d'information. Nous nous concentrons tout d'abord sur la mise en œuvre d'une telle politique.
Nous pensons qu'il n'y a que deux choix possibles pour faire respecter une politique de flux d'information :
- Soit nous sommes capables de prévoir/contraindre tous les flux que les opérations du système rendront possibles.
- Soit nous sommes capables de suivre les flux d'information dynamiquement pour n'admettre que les flux respectant la politique.

1. Dans la suite de ce document nous noterons $\mathcal{C}$ l'ensemble des conteneurs et $\mathscr{P}(E)$ l'ensemble des parties d'un ensemble E

La première approche a été longtemps privilégiée. Les politiques fondées sur les modèles de Denning, Biba, Bell, LaPadula, Myers et Liskov ont été mises en œuvre par des méthodes qui contraignaient les opérations possibles sur le système ou qui prévoyaient par analyse statique tous les flux d'information possibles et n'autorisaient l'exécution d'un programme que si il avait été possible de prouver que celui ci n'engendrait aucun flux illégal. Cette approche a été largement étudiée et l'est encore mais elle n'est pas celle qui a retenu notre attention et sur laquelle nous avons travaillé. Nous avons préféré travailler avec un moniteur de flux d'information capable de suivre dynamiquement la propagation des informations surveillées sur un système. Nous pensons que cette approche offre l'avantage de pouvoir mettre en œuvre des politiques très précises. Cette approche nous permettra d'accepter l'utilisation de programmes qui *pourraient* réaliser des flux d'information illégaux, tant que ces programmes ne réalisent pas effectivement ces flux illégaux. Il existe quelques autres moniteurs de flux d'information comme Flume [KYB+07a], un moniteur de flux d'information pour le noyau Linux dont l'implémentation est faite dans le même esprit que celle de Blare mais gouverné par une politique moins précise. TaintDroid [EGgC+10] qui surveille les flux d'informations au niveau de la machine virtuelle Dalvik. Nous pouvons aussi citer Histar [ZBWKM06a] qui est un système d'exploitation dédié à la surveillance des flux d'information pour des politiques de flux spécifiées via des labels Asbestos.

5.3.1 Suivre l'information

Pour suivre dynamiquement la propagation de l'information, nous allons lui accrocher une étiquette. Ces étiquettes seront liées au conteneur et permettront de caractériser d'une part l'information qu'ils sont autorisé à contenir et d'autre part l'information qu'ils contiennent effectivement. Nous allons ensuite vouloir observer toutes les opérations entrainant un flux d'information sur le système surveillé. Lorsque qu'une opération entrainera un flux, les étiquettes seront mises à jour.

Une telle approche utilise des méthodes dites de *tainting*. Plus précisément nous utiliserons deux étiquettes. La première, s'appellera le *tag information* ou `itag` et dénotera l'ensemble des identifiants ayant servi au calcul de l'information qu'elle caractérise. Chaque conteneur d'information du système surveillé aura un `itag` qui caractérisera l'information que ce conteneur contient :

$$\forall c \in \mathcal{C}, c.\texttt{itag} \in \mathscr{P}(\mathcal{I} \cup \mathcal{X}))$$

identifie les informations utilisées pour le calcul du contenu de c.

La valeur des `itag` devra être mise à jour à chaque observation d'un flux d'information afin de continuer à assurer que la valeur du `itag` caractérise toujours le contenu du conteneur auquel il est attaché. Pour cela, le moniteur peut observer les actions qui modifient le contenu des conteneurs qu'il surveille et mettre à jour le `itag` d'un conteneur à chaque fois que son contenu a été modifié. Nous distinguons deux types d'opérations modifiant un contenu : celles qui remplacent totalement le contenu précédent (mode écrasement) ou celles qui ajoutent du contenu sans écraser le précédent (mode ajout).

Un moniteur observant un flux d'information $a \rightarrow b$ propage `a.itag` vers `b.itag` de la manière suivante :
(mode écrasement) `b.itag:= a.itag` si le contenu de b est écrasé par le contenu de a,
(mode ajout) `b.itag:= a.itag` $\cup$ `b.itag` sinon.

5.3.2 Projeter la politique et la mettre à jour

Nous venons de définir une politique de flux d'information comme une fonction définissant non pas les flux d'information autorisés mais plutôt comme les associations autorisées entre contenus et conteneurs. Une telle fonction définit pour chaque conteneur les différents mélanges d'informations que celui-ci est autorisé à contenir. Pour chacun de ces conteneurs, nous utiliserons une seconde étiquette notée *tag politique* ou `ptag` qui spécifiera exactement la politique de ce conteneur.

$$\forall c \in \mathcal{C}, c.\texttt{ptag} = \mathbb{P}_{\mathcal{C}}(c) \in \mathscr{P}(\mathscr{P}(\mathcal{I} \cup \mathcal{X}))$$

identifie les mélanges d'informations autorisés pour le calcul du contenu de c

Lorsque la politique de flux d'information devra être mise à jour, il suffira de modifier la fonction $\mathbb{P}_{\mathcal{C}}$ et de mettre à jour la valeur des `ptag` des conteneurs concernés. Décider de mettre à jour une politique est un travail délicat qui sera discuté au chapitre suivant. Cela fait, déployer la politique modifiée est, au contraire, une tâche très simple.

5.3.3 Légalité des flux d'information

Un flux $a \rightarrow b$ est légal si la classe de sécurité de l'information impliquée dans le flux est inférieure ou égale à la classe de sécurité du conteneur destination du flux. Dans le modèle que nous proposons, cela signifie que le contenu d'un conteneur après le flux est bien prévu (autorisé) par la politique de flux. Pour vérifier la légalité d'un flux d'information en utilisant les étiquettes que nous venons de proposer, il suffit de vérifier que le `itag` de la source de l'information (donc du conteneur a) est inclus dans l'un des éléments du `ptag` du conteneur destination du flux (du conteneur b) .

Un moniteur chargé de la mise en œuvre d'une politique de flux $\mathbb{P}_{\mathcal{C}}$ et observant un flux d'information $a \rightarrow b$ décide que ce flux est :

- **légal** en regard de $\mathbb{P}_{\mathcal{C}}$, si il existe $e \in$ `b.ptag` tel que
  - `a.itag` $\subseteq e$ si le contenu de b est écrasé par le contenu de a,
  - `a.itag` $\cup$ `b.itag` $\subseteq e$ sinon.

- **illégal** en regard de $\mathbb{P}_{\mathcal{C}}$ sinon.

Si l'on souhaite prévenir les flux interdits, il faut vérifier la légalité d'un flux avant que ce flux ait réellement lieu. Si l'on souhaite seulement alerter un expert, un administrateur qu'un flux interdit s'est produit, il suffit de faire cette vérification a postériori et de lever une alerte.

5.4 Implémentation

Le développement d'un moniteur de flux d'information est un travail techniquement difficile qui a demandé de gros efforts d'implémentation dans mon équipe depuis 2003. Les différentes implémentations de ce moniteur nommé Blare sont disponibles sur `https://www.blare-ids.org/`. Nous avons encadré le travail de Guillaume Hiet qui a développé un moniteur de flux d'information au niveau de la machine virtuelle Java [HMMVTT07]. Nous avons encadré avec Eric Totel le travail de thèse de T. Demongeot qui a proposé un moniteur inspiré de celui-ci dans un interpréteur BPEL permettant l'orchestration de services [DTTT11]. Nous avons enfin encadré le travail de développement de R. Andriatsimanefitra qui a porté un moniteur pour le système d'exploitation Android dédié aux téléphones et aux tablettes, cette implémentation est parfois nommée AndroBlare [And14].
Dans la suite nous utiliserons le nom générique Blare et spécifierons selon le contexte l'implémentation précise que ce terme désigne. Il pourrait être intéressant de compléter ces travaux en proposant une implémentation de ce modèle au niveau des bases de données pour surveiller les mélanges de données.
Dans toutes ces implémentations, les différences les plus notables sont

- l'implémentation effective des `itag` et `ptag` ;
- les moyens utilisés pour observer les flux d'information.

Dans tous les cas, il faut retenir que le moniteur doit se placer en coupure de toutes les opérations entrainant des flux d'information sur le système surveillé. C'est à dire que la maintenance des valeurs des `itag` doit avoir lieu juste avant que le flux se produise.
Dans les implémentations faites au niveau applicatif (dans la machine virtuelle Java ou dans l'orchestrateur BPEL) ou dans celles qui pourraient être développées à l'avenir, l'implémentation se base sur la sémantique du langage de programmation. Plus précisément, pour chaque instruction du langage entrainant un flux, une règle spécifie comment la valeur des `itag` doit être modifiée.
Dans les implémentations (existantes) faites au niveau du système d'exploitation, nous avons identifié que les flux d'information n'ont lieu que lors de l'exécution de certains appels système. Pour cela, il n'y a pas eu d'autres indices que la lecture même du code du noyau Linux. Cela demandera dans la suite à être raffiné. En effet, les appels système sont eux même des fonctions qui appellent d'autres fonctions parmi lesquelles une bonne partie ne réalisent aucun flux d'information. Il est donc possible de voir exécuter un appel système sans qu'il n'y ait eu de flux d'information, parce que le l'appel a échoué et retourné une erreur par exemple. Il existe des cas extrêmes dans lesquels le moniteur déduit l'occurrence d'un flux alors que ce flux n'a pas lieu, parce que la mise à jour des `itag` a été faite trop en amont de l'exécution du flux lui même. Nous sommes conscients de ces cas de faiblesse et travaillons à y remédier.
L'implémentation actuelle de Blare s'appuie aujourd'hui sur le framework LSM (Linux Security Module) prévu initialement pour la mise en œuvre de politiques de contrôle d'accès et qui permet d'effectuer des opérations juste avant l'accès aux ressources du système (juste avant la lecture d'un fichier par exemple).
Il existe assez peu de moniteur de ce type. Nous pouvons néanmoins citer Flume

[KYB+07b], Histar [ZBWKM06b] et Taintdroid [EGgC+10]. Flume est un moniteur de flux d'information qui peut se comprendre comme une sur-couche d'un système d'exploitation de type Unix. Flume considère les flux entre processus, fichiers et sockets. Flume maintient des tags sur les conteneurs d'information, ces tags dénotent les privilèges de l'information contenue et permettent de déterminer si un flux est légal où non. Histar est un système d'exploitation proche d'un système Unix qui a été développé pour permette le suivi de flux d'information. Histar maintient des tags proches de ceux de Flume. Nous reprochons à ces deux systèmes d'utiliser un système de tags issus de labels théoriques Asbestos [EKV+05]. Ce système de labels est difficile à comprendre et plus encore à administrer. Taintdroid quand à lui est un moniteur de flux qui observe comment l'information se propage au sein d'une machine virtuelle Dalvik sous Android. Taintdroid est connu pour avoir été le premier moniteur de flux sous Android, il a permis de mettre en évidence que la grande majorité des applications Android étaient responsables de fuite d'information sensibles vers de serveurs distants. TaindDroid ne maintient qu'une seule valeur de tag qui dénote le fait qu'un contenu soit contaminé ou non.
Comme décrit au début de ce chapitre, nous aimerions que l'implémentation d'un moniteur de flux soit à la fois correcte et complète : qu'elle tienne compte de tous les flux d'information et uniquement des flux d'information.

Correction d'un moniteur de flux d'information. La correction d'un moniteur de flux d'information est une propriété qui spécifie que ce moniteur est capable de ne prendre en compte que des véritables flux d'information. Au niveau applicatif, l'approche usuelle se base sur la sémantique des instructions du langage pour déduire une sur-approximation des flux d'information ayant lieu. Par exemple, lorsqu'un moniteur traite une affectation de la forme $a := b - c$, il applique avec raison une règle qui spécifie que le `itag` de a dépend, à la suite de l'instruction, des `itag` de b et c. Malheureusement, lorsqu'il traite l'instruction $a := 0 \times (b + c)$, il applique la même règle et à la suite du traitement de cette opération, `a.itag` est égal à `b.itag` alors qu'en réalité, la valeur du contenu de a est totalement indépendante du contenu de b puisqu'elle vaut toujours 0. Un moniteur de flux au niveau applicatif ne s'appuyant que sur la sémantique des instructions du langage de programmation est incorrect avec une telle approche. Nous pensons que la propriété de correction n'est pas atteignable pour un moniteur de flux au niveau applicatif en l'état de nos connaissances.
De la même façon, si un moniteur de flux au niveau système détecte qu'un processus est responsable d'un appel système `read` sur un fichier étiqueté par un itag I non vide, son rôle est de propager l'itag du fichier vers le processus. Le processus est alors marqué par I, comme contaminé par l'information que contenait le fichier. Hélas le moniteur ne peut pas savoir ce que le processus fait de cette information. Si le processus n'a pas utilisé cette information, par exemple l'information est stockée dans une variable qui n'est plus ensuite utilisée, le processus reste marqué même si l'information qu'il contient ne dépend pas de I. Par la suite, le moniteur considérera que les flux d'information partant de ce processus concernent I ce qui est une sur-approximation du comportement réel.
Si le moniteur est utilisé comme un détecteur de flux interdits, il est donc responsable

de faux positifs, c'est-à-dire d'alertes levées à mauvais escient. Si l'on souhaite amélioré la correction d'un moniteur de flux, il faut raffiner l'observation de ce moniteur. Pour raffiner un moniteur surveillant les flux au niveau système, on peut affiner la vue du moniteur en passant au niveau applicatif. Néanmoins, nous pensons qu'il restera toujours des cas où ce moniteur décidera à tort qu'il observe un flux d'information.

Complétude d'un moniteur de flux d'information au niveau du système d'exploitation Un moniteur de flux d'information est dit *complet* lorsqu'il est capable de prendre en compte tous les flux d'information explicites [2]. Nous pensons qu'il est possible de montrer que les flux d'information sont effectivement réalisés que lors de l'exécution de certains appels système et que dès lors il est envisageable d'atteindre une implémentation d'un moniteur de flux complet. Nous pensons qu'en déduisant les flux au plus près de la fonction les réalisant effectivement, ce moniteur aura un bon niveau de précision et ne sera pas à l'origine de trop de faux positifs.
Nous avons proposé à Laurent Georget de travailler sur ce problème et de proposer une méthode de validation formelle d'un moniteur de flux système. Laurent est encadré par Frédéric Tronel, Guillaume Piolle, Mathieu Jaume, et moi même. Laurent a débuté sa thèse en septembre 2014 et nous ne doutons pas qu'il sache apporter une réponse élégante à ce problème.

5.5 Bilan

Nous venons de présenter un modèle de fonctionnement d'un moniteur de flux d'information [VTTCM10]. Nous avons précisé que nous pensons que ce moniteur était nécessairement incorrect, c'est-à-dire qu'il pouvait toujours déduire de ses observations des flux d'information qui n'existaient pas réellement. Nous pensons néanmoins qu'il est possible de construire un moniteur de flux qui soit complet, c'est-à-dire qui soit capable d'observer tous les flux ayant réellement lieu. Ce travail est actuellement réalisé par Laurent Georget qui est en seconde année de doctorat.
Le modèle que nous venons de décrire est suffisamment générique pour être implémenté à tous les niveaux d'observation. Le principal problème que rencontrent ces implémentations est l'explosion des valeurs des `itags`. A partir d'un certain temps, il y a trop de contamination du système par les différentes informations surveillées. Nous avons identifié que ce problème venait essentiellement du fait de la centralisation de certains services dans le système. Par exemple, sous Android, le Binder est un driver qui est responsable de la communication inter-processus. Les applications Android tournent dans des processus distincts mais communiquent entre elles grâce à des mécanismes d'IPC tels que les intents et les Content Providers. Ces mécanismes sont fournis par le framework Java Android mais sont implémentés par un driver dans le noyau appelé Binder. Le Binder permet à une application d'appeler une méthode distante, implémentée dans une autre application tournant dans un autre processus. Comme beaucoup d'applications accèdent au Binder, celui ci est rapidement contaminé par beaucoup de valeurs

2. Rappelons que nous n'avons pas pour objectif de prendre en compte les flux implicites.

d'`itag` alors que le Binder n'est qu'un relai dans les demandes de communication qui n'interfèrent pas en réalité. Si Blare voit les processus comme des boites noires, il réalise une sur-approximation trop grossière des comportements réels. Pour répondre à ce problème, Radoniaina Andriatsimendefitra a raffiné la vue de Blare au cœur du Binder. Sous Android, il serait bienvenu de réaliser aussi ce travail pour *System Server* qui est utilisé pour rendre des services aux applications Android dans le système comme délivrer la liste des contacts, les coordonnées GPS etc. Ce problème pourrait être résolu avec une bonne gestion de la politique de sécurité : lorsqu'une information a beaucoup circulé dans le système, nous pourrions considéré qu'elle s'est altérée et que sa politique doit être mise à jour. Ce problème rejoint un problème plus général, celui de la déclassification d'information et la modification de politique. Nous reviendrons sur ce sujet en conclusion de ce document.

Chapitre 6

Construction et utilisation de politiques de flux d'information à grain fin

Nous venons de montrer comment il est possible de mettre en œuvre des politiques de flux d'information dites à grain fin en s'appuyant sur un moniteur de flux d'information. Ces politiques sont extrêmement précises puisqu'à l'échelle du système, elles permettent de préciser les informations, les mélanges d'information que peuvent contenir les processus, les fichiers, les sockets. Le problème sur lequel nous nous penchons maintenant est celui de la définition de telles politiques.
Nous avons exploré plusieurs approches : nous avons d'abord défini des politiques de flux d'information à partir d'une interprétation d'une politique de contrôle d'accès DAC [GVTTM09]. Nous avons proposé de calculer une politique de flux pour Blare à partir de politique AppArmor [GHTVTT11]. Enfin nous avons proposé de définir une politique pour un système Android manuellement afin de comprendre quels besoins réels seraient réellement exprimables dans une politique de flux [AGVTT12]. Aucun de ces travaux ne nous a totalement convaincu, nous avons donc proposé de construire une politique de flux d'information dans un formalisme imaginé spécialement pour cela. Ce travail est l'objet de ce chapitre. Nous présentons ici un langage de spécification de politiques de flux d'information. Ce langage permet la définition décentralisée de politiques de flux. Nous exprimons une propriété de cohérence et un mécanisme de composition garantissant que la composition de deux politiques cohérentes et encore une politique cohérente. Nous proposons un mécanisme de calcul de telles politiques. Enfin, nous mettons à l'épreuve les politiques ainsi calculées.

Pour définir une politique, il paraît illusoire de demander à un utilisateur, même expert, de définir l'ensemble des mélanges d'information possibles et pour chacun des conteneurs de son système. Il nous faudra donc proposer une solution telle que le calcul de la politique soit le plus automatique possible. Ce problème sera traité en section 6.4. En section 6.5, nous nous intéresserons ensuite à vérifier la pertinence de telles politiques.

6.1 Construction de politiques de flux

6.1.1 Qui est responsable de la définition d'une politique ?

Nous nous sommes posé la question de savoir à qui appartient de définir une politique de flux d'information ? Est-ce une définition centralisée, dans laquelle une seule entité est responsable de la définition de la politique ? Est-ce une définition décentralisée dans laquelle la politique globale du système est définie par composition de plusieurs politiques locales ? La première approche est celle qui a été proposée par les modèles de Biba, Bell, LaPadula, Denning, Brewer et Nash. La seconde vient du modèle de Myers et Liskov.

Nous pensons que seuls les propriétaires d'informations sont légitimes pour définir la politique de leurs propres données et de leurs propres conteneurs Nous avons donc proposé, comme Myers et Liskov, une approche décentralisée dans laquelle chaque propriétaire d'information considérée comme sensible par le propriétaire, définit la politique concernant ses propres informations et ses propres conteneurs. La politique du système est ensuite définie par composition.

Nous dirons dans la suite que les propriétaires d'information sont les applications. Derrière cette proposition se cachent plusieurs interprétations puisque derrière les applications se cachent les développeurs, les vendeurs, les utilisateurs de ces applications.

Les informations appartenant à ces applications sont leur code, leurs ressources (autre code, texte, musique, images, etc.) et les informations que ces applications sont susceptibles de créer lors de leur exécution. Dans cette approche, il peut y avoir un propriétaire noté *system* dont le statut n'est pas différent des autres, qui est le propriétaire des informations initialement existantes sur le système surveillé. Par exemple, dans le cas d'une politique pour un système d'exploitation, *system* est le propriétaire des données sensibles créées par défaut lors de l'installation du système d'exploitation.

Avec Stéphane Geller, nous avons proposé un langage appelé *Blare Security Policy Language* ou plus simplement BSPL qui offre une syntaxe et une sémantique claire, non ambigüe permettant à un propriétaire de spécifier la politique de ses données et de ses conteneurs. Ce langage est bien détaillé dans [GVTTM13]. Pour un propriétaire λ, ce langage permet de créer une politique $\mathbb{P}_\lambda$ destinée à définir comment les informations de ce propriétaire peuvent se disséminer dans le système et comment les conteneurs de ce propriétaire peuvent être contaminés par les informations identifiées comme sensibles des autres propriétaires. Cette politique, qui pourra être composée avec d'autres politiques à l'aide de l'opérateur $\oplus$ que nous détaillerons en 6.1.3, pourra ensuite être mise en œuvre par Blare.

6.1.2 Le point de vue des données ou celui des conteneurs ?

En utilisant BSPL, un propriétaire peut définir la politique de ses conteneurs envers les informations qu'il possède et celles des autres propriétaires. Il définit alors partiellement la fonction $\mathbb{P}_{\mathcal{C}}$ telle que nous l'avons utilisée jusqu'à maintenant.

Un propriétaire peut vouloir dualement définir la politique des données qu'il identifie comme sensibles. La politique qu'il définit alors traite de ses données envers les conteneurs (les siens ou ceux des autres propriétaires). Cette définition paraît légitime mais n'entre pas dans le cadre que nous avons mis en place jusqu'ici.

Prenons le cas d'un développeur d'anti-virus. En prenant le point de vue des ses conteneurs, le développeur d'un anti-virus pourrait vouloir spécifier qu'il pense que la base de données de signatures de virus est un conteneur très sensible et que ce conteneur ne peut contenir que des données n'appartenant qu'à lui même. Ce développeur pourrait aussi spécifier qu'un processus exécutant le code du scan du système est un conteneur d'information qui a le droit de contenir toutes les informations du système qu'elles soient sensibles ou non, qu'elles soient à lui ou non.

En prenant le point de vue des informations, ce développeur pourrait aussi vouloir spécifier que le contenu de la base de données de signatures de virus qu'il manipule est un contenu très sensible et que cette information ne peut aller que dans des conteneurs n'appartenant qu'à lui même.

Tout cela pose un problème de cohérence. S'il semble légitime que le processus de scan d'un anti-virus accède à toutes les données sensibles d'un système, il faut aussi que ceci soit accepté par tous les propriétaires de ces informations sensibles existant sur le-dit système.

Ces considérations nous ont menée à proposer une seconde fonction :

Une politique de flux définie en prenant le point de vue des informations est une fonction

$$\mathbb{P}_{\mathcal{I}\cup\mathcal{X}} : \mathcal{I} \cup \mathcal{X} \to \mathscr{P}(C \times \mathscr{P}(\mathcal{I} \cup \mathcal{X}))$$

$$i \mapsto \bigcup_{c_k \text{conteneur d'information}} \{(c_k, \{\{i_{j_1}, \dots, i_{j_n}\}, \dots \{i_{m_1}, \dots, i_{m_n}\}\})\}$$

où c_k est un conteneur autorisé à contenir les mélanges d'information $\{\{i, i_{j_1}, \dots, i_{j_n}\}, \dots \{i, i_{m_1}, \dots, i_{m_n}\}\}$

Cette seconde fonction $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$ est évidemment duale de la première $\mathbb{P}_{\mathcal{C}}$ définie dans le chapitre précédent. Définir une politique peut donc se faire par la définition, éventuellement partielle, de ces deux fonctions. Une telle définition est dite cohérente si elle exprime des contraintes non contradictoires. Plus précisément, si l'on autorise un ensemble d'informations I à se trouver dans un conteneur c, on doit aussi autoriser toutes les informations $i \in I$ à se mélanger avec les autres informations de I, et réciproquement.

Une politique de flux définie par deux fonctions $\mathbb{P}_{\mathcal{C}}$ et $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$ est cohérente si $\forall c \in \mathcal{C}, \forall s \in \mathscr{P}(\mathcal{I} \cup \mathcal{X})$, les faits suivants sont équivalents :

$$I \in \mathbb{P}_{\mathcal{C}}(c) \tag{6.1}$$

$$\forall i \in I, (c, I \setminus \{i\}) \in \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(i) \tag{6.2}$$

Lorsqu'une politique est cohérente, les deux fonctions peuvent se définir l'une en fonction de l'autre de la manière suivante :

$$\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(i) = \{(c, I) | c \in \mathcal{C} \wedge I \cup \{i\} \in \mathbb{P}_{\mathcal{C}}(c)\}$$

$$\mathbb{P}_{\mathcal{C}}(c) = \bigcup_{i \in \mathcal{I}\cup\mathcal{X}} \{I \cup \{i\} | (c, I) \in \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(i)\}$$

A partir de maintenant, une politique de flux $\mathbb{P}$ sera la donnée de ces deux fonctions $\mathbb{P} = (\mathbb{P}_{\mathcal{C}}, \mathbb{P}_{\mathcal{I}\cup\mathcal{X}})$.

Naturellement, ces deux fonctions sont redondantes et nous pourrions choisir d'imposer l'une ou l'autre. Néanmoins, nous visons ici à offrir le moyen pour l'utilisateur non spécialiste de spécifier simplement sa politique et nous pensons que naturellement cet utilisateur spécifiera ses contraintes de confidentialité à l'aide $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$ et ses contraintes d'intégrité à l'aide de $\mathbb{P}_{\mathcal{C}}$. Nous allons détailler en 6.2 comment un propriétaire d'information peut construire une politique par définition partielle de $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$ et $\mathbb{P}_{\mathcal{C}}$. Cette politique ne sera validée que si elle définit une politique cohérente, elle pourra alors être composée avec d'autres politiques cohérentes.

6.1.3 Composition de politiques

Nous détaillons ici comment composer deux politiques cohérentes pour obtenir une politique composée elle même cohérente.

La politique $\mathbb{P}$ obtenue par composition de deux autres politiques $\mathbb{P}_1 = (\mathbb{P}_{\mathcal{C}_1}, \mathbb{P}_{\mathcal{I}_1 \cup \mathcal{X}_1})$ et $\mathbb{P}_2 = (\mathbb{P}_{\mathcal{C}_2}, \mathbb{P}_{\mathcal{I}_2 \cup \mathcal{X}_2})$ est notée $\mathbb{P}_1 \oplus \mathbb{P}_2$ et est définie par $(\mathbb{P}_{\mathcal{C}}, \mathbb{P}_{\mathcal{I} \cup \mathcal{X}})$ où :

1. $(\mathcal{I} \cup \mathcal{X}) = ((\mathcal{I}_1 \cup \mathcal{I}_2) \cup (\mathcal{X}_1 \cup \mathcal{X}_2))$,

2. $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$,

3. $$\begin{array}{rcl} \mathbb{P}_{\mathcal{C}}(c) & = & \{I \in \mathbb{P}_{\mathcal{C}_1}(c) | I \subseteq (\mathcal{I}_1 \cup \mathcal{X}_1) \backslash (\mathcal{I}_2 \cup \mathcal{X}_2)\} \\ & \cup & \{I \in \mathbb{P}_{\mathcal{C}_2}(c) | I \subseteq (\mathcal{I}_2 \cup \mathcal{X}_2) \backslash (\mathcal{I}_1 \cup \mathcal{X}_1)\} \\ & \cup & \{I_1 \cap I_2 | I_1 \in \mathbb{P}_{\mathcal{C}_1}(c) \wedge I_2 \in \mathbb{P}_{\mathcal{C}_2}(c)\} \end{array}$$

4. $$\begin{array}{rcl} \mathbb{P}_{\mathcal{I} \cup \mathcal{X}}(ix) & = & \{(c, I) \in \mathbb{P}^1_{\mathcal{I}_1 \cup \mathcal{X}_1}(ix) | I \cup \{ix\} \subseteq (\mathcal{I}_1 \cup \mathcal{X}_1) \backslash (\mathcal{I}_2 \cup \mathcal{X}_2)\} \\ & \cup & \{(c, I) \in \mathbb{P}^2_{\mathcal{I}_2 \cup \mathcal{X}_2}(ix) | I \cup \{ix\} \subseteq (\mathcal{I}_2 \cup \mathcal{X}_2) \backslash (\mathcal{I}_1 \cup \mathcal{X}_1)\} \\ & \cup & \{(c, I_1 \cap I_2) | (c, I_1) \in \mathbb{P}^1_{\mathcal{I}_1 \cup \mathcal{X}_1}(ix) \wedge (c, I_2) \in \mathbb{P}^2_{\mathcal{I}_2 \cup \mathcal{X}_2}(ix)\} \end{array}$$

Autrement dit,

1. l'ensemble des informations pris en compte par la politique composée est l'union des informations des politiques entrant dans la composition,
2. l'ensemble des conteneurs pris en compte par la politique composée est l'union des conteneurs des politiques entrant dans la composition,
3. un conteneur a le droit de contenir

   les mélanges d'information autorisés dans la politique 1 et non pris en compte par la politique 2

   \+ les mélanges d'information autorisés dans la politique 2 et non pris en compte par la politique 1

   \+ les mélanges d'information autorisés par les deux politiques lorsqu'ils sont effectivement pris en compte dans les deux politiques.
4. une même information a le droit de

   se mélanger et se disséminer comme prévu par la politique 1 avec toutes les informations spécifiées dans la politique 1 mais n'apparaissant pas dans la politique 2.

   \+ se mélanger et se disséminer comme prévu par la politique 2 avec toutes les informations spécifiées dans la politique 2 mais n'apparaissant pas dans la politique 1.

   \+ se mélanger et se disséminer comme prévu à la fois par les politiques 1 & 2 avec toutes les informations spécifiées dans les deux politiques.

La politique composée $\mathbb{P}$ est obtenue par une opération qui s'apparente à une union disjointe des politiques $\mathbb{P}_1$ et $\mathbb{P}_2$. Si ces politiques sont cohérentes alors, intuitivement, la politique composée reste cohérente car les contraintes qui la composent sont issues soit uniquement de $\mathbb{P}_1$, soit uniquement de $\mathbb{P}_2$ soit elles étaient communes à $\mathbb{P}_1$ et $\mathbb{P}_2$ elles sont donc non contradictoires. La preuve formelle de la propriété de cohérence de la politique composée est développée dans [GVTTM13].

6.2 *Blare Security Policy Language*

Avec Stéphane Geller, nous avons proposé un langage (BSPL) permettant de spécifier une politique Blare [GVTTM13]. Avec ce langage, il devrait être simple de définir la nature exacte des informations à surveiller, leur localisation initiale, l'identification des conteneurs d'information, les mélanges d'information autorisés pour chacun des conteneurs pris en charge. BSPL repose sur une syntaxe XML et sa sémantique détermine les fonctions $\mathbb{P}_{\mathcal{C}}$ et $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$.
Une politique définie via BSPL se compose de deux éléments : le premier traite des informations et permet de calculer la fonction $\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$ tandis que traite le second des conteneurs et permet de calculer la fonction $\mathbb{P}_{\mathcal{C}}$. Les informations définies dans le premier élément sont caractérisées par

- leur alias qui permet de les identifier et correspond finalement à un élément nouveau i dans $\mathcal{I} \cup \mathcal{X}$;
- leur conteneur originel ;
- leur propriétaire. En toute généralité, une politique BSPL traite d'informations appartenant à plusieurs propriétaires.
- leur type (est ce un exécutable ou non ?) ;
- la réaction attendue du moniteur. BSPL permet de définir le comportement attendu du moniteur lorsqu'il observe un flux inattendu : bloquer le flux ou lever une alerte. Actuellement seul le second choix est implémenté dans Blare.
- le comportement par défaut lors du calcul de la politique composée. Lorsqu'un autre propriétaire a demandé l'autorisation d'un flux impliquant cette information et pour les conteneurs non définis dans la politique : demander l'avis du propriétaire, toujours accepter ou toujours refuser.

Pour chaque information identifiée, une liste de paires (mélange d'informations, conteneur), indiquant que l'information identifiée peut s'intégrer au mélange et se trouver dans le conteneur spécifié. Les conteneurs étant eux même identifiés dans le second élément de la politique.
Et dualement, le second précise l'identification des conteneurs :

- leur localisation originelle ;
- leur propriétaire ;
- leur type (est ce un fichier, un processus ou encore un utilisateur ?) ;
- la réaction attendue du moniteur lorsqu'il observe un flux interdit ;
- ce qu'autorise le propriétaire lorsqu'une autre politique réclame le droit de disséminer des informations non spécifiées ici dans le conteneur (demander au propriétaire du

conteneur, toujours accepter, toujours refuser).
Pour chacun de ces conteneurs, un élément de BSPL précise les informations que ce conteneur peut recevoir.
La figure 6.1 est un schéma xsd définissant la structure et le contenu exact d'une politique écrite en BSPL.
Les règles définies dans les figures 6.2 et 6.2 définissent la sémantique d'une politique écrite en BSPL. Elles proposent l'utilisation de deux tables $\mathbb{T}_{\mathcal{I}\cup\mathcal{X}}$ et $\mathbb{T}_{\mathcal{C}}$ pour garder trace de l'identification précise des informations et des conteneurs. Ces deux tables complètent le moniteur et servent à comprendre à tout moment comment la politique a été définie et par qui.

6.2.1 Implémentations existantes

Lors de son stage de L3 l'été 2013, Thomas Saliou a réalisé un compilateur et un outil de gestion des politiques BSPL judicieusement nommé *BSPL policy manager* permettant de calculer mais aussi de composer des politiques BSPL pour des applications Android. Dans ce cas, les applications Android sont les propriétaires des informations qu'elles apportent sur le système. Le premier élément de son implémentation parcourt un fichier `.bspl` et vérifie donc qu'il est syntaxiquement correct. Si tel est le cas, il vérifie ensuite que la politique définie via ce fichier est complète, c'est-à-dire qu'elle ne traite que d'informations et de conteneurs parfaitement identifiés. Un conteneur ou une information est parfaitement identifié soit parce que le fichier BSPL le/la décrit soit parce qu'il/elle a été décrit(e) auparavant et existe donc dans une des tables $\mathbb{T}_{\mathcal{I}\cup\mathcal{X}}$ ou $\mathbb{T}_{\mathcal{C}}$. Si la politique spécifiée est à la fois syntaxiquement correcte, et complète, *BSPL policy manager* vérifie qu'elle est cohérente. Si tel est le cas, la politique peut alors être composée avec celle en vigueur sur le système puis être appliquée sur le système en remplacement de la précédente. Elle est appliquée via le calcul des `itag` et `ptag` de tous les conteneurs du système surveillé. Ces tags sont associés aux conteneurs à l'aide des outils `setinfo` et `setpolicy`. Elle est ensuite surveillée par Blare.

6.3 Utilisations possibles

Nous avons proposé un mode d'utilisation possible pour Blare et le langage BSPL dans l'univers Android. Plus précisément, nous proposons de définir une politique par défaut pour le système Android. Thomas Saliou a spécifié une telle politique durant son stage. Ensuite, nous proposons que chaque application installée par la suite soit munie d'une politique BSPL qui spécifiera les éventuelles informations et conteneurs sensibles de l'application ainsi que les informations sensibles appartenant à d'autres applications auxquelles cette application voudrait accéder. Cette politique sera vérifiée par le *policy manager*. Si elle est bien écrite et complète, elle pourra être composée avec la politique courante et la composition résultante appliquée sur le système. Le moniteur Blare pourra ensuite vérifier que la politique est bien respectée. Nous pensons que ce mode d'utilisation permettrait d'épurer le marché des applications en écrivant ensuite un ensemble de critères permettant de vérifier que la politique de flux spécifiée par une application

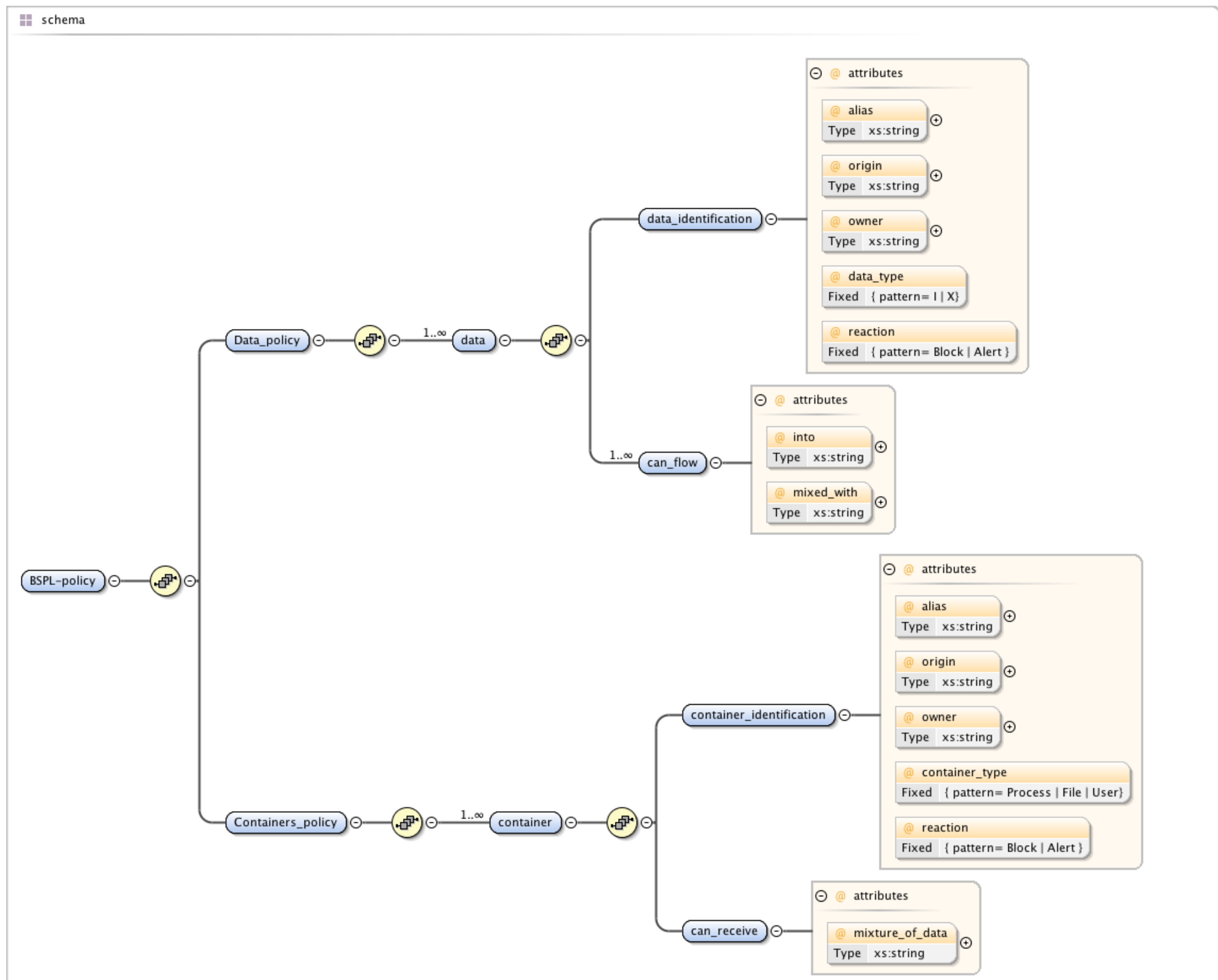


Figure 6.1 – Schéma XSD définissant la structure générale d'une politique en BSPL

Règle 1 : le point de vue des informations ($\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}$)

```
data_identification = {
        alias = al;
        origin = or;
        owner = own;
        data_type = type
        reaction = reac}, }
can_flow = {
        into = c;
        mixed_with_data = mix;
 can_flow = {...
```

if$(al, _, _, _, _) \notin \mathbb{T}_{\mathcal{I}\cup\mathcal{X}}$ then $\mathbb{T}_{\mathcal{I}\cup\mathcal{X}} \leftarrow \mathbb{T}_{\mathcal{I}\cup\mathcal{X}} \cup \{(al, or, own, type, reac)\}$;
for each can_flow element do

$$\mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(al) \leftarrow (c, mix)$$
$$\forall al' \in mix\ \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(al') \leftarrow (c, (mix \cup \{al\}) \setminus \{al'\})$$
$$\mathbb{P}_{\mathcal{C}}(c) \leftarrow \mathbb{P}_{\mathcal{C}}(c) \cup (mix \cup \{al\})$$
done

FIGURE 6.2 – Première partie des règles de sémantique de BSPL

n'est pas une politique dangereuse (par exemple, elle ne demande pas d'accéder à certaines informations sensibles). Nous pensons que le travail de validation d'une politique est plus simple à mettre en œuvre à grande échelle que celui qui consiste à analyser statiquement le code d'une application.
Néanmoins, cette proposition, si elle est alléchante, pose trois problèmes :

- Chaque application doit être munie d'une politique et, à ce stade, il semble encore compliqué de construire de telles politiques le plus automatiquement possible.
- Ces politiques doivent être pertinentes ... c'est-à-dire qu'elles doivent permettre de détecter des flux d'information non prévus par la politique tels que ces flux correspondent effectivement à des actions dangereuses pour le système et non pas simplement des flux inoffensifs mais n'existant pas dans la politique parce que celle ci n'est pas suffisamment complète.
- Les marchés d'applications Android maintiennent un outil de validation de politiques.

Nous nous sommes penchés sur les deux premiers problèmes qui seront étudiés en 6.4 et 6.5. Nous n'avons pas encore étudié le troisième.

6.4 Automatiser la construction des politiques

Construire une politique pour le moniteur Blare (via BSPL ou non) revient à être capable de prévoir l'ensemble des conteneurs (fichiers / processus / socket) dans lesquels pourrait se trouver chaque mélange autorisé d'informations sensibles. Cela semble fastidieux et inatteignable. Nous pourrions imaginer qu'un développeur est à même de

Règle 2.1 : le point de vue des conteneurs, le cas des utilisateurs ($\mathbb{P}_\mathcal{C}$)

$$\frac{\begin{array}{l}\texttt{container_identification} = \{ \\ \quad \textit{alias} = u; \\ \quad \textit{origin} = or; \\ \quad \textit{owner} = own; \\ \quad \textit{container_type} = User \\ \quad \textit{reaction} = reac \quad \} \\ \texttt{can_receive} = \{ \\ \quad \textit{mixture_of_data} = mix; \} \end{array}}{\text{if}(u, or, own, User, _) \notin \mathbb{T}_\mathcal{C} \quad \begin{array}{l} \text{then } \mathbb{T}_\mathcal{C} \leftarrow \mathbb{T}_\mathcal{C} \cup \{(u, or, own, User, reac)\}; \\ \mathbb{P}_\mathcal{C}(al) \leftarrow mix; \\ \forall ix \in mix,\ \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(al) \leftarrow (c, (mix \setminus \{ix\})) \end{array}}$$

Règle 2.2 : le point de vue des conteneurs, le cas des fichiers ($\mathbb{P}_\mathcal{C}$)

$$\frac{\begin{array}{l}\texttt{container_identification} = \{ \\ \quad \textit{alias} = al; \\ \quad \textit{origin} = or; \\ \quad \textit{owner} = own; \\ \quad \textit{container_type} = File \\ \quad \textit{reaction} = reac \quad \} \\ \texttt{can_receive} = \{ \\ \quad \textit{mixture_of_data} = mix; \} \end{array}}{\text{if}(al, _, _, _, _) \notin \mathbb{T}_\mathcal{C} \quad \begin{array}{l} \text{then } \mathbb{T}_\mathcal{C} \leftarrow \mathbb{T}_\mathcal{C} \cup \{(al, or, own, reac)\}; \\ \mathbb{P}_\mathcal{C}(al) \leftarrow mix \\ \forall ix \in mix,\ \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(al) \leftarrow (c, (mix \setminus \{ix\})) \end{array}}$$

Règle 2.3 : le point de vue des conteneurs, le cas des processus ($\mathbb{P}_\mathcal{C}$)

$$\frac{\begin{array}{l}\texttt{container_identification} = \{ \\ \quad \textit{alias} = al; \\ \quad \textit{origin} = or; \\ \quad \textit{owner} = own; \\ \quad \textit{container_type} = Process \\ \quad \textit{reaction} = reac \quad \} \\ \texttt{can_receive} = \{ \\ \quad \textit{mixture_of_data} = mix; \} \end{array}}{\text{if}(al, _, _, _, _) \notin \mathbb{T}_\mathcal{C} \quad \begin{array}{l} \text{then } \mathbb{T}_\mathcal{C} \leftarrow \mathbb{T}_\mathcal{C} \cup \{(\mathcal{E}xecute(al), or, own, reac)\}; \\ \mathbb{P}_\mathcal{C}(\mathcal{E}xecute(al)) \leftarrow mix \\ \forall ix \in mix,\ \mathbb{P}_{\mathcal{I}\cup\mathcal{X}}(al) \leftarrow (c, (mix \setminus \{ix\})) \end{array}}$$

FIGURE 6.3 – Dernière partie des règles de sémantique de BSPL

```
[37.010715] file /data/app/com.rovio.angrybirdsspace.ads-1.apk 32791 > process ndroid.launcher:launcher-loader 339 > itag[3]
[37.010952] file /data/app/com.rovio.angrybirdsspace.ads-1.apk 32791 > process ndroid.launcher:launcher-loader 339 > itag[3]
[37.019919] file /data/data/com.android.providers.media/databases/internal.db 57471 > process d.process.media:MediaScannerSer 360 > itag[3]
```

FIGURE 6.4 – Extrait d'un journal Blare obtenu lors de la surveillance de l'application Angry Birds Space

deviner les conteneurs que l'information qu'il manipule peut contaminer en un saut. Cela semble moins évident à plus d'un saut. Il n'y a qu'un moniteur de flux d'information qui peut savoir précisément comment une information se dissémine dans le système.

Pour construire une politique BSPL pour Blare, nous avons donc proposé d'utiliser Blare lui même.

Nous avons proposé une méthode de construction *semi-automatique* de politique BSPL et nous avons expérimenté cette approche dans l'environnement Android. De manière très simplifiée, sous Android, un utilisateur peut télécharger des applications via des magasins d'applications pour personnaliser son téléphone. Ces applications sont livrées sous forme compressée dans une archive `.apk` qui contient toutes les ressources nécessaires à l'application comme son bytecode, ses images, sa musique. Nous proposons d'apprendre le comportement de l'application dans un environnement considéré comme sain pour en déduire la politique BSPL. Pour cela, nous proposons d'identifier une seule information : le contenu de l'archive `.apk` de l'application. Cette information est identifiée sur le système au téléchargement de l'archive et avant l'installation de l'application par le système. Dans ce système, la politique de tous les conteneurs est réduite à vide ce qui implique que chaque flux d'information observé par Blare sera considéré comme une alerte et donc enregistré dans son journal. La figure 6.4 est un extrait d'un journal Blare. Une entrée dans ce journal décrit un flux d'information unitaire entre deux conteneurs surveillés. Une telle entrée est de forme générale :

```
[Time]
    [type source] [nom source] [identifiant source]
                               >
    [type destination] [nom destination] [identifiant destination]
[itag]
```

Elle décrit l'heure système de l'observation du flux, l'identification du conteneur source par `[type source]` qui décrit le type du conteneur (fichier, processus, socket), `[nom source]` qui décrit son nom et enfin `[identifiant source]` qui décrit son identifiant système (pid ou inode). De la même manière, `[type destination]` `[nom destination]` `[identifiant destination]` identifient précisément le conteneur destination du flux. Enfin l'entrée du log Blare contient la valeur de l'`itag` de l'information impliquée. Ainsi le tout petit extrait du journal de Blare de la figure 6.4 décrit trois flux d'information : les deux premiers ont la même source et la même

destination à savoir l'archive contenant l'application Angry Birds Space (voir `http://space.angrybirds.com/launch/`) et le processus Android responsable du menu du téléphone permettant le lancement des l'applications.
Un tel journal est évidemment assez volumineux puisqu'il contient des dizaines de milliers d'entrées pour une surveillance de quelques minutes. Nous aborderons ce problème dans le chapitre suivant. Pour l'instant, nous acceptons l'idée que la gestion du journal est un problème abordable. Malgré tout, dès maintenant, nous nous devons de remarquer qu'un tel log est volumineux parce qu'extrêmement redondant. L'extrait de log de la figure 6.4 l'était déjà. En effet, les flux sont observés au niveau des appels système engendrant des flux comme `read, write, exec` etc. Pour une même instruction d'un programme, un appel système peut être exécuté plusieurs fois. Par exemple, pour lire un fichier, il faudra répéter plusieurs fois l'appel système read qui ne lit souvent le fichier que par bloc de 4096 octets, ce qui correspond à la taille d'une page mémoire. Ce qui nous intéresse ici est plutôt de savoir combien de conteneurs uniques apparaissent dans ces logs et combien de flux différents entre ces différents conteneurs peuvent avoir lieu. La réponse à cette question est plutôt rassurante car nous avons recensé environ une centaine de conteneurs différents pour les applications que nous avons étudiées (environ 300). Par exemple, la figure 6.4 est extraite d'un log Blare de 49037 entrées qui n'impliquent en fait que 103 conteneurs d'information différents.

A l'aide du journal de Blare, nous pouvons facilement construire une politique BSPL en utilisant l'algorithme suivant :

- identifier le contenu de l'archive de l'application ;
- identifier son conteneur original et l'autoriser à contenir cette information ;
- pour chaque entrée dans le journal de Blare décrivant un flux $c_1 \rightarrow c_2$ impliquant l'itag lié à l'application surveillée, ajouter une entrée BSPL décrivant c_2 et autorisant le conteneur c_2 à contenir l'information identifiée précédemment, si cette entrée n'existe pas déjà.

Nous obtenons dès lors des politiques Blare construites par surveillance d'exécutions de l'application considérées comme saines. Un extrait de politique BSPL pour Angry Birds Space est donné en table 6.1.

6.4.1 Bilan des politiques BSPL

Nous avons proposé un langage permettant de spécifier des politiques pour Blare, puis nous avons proposé une méthode pour initier la construction de telles politiques. Ce travail est encore à approfondir puisque :
- nous n'avons aucune preuve de la complétude de ces politiques. Dans ce qui vient d'être présenté, les politiques sont calculées à partir d'exécutions des applications. Dans la phase de test de notre approche, ces exécutions sont provoquées par un *vrai* utilisateur qui *utilise* l'application. Nous ne pouvons pas affirmer que ces exécutions étaient suffisamment exhaustives pour être représentatives du comportement.
- la construction de la politique est bien avancée mais non terminée ; en particulier, il

```
<BSPL_policy>
  <Data_policy>

      <data>
            <data_identification
                  alias="angrybirds"
                  case_of_unknown_containers="not defined"
                  data_type="I"
                  origin="/data/app/com.rovio.angrybirdsspace.ads-1"
                  reaction="not defined" />
....

      </data>
  </Data_policy>

  <Containers_policy>

      <container>
            <container_identification
                        alias="c_angrybirds"
                        case_of_unknown_data="not defined"
                        container_type="File"
                        origin="/data/data/com.rovio.angrybirdsspace.ads
                           /files/highscores.lua.tmp"
                                       owner="not defined"
                        reaction="alert" />

            <can_receive mixture_of_known_data="angrybirds" />
      </container>

      <container>
            <container_identification
                  alias="c_angrybirdsX_41"
                  case_of_unknown_data="not defined"
                  container_type="File"
                  origin="/data/data/com.android.email/databases/
                     EmailProvider.db-journal"
                  owner="not defined"
                  reaction="not defined" />
            <can_receive mixture_of_known_data="angrybirds" />
      </container>
  </Containers_policy>
</BSPL_policy>
```

TABLE 6.1 – Un extrait de politique BSPL pour Angry Birds Space

faut encore renseigner le comportement attendu lors de la détection de flux interdit, le comportement attendu en cas d'information ou conteneur non pris en compte

Néanmoins, pour avancer sur ce sujet et avoir une idée de la pertinence de l'approche, nous avons proposé de répondre au premier point en explorant l'application avec minutie lors de la phase de surveillance et nous avons proposé des critères par défaut pour répondre aux questions soulevées dans le second point.

6.5 Quelques expérimentations

Nous avons voulu savoir si, *dans cet état*, ces politiques étaient pertinentes. Pour cela, nous avons calculé selon la procédure précédente des politiques pour deux versions de trois applications Android dont nous disposions de versions infectées par trois malware différents. Nous avons répété l'expérience suivante :

1. calculer la politique pour l'application saine ;
2. exécuter l'application saine, la surveiller par Blare paramétré avec la politique calculée au point précédent et vérifier que peu d'alertes sont émises ;
3. exécuter l'application vérolée, la surveiller par Blare avec la même politique et vérifier que des alertes sont émises lors de l'exécution du malware.

Pour cette expérimentation, nous avions à notre disposition trois couples d'applications (saines/vérolées). Le premier couple est le jeu Angry Birds Space et une version infectée par une variante du malware Lena [LPC13]. Ce malware gagne les privilèges administrateur en exploitant une vulnérabilité du système, ensuite Lena modifie et/ou remplace certains fichiers existants sur le système. Ces fichiers permettent à un attaquant de prendre le contrôle distant du téléphone. Lena réalise une attaque contre l'intégrité du système puisqu'il modifie des conteneurs du système.

Notre second couple d'applications est une application permettant de scanner l'empreinte digitale d'un utilisateur ce qui lui permet de déverrouiller son téléphone si cette empreinte est reconnue comme celle de l'utilisateur légitime et une version de cette application infectée par le malware DroidKungFu1. Ce malware, découvert en mai 2011 gagne lui aussi des privilèges administrateur en exploitant une vulnérabilité du système puis installe une nouvelle application sur le téléphone à l'insu de l'utilisateur.

Notre dernier couple d'applications est un jeu et une version de ce jeu infectée par le malware BadNews. Ce malware, découvert en avril 2013, exécute les commandes qu'il reçoit d'un serveur distant. Ces commandes varient permettant par exemple le lancement de nouvelles applications, l'ajout d'icônes sur le menu du téléphone menant l'utilisateur à utiliser le navigateur pour visiter des pages web ou lui proposant d'installer de nouvelles applications.

La première partie de l'expérience a consisté à utiliser les trois versions saines des applications sous surveillance de Blare lui même paramétré à chaque fois par la politique de l'application calculée comme expliqué en section 6.4. L'idée était surtout de savoir si la politique calculée était suffisamment complète pour nous permettre d'utiliser l'application saine sans émettre d'alertes inutiles qui seraient vues comme des faux positifs. Nous

pensions que nos politiques avaient de bonnes chance d'être incomplètes. Nous avons utilisé un filtre permettant de décider si des alertes émises par Blare sont de véritables alertes ou non. Ce filtre élimine les alertes résultant des flux :
- vers `System Server` car ce processus est utilisé pour demander/délivrer des services. Ce type de flux devrait être pris en compte dans BSPL si celui ci était capable de gérer les expressions régulières ce qui n'est pas encore le cas dans l'implémentation existante ;
- vers l'ensemble des services accessibles via `System Server` pour les même raisons que précédemment ;
- vers le répertoire `/data/data/monapp` de l'application car il peut être le lieu de création de fichiers temporaires aux noms aléatoires évidemment non pris en compte par la politique.

Ce même filtre a été utilisé pour les deux versions des trois applications utilisées lors de l'expérience. Les résultats sont encourageants car nous n'avons relevé aucune alerte lors de l'utilisation des trois versions saines de nos applications avec leur politique `.bspl` générée comme décrit en 6.4. Cela signifie que construire une politique BSPL qui accepte le comportement normal de l'application devrait pouvoir se faire selon la méthode décrite précédemment moyennant deux choses :
- la transformation de notre filtre en éléments de politique et donc la prise en compte des expressions régulières dans le *policy manager* (ce qui semble relativement simple).
- complétude de la politique. Dans cette expérience, nous n'avons pas relevé d'alertes produites par Blare lors d'exécutions considérées comme saines. Peut être avons nous eu *de la chance*. Nous n'avons pas de résultats théoriques permettant de savoir si notre politique prend en compte toutes les actions dites normales. Nous savons seulement qu'elle en contient beaucoup. Nous retrouverons exactement ce problème dans le chapitre suivant et en discuterons alors.

Ce travail s'est concentré sur trois paires d'applications, essentiellement parce qu'il a été difficile d'exhiber exactement les même versions saine et infectée d'une même application.

Ce travail a été publié dans [ASVTT13] (*Best Student Paper Awards*) et dans une version étendue publiée dans [AVTT14b].

6.6 Bilan

Nous avons présenté ici des travaux réalisés avec Stéphane Geller, Thomas Saliou et Radoniaina Andriatsimandefitra. Ces travaux se sont concentrés sur la proposition d'un langage de définition de politiques de flux d'information à grain fin en vue d'être mise en œuvre par Blare. Ce langage permet une définition décentralisée de la politique globale d'un système. Pour cela, nous avons dû détailler un peu plus le modèle de politique, définir une notion de cohérence des politiques de flux dans ce cadre, enfin nous avons défini un opérateur permettant la composition de politique. Cet opérateur conserve la cohérence. Nous avons ensuite éprouvé l'utilisation de ce langage en explorant deux axes. Le premier axe répond à la question *"Comment peut-on imaginer construire des*

politiques BSPL pour des applications réelles ?". Le second axe est en réalité la continuité du second et s'intéresse à la pertinence de politiques ainsi définies. Cette partie est plus exploratoire mais est néanmoins encourageante.
Une voie de recherche intéressante de ce travail serait d'étendre le langage BSPL pour qu'il puisse tenir compte de la possibilité de déclassifier des mélanges d'informations. Nous reviendrons sur ce point en conclusion de ce document.

Chapitre 7

Comprendre et caractériser le comportement des applications

Dans le chapitre précédent, nous avons déjà remarqué que le journal de Blare nous apprenait exactement quels conteneurs d'information sont contaminés par l'exécution d'une application. En réalité, ces logs nous apprennent plus que cela : ils nous apprennent l'ordre des flux, leur source et leur destination. Hélas la représentation que nous en avons via le journal de Blare est peu exploitable : l'information est là mais elle est bien camouflée. Nous avons donc proposé d'utiliser une structure arborescente facilement visualisable pour mettre en évidence l'information cachée dans les logs de Blare. Cette structure est détaillée en section 7.1. Nous verrons que cette structure se révèle utile pour la compréhension des comportements malveillants en section 7.2. Nous verrons aussi qu'il est possible de l'utiliser pour calculer des signatures comportementales des malware Android. Nous éprouverons enfin ces signatures sur un ensemble d'applications Android. Le travail présenté dans ce chapitre reprend essentiellement les travaux de thèse de Radoniaina Andriatsimandefitra encadrés du printemps 2011 à fin 2014.

7.1 Représenter les flux d'information

Un flux d'information est une relation entre deux conteneurs (qui sont la source et la destination de l'information) et d'une donnée dépendant d'une information qui a auparavant été identifiée comme sensible. Un ensemble de flux d'information permet de connecter des conteneurs d'information si ils ont été contaminés par la même information dans une même suite de flux. Intuitivement, la structure sous-jacente est évidemment un graphe et nous avons donc proposé de représenter un ensemble de flux

d'information par un graphe. Plus précisément, nous utiliserons un multi-graphe orienté dont les nœuds et les arcs sont étiquetés et pouvant accepter des arcs distincts partant d'une même source et arrivant à une même destination. Cette structure est dédiée à la représentation des flux d'information entre des objets d'un système d'exploitation comme les fichiers, les processus, les sockets. Un tel graphe est appelé *Graphe de Flux Système* ou *System Flow Graph* ou plus simplement SFG.
Un SFG est une structure proche des graphes de dépendances proposés par Samuel T. King et Peter M. Chen dans [KC03]. Ces graphes sont encore des graphes orientés dans lesquels les nœuds sont des conteneurs d'information et les arcs décrivent des flux entre ces conteneurs. La différence principale réside dans le fait qu'on ne sait pas dans ces graphes quelle est l'information impliquée. Ces graphes ne permettent donc ni de distinguer deux flux impliquant deux informations différentes, ni de reconstruire le chemin parcouru par une même information dans le système. Ils sont donc moins précis.

Un Graphe de Flux Système (SFG) est
un multi-graphe (V, E) orienté et étiqueté

- $v \in V$ représente un conteneur d'information et est étiqueté par
  - son type, noté v`.type` :
    `fichier/socket/processus`
  - son nom, noté v`.nom` :
    `chemin complet pour les fichiers/ nom du processus+thread / adresse IP`
  - son identifiant système, noté v`.id` :
    `numéro d'index (inode) pour les fichiers/PID pour les processus`
- $e \in E$ représente un ou plusieurs flux entre deux conteneurs d'information impliquant une même information et est étiqueté par :
  - la liste des instants auxquels les flux ont été observés, notée v`.time` :
    `POSIX timestamp`
  - l'ensemble des identifiants caractérisant l'information concernée, noté v`.info` :
    { `valeurs des itag` }
  - un arc menant d'un nœud v_1 à un nœud v_2 sera noté $v_1 \curvearrowright_{e_1} v_2$

Un SFG appauvri est un SFG dans lequel l'étiquette des nœuds ne contient plus l'identifiant système du conteneur et l'étiquette des arcs ne contient plus de *timestamp*.

La figure 7.1 représente un petit SFG impliquant deux informations sensibles respectivement caractérisées par i_1 et i_2 et trois conteneurs : un fichier représenté par un rectangle, un processus représenté par une ellipse, une socket représentée par un hexagone.

7.1.1 Opérations sur les Graphes de Flux Système

Nous aimerions pouvoir décider si deux graphes issus de deux observations différentes, caractérisent des comportements similaires. Intuitivement, cela revient à décider si il est

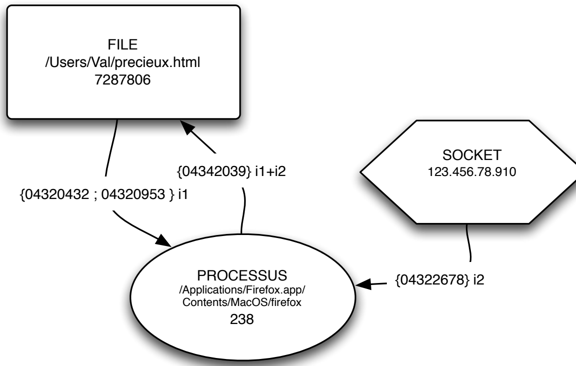


FIGURE 7.1 – Exemple de Graphe de Flux Système (SFG)

possible de simuler le premier graphe à l'aide du second, et réciproquement. Pour cela, nous allons travailler sur la version appauvrie de chacun des graphes, c'est à dire que nous ne tenons compte ni des identifiants systèmes des conteneurs, ni des timestamps qui sont propres à chaque exécution. La relation d'équivalence, l'intersection et l'inclusion ne seront définies que sur des couples de SFGs appauvris.

Soient $g_1 = (V_1, E_1)$ et $g_2 = (V_2, E_2)$ deux graphes de flux systèmes appauvris $\mathcal{R}$ une bisimulation est une relation binaire symétrique dans $V_1 \times V_2$ telle que
- $\forall (v_1, v_2) \in \mathcal{R}$ on a
  - v_1`.type` $=$ v_2`.type`
  - v_1`.nom` $=$ v_2`.nom`

- $\forall (v_1, v_2) \in \mathcal{R}$, si $\exists v_1' \in V_1$ et si $\exists e_1 \in E_1$ tel que $v_1 \curvearrowright_{e_1} v_1'$ alors $\exists v_2' \in V_2$ et $e_2 \in E_2$ tel que $v_2 \curvearrowright_{e_2} v_2'$ vérifiant :

  - e_1`.info` $=$$e_2$`.info`
  - $(v_1', v_2') \in \mathcal{R}$

La relation vide est clairement une bisimulation mais elle nous intéresse peu. Il est aisé de voir que l'union de deux bisimulations est encore une bisimulation, ce qui permet de construire l'union de toutes les bisimulations. Cette dernière relation est la plus grande des bisimulations, c'est la bisimilarité, ce qui sera noté $\sim$. Dans ce travail, deux graphes de flux systèmes seront dits équivalents si ils sont bisimilaires et si la relation de bisimilarité implique tous les nœuds des graphes autrement dit si le domaine de la bisimilarité est V_1 et son co-domaine V_2.

L'intersection de deux SFGs $g_1 = (V_1, E_1)$ et $g_2 = (V_2, E_2)$ est une opération notée $\sqcap$ permettant d'exhiber le plus grand SFG appauvri qui soit bisimilaire à un sous graphe de g_1 et à un sous graphe de g_2.

Nous dirons enfin que pour tout $g_1 = (V_1, E_1)$ et $g_2 = (V_2, E_2)$ SFGs appauvris, g_1 est inclus dans g_2 si il existe $g'_2 \subseteq (V_2, E_2)$ et tel que $g_1 \sim g'_2$. Ce qui sera noté $g_1 \sqsubseteq g_2$. L'inclusion définit une relation d'ordre partiel sur les graphes de flux système.

7.1.2 Construction des SFGs

Le journal de Blare contient des entrées qui décrivent chacune un flux précis. Un SFG peut trivialement se construire à partir d'un journal de Blare selon un algorithme très simple : pour chaque entrée du journal, ajouter les nœuds source et destination dans V s'ils n'existent pas déjà, ajouter un arc e dans E s'il n'existe pas déjà, sinon ajouter simplement une nouvelle valeur à l'étiquette `timestamps`. Cet algorithme a été implémenté et nous a permis de disposer de vues graphiques représentant la dissémination de l'information. Plus précisément, nous avons utilisé Blare comme précisé en section 6.4. Nous n'avons marqué qu'une seule donnée : le contenu de l'archive `.apk` de l'application (Android) et ensuite visualisé le SFG obtenu par l'algorithme précédent à partir du journal de Blare provenant lui même de la surveillance de l'exécution de ladite application. Le SFG représente alors la dissémination de l'information provenant d'une application durant une exécution.
Le figure 7.2 représente le SFG d'une application Android précisément le jeu Angry Birds durant une exécution de cette application. Sur cette figure, les carrés gris sont des fichiers, le carré orange est l'archive `.apk` de l'application qui correspond au point de départ du graphe (seul fichier marqué par Blare à l'initialisation).
Un tel graphe nous apprend essentiellement trois choses. Tout d'abord, nous pouvons comprendre les interactions entre une application et son environnement : les SFG remplissent leur fonction comme attendu. Ensuite nous nous rendons compte à la lecture de ce graphe que cette application ne dissémine pas son information dans tout le système : le graphe est peu profond, c'est une bonne nouvelle. Cette seconde remarque s'est trouvée confirmée dans les tests que nous avons faits (plusieurs centaines). Enfin le SFG obtenu contient toute l'information du journal Blare dans une représentation humainement utilisable. Nous avons ici un graphe de 42 nœuds et 73 arcs calculé à partir d'un journal de 103 362 entrées occupant $16,8Mo$ sur le disque. Cette dernière remarque se généralise : nous avons remarqué qu'un SFG issu de la surveillance d'une application Android est en moyenne un graphe contenant une centaine de nœuds et d'arcs.

7.2 Comprendre une attaque

Un SFG décrit précisément comment une application dissémine de l'information dans son environnement système. Nous venons de voir que cette structure décrit bien un comportement. Nous avons utilisé cette structure pour visualiser des comportements malveillants. Par exemple, la figure 7.3 décrit l'exécution d'une application infectée par un code malveillant nommé *SimpleLocker*. Une simple recherche sur le web nous apprend que ce malware chiffre les données personnelles de l'utilisateur présentes sur le téléphone les rendant ainsi indisponibles puis demande une rançon à l'utilisateur pour lui *rendre* ces données, c'est-à-dire les déchiffrer. Le SFG nous apprend en plus que ce malware

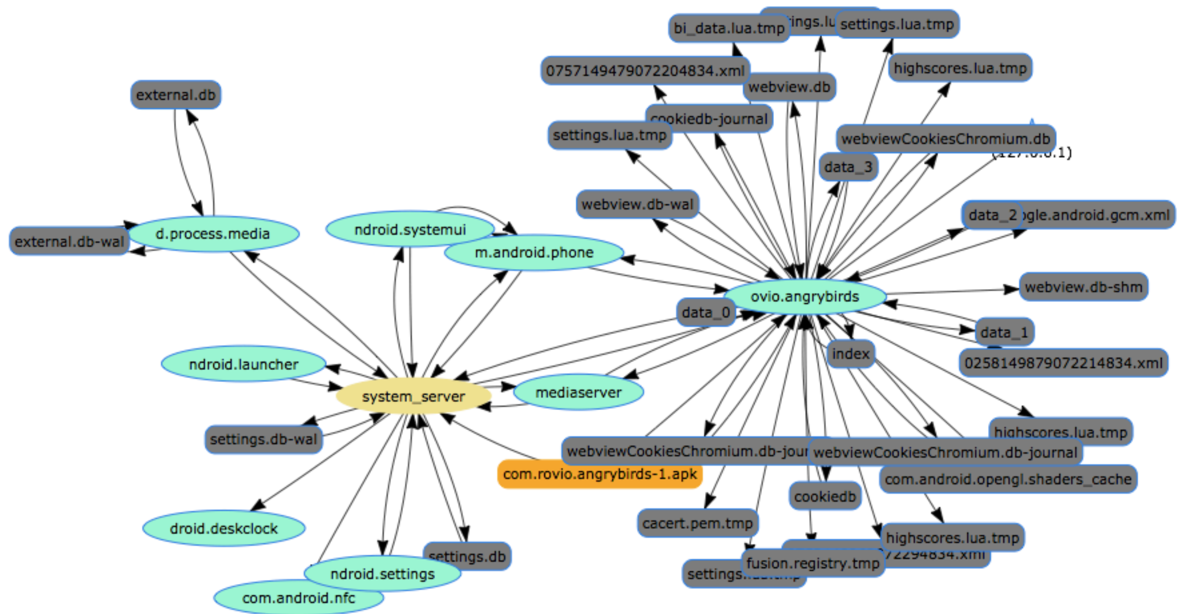


Figure 7.2 – SFG obtenu par surveillance d'une exécution du jeu Angry Birds.

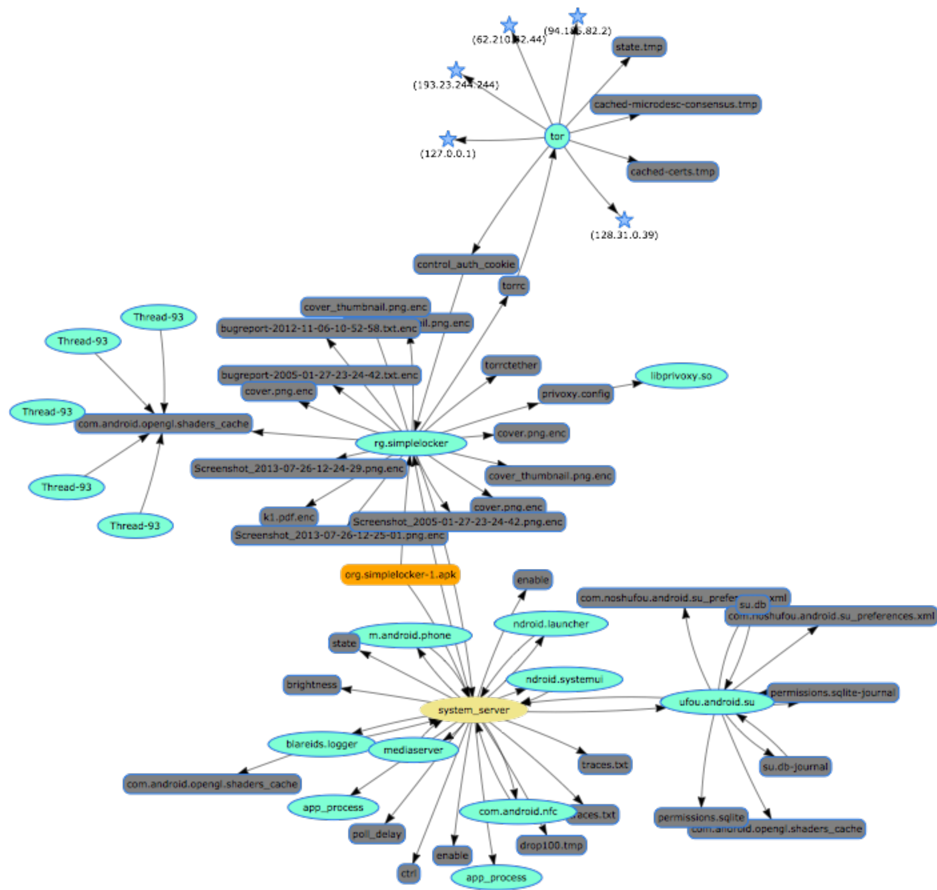


Figure 7.3 – SFG obtenu par surveillance du malware SimpleLocker.

stocke sur la carte SD du téléphone un certain nombre de fichiers `.enc` correspondant sans doute à la version chiffrée des données utilisateurs. Ce malware embarque aussi du code lui permettant de lancer une instance de l'application Tor et se connecte via cette application à des adresses IP qui sont en réalité des nœuds relais Tor. Le SFG nous précise donc le mode de fonctionnement du malware et nous avons obtenu ce résultat à peu de frais, sans qu'il ait été besoin de comprendre le code du malware lui même.
Radoniaina Andriatsimanefitra, Adrien Brunelat, Béatrice Bannier, Sylvan Bale, Nicolas Kiss ont ré-itéré cette expérience sur une quinzaine de malware et obtenu ainsi des descriptions claires de différents comportements malveillants. Ce travail est visible sur la plateforme d'analyse du projet Kharon sur laquelle nous reviendrons en conclusion de ce document. La présentation des SFGs et leur apport dans la compréhension des malware a été publié à [AVTT14c, AVTT14a, AVTTM13].

7.3 Extraction de signatures comportementales

Nous venons de comprendre qu'un SFG est une structure qui peut se révéler bien utile à un expert sécurité pour comprendre comment une application contamine son environnement. Nous allons pousser un peu plus loin cette réflexion en remarquant que dans le monde des applications Android, une pratique courante des développeurs de codes malveillants est de développer leur code puis de l'insérer dans le code d'une application tout à fait légitime avant de la remettre à disposition via une plateforme officielle comme GooglePlay ou via d'autres plateformes non officielles. De cette façon, notre attaquant peut insérer son code dans plusieurs applications différentes et multiplie ainsi les chances de voir son attaque réussir et/ou toucher le plus grand nombre d'utilisateurs. Cette technique s'appelle le *repackaging*. Le code malveillant peut être protégé par des techniques de chiffrements ou d'obfuscation, il peut être du code natif ou du code chargé dynamiquement rendant ainsi sa reconnaissance par analyse statique difficile. Néanmoins, une fois que ce code s'exécute, il nous a paru assez probable qu'il répète à chaque fois les mêmes actions. Nous avons voulu vérifier ces hypothèses et proposé d'extraire un SFG appauvri représentant les actions d'un même malware à partir de SFGs obtenus par surveillance de différentes applications en ne sachant pas si ces applications sont effectivement infectées par un même malware. Autrement dit, nous pensons que si plusieurs applications sont infectées par un même malware et que l'on est capable d'obtenir pour chacune un SFG décrivant les actions de ce malware alors il doit être possible d'exhiber uniquement le SFG appauvri bisimilaire à des sous-SFGs issus de l'observation de ces applications et décrivant le malware. Un tel SFG appauvri serait une signature comportementale de ce malware.
Pour cela, nous considérons un ensemble de SFG $\{g_1, \ldots, g_n\}$ obtenu par surveillance de n applications différentes. Nous cherchons à produire une classification de cet ensemble de la forme $\{(\mathfrak{g}_1, \mathfrak{G}_1), \ldots, (\mathfrak{g}_i, \mathfrak{G}_i)\}$ où les $\mathfrak{g}$ sont des SFGs appauvris appelés *profils* et les $\mathfrak{G}$ sont des ensembles de SFG appelés *classes* tels que si $(\mathfrak{g}_k, \mathfrak{G}_k)$ est un élément de la classification alors

1. $\mathfrak{G}_k \subseteq \{g_1, \ldots, g_n\}$, les éléments classés proviennent de l'ensemble à classer ;

2. $\mathfrak{g}_k$ est $inf(\mathfrak{G}_k)$ pour l'inclusion ;
3. $\forall \mathfrak{g}_i \forall \mathfrak{g}_j$, si $i \neq j$ alors $\mathfrak{g}_i \sqcap \mathfrak{g}_j = \emptyset$, les profils sont deux à deux disjoints ;
4. $\forall g \in \{g_1, \ldots, g_n\}$, $\exists i\, g \in \mathfrak{G}_i$ tout élément de l'ensemble de départ a été classé.

Nous avons proposé un algorithme réalisant cette classification dans [AVTT14a]. Nous avons proposé avec Radoniaina Andriatsimandefitra d'utiliser cet algorithme pour classifier un ensemble d'applications selon le comportement qu'elles ont exhibé durant la phase de surveillance qui a permis la construction de leur SFG. L'idée sous jacente est que l'on espère ainsi obtenir des profils qui soit sont vides, soit représentent des comportements malveillants communs qui ont été observés lors d'au moins deux exécutions différentes. Si cela est possible alors nous sommes capables d'exhiber des signatures comportementales de malware sans connaissance préalable de ces malware, sans savoir si les applications sont infectées ou pas, du moment que le comportement d'un même malware a été observé au moins deux fois.
En revanche, nous ne voulons pas exhiber des profils qui correspondraient à des comportements inoffensifs qu'une application Android serait susceptible d'avoir. Par exemple, il est commun qu'une application Android effectue des flux vers le processus *System Server* pour demander un service. Il est clair que tout SFG obtenu par surveillance d'une application Android a de très bonnes chances de contenir un arc depuis le processus exécutant l'application vers *System Server*. Dès lors, notre algorithme de classification risque de calculer un unique profil contenant uniquement des arcs de ce type, ce qui n'a aucun intérêt. Pour répondre à ce problème, nous avons calculé un graphe composé de nœuds et d'arcs que l'on suppose éventuellement présents dans tout SFG obtenu par surveillance d'une application Android et totalement inoffensifs. Pour cela, nous avons utilisé une *liste blanche* d'applications Android considérées comme saines et calculé l'union de leurs SFG. Nous avons pu utiliser l'algorithme précédent moyennant un nettoyage préalable, qui a consisté à enlever des SFGs à classer les arcs que l'on retrouve dans la liste blanche.
Enfin, pour comparer ce qui est comparable, dans tous les SFGs à classer, nous avons remplacé le nom de l'application par la même constante `blare-anonyme`.

7.4 Evaluation de l'approche

7.4.1 Classification de comportements malveillants

Radoniaina Andriatsimandefitra a appliqué la méthode précédente sur un ensemble de SFGs obtenu par surveillance de dix neuf applications. Nous savions que ces applications étaient infectées par quatre malware différents : BadNews, DroidKungFu1,`DroidKungFu2` et JSMSHider dont nous savions qu'ils étaient responsables d'installations d'applications sur le téléphone à l'insu de l'utilisateur. Ces malware ont été découverts entre 2011 et 2013 et nous les avons récupérés grâce au projet Android Genome [ZJ12], les auteurs de l'outil de rétro-ingénieurie Androguard [And, DG11], le site web Contagio [Par] et des plateformes de téléchargement alternatives russes. BadNews est un malware commandé par un serveur distant, DroidKungFu1 et DroidKungFu2 exploitent des vulnérabilités du

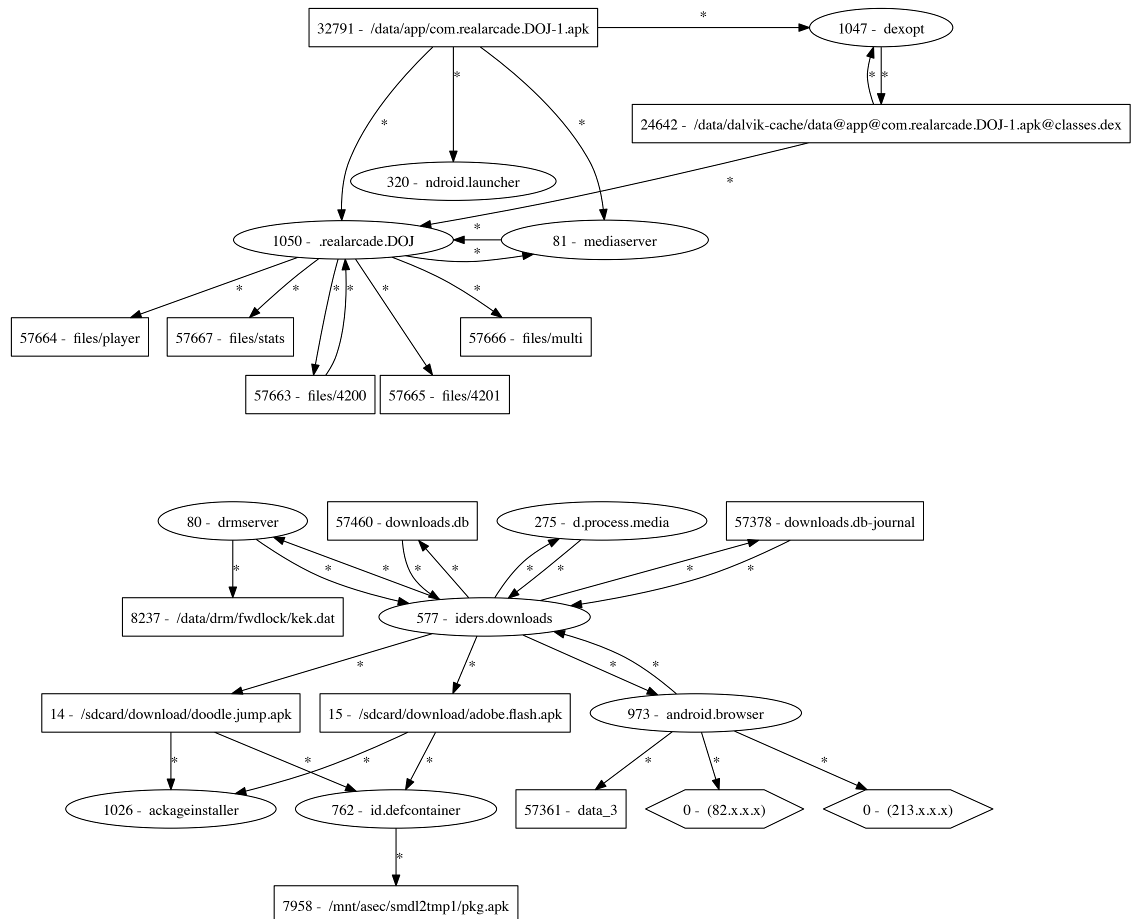


FIGURE 7.4 – Profil attribué au malware Badnews

système. Cette expérience a été menée sur un téléphone Nexus S sous Android Ice Cream Sandwich 4.0.4 connecté à Internet. Cette expérience est bien détaillée dans [AVTT14a, And14]
L'application de l'algorithme précédent a mis en évidence quatre profils et donc quatre classes de SFG. Ces profils sont décrits sur les figures 7.4, 7.5, 7.6,7.7. Ces profils ont été calculés en utilisant une *liste blanche* contenant cinq applications considérées comme saines. Radoniaina Andriatsimandefitra a répété cette expérience en changeant la taille et la nature des applications de la liste blanche, ce qui a permis de comprendre qu'il suffisait de choisir une liste blanche d'applications suffisamment différentes et accédant à la majorité des services Android (comme activer/désactiver le son, activer/désactiver le flash etc). Il a montré durant sa thèse que cinq à dix applications suffisaient.
Nous avons pu vérifier que toutes les applications que nous savions a priori infectées par

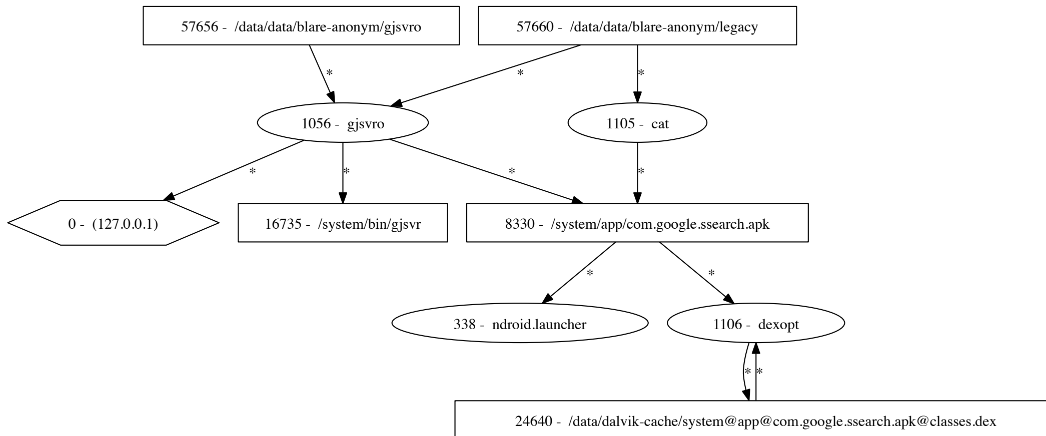


Figure 7.5 – Profil attribué au malware DroidKungFu1

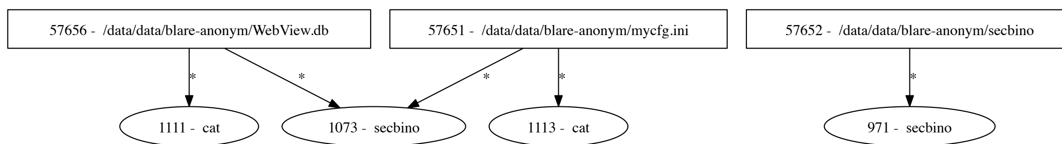


Figure 7.6 – Profil attribué au malware DroidKungFu2

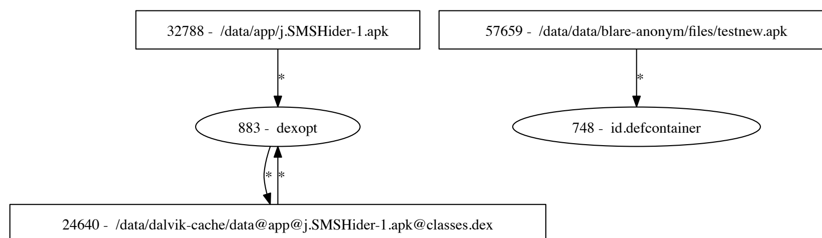


Figure 7.7 – Profil attribué au malware JSMShider

le même malware ont été classées dans la même classe. Nous avons donc attribué chacun des quatre profils aux quatres différents malware. Aucun profil n'est vide et aucun profil ne présente d'arc qui paraîtrait usuel pour toute application Android.
Cette première expérience s'avère donc concluante. Nous avons appelé ces profils des signatures comportementales.

7.4.2 Pertinence des signatures comportementales

Nous avons poussé un peu plus loin la démarche précédente et nous nous sommes posé la question suivante : *Est ce que les signatures comportementales peuvent permettre de détecter des exécutions malveillantes ?* Autrement dit, est-il possible d'utiliser ces signatures pour construire un système de détection d'intrusion (IDS) ? Pour cela, nous avons proposé de surveiller les exécutions des applications et de comparer dynamiquement les flux observés avec les flux attendus. Nous voulions précisément

1. évaluer les faux positifs, c'est-à-dire savoir combien d'arcs présents dans les signatures comportementales on retrouve dans des exécutions considérées comme normales ;
2. évaluer les faux négatifs, c'est à dire savoir combien d'exécutions de malware dont nous avons une signature ne sont pas détectées ;
3. évaluer le pourcentage du graphe reconnu lorsque le moniteur reconnait une signature.

Pour répondre à ces questions, nous avons tout d'abord expérimenté la démarche sur les soixante-dix applications gratuites les plus téléchargées sur Google Play en juin 2013. Nous avons considéré ces applications comme saines. Notre critère de décision vient de la simple remarque que ces applications ont été téléchargées par des millions d'utilisateurs, si elles avaient été malveillantes, elles auraient été détectées. Cette première expérience nous a menée à étudier plus de neuf millions de flux d'informations dont aucun n'apparaissait dans une signature comportementale que nous connaissions. Ce fut une bonne nouvelle. Il semble que nos signatures comportementales soient suffisamment discriminantes pour ne pas confondre des exécutions normales avec des exécutions malveillantes.

Néanmoins, peut être que notre IDS n'a pas de faux positif parce qu'il ne détecte jamais rien. Cela serait ennuyeux.

Pour s'en rendre compte, nous avons réitéré l'expérience sur un ensemble de trente neuf applications dont nous savions qu'elles étaient infectées par les malware dont nous avions une signature. Parmi ces trente neuf applications, dix neuf avaient servi au calcul de signatures comportementales. Cette base de test comprenait toutes les applications à notre disposition et infectées par les malware étudiés. Cette seconde expérience a montré que nous étions capable de reconnaître la totalité des exécutions malveillantes. Le détail de ces résultats est donné sur la table 7.1.

Empreinte MD5 des échantillons	Label[†]	Origin[§]	Log size	$p_1^‡$	$p_2^‡$	$p_3^‡$	$p_4^‡$
d25008db2e77aae53aa13d82b20d0b6a	JSH	A	79598				4/4
24663299e69db8bfce2094c15dfd2325	JSH	A	179608				4/4
39d140511c18ebf7384a36113d48463d	DKF1	A / G	3565		11/11		
7f5fd7b139e23bed1de5e134dda3b1ca	DKF1	A	5772		11/11		
a81dc5210b3444b8e6f002605a97292d	DKF1	A	3233		3/11		
107af5cf71f1a0e817e36b8deb683ac2	DKF1	A	7257		11/11		
ac2a5a483036eab1b363a7f3c2933b51	DKF1	A	3596			5/5	
e741a9bc460793b9afdadc963d6e8c1d	DKF1	A	3230		3/11		
6b7c313e93e3d136611656b8a978f90d	DKF1	A	7740			5/5	
389b416fb0f505d661716b8da02f92a2	JSH	G	179702				4/4
a3c0aacb35c86b4468e85bfb9e226955	JSH	G	7527				4/4
0417b7a90bb5144ed0067e38f7a30ae0	JSH	G	32145				4/4
d25008db2e77aae53aa13d82b20d0b6a	JSH	G	122951				4/4
f0fcef1c52631ae36f489351b1ba0238	JSH	G	211823				4/4
06dea6a4b6f77167eaf7a42cb9861bbe	DKF1	G	72706		6/11		
994af7172471a2170867b9aa711efb0d	DKF1	G	13959		11/11		
107af5cf71f1a0e817e36b8deb683ac2	DKF1	G	187221		11/11		
71fe80d5bf6d08890de3c76a3292fc09	DKF1	G	15709		11/11		
ecc4aad77ab042a4fa1693fc77afb8ac	DKF1	G	107910		11/11		
b763bc07f641bb915a4e745f1deff315	DKF1	G	180984		8/11		
6625f4a711e5afaee5f349c40ad1c4ab	DKF1	G	4982		11/11		
5c593a7ab5e61f76d2e0e61c870da986	DKF1	G	99363		11/11		
41f7b03a94d38bc9b61f8397af95a204	DKF1	G	13474		4/11		
f438ed38b59f772e03eb2cab97fc7685	DKF2	G	34906			5/5	
4f6be2d099b215e318181e1d56675d2c	DKF2	G	283990			5/5	
805bbc6ff9ef376c4b5f2c1b1c1006d2	DKF2	G	49404			5/5	
13a491126dd11f1ef51a4b067f10f368	DKF2	G	278425			5/5	
72dc94b908b0c6b7e3cb293d9240393c	DKF2	G	294163			5/5	
e4d348e97db481507a0cea64232c8065	DKF2	G	64616			5/5	
47ffc035dd1288bad27b3681535e68c8	BN	I	298819	36/36			
d8943ed5be382c22c9a206af0815ff0a	BN	I	363578	36/36			
ccab22538dd030a52d43209e25c1f07b	BN	I	167837	36/36			
3a648e6b7b3c5282da76590124a2add4	BN	I	332519	36/36			
4ecf985980bcc9b238af1fdadd31de48	BN	I	125167	36/36			
5b08c96794ad5f95f9b42989f5e767b5	BN	C	132846	36/36			
422d1290422ebfbf48ec34f0990fba21	BN	I	634202	35/36			
98cfa989d78eb85b86c497ae5ce8ca19	BN	C	568920	36/36			
e70964e51210f8201d0da3e55da78ca4	BN	I	253320	36/36			
8b9e8a2e93c3f3c18b8f5820f21e2458	BN	I	149248	36/36			

[†] BN : BadNews, DKF1 : DroidKungFu1, DKF2 : DroidKunFu2, JSH : jSMSHider

[§] C : Contagio, G : Genome Project, A : Androguard, I : Internet

[‡] p_1, p_2, p_3, p_4 correspondent respectivement aux profils attribués à BadNews, DroidKungFu1, DroidKunFu2, et JSMShider

TABLE 7.1 – Résultats de la détection sur 39 échantillons de malware

7.5 Limites actuelles de l'approche

Ces premiers résultats sont excellents et nous sommes convaincue que lorsque deux applications sont infectées par le même malware, elles exécutent les mêmes actions dûes à ce malware, il nous semble donc tout à fait normal que nous n'ayons pas de faux négatifs dès lors que le malware s'est effectivement exécuté. Aujourd'hui, les auteurs de malware sont capables d'obfusquer le code de leur malware, mais a priori ne se préoccupent pas encore tellement d'obfusquer leur comportement. Par ailleurs, nous avons pu remarquer qu'une application saine exhibe toujours à peu près le même type de graphe qui n'a rien à voir avec ceux que nous avons calculés dans les signatures des malware, le taux bas de faux positifs n'est donc pas surprenant pour les signatures qui ont été utilisées dans ces expériences.
Néanmoins, nous identifions trois limites principales dans cette approche :

1. Cette approche est performante dès lors que les malware s'exécutent effectivement. En pratique, les auteurs de malware ne manquent pas d'imagination pour échapper à l'analyse dynamique et il ne suffit pas de lancer une application infectée pour que le code malveillant qu'elle contient s'exécute à son tour. Par exemple, le code malveillant peut se déclencher après le redémarrage du téléphone, après la réception d'une commande d'un serveur distant, après 10 minutes, après que l'utilisateur ait gagné le premier niveau, etc. Dans nos expériences, nous savions comment provoquer le déclenchement des malware et nous avons forcé leur exécution. Nous étions donc sûrs que le comportement malveillant serait observé. Ce problème rejoint celui du calcul complet de la politique BSPL que nous avions identifié à la fin du chapitre précédent : comment savoir si l'exécution / les exécutions que l'on considère sont suffisantes et pertinentes ? Nous avons proposé à Adrien Brunelat, Adrien Abraham et Nicolas Kiss d'aborder ce problème selon des angles d'attaque différents durant leur stage de fin d'études. Ces stages sont co-encadrés avec Jean-François Lalande. Nous pensons que ce problème ne peut se régler sans une analyse statique du code. C'est dans le sens de cette remarque que nous travaillons au montage d'un sujet de thèse avec Jean-François Lalande et Thomas Genet qui pourrait débuter à l'automne 2015.
2. Cette approche sera certainement performante pour caractériser tous les malware qui réalisent une attaque à l'intégrité des données du téléphone comme par exemple installer des applications à l'insu de l'utilisateur, remplacer des données par d'autres, chiffrer des données pour demander ensuite des rançons etc. Nous n'avons pas évalué la pertinence de l'approche sur les malware qui enverraient des SMS à des numéros sur-taxés mais il est probable que cette approche soit bien moins efficace car *envoyer un SMS* sera sans doute une action vue comme normale si jamais elle engendre un flux d'information au niveau du système.
3. Si les malware étaient capables de modifier leur comportement lors de chacune de leur exécution, nous courrions le risque d'avoir des signatures vides. En effet, si le comportement d'un même malware était différent à chaque exécution, notre calcul d'intersection se ramènerait rapidement au calcul du SFG vide. Néanmoins,

on peut se demander si cela est réellement possible : cela voudrait dire en effet que le code malveillant maîtrise l'ensemble des flux d'information partant de lui et donc maîtrise totalement son environnement. Cela nous semble peu probable. Nous pensons en revanche que si un malware était capable d'obscurcir *un peu* son comportement alors les signatures pourraient devenir très petites. Cela reste à expérimenter en pratique.

7.6 Bilan

Dans ce dernier chapitre, nous avons proposé une représentation des flux observés par Blare sous forme de graphe. Nous avons montré dans un premier temps que cette représentation permettait de comprendre comment l'exécution d'une application contamine son environnement. Nous avons aussi montré que nous pouvions utiliser cette représentation pour calculer des signatures comportementales à l'aide d'un algorithme de classification. Enfin, nous avons proposé d'utiliser ces signatures dans un système de détection d'intrusion qui nous a donné de premiers très bons résultats. Nous aurions aussi pu remarquer que cette structure était une bonne solution au problème de stockage des logs Blare que nous évoquions au chapitre précédent : comme les SFG représentent les logs Blare sans perte d'information, il serait plus judicieux de stocker les logs sous forme de SFG qui sont bien moins volumineux. Pour revenir au chapitre précédent, nous avons montré dans [AVTT14b] que les alertes produites par Blare lorsqu'il surveille une application malveillante avec la politique Blare provenant de l'application saine, permettent de construire un SFG qui correspond à la signature comportementale calculable avec la méthode présentée dans ce chapitre. La structure de SFG nous paraît donc bien appropriée pour représenter des comportements malveillants.

Chapitre 8

Pour conclure

8.1 Bilan

Dans ce document, nous avons présenté nos travaux sur le suivi de flux d'information réalisés dans l'équipe projet CIDRE. J'ai commencé à travailler sur ce sujet en 2006 et encadré / participé à l'encadrement de cinq doctorants (toujours sur ce sujet) : Guillaume Hiet, Stéphane Geller, Thomas Demongeot, Radoniana Andriatsimendefitra et Laurent Georget ; six stagiaires de M2 : Laurent George, Radoniana Andriatsimendefitra, Adrien Brunelat, Laurent Georget, Adrien Abraham et Nicolas Kiss et un stagiaire de L3 : Thomas Saliou.

Ces travaux ont tout d'abord consisté à proposer un modèle de moniteur de flux d'information. Ensuite nous avons exploré deux voies.

La première s'intéresse aux politiques de sécurité qui peuvent être mises en œuvre par cette approche. Nous avons proposé un modèle de politique décentralisé. Nous avons défini une notion de cohérence pour ces politiques. Nous avons proposé un opérateur de composition respectant la cohérence. Ce modèle de politique s'accompagne d'un langage de spécification de politique de haut niveau BSPL. Nous avons aussi encadré l'implémentation d'un *policy manager* pour ce langage permettant ainsi de mettre en œuvre ces politiques sous Android. Enfin, nous avons proposé une méthode semi-automatique facilitant la construction de telles politiques. Nous avons confronté ces idées théoriques à des applications Android. Les applications utilisées n'étaient pas des applications jouets. Nous avons aussi expérimenté la pertinence de notre approche face à des applications Android malveillantes récentes (2011, 2013, 2014). Ces expérimentations sont concluantes et nous persuadent de continuer dans cette voie.

Dans un second temps, nous avons proposé une nouvelle approche pour l'étude des comportements des applications. Nous avons proposé une nouvelle structure permettant de comprendre la contamination de l'environnement par les actions d'une application. Nous avons montré que cette approche est tout à fait adaptée à la caractérisation des comportements malveillants. Nous avons proposé un algorithme de classification permettant de calculer des *signatures comportementales* de malware Android. Nous avons là encore confronté ces idées théoriques à des malware actuels. Nous pensons

que ce deuxième axe de travail est particulièrement enthousiasmant, pertinent et prometteur. Nous avons proposé le projet Kharon au Labex CominLabs. Ce projet est une collaboration avec l'équipe projet INRIA Celtique. Il vise à proposer une plateforme d'analyse des applications Android permettant la reconnaissance des comportements malveillants dans les exécutions Android. Nous souhaitons pouvoir analyser des milliers de malware avec cette approche et pouvoir ainsi mener des expérimentations à plus grande échelle. Cette plateforme sera publique et permettra à tous d'analyser ses applications. Nous souhaitons aussi que cette plateforme puisse offrir aux autres chercheurs du domaine une bibliothèque de comportements bien documentée ce qui serait à notre avis un apport intéressant pour la communauté.

La suite de ces travaux comporte plusieurs volets.

8.2 Etendre l'approche à d'autres niveaux d'observations

Nous avons présenté un modèle de suivi de flux des informations suffisamment générique pour qu'il soit adaptable à tout niveau d'observation. Nous nous sommes concentré sur le niveau système. Guillaume Hiet s'est intéressé à améliorer la précision en portant l'implémentation dans la machine virutelle java ce qui permet de raffiner l'observation de flux entre deux applications java [HMMVTT07]. Il poursuit ce travail d'amélioration de la précision en proposant d'observer les flux d'information au niveau matériel. Christophe Hauser et Frédéric Tronel [HTRF12], [Hau13] ont proposé une implémentation distribuée du modèle ce qui permet d'élargir le champ de vision des flux d'information. Nous pensons qu'il serait pertinent de porter ce travail dans les bases de données ce qui permettrait de garantir que des informations n'interfèrent pas dans une base contenant des informations de différentes sensibilités. Il serait aussi très intéressant de porter ce travail au niveau cloud en réalisant une implémentation au niveau de l'hyperviseur. Cela permettrait d'assurer la confidentialité des données des utilisateurs qui y sont stockées.

8.3 Complétudes des approches - validation des implémentations

Nous pensons qu'une réflexion forte est à engager sur le problème de la complétude des implémentations. Théoriquement, il devrait être possible d'observer, hormis les canaux cachés, *toutes* les opérations engendrant des flux explicites. En pratique, aujourd'hui, beaucoup de travaux se sont penchés sur ce problème au niveau applicatif. Autrement dit, de nombreuses études ont proposé des mécanismes complets de suivi de flux d'information dans les exécutions de programme, majoritairement par des approches statiques [PS03, Mye99b, CVM07].Dans ces travaux, nous avons soutenu le suivi de flux d'information au niveau du système d'exploitation. L'implémentation de Blare, le moniteur de flux résultant du modèle théorique se base sur l'hypothèse raisonnable, mais empirique, que les flux d'information sont causés par *certains appels systèmes*. Le moniteur Blare est aujourd'hui la base d'autres travaux de recherche sur la caractérisation de malware

par exemple et nous devons pouvoir lui porter une grande confiance, ce qui signifie qu'il faut une preuve de la complétude de l'implémentation. Nous avons proposé à Laurent Georget d'étudier ce problème durant sa thèse. Laurent a débuté sa thèse en octobre 2014 ; il est encadré par Mathieu Jaume, Frédéric Tronel, Guillaume Piolle et moi même. Laurent se concentre sur le système d'exploitation Linux. Il a défini une représentation du code du noyau Linux et donc des appels systèmes. Cette représentation est calculée par un plugin pour gcc lors de la compilation du noyau Linux : elle est donc fidèle à l'implémentation du noyau. Cette représentation peut se traduire en un modèle symbolique exécutable par réécriture. Il devient donc possible de simuler des appels concurrents à plusieurs appels systèmes et de calculer l'ensemble des flux d'information théoriquement effectivement possibles selon l'ordonnancement effectué par le noyau. Nous pensons que son travail fournira une base de confiance solide pour Blare. Nous espérons aussi qu'il mettra en évidence une méthode d'implémentation complète pour tout autre moniteur de flux.

8.4 Complétude des comportements étudiés

Dans le chapitre 6, nous avons proposé une méthode semi-automatique de construction des politiques de flux d'information. Pour que cette méthode soit effectivement intéressante, il faudrait que nous soyons capables d'observer tous les comportements de l'application générant des flux d'information (et idéalement, seulement ceux là). Nous retrouvons exactement le problème évoqué en fin de chapitre . Nous aimerions être capable d'observer toutes les exécutions induisant des comportements système différents afin d'en calculer une représentation.

Nous pensons que ce problème est un sous-problème de celui qui consiste à observer toutes les exécutions d'une application. En effet, beaucoup d'exécutions induisent des comportements systèmes équivalents. Par exemple, le programme ci-dessous, a autant d'exécutions possibles que de valeurs possibles pour `a`. Néanmoins, aucune de ces exécutions n'induit de flux d'information dans le système d'exploitation. Nous pourrions dire que toutes ces exécutions sont équivalentes pour l'observation des comportements systèmes ; en observer une seule est suffisant.

Algorithm 2: `Plusieurs exécutions, un seul comportement système`

```
begin
    Data: file
    /* un fichier                                                   */
    Data: input(a)
    /* a une valeur entrée par l'utilisateur et supposée entière   */
    while random(0, a)! = 0 do
        a :=(a+1);
    file.write( "fin")
```

Ce travail devrait donc
– formaliser la notion de *comportement système* ;
– définir une l'équivalence des exécutions en regard de leur comportement système ;
– proposer une méthode d'analyse statique permettant d'exhiber une exécution *représentante* pour chaque classe des comportements système possibles ;
– déclencher ces exécutions représentantes et les observer.

8.5 Obfuscation comportementales

Notre travail sur la caractérisation des malware détaillé, dans le chapitre 8.4 a de bons résultats parce que les différentes exécutions d'un même malware ont des comportements systèmes identiques. Les auteurs de malware sont aujourd'hui capables d'obfusquer leur code malveillant, c'est-à-dire sa forme, mais pas d'obfusquer leur comportement. Nous devrions réfléchir à ce problème avant les auteurs de malware. Nous pensons aujourd'hui que si un malware a une finalité précise (voler des données, installer des services à l'insu de l'utilisateur, corrompre le système, etc.) alors nous devrions être capable de calculer une *forme normale* de ce comportement. Nous pensons que si c'est le cas alors nous pourrions réfléchir la manière d'établir qu'un comportement peut (ou pas) se réduire à une forme normale donnée. Nous n'avons pas du tout étudié ce problème aujourd'hui ; nous pensons qu'il est important de le faire dans les années qui viennent.

8.6 Affiner l'observation des tags

Le chapitre 6, a présenté un langage de spécification de politiques de flux d'information. Nous avons évalué la pertinence des politiques mises en œuvre en analysant des exécutions solitaires. Autrement dit, une fois notre étude réalisée, nous avons mené d'autres expériences et les tags des conteneurs d'information ont été ré-initialisés. Il est probable que si nous n'avions pas fait cette réinitialisation, nous aurions dû faire face à une explosion des tags. Nous pensons que ce problème d'explosion est d'une part lié aux services qui centralisent des mécanismes dans le système comme les bus de communication, `system_server`, le binder d'Android etc. Pour réduire la sur-approximation du suivi de flux d'information, et donc limiter l'explosion des tags, il faut donc impérativement descendre au niveau applicatif dans ces objets là.

8.7 Déclassifier l'information

Le problème général de la déclassification de l'information a été bien formalisé par A. Sabelfeld et D. Sands dans [SS09]. Ce problème est aujourd'hui étudié en partie dans les langages de programmation. Mantel et Reinhard se sont intéressés dans [MR07] à ce qui pouvait être déclassifié et au devenir de l'information déclassifiée.
Déclassifier une information revient à modifier sa politique. En soit, la modification de politique n'est pas un problème compliqué : il suffit de changer la valeur de son `itag`. Ce qui est difficile est de savoir dans quelles conditions faire ce changement.

Dans [DTTT11], nous avons proposé que les propriétaires de l'information concernée s'accordent sur une éventuelle déclassification. Dans ce travail, il s'agissait de permettre un flux d'information illégal mais éventuellement nécessaire au fonctionnement d'un service web. Ce flux pouvait avoir lieu `jamais - une fois - toujours` selon la volonté de tous les propriétaires de l'information. Ce travail est une première réflexion qui propose essentiellement que la décision de la déclassification soit prise conjointement par les propriétaires de l'information. Il reste encore à réfléchir à la manière de mettre en œuvre cet accord, surtout lorsque les propriétaires de l'information ne sont pas atteignables, parce qu'ils sont développeurs d'une application par exemple. Il faut aussi s'intéresser au devenir de l'information déclassifiée, comment définit-on la politique de ce qui est devenu une nouvelle information dans le système ? Enfin comment intégrer ces principes au langage de spécification de politique ?

Bibliographie

[AGVTT12] R. Andriatsimandefitra, S. Geller, and V. Viet Triem Tong. Designing information flow policies for android's operating system. In *Proceedings of the International Conference on Computer Communications (ICC)*, 2012.

[And] Androguard. `https://github.com/androguard/androguard`.

[And14] Radoniaina Andriatsimandefitra. *Characterization and detection of android malware based on information flows*. Theses, Supélec, December 2014.

[ASVTT13] Radoniaina Andriatsimandefitra, Thomas Saliou, and Valérie Viet Triem Tong. Information flow policies vs malware. In *IAS - Information assurance and security - 2013*, Yassmine Hammamet, Tunisia, 2013.

[AVTT14a] Radoniaina Andriatsimandefitra and Valérie Viet Triem Tong. Capturing Android Malware Behaviour using System Flow Graph. In *NSS 2014 - The 8th International Conference on Network and System Security*, Xi'an, China, October 2014.

[AVTT14b] Radoniaina Andriatsimandefitra and Valérie Viet Triem Tong. Information Flow Policies vs Malware – Final Battle –. *Journal of Information Assurance and Security*, 9(2) :72–82, 2014.

[AVTT14c] Radoniaina Andriatsimandefitra and Valérie Viet Triem Tong. Poster Abstract : Highlighting Easily How Malicious Applications Corrupt Android Devices. Research in Attacks, Intrusions, and Defenses, September 2014.

[AVTTM13] Radoniaina Andriatsimandefitra, Valérie Viet Triem Tong, and Ludovic Mé. Diagnosing intrusions in android operating system using system flow graph. In *Workshop Interdisciplinaire sur la Sécurité Globale*, 2013.

[BL73] D.E. Bell and L.J. LaPadula. Secure computer systems : Mathematical foundations. MTR-2547 (ESD-TR-73-278-I) Vol. 1, MITRE Corp., Bedford, 1973.

[CVM07] Stephen Chong, K. Vikram, and Andrew C. Myers. Sif : Enforcing confidentiality and integrity in web applications. In *16th USENIX Security Symp.*, August 2007.

[DD77] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7) :504–513, July 1977.

[Den76a] Dorothy Denning. On the derivation of lattice structured information flow policies. Technical report, Dep. of Compter. Science, Purdue University, March 1976.

[Den76b] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5) :236–243, 1976.

[DG11] Anthony Desnos and Geoffroy Gueguen. Android : From reversing to decompilation. Technical report, Black Hat Abu Dhabi 2011, 2011.

[DTTT11] Thomas Demongeot, Eric Totel, Valerie Viet Triem Tong, and Yves Le Traon. Preventing data leakage in service orchestration. In *Proceedings of the International Conference on Information Assurance and Security (IAS 2011)*, december 2011.

[EGgC+10] William Enck, Peter Gilbert, Byung gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, , and Anmol N. Sheth. Taintdroid : An information-flow tracking system for realtime privacy monitoring on smartphones. In *In Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[EKV+05] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 17–30, New York, NY, USA, 2005. ACM.

[GHTVTT11] Stéphane Geller, Christophe Hauser, Frédéric Tronel, and Valérie Viet Triem Tong. Information flow control for intrusion detection derived from mac policy. In *Proceedings of the International Conference on Computer Communications (ICC)*, 2011.

[GVTTM09] Laurent George, Valérie Viet Triem Tong, and Ludovic Mé. Blare Tools : A Policy-Based Intrusion Detection System Automatically Set by the Security Policy, September 2009. Poster and short paper.

[GVTTM13] S. Geller, V. Viet Triem Tong, and L. Mé. Bspl : a language to specify and compose fine-grained information flow policies. In *Proceedings of the The Seventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, 2013.

[Hau13] Christophe Hauser. *A basis for intrusion detection in distributed systems using kernel-level data tainting.* Theses, Supélec ; QUEENSLAND UNIVERSITY OF TECHNOLOGY, June 2013.

[HMMVTT07] G. Hiet, L. Mé, B. Morin, and V. Viet Triem Tong. Monitoring both os and program level information flows to detect intrusions against net-

work servers. In *IEEE Workshop on "Monitoring, Attack Detection and Mitigation"*, 2007.

[HTRF12] C. Hauser, F. Tronel, J Reid, and C Fidge. A taint marking approach to confidentiality violation detection. In *10th Australasian Information Security Conference (AISC 2012)*. Australian Computer Society, 2012.

[JVTTH12] Mathieu Jaume, Valérie Viet Triem Tong, and Guillaume Hiet. Spécification et mécanisme de détection de flots d'information illégaux. *Technique et Science Informatiques (TSI)*, 31(6) :713–742, 2012.

[JVTTM10] Mathieu Jaume, Valérie Viet Triem Tong, and Ludovic Mé. Contrôle d'accès versus Contrôle de flots. In LISI/ENSMA, editor, *10emes Journées Francophones sur les Approches Formelles dans l'Assistance au développement des logiciels*, pages 27–41, Poitiers, France, June 2010.

[JVTTM11] Mathieu Jaume, Valérie Viet Triem Tong, and Ludovic Mé. Flow based interpretation of access control : Detection of illegal information flows. In *7th International Conference on Information Systems Security (ICISS)*, volume 7093, pages 72–86, Kolkata, India, December 2011.

[KBB+03] Anas Abou El Kalam, Rania El Baida, Philippe Balbiani, Salem Benferhat, Frederic Cuppens, Yves Deswarte, Alexandre Miege, Claire Sorel, and Gilles Trouessin. Abstract organization based access control, 2003.

[KC03] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 223–236, Bolton Landing, NY, October 2003. ACM Press.

[KYB+07a] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In *Proceedings of the 21st Symposium on Operating Systems Principles*, Stevenson, WA, October 2007.

[KYB+07b] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 321–334, New York, NY, USA, 2007. ACM.

[LPC13] Hwan-Taek Lee, Minkyu Park, and Seong-Je Cho. Detection and prevention of lena malware on android. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4) :63–71, November 2013.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. *SIGOPS Oper. Syst. Rev.*, 31(5) :129–142, 1997.

[MR07] Heiko Mantel and Alexander Reinhard. Controlling the what and where of declassification in language-based security. In *Proceedings of the 16th European Conference on Programming*, ESOP'07, pages 141–156, Berlin, Heidelberg, 2007. Springer-Verlag.

[Mye99a] Andrew C. Myers. Jflow : Practical mostly-static information flow control. In *Proceedings of the 26th ACM on Principles of Programming Languages*, 1999.

[Mye99b] Andrew C. Myers. Jflow : practical mostly-static information flow control. In *26th ACM Symp. on Principles of Programming Languages (POPL)*, page 228–241, January 1999.

[Par] Mila Parkour. Contagio mobile. `contagiominidump.blogspot.com`.

[PS03] François Pottier and Vincent Simonet. Information flow inference for ml. *ACM Trans. Program. Lang. Syst*, 2003.

[San93a] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11) :9–19, November 1993.

[San93b] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11) :9–19, 1993.

[SS09] Andrei Sabelfeld and David Sands. Declassification : Dimensions and principles. *J. Comput. Secur.*, 17(5) :517–548, October 2009.

[VTTCM10] Valérie Viet Triem Tong, Andrew Clark, and Ludovic Mé. Specifying and enforcing a fine-grained information flow policy : Model and experiments. In *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications*, 2010.

[ZBWKM06a] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *In Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[ZBWKM06b] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in histar. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 19–19, Berkeley, CA, USA, 2006. USENIX Association.

[ZJ12] Yajin Zhou and Xuxian Jiang. Dissecting android malware : Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.