Développement formel d'un moniteur détectant les violations de politiques de sécurité de réseaux

Vianney Darmaillacq* et Nicolas Stouls †
{Vianney.Darmaillacq, Nicolas.Stouls}@imag.fr
LSR-IMAG - 681, rue de la Passerelle, BP72, 38402 St Martin d'Hères Cedex

Résumé

La gestion de la sécurité des réseaux de type TCP/IP est rendue très difficile par leur hétérogénéité et leur complexité. Pour s'abstraire de ces contraintes on préconise la spécification de politiques de sécurité en termes de haut niveau. L'inconvénient est qu'il s'avère difficile d'établir une concordance entre de telles politiques et les moyens matériels et logiciels d'application de la sécurité. Le but de cet article est de développer formellement, par raffinements successifs, un moniteur qui surveille la conformité d'un réseau à une politique de sécurité abstraite. A chaque niveau de raffinement nous décrivons plus précisément le réseau. Une fonction permet de lier ces différents niveaux de description aux concepts de la politique de sécurité. Nous avons choisi d'utiliser la méthode B, ce qui apporte la garantie, par la preuve, que le moniteur construit détecte toute violation de la politique de sécurité.

Mots clefs : Réseaux, politique de sécurité, méthode formelle, méthode B, raffinement

1 Introduction

Dans les réseaux TCP/IP modernes, l'hétérogénéité et la distributivité croissante des applications rendent la gestion de la sécurité de plus en plus complexe. Afin de donner une vision globale et compréhensible de la sécurité d'un réseau, on cherche à s'abstraire des contraintes techniques en utilisant des politiques de sécurité de haut niveau. Celles-ci expriment les objectifs de sécurité du réseau et spécifient le comportement désiré du système [13].

On identifie différentes manières de garantir le respect d'une politique de sécurité. La **dérivation** qui construit une configuration du système conforme à une politique de sécurité. La **vérification** qui établit la preuve de la conformité de la configuration du système à une politique de sécurité. Le **test actif** qui valide la conformité du système à une politique de sécurité. Enfin la **surveillance** consiste à observer à l'exécution la conformité du comportement du système à la politique de sécurité *(test passif ou monitoring)*.

Pour configurer un réseau conformément à une politique de sécurité, un administrateur se fie à son expertise technique. Il écrit les configurations en fonction de

[0]Article publié dans les actes de la conférence AFADL'2006 (http ://cedric.cnam.fr/AFADL06/)
*Ce travail est supporté par une bourse doctorale ministérielle.
†Ce travail est supporté par une bourse BDI cofinancée par le CNRS et ST Microelectronics.

la politique de sécurité, puis procède à des vérifications et à des tests pour voir s'il n'a pas fait d'erreurs. Cependant, ces dernières ne sont souvent découvertes qu'à l'exécution.

L'ensemble de ces processus peut être formalisé et automatisé. Nous nous proposons dans cet article d'automatiser le processus de surveillance des violations de la politique de sécurité.

Si le haut degré d'abstraction des politiques permet de les comprendre et de les rédiger aisément il rend également le réseau difficile à configurer, car la correspondance entre les deux vocabulaires n'est pas immédiate. La problématique principale de ce travail consiste donc à déterminer comment comparer des événements du réseau et la politique de sécurité. Pour traiter ce problème nous avons décidé de décrire le réseau et lo logiciel surveillant (appelé par la suite le moniteur) par raffinements successifs. A partir d'une représentation abstraite du réseau, chaque raffinement modifie la représentation des événements pour prendre en compte de nouvelles informations techniques. Pour ce faire, nous utilisons la méthode B, un formalisme dédié au raffinement prouvé de logiciels. Nous utiliserons plus exactement le B événementiel dont la sémantique basée sur le déclenchement spontané d'événements nous a paru plus adaptée à cette application.

Dans une première partie, nous présentons rapidement la méthode B. La seconde partie introduit les concepts nécessaires à la compréhension des politiques de sécurité réseau. La troisième partie présente notre approche et nos choix de modélisation. La quatrième partie décrit formellement les différents raffinements qui composent le modèle, mettant en évidence la correction des raffinements et les garanties obtenues. Enfin, nous concluons cet article par un bilan de notre travail, une comparaison avec les travaux existants et une présentation des suites de ce travail.

2 Notions de B événementiel

La méthode B[2] est une méthode de développement formel en même temps qu'un langage de spécification. Elle permet de décrire des modèles mathématiques que l'on peut ensuite raffiner jusqu'à obtenir un code compilable et exécutable. Un mécanisme de génération d'obligations de preuve permet de valider la cohérence des modèles et la correction des raffinements.

Le B événementiel est une extension du langage B [1, 3] où les modèles sont décrits en termes d'événements. Le niveau le plus abstrait est contenu dans un composant appelé *système*. Il peut être raffiné par des composants de *raffinement*. Chaque composant B événementiel contient des clauses décrivant les données (par des ensembles), leurs propriétés (en logique du premier ordre ensembliste) et la dynamique du système (par des substitutions généralisées).

Tout événement a la forme générale suivante : SELECT G THEN S END. Si sa garde G est vérifiée, alors son corps S est exécuté. Si plusieurs événements ont leur garde vérifiée au même moment alors l'un d'entre eux se déclenche (le choix se faisant de manière non déterministe).

2.1 Notations

Opérateur	Signification
$A \mathrel{+\!\!\!\rightarrow} B$	Fonction partielle de A vers B
$A \longrightarrow B$	Fonction totale de A vers B
$\mathsf{dom}(R)$	Domaine de la relation R
$\mathsf{ran}(R)$	Codomaine de la relation R
$\mathbb{F}(A)$	Ensemble des parties finies de A
$\mathbb{F}_1(A)$	Parties finies non vides de A
$A \times B$	Produit cartésien de A et B

Table 1: Opérateurs sur les ensembles

Les données étant définies par des ensembles, les propriétés sont exprimées en logique du premier ordre ensembliste (tableau 1). Notons qu'une *fonction* est une *relation* qui n'associe au plus qu'une image à chaque élément du domaine. Une fonction est *totale* si tous les éléments du domaine ont une image. Sinon elle est *partielle*.

Les substitutions généralisées sont l'équivalent des instructions d'un langage de programmation. Le tableau 2 présente celles que nous utiliserons par la suite.

Substitution	Notation syntaxique	Notation mathématique
Ne rien faire	skip	skip
Assignation	$x := E$	$x := E$
Garde	SELECT P THEN S END	$P \Longrightarrow S$
Choix borné	CHOICE S_1 OR S_2 END	$S_1 \parallel S_2$
Choix non borné	ANY z WHERE P THEN S END	$@z \cdot (P \Rightarrow S)$
Conditionnelle	IF P THEN T_1 ELSE T_2 END	$P \Longrightarrow T_1 \parallel \neg P \Longrightarrow T_2$

Table 2: Substitutions primitives

2.2 Obligations de preuve

Dans un système B événementiel, les obligations de preuve servent à vérifier que l'invariant est vrai. Il doit donc être établi par l'initialisation et préservé par chaque événement. La génération des obligations des preuves est basée sur le calcul de la plus faible pré-condition ($\mathcal{WP}$), introduit par Dijkstra [7]. On note $[S]R$ la plus faible pré-condition telle que la substitution S termine et mène à la post-condition R.

Un système M est dit **cohérent** par rapport à l'invariant I si ni l'initialisation $Init$, ni aucun des événements Ev ne permettent de violer I.

2.3 Le raffinement en B événementiel

Lors du raffinement, on peut modifier la représentation des données. L'invariant de liaison L permet alors de décrire la relation liant les variables abstraites aux variables concrètes.

Une substitution A est raffinée par une substitution R pour l'invariant L si R peut au moins traiter les mêmes états du système que A et que tous les états de sortie possibles de R ont été prévus dans A.

Les raffinements sont effectués par partie : l'initialisation $Init_R$ raffine l'initialisation $Init_A$ et chaque événement Ev_A est raffiné par un événement Ev_R plus concret. Il est également possible d'introduire de nouveaux événements correspondant à l'événement skip. Cela correspond au *stuttering* (ou *bégaiement*) en TLA [9].

En B événementiel, les gardes peuvent être restreintes par raffinement. C'est pourquoi il faut prouver qu'il y a toujours au moins un événement déclenchable. Enfin, lors de l'introduction de nouveaux événements, on doit également prouver que ceux-ci ne prennent pas la main indéfiniment.

3 Introduction aux politiques de sécurité sur les réseaux

Soit $Action$ l'ensemble des événements qui peuvent être autorisés ou interdits par une politique de sécurité. Dans le cadre de cet article, ces événements représentent l'accès d'un utilisateur du réseau à une ressource. Nous inspirant de [11], nous donnons la définition suivante d'une politique de sécurité :

Une politique de sécurité définie sur un ensemble d'actions est une propriété indiquant pour chaque action si elle est autorisée ou non.

Voici trois manières différentes de définir une politique :

1. avec un ensemble A_+ spécifiant les actions autorisées (politique *restrictive*),
2. avec un ensemble A_- spécifiant les actions interdites (politique *permissive*),
3. avec un ensemble A_+ spécifiant les actions autorisées et un ensemble A_- spécifiant les actions interdites.

Une politique est dite *incomplète* s'il existe une action pour laquelle aucune permission ou interdiction n'est spécifiée. Ce problème ne se pose pas dans les cas 1 et 2, car une règle par défaut est utilisée, spécifiant que toute action non autorisée est interdite ou inversement. Par contre dans le cas 3, l'*incomplétude* est caractérisée par l'existence d'une action qui n'est ni autorisée ni interdite par la politique ($A_+ \cup A_- \subset Action$).

Une politique est dite *incohérente* si une action est interdite et autorisée par la politique. Un tel *conflit* n'est possible que dans le cas 3, si $A_+ \cap A_- \neq \emptyset$. Pour l'éviter, on peut soit interdire que deux règles se contredisent, soit proposer une procédure pour résoudre les conflits. On se rapportera à [8] ou [10] pour une étude du sujet dans le cadre de politiques de contrôle d'accès.

L'application d'une politique de sécurité à un système réel pose le problème de la correspondance entre les actions spécifiées par la politique et les événements du système. Notons $Event$ l'ensemble des événements observables sur le réseau. Nous définissons alors la conformité d'un réseau à une politique de sécurité comme suit :

Définition 1 *Un réseau $\mathcal{R}$ est conforme à une politique $\mathcal{P}$ si aucun événement de $\mathcal{R}$ n'est lié à une action non autorisée par $\mathcal{P}$.*

Le lien entre les événements du réseau et les actions de la politique est établi par la fonction $mapping$:

$$mapping \in Event \longrightarrow \mathbb{F}(Action)$$

$mapping$ associe à un événement un ensemble d'actions[1] plutôt qu'une action unique car, dans le cadre d'une application réseau, les observations bas niveau sont souvent insuffisantes pour définir précisément l'action correspondant à un événement donné (cf. exemple de la section 4.1).

Le lien entre un réseau et une politique est dit *incomplet* (figure 1.a) s'il existe un événement de bas niveau qui n'est associé à aucune action de haut niveau.

Le lien entre un réseau et une politique est dit *incohérent* (figure 1.b) si la fonction $mapping$ associe à un événement de haut niveau au moins deux actions telles que l'une est autorisée par $\mathcal{P}$ et l'autre non.

a. Incomplétude b. Incohérence

Figure 1: Exemple d'incomplétude et d'incohérence du lien entre $\mathcal{R}$ et $\mathcal{P}$ dans le cas d'une politique restrictive (A_+)

4 Présentation générale du modèle

4.1 Objectifs

Notre but est de construire, par raffinement, un exécutable pouvant surveiller un réseau et garantir que toute violation de la politique de sécurité sera détectée et signalée (Vérification de la conformité, définition 1). Pour cela, nous devons garantir la propriété suivante à tous les niveaux de raffinement du moniteur :

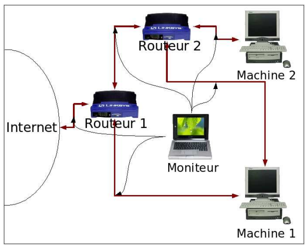
Toute violation de la politique de sécurité dans le réseau est détectée et archivée.

De plus, le moniteur doit assurer une cohérence minimale des données de configuration fournies par l'utilisateur.

Afin d'illustrer ces objectifs, voici un court exemple (figure 2) décrivant un réseau avec sa politique de sécurité, sa configuration et sa disposition physique. Notons que le moniteur a été installé de telle sorte qu'il puisse récupérer une copie de chaque paquet circulant sur le réseau.

Si l'on observe une connexion de $Machine_1$ vers le port 80 de $Machine_2$, alors l'émetteur ne peut être que $Jean$ ou $Alice$, car ils sont les seuls à être référencés comme utilisateurs de $Machine_1$. De plus, le service accédé est $Intranet$,

[1] Rappelons que $\mathbb{F}(A)$ est l'ensemble des parties finies de A.



Disposition matérielle du réseau

$$A_+ = \{(Jean, Intranet), (Jean, Mail), (Alice, Mail)\}$$

Politique de sécurité

$Machine_1$ est utilisée par $Jean$ ou $Alice$
$Intranet$ est hébergé sur le port 80 de $Machine_2$
$Mail$ est hébergé sur le port 993 de $Machine_2$.

Configuration du réseau

Figure 2: Exemple complet d'utilisation d'un moniteur

car c'est lui qui tourne sur le port 80 de $Machine_2$. Enfin, seul $Jean$ a l'autorisation de s'y connecter d'après l'ensemble des autorisations A_+. On conclut alors à un risque de violation de politique.

Si par contre le message va de $Machine_1$ au port 993 de $Machine_2$, alors le service demandé est le $Mail$, auquel tous les utilisateurs de $Machine_1$ ont le droit d'accès. On est donc sûr que ce message est conforme à la politique.

4.2 Description des niveaux de raffinement

Dans un réseau TCP/IP, les machines utilisent pour communiquer entre elles des protocoles de communication répartis sur plusieurs couches. Nous avons choisi de faire correspondre les différents niveaux de raffinement aux couches du modèle TCP/IP (tableau 3).

Niveau de spécification	Concepts réseau	Couche TCP/IP
0	utilisateurs, ressources	
1	serveurs, serveurs de terminaux	application
2	machines, ports	transport
3	interfaces	réseau
4	adresses	réseau

Table 3: Evénements observables suivant le niveau de raffinement

Au niveau 0 du modèle, les événements observés représentent l'accès d'un utilisateur à une ressource. Au niveau 1, les événements observés représentent les communications entre un serveur de terminaux et un serveur[2] (couche *application* du modèle TCP/IP). Au niveau 2, les événements observés représentent les connexions d'une machine sur un port de communication d'une autre machine (couche *transport* du modèle TCP/IP). Au niveau 3, les événements observés représentent les connexions d'une interface à un port de communication d'une autre interface (couche *réseau* du modèle TCP/IP). Le niveau 4 est l'implémentation du moniteur et se situe au même niveau de détail que le précédent.

[2]Un serveur est un logiciel hébergeant des ressources sur lequel des utilisateurs peuvent se connecter en utilisant un terminal (ressource fournie les serveurs de terminaux).

La deuxième colonne du tableau 3 montre les concepts réseaux introduits à chaque niveau de raffinement et représentant la configuration du réseau. Ces informations sont fournies par l'administrateur et doivent être cohérentes entre elles, conformément aux contraintes décrites dans le tableau 4.

	Contraintes Logicielles
1.	Tout serveur fournit au moins une ressource.
2.	Toute ressource est fournie par au moins un serveur.
3.	Tout serveur de terminal peut être utilisé par au moins un utilisateur.
4.	Tout utilisateur peut accéder à au moins un serveur de terminal.
5.	Tout serveur hébergé sur une machine est associé à au moins un port.
6.	Toute machine héberge au moins un serveur.
7.	Toute machine possède au moins une interface.
	Contraintes Matérielles
8.	Tout serveur est hébergé par une machine.
9.	Tout serveur de terminal est hébergé sur une machine.
10.	Tout serveur associé à un port d'une machine y est hébergé.
11.	Chaque interface n'est que sur une seule machine.

Table 4: Exemple de règles de cohérence entre les données de configuration

4.3 Choix de modélisation

Nous avons choisi d'utiliser une politique restrictive où l'ensemble A_+ contient les actions autorisées. Les actions qui ne sont pas autorisées sont interdites.

$$A_+ \subseteq Action$$

Nous notons $User$ l'ensemble des personnes physiques identifiées comme des utilisateurs du réseau et $Resource$ l'ensemble des ressources fournies par le réseau.

$$Action = User \times Resource$$

Notons $Event_i$ l'ensemble des événements observables sur le réseau, dont la représentation au i^{ème} niveau de raffinement est donnée dans le tableau 5. $Monitored_i$ est le journal des événements observés sur le réseau au i^{ème} niveau de raffinement, idE est un ensemble d'identificateurs servant à dissocier les occurrences d'événements et $IdUsed$ est l'ensemble des identificateurs de idE qui sont déjà utilisés ($IdUsed \subseteq idE$) :

Niveau de raffinement	$Event_i$
0	$User \times Resource$
1	$Server \times Server$
2	$Machine \times Machine \times Port$
3	$Interface \times Interface \times Port$
4	$Interface \times Interface \times Port$

Table 5: Evénements observables suivant le niveau de raffinement

$$Monitored_i \in IdUsed \longrightarrow Event_i$$

Les différentes représentations de $Event_i$ sont reliées par la fonction $refines_i$, de profil :

$$refines_i \in Event_i \longrightarrow \mathbb{F}(Event_{i-1})$$

Comme montré dans l'exemple de la section 4.1, les événements observés sur le réseau ne permettent pas toujours de trouver un unique événement abstrait correspondant à cet événement concret.

Enfin, nous définissons la fonction $mapping_i$ qui lie la représentation des événements, au niveau i, aux actions de la politique de sécurité :

$$mapping_i \in Event_i \longrightarrow \mathbb{F}(Action)$$

Cette fonction est définie par la formule (φ_1) suivante :

$$\boxed{mapping_i(e_i) = \{a \mid \exists e_{i-1} \cdot (e_{i-1} \in refines_i(e_i) \wedge a \in mapping_{i-1}(e_{i-1}))\} \quad (\varphi_1)}$$

L'événement $check\_event$ analyse la conformité des événements observés sur le réseau par rapport à la politique. Ceux qui ne sont pas conformes voient leur identificateur ajouté à l'ensemble $Alerte_i$, qui représente le journal des violations détectées par le moniteur. Nous avons choisi d'implanter une propriété invariante à chacun des niveaux de raffinement afin de garantir par la preuve que la conformité est correctement vérifier :

Propriété 1

$$\forall j \cdot (j \in IdUsed \Rightarrow (j \in Alerte_i \Leftrightarrow mapping_i(Monitored_i(j)) \not\subseteq A_+))$$

Afin de vérifier cette propriété à chaque niveau de raffinement nous avons introduit l'observateur $IsAlert$. Ce sont les obligations de preuve liées au raffinement de cet événement qui nous permettent de garantir la conservation de la propriété.

Notons enfin que pour simplifier le modèle présenté, nous considérons ici que tous les messages vont d'un client vers un serveur. Sinon la source et la destination sont interverties avant le déclenchement de $check\_event$.

5 Description du modèle

Dans cette section nous allons présenter chacun des niveaux de spécification en mettant l'accent sur la définition et la validation de la propriété 1.

5.1 Niveau 0 : vue utilisateur-ressource du moniteur

Les ensembles $User$, $Resource$, $Action$, $Event_0$, A_+ et $mapping_0$ sont des constantes données par l'utilisateur. Les autres données sont des variables du système. Notons que l'ensemble $Alerte_0$ est défini ici par la variable $FAIL$.

$$\begin{aligned}
& Action = User \times Resource \\
\wedge\ & A_+ \subseteq Action \\
\wedge\ & Event_0 = User \times Resource \\
\wedge\ & \forall e \cdot (e \in Event_0 \Rightarrow mapping_0(e) = \{e\}) \\
\wedge\ & Monitored_0 \in IdUsed \longrightarrow Event_0 \\
\wedge\ & FAIL \subseteq IdUsed
\end{aligned}$$

En particulier, notons que nous définissons ici le cas de base $mapping_0$ par l'association $mapping_0(e) = \{e\}$. Enfin, par instanciation de la propriété 1 nous décrivons l'*invariant de surveillance* (φ_2) suivant :

$$\boxed{\forall i \cdot (i \in IdUsed \Rightarrow (i \in FAIL \Leftrightarrow mapping_0(Monitored_0(i)) \not\subseteq A_+)) \qquad (\varphi_2)}$$

Initialement, les journaux sont vides : $mapping_0 := \emptyset$, $IdUsed := \emptyset$ et $FAIL := \emptyset$.

Lorsqu'un message est observé sur le réseau, $check\_event$ (figure 3.a) se déclenche et le stocke dans le journal $Monitored_0$. Si la politique A_+ a été violée alors l'identificateur de l'événement observé est enregistré dans $FAIL$. Notons qu'à ce niveau de spécification nous savons encore avec certitude si la politique a été violée ou non car $mapping_0(e) = \{e\}$. Ce ne sera plus le cas dans les raffinements suivants, comme le montre l'exemple de la section 4.1. L'événement $IsAlert$ (figure 3.b) est l'observateur qui, pour un identificateur Id donné, dit si l'événement associé viole la politique.

```
check_event ≙ ANY e WHERE e ∈ Event0 THEN
  ANY idf WHERE idf ∈ idE − IdUsed THEN
    IdUsed := IdUsed ∪ {idf} ||
    Monitored0(idf) := e ||
    IF mapping0(e) ⊈ A+ THEN
      FAIL := FAIL ∪ {idf}
    END
  END
END
```

a. Evènement $check\_event$

```
Id, Res ⟵ IsAlert ≙
ANY i WHERE i ∈ idE THEN
  Id := i
  IF i ∈ FAIL
  THEN Res := true
  ELSE Res :∈ BOOL
  END
END
```

b. Observateur de $FAIL$

Figure 3: Code de l'événement $check\_event$ et de l'observateur $IsAlert$

Comme dans ce niveau de spécification la propriété 1 est décrite dans l'invariant alors la vérification des obligations de preuve permet de garantir que tout violation de la politique est notée dans $FAIL$.

5.2 Niveau 1 : vue serveurs

Conformément au tableau 3, les ensembles de serveurs ($Server$) et de serveurs de terminaux ($TerminalServer$) sont décrites à ce niveau. Les constantes $provide$ et $used\_by$ sont des fonctions permettant respectivement de connaître les ressources fournies par chaque serveur et les utilisateurs pouvant accéder à un serveur de terminaux. Afin de garantir la cohérence des données de configuration nous décrivons ici les contraintes 1 à 4 du tableau 4, qui seront vérifiées à la valuation :

(1) $\quad provide \in Server \longrightarrow \mathbb{F}_1(Resource)$
(2) $\wedge\, \forall r \cdot (r \in Resource \Rightarrow \exists s \cdot (s \in Server \;\wedge\; r \in provide(s)))$
(3) $\wedge\, used\_by \in TerminalServer \longrightarrow \mathbb{F}_1(User)$
(4) $\wedge\, \forall u \cdot (u \in User \Rightarrow \exists s_t \cdot (s_t \in TerminalServer \;\wedge\; u \in used\_by(s_t)))$

La représentation des événements réseau est maintenant définie par :

$$
\begin{aligned}
& Event_1 = TerminalServer \times Server \\
\wedge\ & TerminalServer \subseteq Server \\
\wedge\ & \forall (s_t, s) \cdot ((s_t, s) \in Event_1 \Rightarrow refines_1(s_t, s) = used\_by(s_t) \times provide(s))
\end{aligned}
$$

Par application de la formule (φ_1) on a :

$$(u, r) \in mapping_1(s_t, s) \Leftrightarrow \exists e_0 \cdot (e_0 \in refines_1(s_t, s) \wedge (u, r) \in mapping_0(e_0))$$

Les différentes actions associées par $mapping_1$ à un même événement peuvent être conflictuelles : l'une peut être autorisée et une autre interdite (cf exemple de la section 4.1). C'est pourquoi nous introduisons l'ensemble $CONFLICT$, contenant les événements observés pour lesquels on ne peut pas décider s'ils correspondent ou non à une violation. Ainsi, nous décrivons ici $Alerte_1$ avec les deux nouvelles variables $CONFLICT$ et $FAIL_1$, remplaçant la variable $FAIL$, tel que $FAIL_1$ soit l'ensemble des éléments de $FAIL$ n'introduisant pas de conflit. Nous avons :

$$
\begin{aligned}
& FAIL \subseteq FAIL_1 \cup CONFLICT \\
\wedge\ & FAIL_1 \subseteq FAIL \\
\wedge\ & FAIL_1 \cap CONFLICT = \emptyset \\
\wedge\ & (i \in FAIL_1 \Leftrightarrow (mapping_1(Monitored_1(i)) \cap A_+ = \emptyset)) \\
\wedge\ & (i \in CONFLICT \Leftrightarrow (i \notin FAIL_1 \ \wedge\ mapping_1(Monitored_1(i)) \nsubseteq A_+))
\end{aligned}
$$

L'observateur $IsAlert$ est donc raffiné comme suit :

IF $i \in FAIL_1 \cup CONFLICT$ THEN $Res :=$ true ELSE $Res :=$ false END

Le raffinement de cet observateur nous permet de garantir que la propriété 1 est vérifiée, car l'invariant de surveillance de la spécification abstraite, donné formule φ_2, garantit que tout événement violant la politique est dans $FAIL$. Il suffit alors de montrer que $i \in FAIL \Rightarrow i \in FAIL_1 \cup CONFLICT$. Or, l'obligation de preuve liée au raffinement de $IsAlert$ [2, §11.3.3] donne :

$$L \ \wedge\ i \in idE \quad \Rightarrow \quad i \in FAIL \Rightarrow i \in FAIL_1 \cup CONFLICT$$

où L est la conjonction des invariants des niveaux de spécification 0 et 1. Cette obligation de preuve se vérifie par l'hypothèse $FAIL \subseteq FAIL_1 \cup CONFLICT$, garantissant ainsi que toute violation de la politique est détectée.

5.3 Niveau 2 : introduction des machines

Comme décrit dans le tableau 3, ce niveau introduit les notions de $Machine$ et de $Port$. La fonction run_on indique quel serveur est hébergé sur une machine et un port donnés et la fonction $hosting$ définie l'ensemble des serveurs hébergés par une machine. Notons que la fonction run_on est partielle, car tous les ports d'une machine ne sont pas nécessairement utilisés. Les deux constantes run_on et $hosting$ sont étroitement liées (Conditions 5, 6 et 8 à 10 du tableau 4) :

$$
\begin{array}{ll}
 & run\_on \in Machine \times Port \mapsto Server \\
(6) & \wedge\ hosting \in Machine \longrightarrow \mathbb{F}_1(Server) \\
(5+10) & \wedge\ \forall (s,m) \cdot (s \in Server\ \wedge\ m \in Machine \Rightarrow \\
 & \qquad (s \in hosting(m) \Leftrightarrow \exists p \cdot (p \in Port\ \wedge\ s \in run\_on(m,p)))) \\
(8+9) & \wedge\ \forall s \cdot (s \in Server \Rightarrow \exists m \cdot (m \in Machine\ \wedge\ s \in hosting(m)))
\end{array}
$$

On représente ici les événements et leur lien de raffinement $\mathit{refines}_2$ comme suit :

$$
\begin{array}{l}
\quad Event_2 = \{m_1, m_2, p \mid m_1 \in Machine\ \wedge\ (m_2, p) \in \mathsf{dom}(run\_on) \\
\qquad\qquad\qquad\qquad \wedge\ hosting(m_1) \cap TerminalServer \neq \emptyset\} \\
\wedge\ \forall (m_1, m_2, p) \cdot ((m_1, m_2, p) \in Event_2 \Rightarrow \\
\quad \mathit{refines}_2(m_1, m_2, p) = (hosting(m_1) \cap TerminalServer) \times \{run\_on(m_2, p)\})
\end{array}
$$

Pour différentes raisons (paquets forgés, connexions anonymes, ...), les événements circulant sur le réseau ne sont pas forcément bien formés. Ils peuvent ainsi être, par exemple, à destination d'un port inutilisé d'une machine. Ces événements ne sont pas prévus dans la configuration du réseau et n'existent pas dans les niveaux d'abstraction supérieurs. Il en découle qu'ils ne peuvent pas être comparés à la politique de sécurité. C'est pourquoi nous considérons qu'ils ne peuvent pas être décrit par l'ensemble $Event_2$, ce qui explique sa définition. Nous introduisons donc l'événement $event\_filter$ (figure 4) traitant uniquement ces messages en les stockant dans un nouvel ensemble : $UNSPEC$. Ils y sont identifiés par des identificateurs idE_F, différents de ceux de idE. On note $IdUsed_F$ ceux qui sont déjà utilisés.

$$
\begin{array}{l}
\quad idE_F \cap idE = \emptyset\ \wedge\ IdUsed_F \subseteq idE_F \\
\wedge\ UNSPEC \in IdUsed_F \longrightarrow ((Machine \times Machine \times Port) - Event_2)
\end{array}
$$

```
event_filter ≙ ANY e WHERE e ∈ Machine × Machine × Port ∧ e ∉ Event₂
THEN
  ANY idf WHERE idf ∈ idE_F − IdUsed_F THEN
    IdUsed_F := IdUsed_F ∪ {idf} ||
    UNSPEC(idf) := e
  END
END
```

Figure 4: Code de l'événement $event\_filter$

Notons que la garde de $check\_event$ ne change pas et que celui-ci ne se déclenche pas si l'événement réseau reçu ne fait pas partie des cas décrits par la configuration. Les deux événements ont donc des gardes complémentaires.

Enfin, l'événement $check\_event$ et les ensembles $CONFLICT$ et $FAIL_1$ restent inchangés. Ils garantissent donc toujours la propriété 1.

5.4 Niveau 3 : introduction des interfaces

Dans ce dernier raffinement, nous introduisons la notion d'$Interface$. Une interface peut se résumer dans la plupart des cas à une carte réseau. La fonction $interface\_of$ définit la machine sur laquelle une interface est installée (Conditions 7 et 11 du tableau 4) :

$$\begin{array}{ll} (11) & interface\_of \in Interface \longrightarrow Machine \\ (7) & \mathsf{ran}(interface\_of) = Machine \end{array}$$

On représente ici les événements de la manière suivante :

$$\begin{array}{l} Event_3 = \{I_1, I_2, p \mid I_1 \in Interface \ \wedge \ I_2 \in Interface \\ \qquad\qquad \wedge\ (interface\_of(I_2), p) \in \mathsf{dom}(run\_on) \\ \qquad\qquad \wedge\ hosting(interface\_of(I_1)) \cap TerminalServer \neq \emptyset\} \\ \wedge\ \forall(I_1, I_2, p) \cdot ((I_1, I_2, p) \in Event_3 \\ \qquad \Rightarrow refines_3(I_1, I_2, p) = \{(interface\_of(I_1), interface\_of(I_2), p)\}) \end{array}$$

Dans ce niveau de spécification, nécessaire pour aller vers une implantation, nous conservons les mêmes journaux et les mêmes événements. La propriété 1 est donc préservée.

5.5 Niveau 4 : implantation

Comme le langage B événementiel n'est pas directement implantable, nous avons choisi de traduire notre modèle en B classique. Pour cela, nous avons utilisé une méthode ad-hoc basée sur le fait que la garde de $check\_event$ est restée inchangée par raffinement et que la seule modification de la garde de l'événement $event\_filter$ correspond au raffinement de skip. Nous avons alors simplement remplacé les événements $e \mathrel{\hat{=}}$ ANY V WHERE C THEN S END par des opérations $e(V) \mathrel{\hat{=}}$ PRE C THEN S END.

De plus, la sémantique du B événementiel diffère du B classique par l'internalisation du contrôle et la garantie de l'absence de deadlock. C'est pourquoi nous avons également développé une machine d'appel et mis une assertion vérifiant qu'au moins une opération est toujours appelable (preuve d'absence de deadlock). Notons que comme les deux gardes forment une partition de l'invariant, la machine d'appel est totalement déterministe.

Les obligations de preuves restent similaires et ont été revérifiées de manière automatique, garantissant la correction de notre adaptation. Nous en concluons que la propriété 1 est encore vraie dans le modèle B classique.

Pour l'implantation, nous avons choisi d'encapsuler les constantes représentant la configuration du réseau dans un composant à part ($ConfReseau$). Cela nous permet de construire un moniteur générique, indépendamment de la valeur des constantes. Le moniteur est donc prouvé une fois pour toute et l'utilisateur n'a plus qu'à valuer les constantes et prouver les obligations de preuve de cohérence des constantes (tableau 4). Une approche similaire a été utilisée dans Météor [4], le métro parisien sans chauffeur, pour développer des composants réutilisables.

Pour vérifier la cohérence des messages avec la politique, nous devons stocker cette dernière en mémoire. Nous avons alors le choix entre traduire les actions de la politique en événements de bas niveau ou conserver une politique de haut niveau avec des fonctions de traduction. Afin d'éviter l'explosion combinatoire de l'espace mémoire nécessaire, nous avons choisi la seconde solution. De plus, nous avons choisi de ne pas mémoriser les fonctions $mapping_i$ et $refines_i$ mais seulement les données de configurations permettant de réaliser dynamiquement le calcul correspondant à ces fonctions.

Les constantes représentant la configuration du réseau doivent être valuées dans l'implantation de $ConfReseau$. Par exemple, chaque interface doit être valuée par une adresse IP. La principale difficulté de la valuation réside dans la récupération des données de configuration du réseau. Cependant, ce travail peut être automatisé à partir de fichiers existants sur les machines pour la quasi-totalité des cas, comme le montre le tableau 6 dans le cas de la distribution Fedora de Linux. Au final, seules la politique de sécurité et la correspondance ressource-serveur doivent être saisies par l'utilisateur.

Constante	Source des données
$TerminalServer$ et $User$	/etc/passwd
run_on et $Server$	/etc/init.d/
$provide$ et $used\_by$	fichiers de configuration des différents serveurs
$port\_of$	/etc/services
$interfaces\_of$	/etc/sysconfig/networking/devices/

Table 6: Exemple de fichiers de configuration sur un système Fedora Linux

Les variables $FAIL_1$, $CONFLICT$ et $UNSPEC$ ne correspondent pas à des structures de données du langage B0. Nous avons choisi de ne conserver en mémoire que les $LogSize$ derniers événements et de stocker les plus anciens dans des fichiers. Cependant, la propriété 1 est toujours garantie car il suffit qu'elle soit vraie sur le dernier événement observé et à l'initialisation pour qu'elle soit vraie sur l'historique. Les tableaux $IdLog$ et $StatusLog$ permettent d'associer un statut ($Fail$, $Conflict$, $Unspec$ ou $Pass$) à chacun des identificateurs en mémoire remplaçant ainsi les variables $FAIL_1$, $CONFLICT$ et $UNSPEC$.

5.6 Retour sur les objectifs fixés

Sur le modèle présenté et implanté, l'*AtelierB* génère un total de 553 obligations de preuve (plus 1410 obligations de preuve triviales). Celles-ci ont pu être prouvées à 100% (mais pas toujours automatiquement) nous permettant de garantir la propriété 1 sur l'implantation. De plus, les données de configuration à fournir sont décrites avec un certain nombre de contraintes qui, si elles sont vérifiées, permettent de garantir la cohérence des données.

6 Bilan et perspectives

Ce travail s'intègre dans les avancées du projet POTESTAT[3] [6], dont le but est la recherche de méthodes de test de sécurité sur un réseau à partir d'une description haut niveau de la politique de sécurité.

Cet article montre la faisabilité de notre approche, qui consiste à surveiller la conformité des événements se produisant sur un réseau vis-à-vis d'une politique de sécurité. Pour ce faire, nous avons décrit une relation de conformité correspondant au respect de la politique par le réseau aux différents niveaux d'abstraction. Nous avons également défini une propriété de sécurité garantissant l'alerte en cas de non respect de cette relation (propriété 1).

Les travaux [14] et [12] se sont également intéressés à la modélisation formelle de la conformité d'un réseau à une politique de sécurité, dans le cadre du test actif. Le modèle de [14] supporte l'ensemble des couches TCP/IP tandis que le modèle de [12] restreint son champ d'application aux couches réseau et transport. L'approche présentée dans notre article se différencie de ces travaux par la formalisation de la politique de sécurité et la correspondance établie entre elle et les événements de bas niveau.

L'approche de surveillance choisie ici peut être améliorée avec la prise en compte d'un plus grand nombre d'informations. Nous pourrions, par exemple, inspecter le contenu des paquets, afin de construire une trace des commandes observées ou vérifier que le protocole utilisé correspond au numéro de port. Une autre possibilité d'extension serait d'observer les routes suivies par les messages pour diminuer les risques de *spoofing d'adresse*[4].

Dans l'approche surveillance présentée ici nous analysons la cohérence des données de configuration fournies par l'administrateur, ou extraites des fichiers de configuration. L'approche *vérification* présentée dans l'introduction requiert l'utilisation du même modèle du comportement du réseau basé sur sa configuration. Nous prévoyons donc de continuer à étudier les contraintes de cohérence nécessaires au bon fonctionnement du réseau et à l'expression de la politique de sécurité.

Nous avons l'intuition que certains aspects de la sécurité ne sont observables que par une combinaison de différentes approches (vérification, surveillance, test). C'est pourquoi nous voulons fournir un modèle permettant de faire cohabiter ces techniques à partir d'une même modélisation. Un tel résultat nous permettrait de proposer une méthode pour couvrir formellement l'ensemble des moyens d'assurance de sécurité décrits par les méthodes de certification telles que les Critères Communs [5].

---

[3] POlitiques de sécurité : TEST et Analyse par le Test de systèmes en réseau ouvert. Projet financé dans le cadre des ACI Sécurité - http ://www-lsr.imag.fr/POTESTAT/

[4] Attaque consistant à mettre une adresse d'émetteur qui n'est pas la sienne.

Remerciements Nous tenons à remercier *Marie-Laure Potet*, *Roland Groz* et *Jean-Luc Richier* pour leurs innombrables relectures et leurs conseils aussi nombreux que pertinents.

References

[1] J.R. Abrial. Extending B Without Changing it. In H. Habrias, editor, *First B conference*, 1996.

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

[3] J.R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert, editor, *Proceedings of the Second International B Conference*, volume 1393 of *LNCS*. Springer-Verlag, 1998.

[4] P. Behm, P. Benoit, A. Faivre, and J-M. Meynadier. Meteor : A Successful Application of B in a Large Project. In *FM'99 - Formal Methods : World Congress on Formal Methods in the Development of Computing Systems*, volume 1708 of *LNCS*, pages 369–387. Springer-Verlag, 1999.

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

[6] V. Darmaillacq, J.-C. Fernandez, R. Groz, L. Mounier, and J.-L. Richier. Eléments de modélisation pour le test de politiques de sécurité. In *Colloque sur les RIsques et la Sécurité d'Internet et des Systèmes, CRiSIS, Bourges, France*, 2005.

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

[8] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symp. on Research in Security and Privacy*, 1997.

[9] L. Lamport. A Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3) :872–923, may 1994.

[10] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6), 1999.

[11] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1), 2000.

[12] D. Senn, D. Basin, and G. Caronni. Firewall Conformance Testing. In *TestCom*, volume 3502 of *LNCS*, pages 226–241, 2005.

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

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