Développement formel d'un moniteur.

Vianney Darmaillacq* **et Nicolas Stouls†**
Laboratoire Logiciels Systèmes Réseaux - LSR-IMAG - Grenoble, France
{Vianney.Darmaillacq, Nicolas.Stouls}@imag.fr

Résumé : Dans le domaine des réseaux informatiques, de nombreux modèles existent pour permettre l'expression de politiques de sécurité. Les plus utilisés sont les modèles de contrôle d'accès. Cependant, peu de méthodes sont proposées pour s'assurer que ces politiques de haut niveau sont effectivement respectées par le système réel. Nous nous proposons ici d'assurer cette garantie par la mise en place d'un moniteur surveillant le réseau. Ce moniteur est développé par un processus basé sur le raffinement. Nous illustrerons nos propos avec une étude de cas modélisant un réseau de type intranet, protégé par des pare-feux.
Mots-clés : Réseaux, politique de sécurité, méthode formelle, méthode B, raffinement

1 INTRODUCTION

Il existe principalement quatre moyens permettant de faire appliquer la sécurité dans un système informatique. Ce sont le *contrôle d'accès*, le *contrôle des flux d'informations*, la *cryptographie* et *l'authentification*. Le contrôle d'accès est indispensable dans un système informatique [SCFY96]. Ce paradigme suppose l'existence d'une liste définissant les actions autorisées dans le système. Le contrôle d'accès est effectif lorsque toute action est interceptée et que celles qui ne sont pas autorisées sont bloquées. Citons par exemple, l'authentification sur un serveur, les droits des fichiers ou le filtrage de paquets comme mécanismes de contrôle d'accès.

Dans les réseaux modernes, on cherche à s'abstraire des contraintes techniques. Pour cela, on préconise la définition de politiques qui sont dérivées des objectifs de haut niveau et qui spécifient le comportement désiré du système. Leur utilisation permet de séparer les règles de sécurité des fonctionnalités du système [Slo94].

L'utilisation de telles abstractions pose cependant un problème : comment décider si le système respecte effectivement la politique ? Ce problème est celui de la *conformité* entre le système et la politique de sécurité.

On identifie quatre manières de garantir le respect d'une politique de sécurité :

Dérivation : Génération des configurations du système depuis son architecture et une politique de sécurité ;

Vérification : Preuve de la conformité de l'architecture et des configurations d'un système par rapport à une politique de sécurité *(analyse statique)* ;

Test : Test de la conformité du comportement du système par rapport à la politique de sécurité *(test actif)* ;

Surveillance : Observation de la conformité du comportement du système par rapport à la politique de sécurité *(test passif ou monitoring)* ;

Dans cet article, c'est ce dernier cas qui va nous intéresser.

[Mas93] est le premier à remarquer que les politiques peuvent s'abstraire plus ou moins du système réel et en conséquent, à proposer une hiérarchie de classes de politiques basée sur leur degré d'abstraction. Les niveaux les plus élevés correspondent à des niveaux organisationnels et les niveaux les plus bas aux mécanismes matériels et logiciels constituant le système.

Les politiques de haut niveau sont intéressantes car elles peuvent être comprises et rédigées par des personnes qui n'ont pas de connaissances techniques relatives à leur mise en œuvre. Par contre, ces politiques sont difficilement utilisables car trop éloignées des réalités du système. On aimerait disposer de politiques dont le vocabulaire est proche du vocabulaire technique.

Une solution est de raffiner des politiques de haut niveau en politiques techniques. Le concept de raffinement de politiques est introduit dans [Wie95] et étendu par [NW96]. Malheureusement, on trouve peu d'exemples concluants de raffinement de politiques de sécurité dans la littérature.

Notre but dans cet article est de produire un moniteur surveillant la conformité des comportements d'un réseau à une politique de sécurité. Cependant, comme la politique de sécurité sera exprimée à un haut niveau d'abstraction, nous devrons la raffiner pour qu'elle soit comparable avec le comportement du réseau. C'est pourquoi, nous décrirons le comportement du moniteur par raffinement. De plus, cette méthode de développement permet de s'inscrire dans un processus de certification tel que celui des critères communs [Cc05].

Pour ce faire, nous utiliserons la méthode B, un formalisme dédié au raffinement prouvé de logiciels. Le processus de raffinement du B étant contrôlé par la résolution d'obligations de preuve, une certaine confiance est acquise quant à l'implantation produite.

Dans une première partie, nous présentons rapidement la méthode B. Dans une seconde partie, nous présentons notre étude de cas en insistant sur nos choix de modélisation et la correction de nos raffinements. Enfin, nous concluons cet article en comparant notre approche aux travaux déjà existants dans le domaine et nous exposons les perspectives de ce travail.

* Ce travail est supporté par une bourse doctorale ministérielle.
† Ce travail est supporté par une bourse doctorale BDI cofinancée par le CNRS et ST Microelectronics.
0 Article publié dans les actes de la conférence Majecstic'2005 (http ://majecstic05.irisa.fr/)

2 NOTIONS DE B

La méthode B, introduite dans [Abr96], est une méthode de développement formel en même temps qu'un langage de spécification. Cette méthode permet d'écrire des modèles mathématiques que l'on peut ensuite raffiner jusqu'à obtenir un code compilable et exécutable.
On appelle raffinement le processus qui consiste à affiner la vue que l'on a d'un concept. Pour citer l'image utilisée par J.R. Abrial, le concept du raffinement est comparable à ce que voit un parachutiste : d'abord des formes grossières, puis de plus en plus de détails. Ainsi, on pourra dire par exemple qu'un ballon est un raffinement d'une sphère.
En B, le niveau le plus abstrait est contenu dans une *machine*. Elle peut être raffinée par des composants nommés *raffinement*. Le dernier d'entre eux est une *implantation* s'il respecte les contraintes des systèmes informatiques (déterminisme, contraintes mémoire, etc).

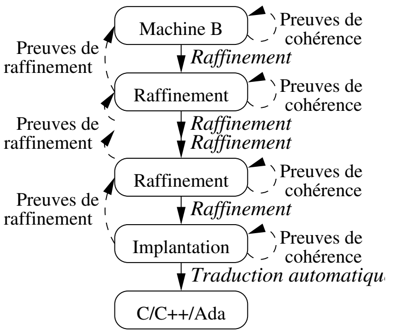

FIG. 1 – Principe de la méthode B

La méthode B permet de prouver que les différentes étapes d'un raffinement sont réalisées correctement grâce à des obligations de preuve. Si celles-ci sont résolues, alors elles garantissent la cohérence du modèle et la correction du raffinement (figure 1).

2.1 Composants B

Un composant B est une machine ou un raffinement contenant des clauses qui permettent de décrire les données *(variables)*, leurs propriétés *(en logique du premier ordre ensembliste)* et la dynamique du système *(initialisation et opérations)*.

2.2 Calcul des obligations de preuve

Dans une composant B, les obligations de preuve servent à vérifier que l'invariant est toujours vrai. La génération de ces obligations est basée sur le calcul du $\mathcal{WP}$ (Weakest Precondition - Plus faible précondition), introduit par Dijkstra [Dij76].

Definition 1 (Cohérence d'une machine.) *Une machine est dite cohérente si, ni l'initialisation, ni aucune des opérations ne permettent de violer l'invariant.*

2.3 Le raffinement en B

Dans la méthode B, les raffinements sont effectués par partie. C'est à dire qu'une initialisation raffine l'initialisation et que chaque opération est raffinée par une opération plus concrète.
Enfin, lors du raffinement on peut changer la représentation des données. Pour cela il suffit de définir de nouvelles variables et de décrire par un invariant leur lien avec les variables abstraites.

Definition 2 (Raffinement de substitutions) *Le raffinement d'une substitution A par une substitution R garantit que R peut au moins traiter les mêmes données que A et que tous les résultats possibles de R sont prévus dans A.*

Par exemple, la substitution $S_1 \parallel S_2$[1] peut se raffiner par S_1 ou par S_2 ou encore par IF C THEN S_1 ELSE S_2 END.

3 PRÉSENTATION DU MODÈLE

3.1 Objectifs et méthodologie

Nous voulons décrire, par raffinement, un logiciel dont le rôle est de surveiller un réseau et d'informer par des alertes toute infraction à la politique de sécurité. De plus, nous voulons que la politique de sécurité soit décrite de manière simple. Celle que nous utilisons représente le contrôle de l'accès des personnes physiques aux services du réseau. C'est pourquoi, nous avons choisi de représenter notre politique par des couples d'utilisateurs et de services ayant ou non le droit d'être connectés les uns aux autres.
Notre modèle B présente 4 niveaux de raffinement. Le plus abstrait d'entre eux décrit la politique de sécurité, tandis que les différents composants du réseau sont ajoutés progressivement par raffinement. La figure 2 résume les étapes de l'introduction des différentes notions.

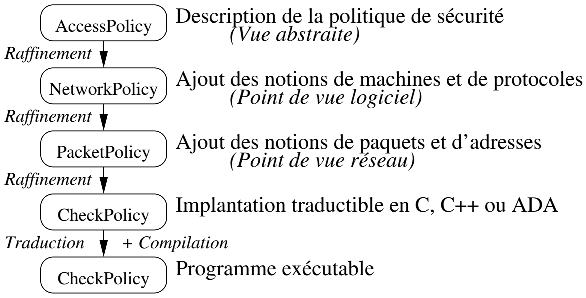

FIG. 2 – Présentation du modèle proposé.

Cette surveillance consiste à vérifier la *conformité* d'un réseau à sa politique. La définition de la notion de conformité est un problème classique, initialement posé par [Abr87], et de nombreuses relations de conformité ont déjà été définies. Nous en retiendrons trois principales :

Sûreté : Tout comportement possible dans le système est prévu par la politique.

Vivacité : Tout comportement prévu par la politique est possible dans le système.

[1] $S_1 \parallel S_2$ revient à dire que l'on exécute soit le code S_1 soit le code S_2, mais que le choix est non déterministe (Comme un IF dont la condition ne serait pas déterminé).

Equivalence : Les comportements prévus par la politique sont les comportements possibles du système.

En ce sens, notre relation de conformité, modélisée par les alertes, est une relation de sûreté mais pas de vivacité. Le schéma figure 3 montre l'insertion d'un moniteur dans une architecture réseau classique. Le moniteur peut voir le trafic sur l'ensemble du réseau grâce aux routeurs. C'est d'après ce trafic qu'il contrôle la conformité à la politique de sécurité.

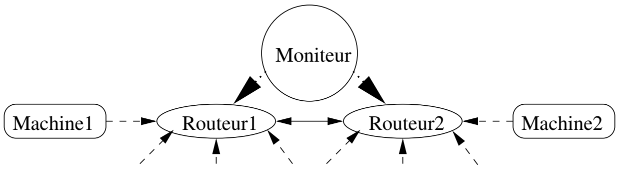

FIG. 3 – Exemple d'implantation du moniteur.

Pour définir une politique de contrôle d'accès, on peut utiliser au choix :

1. Une liste de permissions ;
2. Une liste de permissions et une liste d'interdictions.

Nous avons choisi la première solution, car elle est plus simple, mais surtout plus classique. On considère généralement que toute action qui n'est pas dans la liste des permissions est interdite. On résout ainsi les questions de complétude et de cohérence du modèle qui se seraient posées si nous avions choisi le modèle à deux listes.
Pour une personne et un service donnés, on peut ainsi définir une autorisation ou une interdiction, dont la signification est que la personne a (resp. n'a pas) la permission d'accéder au service. Par exemple, on donne l'autorisation à *Jean-Louis Martin* d'accéder au service *mail IMAG*.

3.2 Description abstraite ($AccessPolicy$)

Dans le modèle, les permissions sont données par l'ensemble $Politique$. Si le couple (u, s) appartient à $Politique$, l'utilisateur u a le droit d'accéder au service s. Sinon u n'a pas la permission d'accéder à s. La variable $message$ représente une abstraction de l'accès réseau observé par le moniteur ($message \in User \times Service$[2]). Enfin, la variable $Alerte$ est un ensemble qui est soit vide, si l'accès ne viole pas la politique, soit ne contenant que le couple (u, s), si u accède à s mais n'en a pas l'autorisation. Ces différentes variables sont liées ensembles par l'invariant suivant :

$$Alerte = \{message\} - Politique \tag{1}$$

Où le moins ensembliste $A - B$ représente l'ensemble des éléments de A qui ne sont pas dans B. Nous appellerons la formule 1 ci-dessus l'invariant de surveillance.

3.3 Premier raffinement ($NetworkPolicy$)

Dans ce premier niveau de raffinement nous introduisons les concepts de machine, de comptes et d'hébergement de services. Cela nous permet de nous rapprocher des réalités physiques, car le message reçu n'est plus un couple (u, s) mais un couple (M_1, M_2) de machines. Comme nous savons quels utilisateurs ont un compte sur M_1 et quels services tournent sur M_2, nous en déduisons l'ensemble des couples (u, s) tels que u ait un compte sur M_1 et s tourne sur M_2.
Cette approximation (ensemble de couples (u, s) au lieu d'un seul couple) se répercute alors sur le message reçu et sur les alertes. Ainsi, nous nous intéresserons ici à la variable $Messages$, qui est l'ensemble des couples (u, s) possibles et telle que $message \in Messages$, et aux variables $AlertesPossibles$ et $AlertesSures$ définies comme suit :

$$AlertesPossibles = Messages - Politique \tag{2}$$

$$(Messages \cap Politique = \emptyset) \Rightarrow (AlertesSures = Messages) \tag{3}$$

$$(Messages \cap Politique \neq \emptyset) \Rightarrow (AlertesSures = \emptyset) \tag{4}$$

Implicitement, une alerte est sûre si tous les couples (u, s) possibles violent la politique, et une alerte est possible si un au moins des couples n'est pas autorisé. Ces invariants de surveillance plus fins serons conservés jusqu'à l'implantation et nous permettent de garantir la correction des alertes produites.
Remarquons que la variable $Alerte$ est liée de implicitement aux nouvelles variables de la manière suivantes :

$$Alerte \subseteq AlertesPossibles \tag{5}$$

$$Alerte = Alerte \cap AlertesSures \tag{6}$$

Ce lien nous permet de diminuer la complexité des obligations de preuve et de faciliter la compréhension du raffinement par rapport à la spécification.
Les informations de configuration du réseau, qui sont utilisées pour retrouver la liste des utilisateurs d'une machine et des services hébergés, peuvent être retrouvées de manière très simple. Par exemple, sur un système Fedora Linux, on obtient la liste des comptes par le fichier */etc/passwd* et la liste des services hébergés dans le dossier */etc/init.d*.

3.4 Deuxième raffinement ($PacketPolicy$)

Dans ce composant, les messages qui circulent sont réels. C'est à dire que l'on connaît également le port d'émission et le port de réception de chacun des messages. Le port est un numéro qui identifie chaque connexion à une machine, de la même manière qu'une maison est identifiée par un numéro dans une rue. Cela est intéressant car certains de ces numéros sont réservés à certains services tels que l'accès à une page web, le retrait des mails, ou la connexion distante à une machine. Ainsi, en rajoutant l'information du port du serveur on peut savoir avec précision quel service est sollicité. Les invariants et propriétés de sécurité sont donc conservés par raffinement.
Sur un système Fedora Linux, ces numéros de ports sont récupérables par la commande *netstat -atpn*.

3.5 Implantation ($CheckPolicy$) et Bilan

Le modèle produit est implanté et peut être compilé et exécuté. L'exécutable obtenu est un moniteur qui signale

[2] $message$ est un couple $(user, service)$).

toute infraction à la politique de sécurité, à condition qu'une copie de chaque paquet circulant sur le réseau lui soit transmise. Une première version de ce modèle a été réalisée jusqu'à l'implantation. Il nécessite la preuve de 1137 obligations de preuves, dont 86 sont encore à prouver.

Enfin, ce moniteur ayant été développé par raffinement et ceux-ci étant justifiés par une facilité de compréhension du modèle grâce à une introduction progressive des concepts, alors il permet de rentrer dans une dynamique de certification selon les critères communs [Cc05].

4 BILAN ET PERSPECTIVES

Ce travail s'intègre dans les avancées du projet PoTestAT[3] et montre la faisabilité de notre approche consistant à surveiller la conformité d'un système à une politique de sécurité. Partant d'une spécification abstraite, où les politiques sont simples à écrire, nous avons dérivé des politiques décrites à un niveau de granularité proche du matériel. L'intérêt principal de ce modèle est que les règles de sécurité sont décrites dès le plus haut niveau. Il suffit alors que ces règles puissent être raffinées pour exprimer les comportements autorisés du système à différents degrés d'abstraction.

[Vig96], [SBC05] proposent des modèles formels pour le test de pare-feux. Bien que le modèle de [Vig96] ressemble au notre, il ne propose pas de définition formelle de la politique de sécurité et les tests doivent être créés manuellement. De même, le modèle de [SBC05] est orienté bas niveau, et il ne présente pas de nécessité de raffinement de politiques.

Par la suite, nous comptons orienter nos travaux vers la dérivation de configuration à partir d'une politique de sécurité. Dans ce domaine, il existe déjà quelques résultats de recherche [Bar99, CCBSM04, MBG99]. Firmato [Bar99] permet de dériver la configuration des pare-feux d'un modèle représentant la topologie et la politique de sécurité du réseau. La non-séparation de la topologie et de la politique de sécurité est gênante. POWER [MBG99] est un outil développé chez Hewlett-Packard pour raffiner des politiques de sécurité. Le processus de raffinement n'est pas automatique et la cohérence des politiques n'est pas analysée. [CCBSM04] propose de modéliser une politique de sécurité en Or-BAC [KBB+03], et d'en dériver les configurations des pare-feux. La politique décrite en Or-BAC est trop technique pour qu'il y ait besoin de la raffiner. Nous nous démarquons de ces travaux par le degré d'abstraction élevé de notre politique et l'automatisation du processus de raffinement.

Quelle que soit l'approche (surveillance, vérification ou dérivation), les mêmes questions se posent (modélisation formelle d'une politique de sécurité, conformité, cohérence, complétude ...). Notre objectif final est de fournir une approche globale en matière de sécurité basée politique. Nous espérons que cette approche permettra de couvrir l'ensemble des moyens apparaissant parmi les méthodes officielles de certification de la sécurité comme les Critères Communs [Cc05].

[3] POlitiques de sécurité : TEST et Analyse par le Test de systèmes en réseau ouvert. Projet financé dans le cadre des ACI Sécurité.

BIBLIOGRAPHIE

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Journal of Theoretical Computer Science*, 53, 1987.

[Abr96] J.R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[Bar99] Bartal. Firmato : A novel firewall management toolkit. *ACM Transactions on Computer Systems*, 22(4), 1999.

[Cc05] Critères-communs. *Common Criteria for Information Technology Security Evaluation, Norme ISO 15408 - version 3.0 Rev. 2*. Common Criteria, 2005.

[CCBSM04] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy. In *Workshop FAST*, 2004.

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[KBB+03] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *IEEE POLICY*, 2003.

[Mas93] M. Masullo. Policy Management : An Architecture and Approach. In *IEEE International Workshop on System Management*, 1993.

[MBG99] M. Casassa Mont, A. Baldwin, and C. Goh. POWER Prototype : Towards Integrated Policy-Based Management. Technical report, HP Laboratories, 1999.

[NW96] B. Neumair and R. Wies. Case study : applying management policies to manage distributed queuing systems. In *Distributed Systems Engineering*, 1996.

[SBC05] D. Senn, D. Basin, and G. Caronni. Firewall Conformance Testing. In *TestCom*, 2005.

[SCFY96] R. S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2) : 38–47, 1996.

[Slo94] M. Sloman. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2(4) : 333–360, 1994.

[Vig96] G. Vigna. A Topological Characterization of TCP/IP Security. Technical report, Technical Report TR-96.156, Politecnico di Milano, 1996.

[Wie95] Wies. Using a classification of Management Policies for Policy Specification and Policy Transformation. In *IFIP/IEEE International Symposium on Integrated Network Management*, 1995.