MASTER EN SCIENCES INFORMATIQUES

Modélisation et déploiement des politiques de sécurité

Vu, Huu Thanh

Award date:
2009

Awarding institution:
Universite de Namur

Modélisation et Déploiement des politiques de sécurité

VU Huu Thanh

Master 120 en sciences informatiques

Année académique 2008-2009

Remerciements

Je tiens tout d’abord à remercier chaleureusement mon encadrant, Madame Nora CUPPENS-BOULAHIA qui m’a admis dans le stage à l’équipe SERES de l’école TELECOM Bretagne, qui m'a guidé tout au long de ce stage et qui m'a apporté, jour après jour, son jugement scientifique pertinent et enrichissant mais aussi des nouvelles perspectives et orientations pour mon travail de recherche.

Je suis profondément reconnaissant à mon promoteur, Monsieur Professeur Jean-Noël COLIN pour sa patience et ses renseignements qui ont largement contribué aux avancement du mémoire.

Je tiens à remercier Monsieur Professeur Laurent SCHUMACHER qui a accepté de diriger mon mémoire de fin d’étude au départ.

Je tiens à remercier Monsieur Professeur Frédéric CUPPENS qui m’a envoyé les cours du modèle Or-BAC.

J’adresse aussi mes remerciements à Doctorant Stere PREDA pour les orientations et l’aide qu’il m’a apportées.

Je remercie également toutes les personnes qui m’ont apporté leur aide précieuse.

Résumé

Nous présentons dans ce mémoire une approche systématique pour modéliser et déployer des politiques de sécurité vers des composants dédiés et non-dédiés à la sécurité dans le réseau d'une organisation. Cette approche est basée sur l'architecture d'un système répondant aux menaces en intégrant la méthode formelle Event-B pour spécifier des politiques de sécurité et vérifier formellement leur propriétés, les technologies XML (*eXtensible Markup Language*) pour représenter des politiques de sécurité et les protocoles de la gestion et la configuration de réseau NETCONF (*Network Configuration Protocol*) et SNMP (*Simple Network Management Protocol*) pour déployer des règles de sécurité concrètes. Pour ce faire, le modèle de contrôle d'accès basé sur l'organisation Or-BAC (*Organization Based Access Control)* a été utilisé. Il a distingué la definition générique de la politique de son implémentation effective en fonction du contexte, pour la représentation des politiques de sécurité et leur fonctionnement.

Mots-clés : Politiques de sécurité, Or-BAC, Event-B, XML, NETCONF, SNMP

Abstract

This thesis shows how to model all security policies and deploy them into security devices or even for non dedicated ones of an organized network in a systematic way. It's based on threat response system architecture in the way of integrating the formal method Event-B to specify security policies and to check formally their properties, the XML technologies (eXtensible Markup Language) to express security policies, the network configuration and management protocols NETCONF (Network Configuration Protocol) and SNMP (Simple Network Management Protocol) which is used to deploy concrete rules. By this way, the Organization-Based Access Control (Or-BAC) model, which distinguishes generic policy definition from its actual instantiation depending on contextual conditions, is used to represent security policies and their functionality.

Keywords : Security policy, Or-BAC, Event-B, XML, NETCONF, SNMP

Table des matières

Table des figures

Table des tableaux

Introduction

1 Motivation

Dans le contexte de la supervision de la sécurité des réseaux informatiques, plus particulièrement les techniques de détection d'intrusions et de réaction aux attaques, le concept de la corrélation semi-explicite d'alertes a été défini pour détecter des scénarios d'attaques structurés. C'est une technique qui consiste à développer un module de coopération pour analyser et corréler des alertes et générer un diagnostic plus global et synthétique [15]. Lorsque de tels scénarios d'attaques sont détectés, il convient de réagir le plus rapidement et tôt possible. Pour cela, nous envisageons deux procédés: (1) l'utilisation du concept de corrélation semi-explicite défini dans le module de corrélation pour sélectionner automatiquement les réactions adaptées au scénario détecté, (2) l'utilisation du modèle de contrôle d'accès et d'usage Or-BAC (Organization-Based Access Control) pour modéliser une politique de réaction et générer automatiquement des configurations de différents composants de sécurité comme les firewalls, les IDS (*Intrusion Detection System*) et les tunnels IPSec. Selon le deuxième choix, le processus d'obtention de paquets des règles de sécurité (un ensemble des règles de sécurité concrètes) se déroule de manière automatique et centralisée, mais le déploiement de chaque paquet des règles sur chaque composant de sécurité est dans la plupart des cas, pour l'instant, la tâche de l'administrateur.

Pour simplifier le travail de l'administrateur, un protocole de gestion des composants de réseau et de déploiement automatique de règles de sécurité dans le réseau devient avantageux, favorable et adéquat. NETCONF (*Network Configuration Protocol*) est un tel protocole. Il fourni des mécanismes pour la manipulation des configurations sur des composants de réseau. De même, SNMP (*Simple Network Management Protocol*) permet aux administrateurs de réseau de gérer les équipements du réseau, superviser et diagnostiquer des problèmes de réseaux à distance. Ces protocoles sont également intégrés dans un système unifié pour répondre aux menaces et aux attaques qui se produisent dans le réseau.

Un système répondant aux menaces basé sur des politiques de sécurité est capable de recevoir des alertes d'attaque *multi-types* générées par des sondes des IDS (*Intrusion Detection System*), puis de regrouper ces alertes, de les fusionner et de les corréler pour reconnaître l'intention d'attaque et générer son contexte. À partir de ce contexte, le système répondant aux menaces va modéliser des politiques de réaction à l'aide du modèle Or-BAC pour produire des règles de sécurité concrètes et les déployer vers des composants de réseau pour répondre aux attaques en utilisant un protocole de gestion et de configuration du réseau tel que NETCONF et SNMP. L'équipe SERES chez TELECOM Bretagne a développé l'outil CRIM [*http://www.crim-platinum.org/*] permettant de regrouper, fusionner et corréler des alertes d'attaque. Les autres modules du système répondant aux menaces sont encore

développés.

Dans le cadre de ce mémoire, nous présenterons les deux processus restants du système répondant aux menaces: (1) la modélisation des politiques de sécurité d'Or-BAC et (2) le déploiement des règles de sécurité concrètes vers des composants de réseau. Le processus de la modélisation consistera à spécifier des politiques de sécurité d'Or-BAC en méthode formelle Event-B qui est une méthode de spécification, de conception et d'implémentation des systèmes réactifs et à représenter ces politiques en langage XML qui est un standard *de facto* pour la réprésentation de données dans un environnement distribué. À partir de cette représentation, les règles de sécurité concrètes seront générées. Le processus du déploiement recevra ces règles de sécurité concrètes comme *input* et les enverra aux composants de réseau en utilisant un protocole de communication tel que SNMP et spécialement NETCONF qui sera un protocole de configuration de réseau très prometteur [*RFC 4741*].

2 Objectif

L'objectif de mémoire est d'accomplir ces deux processus du système répondant aux menaces. Il comporte les tâches suivantes :

- ✓ Étude du modèle de contrôle d'accès Or-BAC.
- ✓ Étude des technologies XML (*eXtensible Markup Language*), de la méthode formelle B et Event-B pour modéliser des politiques de sécurité d'Or-BAC.
- ✓ Étude comparative des protocoles SNMP et NETCONF et de leur implémentations disponibles pour déployer des politiques de sécurité d'Or-BAC.
- ✓ Mise en oeuvre d'un cas pratique pour appliquer ces études. C'est un réseau d'une organisation qui contient des composant dédiés et non-dédié à la sécurité tels que le firewall, le serveur, le système de détection d'intrusion.

Afin d'atteindre cet objectif, ce mémoire est divisé en quatre chapitre:

- Le premier chapitre est consacré à la présentation des concepts concernant des politiques et des modèles de sécurité . Nous arborderons en particulier des politiques de contrôle d'accès et le modèle Or-BAC (*Organization-Based Access Control*).
- Le deuxième chapitre décrit la modélisation des politiques de sécurité organisationnelles à l'aide du modèle Or-BAC. Pour ce faire, nous décrirons la spécification des politiques de sécurité basée sur le modèle Or-BAC en utilisant le langage formelle Event-B et le langage de balisage XML.
- Le troisième chapitre présente le déploiement des politiques de sécurité. Pour cela, nous présenterons le système répondant aux menaces et attaques dans un réseau informatique. Ce système est l'infrastructure pour réaliser

les deux processus de la modélisation et du déploiement des politiques de sécurité. La suite du chapitre exposera des protocoles de communication pour déployer des politiques de sécurité sur ce système. En premier lieu, nous présenterons le protocole COPS-PR (*Common Open Policy Service for policy provisioning*) qui décrit un modèle extensible pour l'échange d'informations d'administration de réseau entre un serveur et ses clients. En second lieu, nous aborderons le protocole SNMP (*Simple Network Management Protocol*) qui permet de gérer les équipements du réseau, de superviser et diagnostiquer des problèmes de réseaux à distance. Enfin, nous exposerons le protocole NETCONF (NETwork Configuration) qui est un protocole de configuration de réseau très prometteur actuellement.

- Le dernier chapitre est consacré à une étude de cas basée sur le réseau pratique d'une organisation. En premier lieu, une topologie du réseau et un ensemble des politiques appliqué sur ce réseau seront décrites. En second lieu, des politiques de contrôle d'accès seront spécifiées pour générer des règles de sécurité concrètes. Enfin, ces règles de sécurité concrètes seront déployées sur les composants dédiés et non-dédiés à la sécurité de ce réseau.

Chapitre 1: Politiques et Modèles de sécurité

1 Introduction

La sécurité d'un système d'information a pour but la protection des resources incluant des données, des logiciels et des matériels contre la révélation, la modification et la destruction accidentelle ou malfaisante tout en garantissant l'accès pour les utilisateurs légitimes. Généralement, la sécurité se définit par les trois propriétés suivantes: la confidentialité, l'intégrité et la disponibilité. Afin d'assurer la sécurité, il existe plusieurs techniques autour de ces trois propriétés dont la cryptographie, l'authentification et le contrôle d'accès sont principaux. Dans le cadre de notre travail, nous nous intéressons au contrôle d'accès. Il vise à intercepter toutes les tentatives d'accès aux informations critiques. À cet égard, nous envisageons des aspects différents. Tout d'abord, les politiques de sécurité définissent les règles de haut niveau régissant les accès. Ensuite, les modèles de sécurité dressent à une représentation formelle des politiques de sécurité. Nous présentons en détail le modèle de contrôle d'accès Or-BAC. Enfin, les phases principaux pour développer des politiques de sécurité seront présentées.

2 Politiques de sécurité

Une politique de sécurité définit des règles de haut niveau qui régissent les accès et décide lesquels sont autorisés pour la protection d'un système. Autrement dit, il s'agit d'un ensemble des lois des règles et des pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique [16].

Il existe plusieurs types des politiques de sécurité (politiques d'authentification, politiques d'administration et délégation, politiques de contrôle d'accès,…). Dans le cadre de notre travail, on se concentre sur les politiques de contrôle d'accès, celles qui peuvent être groupées en trois classes principales comme suit:

- Les politiques discrétionnaires accordent au propriétaire de l'information, généralement le créateur, tous les droits d'accès, selon sa discrétion.
- Les politiques obligatoires décrètent des règles incontournables qui régissent les droits des sujets et des objets.
- Les politiques basées sur les rôles sont les plus récentes, elles servent à décrire d'une manière plus expressive et plus puissante les fonctionnalités dans les organisations.

Chaque classe des politiques de contrôle d'accès correspond à un modèle de sécurité, ce qui sera présenté dans la partie suivante.

3 Modèles de sécurité

Un modèle de sécurité décrit une représentation formelle des politiques de sécurité et de leur fonctionnement. Il permet de faciliter les preuves sur la sécurité d'un système, c'est la raison pour laquelle les efforts se sont focalisés autour de la construction des modèles formels pour la sécurité [17].

Un modèle de sécurité est souvent rattaché à un mécanisme de sécurité. Un mécanisme de sécurité définit les fonctions des logiciels et matériels permettant d'implémenter les contrôles imposés par la politique de sécurité.

Dans le cadre de notre travail, nous nous intéressons aux modèles de contrôles d'accès. Jusqu'à maintenant, il y a plusieurs modèles de contrôles d'accès qui ont été proposés: DAC, MAC, RBAC,... Nous rappellons ci-dessous ces modèles:

- **Modèles d'autorisation discrétionnaires (*Discretionary Access Control* - DAC)**:

Le modèle DAC est proposé par Lampson [27]. Puis, il a été redéfini par Graham et Dening qui ont précisé des règles pour créer, supprimer des objets transférer et donner des permissions d'accès afin de mettre à jour la matrice d'accès [28]. Ensuite, ce modèle fut formalisé par Harrison, Ruzzo et Ullman pour porter un nouveau nom le modèle HRU où ses auteurs ont abordé le problème de la protection d'un système [29]. DAC a pour objectif d'assurer que tous les accès directs aux objets correspondent à des accès permis par la politique de sécurité. Dans DAC, les sujets ont des permissions de réaliser des actions sur les objets. Les sujets ont l'autorisation de transférer certaines permissions à autres sujets. La gestion des accès aux fichiers du système d'exploitation UNIX constitue un exemple classique de mécanisme de contrôle d'accès basé sur une modèle d'autorisation discrétionnaires.

- **Modèles de sécurité obligatoire (*Mandatory Access Control* - MAC):**

Le modèle MAC a pour but de contrôler la diffusion de l'information même si les applications n'ont pas de confiance d'assurer le cloisonnement des informations. Il permet de modéliser des politiques de sécurité multi-niveaux basées sur une classification des sujets et des objets imposés par une autorité centrale. Les politiques multi-niveaux effectuent un classement des sujets et des objets. Elles introduisent la notion de classe d'accès qui est affectée à chaque sujet et objet. Une relation d'ordre partiel fut définie sur l'ensemble des classes d'accès, il s'agissait de la relation de dominance.

- **Modèle de contrôle d'accès à base de rôles (*Role Based Access Control* - RBAC):**

Le modèle RBAC propose de structurer l'expression de la politique d'autorisation autour du concept de rôle. Des rôles sont affectés aux utilisateurs comformément à la fonction de ces utilisateurs dans l'organisation. Le principe de base du modèle RBAC est de considérer que les autorisations sont directement associées aux rôles. Dans le modèle RBAC, les rôles

recoivent donc des autorisations pour réaliser des actions sur des objets. Le modèle RBAC ne considère que des autorisations positives (permissions). Il introduit le concept *session* . Afin de réaliser une action sur un objet, un utilisateur doit d'abord créer une session dans laquelle il active un rôle qui a reçu l'autorisation pour réaliser cette action sur cet objet. Le modèle RBAC introduit la notion de *contrainte* pour spécifier des politiques d'autorisation incluant des situations plus restrictives. Ainsi, une contrainte de séparation statique spécifie que certains rôles ne peuvent pas être simultanément affectés à un utilisateur. Une contrainte de séparation dynamique spécifie que, même si certains rôles peuvent être affectés à un utilisateur, ces rôles ne peuvent pas être activités simultanément dans une même session. Dans le modèle RBAC, il est également possible d'organiser les rôles de façon *hiérarchique*. Les rôles héritent les autorisations des autres rôles qui lui sont hiérarchiquement inférieurs.

Les modèles de contrôles d'accès comme DAC, MAC et RBAC ne permettent de modéliser que des politiques de sécurité qui se restreignent à des permissions statiques. Avant d'aborder à un modèle de contrôle d'accès récent Or-BAC qui est capable de surmonter des inconvénients de ces trois modèles, nous tentons de citer ces inconvénients:

- Les modèles DAC, MAC et RBAC sont limités à l'expression de permissions. Il n'est pas possible de définir explicitement des interdictions, et d'établir des obligations.
- Ils n'offrent pas la possibilité d'exprimer des règles contextuelles relatives aux permissions, aux interdictions et aux obligations.
- Ils n'offrent pas la capacité d'exprimer des règles spécifiques à l'organisation. Le modèle devra ainsi proposer un moyen de spécifier au sein d'une même organisation plusieurs politiques de sécurité.

Le modèle Or-BAC tente de prendre en compte de ces différents points. On va le voir en détail dans la partie suivante.

4 Modèle OR-BAC (*Organization Based Access Control)*

Cette partie suivante est inspirée du cours FR403b: "Le modèle Or-BAC" du professeur Frédéric Cuppens chez Télécom Bretagne.

Le modèle Or-BAC est un modèle de contrôle d'accès. Il vise à pallier les limites des modèles de sécurité existants tout en simplifiant la spécification d'une politique de sécurité. Il distingue la rédaction de la politique de son implantation en abstrayant des entités traditionnelles du contrôle d'accès (sujet, action, objet) en méta-entités (rôle, activité,vue). Le contrôle d'accès qui est géré au niveau de l'organisation permet de définir ses politiques de sécurité de façon modulaire. On peut ainsi analyser l'interopérabilité d'organisations ayant chacune leur politique de sécurité ou décomposer une politique de sécurité en suivant une hiérarchie d'organisations. Dans le modèle Or-BAC, la possibilité d'exprimer simultanément des permissions, des obligations et des

interdictions, qui dépendent de contextes est un élément qui va vers une plus grande expressivité et la notion d'*organisation* devient centrale de ce modèle.

Une organisation doit gérer un ensemble des règles de sécurité. Le contrôle d'accès consiste à accorder des privilèges à des sujets afin qu'ils puissent réaliser des actions sur des ressources. En pratique, une organisation est censée n'importe quel composant qui gère un ensemble des règles de sécurité. Alors, ses sujets peuvent être des utilisateurs, c'est-à-dire des personnes physiques, mais aussi les processus. Des ressources, aussi appelées objets, sont les fichiers du système, mais peuvent être également des relations dans une base de données, des imprimantes, etc. Les actions possibles dans un système d'information sont "lire", "écrire", "exécuter", etc. Dans le modèle Or-BAC de base, des règles de sécurité sont restreintes aux permissions, mais elles peuvent être étendues pour inclure les interdictions et les obligations. Les permissions dans le modèle Or-BAC ne s'appliquent pas directement aux sujets, actions et objets. Les sujets, actions et objets sont respectivement abstraits par rôles, activités et vues. On va voir en détail les entités et leurs relations (*voir figure 1*). Les entités sont représentées par des rectangles et les relations par des ovales. On remarque également que la position centrale de l'organisation a été citée.

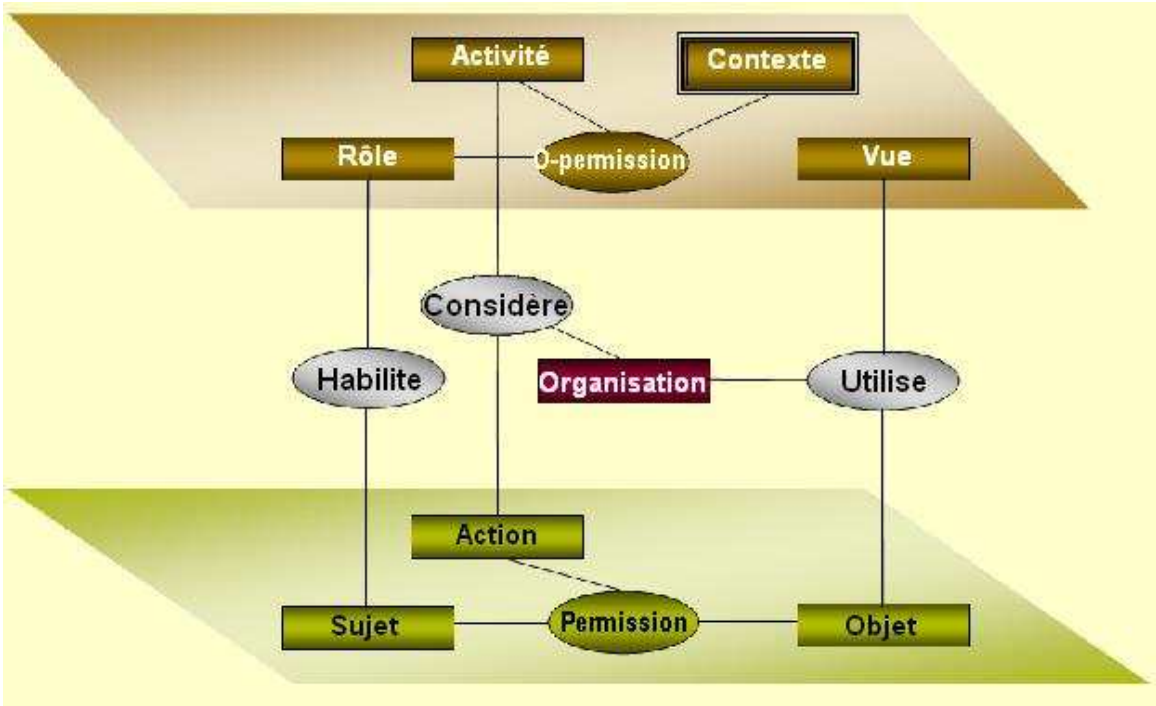


Figure 1: Le modèle Or-BAC [2]

4.1 Organisation

Le concept d'***organisation*** (*Organization* en anglais) est central dans ce modèle. Une organisation peut être vue comme un groupe organisé d'entités actives, c'est-à-dire que les sujets jouent certains rôles. Notons qu'un groupe de sujets n'est pas nécessairement considéré comme une organisation. L'organisation est un des paramètres des règles de sécurité de sorte qu'il est

possible de gérer simultanément plusieurs politiques de sécurité associées à différentes organisations. Or-BAC définit des règles spécifiques à l'organisation. En particulier, l'organisation peut être structurée en plusieurs sous-organisations qui ont chacune leur propre politique de sécurité. Il est également possible de spécifier une politique de sécurité générique au niveau d'une organisation mère. Ses sous-organisations peuvent alors hériter de sa politique de sécurité mais aussi ajouter ou supprimer des règles et ainsi définir leur politique de sécurité. La définition d'une organisation et de la hiérarchie de sous-organisations qui la compose permet ainsi de faciliter l'administration de la politique de sécurité. D'un point de vue pratique, cette hiérachie permet de modéliser la structure des organisations réelles qui peuvent être constituées de départements, d'entité, d'unité,…Un projet ou un groupe de travail peut également être modélisé par une organisation. [4]

4.2 Sujets et rôles

L'entité ***Sujet*** (*Subjects* en anglais) est utilisée différemment selon les modèles de sécurité. Dans le modèle Or-BAC, un sujet peut être soit une entité active, c'est-à-dire un utilisateur, soit une organisation. La notion de ***rôle*** (*Role* en anglais) a déjà été introduite à travers l'évocation du modèle RBAC [3], mais elle diffère quelque peu dans le modèle Or-BAC. Ici, nous considérons qu'un sujet joue un rôle dans une organisation, c'est-à-dire *un sujet obtient des permissions en fonction du ou des rôles qu'il joue dans une certaine organisation* [2]. Ainsi, l'entité *Rôle* est utilisée pour structurer le lien entre les sujets et les organisations. Nous dirons par exemple que l'utilisateur "Nicolas" joue le rôle "administrateur de réseau" dans l'organisation "département informatique". Les permissions obtenues par Nicolas dépendent ainsi de son rôle et de l'organisation dans laquelle il l'exerce.

La relation qui relie le sujet, le rôle et l'organisation est appelée *Empower* (*Habilite* en français, *voir figure 1*). L'exemple précédent peut alors s'écrire de la manière suivante:

Empower(département_informatique,Nicolas, administrateur_réseau)

Un sujet peut également être une organisation. Par exemple, l'organisation "Wanadoo" joue le rôle "fournisseur d'accès" dans l'organisation "France Télécom". Des permissions sont accordées aux rôles. Les sujets obtiennent alors les permissions accordées aux rôles qu'ils jouent, sachant qu'un sujet peut jouer plusieurs rôles. C'est pourquoi, il est possible de hiérarchiser l'ensemble des rôles définis dans une organisation. Un mécanisme d'héritage permet alors d'accorder toutes les permissions d'un rôle à ses sous-rôles. La définition de rôles, l'affectation de rôles aux sujets et l'héritage des permissions à travers la hiérarchie ont pour but de structurer l'ensemble des sujets d'une organisation et de simplifier ainsi la gestion de la politique de sécurité [4].

4.3 Objets et vues

L'entité ***Objet*** (*Object*, en anglais) représente principalement les entités

non actives, c'est-à-dire toutes les ressources de l'organisation, comme les fichiers, les courriers électroniques, les formulaires imprimés, etc. Comme nous venons de le voir, les rôles nous permettent de structurer les sujets et de faciliter la mise à jour de la politique de sécurité. Dans la mesure où il est également nécessaire de structurer les objets et d'ajouter de nouveaux objets au système, une entité comparable au rôle pour les sujets est indispensable pour les objets. Nous l'appelons entité *View* (***Vue***, en français, *voir figure 1*).

De manière intuitive, une vue correspond, comme dans les bases de données relationnelles, à un ensemble d'objets qui satisfont une propriété commune. Prenons l'exemple des fichiers « client » d'un fournisseur d'accès. Chaque organisation peut choisir la manière dont ces fichiers sont implantés. Ces informations peuvent être gérées par des fichiers papier ou stockées dans une base de données. L'organisation peut ainsi avoir à manipuler des objets de nature diverse. Dans ce cas, nous créons une vue "fichier client". Cette vue regroupe l'ensemble des objets correspondants aux fichiers clients quelle que soit leur nature. Une vue est donc une abstraction d'un ensemble d'objets. Dans la mesure où les vues caractérisent la manière dont les objets sont utilisés dans l'organisation, on a besoin d'une relation qui relie ces trois entités: la relation *Use* (***Utilise***, en français, *voir figure 1*). On peut alors écrire qu'une certaine organisation "VST" utilise un certain fichier "fichier253.xls" dans la vue "fichier client" comme suit:

Use (VST, fichier253.xls, fichier_client)

Enfin, comme pour les rôles, il est possible de définir des hiérarchies de vues [4].

4.4 Actions et activités

Les politiques de sécurité spécifient les accès autorisés aux entités passives par des entités actives et régulent les actions effectuées sur le système. Dans le modèle Or-BAC, l'entité *Action* englobe principalement les actions informatiques comme "lire", "écrire", "envoyer", etc. De la même manière que les rôles et les vues sont des abstractions des sujets et des objets, nous définissons une nouvelle entité utilisée comme abstraction des actions: l'entité *Activity* (*Activités* en français, *voir figure 1*). Ainsi, les rôles associent des sujets qui remplissent les mêmes fonctions, les vues regroupent des objets qui satisfont une propriété commune, et les activités correspondent à des actions qui ont un même objectif. L'activité est une abstraction de l'action, c'est-à-dire une activité est un ensemble d'actions ayant des propriétés communes. Nous pouvons, par exemple, définir l'activité "consulter". L'action "acroread", c'est-à-dire utiliser Acrobat Reader, pourra être considérée par une certaine organisation comme l'implémentation de l'activité "consulter".

Dans la mesure où différentes organisations peuvent considérer qu'une même action est employée pour réaliser différentes activités, la relation *Consider* (*Considère*, en français, *voir figure 1*) sera utilisée pour associer les entités *Organization*, *Action* et *Activity*. Nous pourrons alors écrire une relation du type:

Consider (département_informatique, acroread, consulter)

Comme pour les rôles et les vues, le modèle Or-BAC offre la possibilité de définir une hiérarchie d'activités [4].

4.5 Une politique de sécurité avec deux niveaux

Nous venons de voir que les sujets, les actions et les objets sont respectivement abstraits en rôles, en activités et en vues, comme la représentation dans la figure suivante:

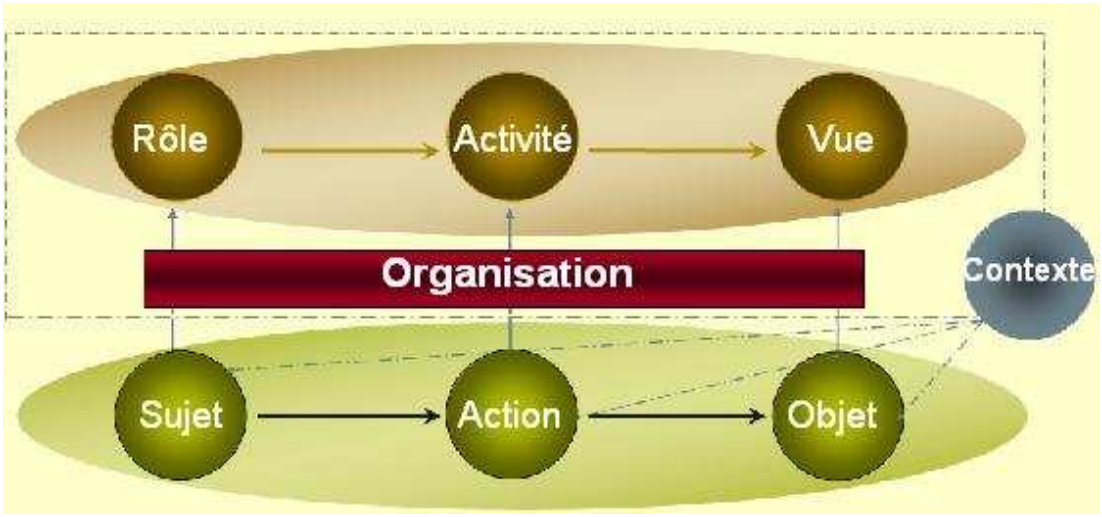


Figure 2 : Une politique de sécurité à deux niveaux [2]

Nous obtenons alors une politique de sécurité à deux niveaux. Le modèle Or-BAC permet ainsi d'établir une politique de sécurité abstraite (rôle, activité, vue) indépendante des choix d'implémentation (sujet, action, objet). Nous devons à présent définir les relations correspondant aux permissions. Nous appelons *Is_permitted* la relation entre un sujet, une action et un objet:

Is_permitted (Subject, Action, Object)

De telles règles de sécurité sont dites concrètes, et sont similaires aux règles de contrôle d'accès utilisées dans le modèle DAC (Discretionary Access Control).

Nous appelons *Permission* la relation abstraite entre un rôle, une activité et une vue. L'organisation dans laquelle une permission est valide est aussi indiquée dans cette relation:

Permission (Organisation, Role, Activity, View)

Cette relation signifie que l'organisation donne la permission à un rôle de réaliser une activité sur une vue.

L'objectif dans le modèle Or-BAC est de rédiger la politique de sécurité à l'aide de permissions abstraites. Les permissions concrètes sont alors dérivées des permissions abstraites. La règle des dérivation est comme suit:

Si *Permission (Organization, Role, Activity, View)* et

Empower (Organization, Subject, Role) et

Consider (Organization, Action, Activity) et

Use (Organization, Object, View)

Alors *Is_permitted (Subject, Action, Object)*

Pour illustrer cette règle, considérons l'exemple suivant: l'utilisateur "Jean" désire ouvrir le fichier "fiche_client_21.pdf" à l'aide d'Acrobat Reader. Le contrôle d'accès associé à cette requête correspond à la permission suivante:

Is_permitted (Jean, acroread, fiche_client_21.pdf)

Nous dirons alors que si nous avons la permission abstraite:

Permission (département_informatique, administrateur, consulter, fiche_client)

et que l'organisation "département_informatique" emploie Jean dans le rôle "administrateur", que cette organisation considère l'action "acroread" comme une activité "consulter", et que cette organisation utilise l'objet "fiche_client_21.pdf" dans la vue "fiche_client", alors Jean obtient l'accès demandé.

Si on considère maintenant la figure 1, il nous reste , pour être complet, à introduire une entité très importante, l'entité *Context*.

4.6 Contexte

Les modèles de contrôle d'accès classiques comme DAC, MAC et RBAC ne permettent pas d'exprimer des règles contextuelles. En effet, il est fréquent d'avoir des règles de sécurité spécifiques à un certain contexte. Un réseau informatique, par exemple, peut fonctionner en mode dégradé suite à un incident technique ou à une action malveillante. En considérant le contexte "mode dégradé", on pourra exprimer des règles de sécurité différentes de celles à appliquer dans un contexte normal. Pour cette raison l'entité *Context* (Contexte en français) a été introduite dans le modèle Or-BAC [5]. Les contextes sont utilisés pour spécifier les circonstances concrètes dans lesquelles les organisations accordent aux sujets des permissions de réaliser des actions sur les objets. Par conséquent, les entités *Organization, Subject, Object, Action* et *Context* sont liées par une nouvelle relation appelée *hold*. Les contextes sont définis par des règles concluant sur cette relation. Par exemple, le contexte *working_hours* (heure de travail) pour un sujet jouant le rôle *administrateur* dans l'organisation *département_informatique* peut être défini de la façon suivante:

Si *Empower (département_informatique, Subject, administrateur)* et

09:00 ≤ global_clock et global_clock ≤ 19:00

Alors *hold (département_informatique, Subject, Action, Object, working_hours)*

C'est-à-dire un sujet jouant le rôle d'administrateur est dans le contexte

working_hours entre 9 heures et 19 heures.

Notons que la relation *Permission* prend alors la forme:

Permission (*Organization, Role, Activity, View,* ***Context***)

Par exemple, on peut spécifier la règle suivante:

Permission (*département_informatique, administrateur, consulter, fiche_client, working_hours*)

Cette règle signifie que les sujets jouant le rôle “administrateur” pourront consulter les objets appartenant à la vue “fiche client” uniquement pendant les heures de travail.

La règle de dérivation précédemment évoquée est aussi modifiée:

Si *Permission* (*Organization*, *Role*, *Activity*, *View*, *Context*) et

Empower (*Organization*, *Subject*, *Role*) et

Consider (*Organization*, *Action*, *Activity*) et

Use (*Organization*, *Object*, *View*) et

hold (*Organization*, *Subject*, *Action*, *Object*, *Context*)

Alors *Is_permitted* (*Subject*, *Action*, *Object*)

Plusieurs types de contexte ont été définis dans le modèle Or-BAC. Dans le cadre de notre travail, nous nous intéressons aux types de contexte suivants :

- Le contexte temporel
- Le contexte spatial
- Le contexte déclaré par l’utilisateur

Le contexte temporel permet de contraindre la date et la durée de validité d’une permission. Le contexte spatial correspond au lieu d’où un utilisateur peut effectuer une activité. Le contexte déclaré par l’utilisateur correspond à un contexte dans lequel l’utilisateur décide de se placer pour effectuer une activité. On peut trouver que le contexte de menace (voir la section 4.2.4 du chapitre 2) appartient à ce dernier type de contexte. D’autres types de contexte (*Contexte prérequis, Contexte provisionnel* ,…) sont présentés sur le site [*www.orbac.org*].

La décision d’accepter une requête nécessite l’évaluation du contexte. Ceci requiert d’avoir à la disposition un certain nombre d’informations pour tester l’activation du contexte. La liste suivante décrit l’ensemble des informations que le système doit pouvoir fournir:

- Une horloge interne, appelée *global_clock*, pour évaluer le contexte temporel.
- L’environnement des utilisateurs et des informations relatives à l’architecture logicielle et matériel, pour évaluer le contexte spatial.
- L’objectif de l’utilisateur pour évaluer le contexte déclaré par

l'utilisateur

La figure suivante récapitule la taxonomie des contextes et les informations nécessaires à leur évaluation:

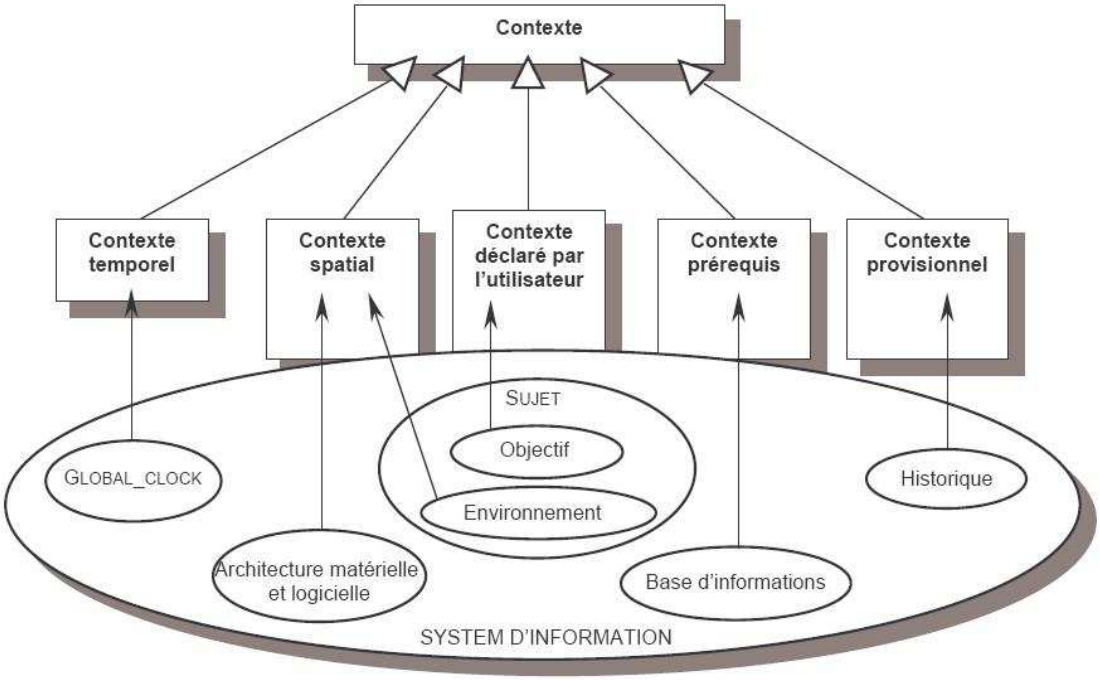


Figure 3: Taxonomie de contextes et données requises [2]

4.7 Gestion des conflits

Quand des politiques de sécurité comportes tous les permissions, interdictions et obligations, des conflits entre eux sont inévitables. Cette section présentera une manière pour résoudre ces conflits.

4.7.1 Les interdictions et les obligations

Dans le modèle Or-BAC, il est possible d'exprimer des permissions, mais aussi des interdictions et des obligations. La forme des interdictions et des obligations est la même que celle des permissions. Nous avons ainsi d'une part les relations *Prohibition* et *Is_prohibited* et d'autre part les relations *Obligation* et *Is_obliged* :

Si *Prohibition* (*Organization*, *Role*, *Activity*, *View*, *Context*) et
Empower (*Organization*, *Subject*, *Role*) et
Consider (*Organization*, *Action*, *Activity*) et
Use (*Organization*, *Object*, *View*) et
hold (*Organization*, *Subject*, *Action*, *Object*, *Context*)
Alors *Is_prohibited* (*Subject*, *Action*, *Object*)

Si *Obligation* (*Organization*, *Role*, *Activity*, *View*, *Context*) et

Empower (*Organization*, *Subject*, *Role*) et

Consider (*Organization*, *Action*, *Activity*) et

Use (*Organization*, *Object*, *View*) et

hold (*Organization*, *Subject*, *Action*, *Object*, *Context*)

Alors *Is_obliged* (*Subject*, *Action*, *Object*)

On trouve que si l'utilisation d'interdictions et d'obligations élargit l'expressivité de la politique de sécurité, il existe une contrepartie. En effet, il peut survenir des conflits entre une permission et une interdiction ou entre une obligation et une interdiction. Pour simplifier notre propos, nous ne considérons ici que les conflits entre les permissions et les interdictions. Pour ce faire, il faut avoir une politique de gestion des conflits

4.7.2 Politique de gestion des conflits

Dans le modèle Or-BAC, des conflits peuvent apparaître au niveau concret et au niveau abstrait. Ainsi, il est possible de détecter et de gérer les conflits au niveau des privilèges abstraits. Nous pourrions avoir par exemple [2]:

Permission(*département_informatique, administrateur, consulter,fiche_client*) et
Prohibition(*département_informatique,administrateur, consulter, fiche_client*)

En effet, si les conflits sont résolus au niveau abstrait, c'est-à-dire si on a la garantie qu'une politique de sécurité abstraite n'est pas conflictuelle, alors aucun conflit ne pourra apparaître au niveau concret. Par conséquent, une même politique de sécurité abstraite pourrait être appliquée à des organisations différentes dans des domaines différents tout en ayant la garantie qu'aucun conflit n'est possible. Afin de régler les conflits dans une politique de sécurité utilisant des permissions et des interdictions, nous introduisons des niveaux de priorités dans les règles de sécurité. Nous appelons *Permission'* et *Prohibition'* les nouvelles relations abstraites obtenues, et *Is_permitted'* et *Is_prohibited'* les nouvelles relations concrètes. Elles sont de la forme:

Permission'(*Organisation*, *Role*, *Activity*, *View*, *Context*, ***Level*)**

Is_permitted'(*Subject*, *Action*, *Object*, ***Level***)

Supposons, par exemple, que les niveaux de priorité sont des entiers naturels. Il suffit de choisir correctement les niveaux de sécurité afin d'éviter les conflits. Par exemple:

Permission (*département_informatique, administrateur, consulter,fiche_client,**2***)

Prohibition(*département_informatique, administrateur, consulter, fiche_client,* **4**)

Dans ce cas, la permission l'emporte sur l'interdiction. La figure 4 suivante présente la démarche générale utilisée dans le modèle Or-BAC pour la gestion des conflits. Nous avons le même schéma pour les interdictions. La flèche en

pointillée représente la règle de dérivation entre les permissions abstraites et les permissions concrètes telle que nous l'avons précédemment explicitée. Les flèches pleines définissent les trois transformations de dérivation des règles concrètes. D'abord, il faut établir une stratégie de gestion de conflits pour généraliser le choix des niveaux de sécurité. Celle-ci doit être indépendante de la politique de sécurité. La stratégie de gestion des conflits permettra de dériver automatiquement les règles *Permission'* des règles *Permission*, ainsi les règles *Prohibition'* des règles *Prohibition*. Pourtant, en suivant la stratégique de gestion des conflits choisie, il peut subsister des conflits entre des permissions et des interdictions abstraites. Ce sera le cas si les deux règles dans l'exemple précédent ont toutes les deux un niveau de priorité égal à trois. Si ce cas se présente, le concepteur de la politique de sécurité doit alors intervenir et soit supprimer ou ajouter des politiques, soit changer les niveaux de priorité des politiques conflictuelles.

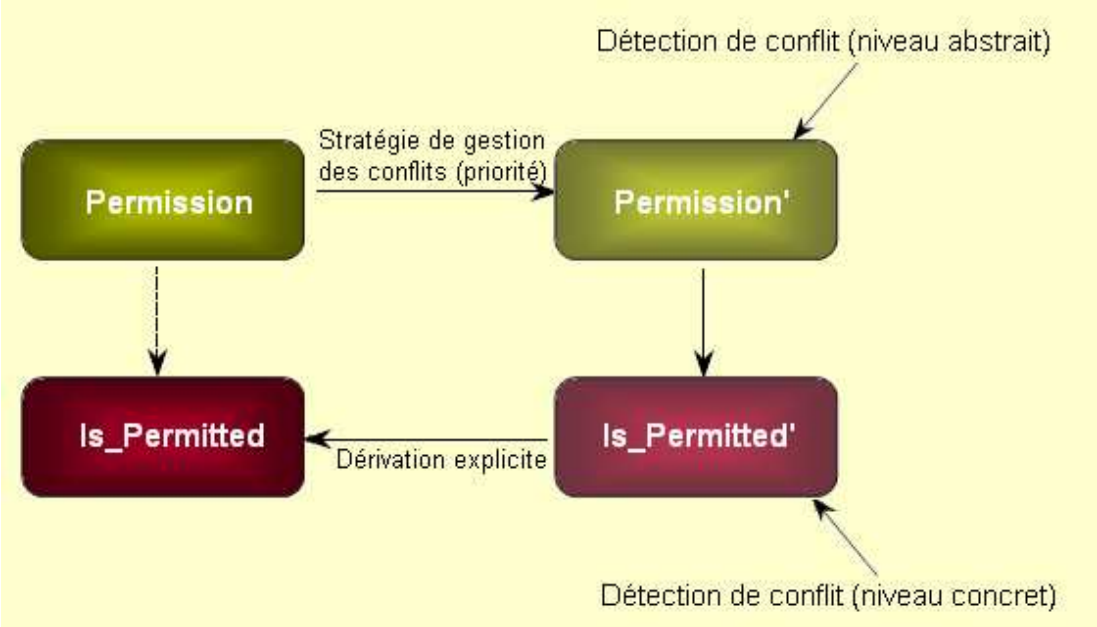


Figure 4: La gestion des conflits dans Or-BAC [2]

Une fois cette première transformation franchie nous obtenons une politique de sécurité abstraite qui ne comporte aucun conflit. Cette propriété est particulièrement intéressante parce qu'à présent, quels que soient les choix d'implémentation (c'est-à-dire quel que soit le choix des sujets, des actions et des objets), la politique de sécurité concrète ne comportera aucun conflit.

Ensuite, il suffit de dériver les privilèges *Is_permitted'* et *Is_prohibited'* à partir des privilèges *Permission'* et *Prohibition'*. La règle de dérivation est la même que celle que nous avons expliquée et le niveau de priorité reste inchangé lors de la dérivation. Enfin, les privilèges *Is_permitted* sont dérivées des privilèges *Is_permitted'* en appliquant la règle suivante:

Si *Is_permitted'*(*Subject, Action, Object, Level*) et

¬*Is_higher_prohibited'*(*Subject, Action, Object, Level*)

Alors *Is_permitted*(*Subject, Action, Object)*

où *Is_higher_prohibited'* est défini de la façon suivante:

Si *Is_prohibited'(Subject, Action, Object, Level')* et *Level < Level'*

Alors *Is_higher_prohibited'(Subject, Action, Object, Level)*

Les privilèges *Is_prohibited* sont dérivées des privilèges *Is_prohibited'* en appliquant des règles similaires.

Le processus que nous venons de présenter donne la possibilité pour concevoir des politiques de sécurité des permissions, des interdictions et des obligations tout en se prémunissant des éventuels conflits. Nous obtenons alors une politique de sécurité d'une grande expressivité qui reste cohérente.

4.8 Administration du modèle Or-BAC

L'administration est un élément important d'une politique de sécurité. Un modèle d'administration doit permettre de déterminer les droits correspondant à la réalisation des tâches administratives (gestion des organisations ; gestion des rôles, des activités, des vues et des contextes ; gestion des utilisateurs, des actions et des objets ;…). Sans procédure administrative, on ne pourrait pas contrôler les évolutions de la politique de sécurité. Alors, l'administration permet d'ajouter et de révoquer des entités ainsi que des privilèges. Le modèle AdOr-BAC, *Administration model for Or-BAC*, permet de gérer une politique de sécurité Or-BAC. Il comporte les trois sous-modèles suivants [2]:

- PRA (*Permission-Role Assignment*) : Création et suppression de permissions.
- URA (*User-Role Assignment*) : Habilitation des sujets dans des rôles.
- UPA (*User-Permission Assignment*) : Affectation de permissions à des utilisateurs.

En conclusion, le modèle Or-BAC présente plusieurs avantages sur les modèles existants (RBAC, DAC. MAC). Les concepts de rôle, d'activité et de vue permettent de généraliser le modèle RBAC et de spécifier une politique de sécurité de façon complètement indépendante de l'implantation. De plus, le modèle Or-BAC a offert la possibilité de gérer des permissions, des interdictions ainsi que des obligations avec une méthode précise et claire pour gérer les éventuels conflits pouvant apparaître dans la politique de sécurtiré. Grâce au concept de *contexte*, il est possible de spécifier les conditions particulières dans lesquelles les privilèges sont accordés. Ainsi, le modèle Or-BAC n'est pas restreint à l'expression de politiques de sécurité *statiques*.

5 Phases pour développer des politiques de sécurité

Le fait de garantir la sécurité d'un système d'information cerne le développement des politiques de sécurité. C'est le processus itératif et continu qui comporte les phases suivantes :

- **Modélisation** : L'objectif de cette phase est de spécifier des politiques et des propriétés de sécurité et de définir des formalismes pour guider le déploiement et le test des mécanismes de sécurité sur les éléments du réseau.
- **Déploiement**: L'objectif de cette phase est de définir des outils pour le déploiement des politiques de sécurité dans un environnement complexe et pour la décomposition de politiques de sécurité en fonction de la topologie et des capacités des composants avant de participer à l'implantation de la politique de sécurité.
- **Test**: L'objectif de cette phase est de valider la conformité des politiques de sécurité par rapport à un système d'information avec l'aide des méthodes formelles et des représentations sémantiques des politiques de sécurité.
- **Surveillance**: L'objectif de cette phase est de détecter des violations à la politique de sécurité. Elle comporte le processus de l'audit de système et des solutions de détection et prévention des intrusions en temps réel. Cette étape valide également l'implémentation des politiques de sécurité.
- **Gestion et amélioration**: L'objectif de cette phase est de mettre à jour la politique de sécurité contre de nouvelles vulnérabilités et risques.

Dans le cadre de notre travail, nous concentrerons d'étudier et présenter deux de ces phases: la modélisation et le déploiement des politiques de sécurité.

6 Conclusion

Ce chapitre a présenté une base de théorie pour le reste du mémoire. Il a été commencé par la définition des politiques et des modèles de sécurité. Ensuite, le modèle de contrôle d'accès Or-BAC qui permet de modéliser des politiques de sécurité de façon complètement indépendante de l'implémentation a été présenté en détail. Ce chapitre s'est terminé par les phases du développement des politiques de sécurité. Ces phases, surtout la modélisation et le déploiement seront abordées dans les chapitres suivants.

Chapitre 2: Modélisation des politiques de sécurité

1 Introduction

Dans ce chapitre, nous aborderons la modélisation des politiques de sécurité organisationnelles à l'aide du modèle Or-BAC. Pour ce faire, des politiques de sécurité doivent être spécifiées et représentées en fonction d'une manière qui facilite la génération des paquets de règles de contrôle d'accès concrètes et augmente la disponibilité pour les distribuer vers des composants de sécurité (par exemple, firewall, IDS/IPS,…). De plus, cette manière de modélisation permettra de vérifier formellement des politiques de sécurité et leurs propriétés.

Nous débuterons ce chapitre par une présentation générale des technologies XML (*Extensible Markup Language*) et de la méthode B. Ensuite, le modèle Or-BAC sera spécifié en utilisant Event-B. Cette spécification permettra de vérifier par la preuve certaines propriétés du modèle Or-BAC exprimées avec des notations mathématiques. Puis, ses éléments (*Organisation, Rôle, Activité, Vue, Contexte*) seront représentés en langage XML pour être prêt à générer des paquets de règles de contrôle d'accès concrètes. Enfin, nous terminerons ce chapitre par une conclusion ouverte.

2 Vue d'ensemble des technologies XML

Le langage de balisage XML est devenu en quelques années une technologie remarquable dans de nombreux domaines. Ce langage vise à permettre une structuration hiérarchique de l'information grâce aux balises. Une balise peut être envisagée comme une étiquette permettant d'identifier des éléments d'information. XML tire sa force de sa grande simplicité car un document XML est un simple fichier textuel. Dès lors qu'il est possible de le *parser*, générer et manipuler à l'aide d'outils appropriés. De plus, cette propriété lui assure une grande portabilité et l'interopérabilité des systèmes qui l'utilisent. En outre, un fichier XML est relativement aisé à comprendre pour un être humain. Néanmoins, le langage XML ne constitue qu'un moyen d'encoder de l'information [18].

Dans cette section, on va voir une brève introduction à XML, à XPath permettant de parcourir ce type de ce document, au langage de requête XQuery permettant d'effectuer des recherches au sein d'un document et à XSLT étant un langage de transformation d'un document XML.

2.1 XML

Le langage XML (*eXtensible Markup Language*) est un langage de balisage extensible. Ce langage est proposé et défini par le *World Wide Web Consortium* ou W3C . Il a pour but de faciliter l'échange d'information entre différents systèmes, principalement sur Internet. XML est basé sur deux idées : (1) représenter le document au même titre que les données par des arbres et (2) représenter le type de chacun d'entre eux comme des grammaires

appliquées à ces mêmes arbres [19].

Comme son nom l'indique, XML est un langage extensible car il permet aux utilisateurs de définir leurs propres éléments. Pourtant, un document XML se doit de respecter la syntaxe du langage XML. Dans un document XML, il est nécessaire que tout élément possède un et un seul parent, exception faite de la racine qui n'a pas de parent. Des éléments XML peuvent avoir des attributs. Un attribut est une paire nom-valeur rattachée à la balise ouvrante d'un élément. Il est aisé de voir qu'une balise ouvrante doit commencer par le caractère < tandis qu'une balise fermante par </. Il n'est pas autorisé d'utiliser des caractères tels que <, >, & dans le contenu d'un élément ou d'un attribut XML. Le document XML peut éventuellement commencer par une déclaration XML avec trois attributs : `version`, `standalone` et `encoding`. En outre, un document XML doit contenir un et un seul élément racine .

```
<?xml version="1.0" encoding="ISO-8859-15" standalone="yes"?>
 <AbstractPolicy name=”exp” type=”permission” context=”default”>
         <relevantRole name="expRole">
         </relevantRole>
         <relevantActivity name="expActivity">
         </relevantActivity>
         <relevantView name="expView">
         </relevantView>
 </AbstractPolicy>
```

Tableau 1 : Exemple d'un fichier XML

2.2 XPath

XPath permet de parcourir aisément un fichier XML en tirant parti de sa structure arborescente. En général, une expression XPath sélectionne des éléments ou des attributs XML. Dans une expression XPath, les attributs XML sont sélectionnés en utilisant le symbole @. Ces expressions peuvent contenir des prédicats permettant de filtrer des éléments ou des attributs sur base d'une condition (*voir l'exemple de l'utilisation de XPath décrivant un message du protocole NETCONF dans la figure 22*).

2.3 XSLT

XSLT (*Extensible Stylesheet Language for Transformations*) est un langage de transformation d'un document XML. Dans cette section, nous ne présenterons qu'une petite partie des possibilités offertes par XSLT. Nous renvoyons le lecteur à la définition du standard pour plus d'informations [30].

Une feuille du type XSLT est un document XML. XSLT se base sur le principe dit de *pattern matching*, c'est-à-dire qu'une feuille XSLT est composé d'un ensemble de règles dont la signification est: “lorque tel élément est rencontré, alors exécuter l'action suivante”. XSLT est inspiré des langages fonctionnels (c'est-à-dire, un programme peut donc être vu comme une série de fonctions). Par conséquent, il est impossible d'exécuter une boucle classique (par exemple, une boucle *for*) en XSLT. Pour ce faire, il faudra

utiliser les principes d'itération et de récursivité. Un document XSLT contient un ensemble d'instructions de type *pattern-template*, la partie pattern est utilisée pour sélectionner un élément dans le document XML source. La partie *template* décrit le format du fichier destination correspondant aux éléments sélectionnés (*voir l'exemple d'une feuille de style XSLT dans le tableau 11*).

3 Vue d'ensemble de méthode B

Pendant ces dernières années, l'application des méthodes formelles pour la sécurité des systèmes d'information s'est développée, grâce notamment aux nouveaux problèmes de sécurité posés par le développement de l'Internet. Les méthodes formelles permettent de vérifier par la preuve certaines propriétés d'un système exprimées avec des notations mathématiques. La méthode B fait partie des méthodes de spécification formelle orientées modèle. Elle est fondée sur un langage de spécification relevant de la théorie des ensembles et la logique du premier ordre, et où l'état du système est modélisé par des types abstraits de données prédéfinis. Elle permet une modélisation des aspects statiques et dynamiques du logiciel. L'aspect statique concerne les données manipulées et est caractérisé par un ensemble de variables ayant une valeur typée (numérique, booléen,…). L'état du système spécifié est décrit par l'ensemble des couples (donnée, valeur). L'aspect dynamique exprimé par un ensemble d'opérations décrivant des changements (ou *transitions*) d'états.

Quand un système d'information est en train de développer, il est nécessaire de considérer la documentation des exigences. À cet égard, la politique de sécurité peut être envisagée comme une partie possible de ce document et des politiques des types *permission*, *prohibition*, *obligation* liées au système d'information peuvent être spécifiées et validées par un langage formel comme B. La question dominante est d'assurer que ce système est conformé à ces politiques de sécurité. Pour répondre à cette question et avant de modéliser des politiques de sécurité en méthode B, on va revoir une brève introduction à cette méthode. Dans les sections suivantes, nous ne présenterons qu'une petite partie des possibilités offertes par méthode B. Nous renvoyons le lecteur au détail du livre *The B-Method an introduction* [21] pour plus d'informations.

3.1 Machine abstraite

Le bloc de base de la méthode B est la notion de machine abstraite. Chaque machine abstraite est écrite en langage de pseudo-programmation AMN (*Abstract Machine Notation*) qui permet d'exprimer à la fois une spécification abstraite et son implantation au niveau du code. Une machine abstraite possède des données (appelées variables) et des opérations manipulant ces données. Une machine abstraite représente l'abstraction de l'état du système. Cet état est représenté par les variables qui sont typées par des expressions mathématiques ensemblistes dans la clause INVARIANT. Celle-ci contient aussi les propriétés, que les variables doivent vérifier, sont exprimées au moyen de prédicats en logique du premier ordre.

La structure d'une machine abstraite est composée de trois parties : la partie entête, la partie statique et la partie dynamique :

- La partie entête permet l'identification de la machine abstraite. Elle contient la clause **MACHINE** décrivant le nom de la machine suivi de paramètres, aisi que la clause **CONSTRAINTS** où il s'agit de caractériser ces paramètres.
- La partie statique regroupe les déclarations d'ensembles (clause **SETS**), de constantes (clause **CONSTANTS**) et de variables (clause **VARIABLES**). Ces déclarations sont complétées par un ensemble de prédicats décrivant les propriétés des constantes (clause **PROPERTIES**) ainsi que les invariants (clause **INVARIANT**) qui explicitent précisément les propriétés qui doivent être toujours satisfaites par l'état de la machine.

```
MACHINE m(p1,p2)
CONSTRAINTS C
SETS se
CONSTANTS k
PROPERTIES T
VARIABLES v
INVARIANT I
INITIALIZATION L
OPERATIONS
y ← op(x) =
          PRE P THEN S
          END ;
…
END
```

Tableau 2 : Structure d'une machine abstraite

- La partie dynamique décrit l'évolution de l'état de la machine. Ceci comprend son initialisation et les opérations offertes par la machine. Les transitions entre états peuvent s'effectuer de manière déterministe et indéterministe. Pour ce faire, le langage défini est celui des substitutions généralisées qui est une notation spécifique à B (par exemple, la substitution simple *x := E* et la substitution multiple simple *x,y := E,F*). Les substitutions font passer une machine abstraite d'un état initial *pré* en un autre état *post*. Elles sont utilisées comme les instructions des langages de programmation.

Il existe deux catégories de types en B : les types de base et les types construits à l'aide d'un constructeur de type [20].

- Les types de base contiennent l'ensemble des entiers, l'ensemble des booléens, l'ensemble des chaînes de caractères et les ensembles abstraits et énumérés définis au niveau de la clause SETS.
- Les types construits à l'aide d'un constructeur de type comprennent l'ensemble des parties finies d'un ensemble (par exemple, *P(E)* définit l'ensemble dont les éléments sont des ensembles d'éléments de *E* et le

produit cartésien entre deux ensembles (par exemple, *E1 x E2* définit l'ensemble des paires ordonnées dont le premier élément est de type *E1* et le seconde est de type *E2*.

Les opérations décrivent un changement atomique dans l'état et l'affectation des variables de sortie. En général, la spécification des opérations utilise les informations : *Nom*, *Paramètre d'entrée*, *Paramètre de sortie*, *Pré-requis* (restrictions sur les paramètres), *Modifications* (données et variables globales qui sont modifiées) et *Effets* (le comportement du processus).

Le corps d'une opération est décrit par la section *PRE P THEN S END* où *P* est pré-condition de l'opération qui décrit les restrictions sur les paramètres et l'état de la machine. Une opération ne peut être invoquées que quand la pré-condition est vraie. La partie *S* décrit son effet. Autrement dit, elle décrit ce que l'opération effectue. Elle doit décrire comment l'état de la machine est modifié et fournir la valeur des sorties. Cela est fait par des substitutions généralisées par lesquelles des variables d'état peuvent être modifiées.

3.2 Obligation de preuve

Un développement en B est basé sur la formalisation des propriétés auxquelles le système doit répondre. C'est pourquoi, la preuve est la vérification que le système en cours de développement possède les propriétés attendues. Les propriétés qui doivent être préservées sont notamment exprimées à travers des invariants. La technique de preuve se base sur la préservation de l'invariant dans la spécification. Ceci a pour but de vérifier la cohérence interne d'un composant et la correction des raffinements par rapport aux spécifications. En effet, la méthode B est capable de découvrir et corriger les erreurs dès les phases amont de conception, grâce à l'élaboration des obligations de preuves réalisées au fur et à mesure du développement.

Une obligation de preuve est une formule mathématique à démontrer pour assurer qu'une machine abstraite est correcte. Plus concrètement, elle est constituée d'un but et d'un ensemble d'hypothèses. Pour démontrer une obligation de preuve, il faut démontrer ce but en supposant que toutes les hypothèses sont vérifiées. Elle représente des conditions de validité permettant de vérifier la correction d'une machine abstraite. Il s'agit de vérifier que l'invariant est respecté aussi bien par l'initialisation que par les opérations de la machine :

- Obligation de preuve de l'initialisation : Soit *I* l'invariant (clause INVARIANT), *T* les propriétés (clause PROPERTIES) et *C* les contraintes (clause CONSTRAINTS) de la spécification, et soit *L* la substitution définissant l'initialisation de la machine ; alors l'initialisation est dite correcte si et seulement si : $T \wedge C \Rightarrow [L]I$.
- Obligation de preuve des opérations : a pour but de vérifier que l'invariant de la machine est vrai après l'application de la substitution de l'operation sachant que l'invariant était vrai avant l'appel de l'opération. Étant donné que la forme générale d'une opération en B

est : $y \leftarrow op(x) = PRE\ P\ THEN\ S\ END$; alors l'obligation de preuve associée revient à vérifier que le prédicat suivant est satisfait : $I \wedge P \rightarrow [S]I$.

3.3 Raffinement

Le raffinement est une technique de développement incrémental permettant de préciser progressivement les données et les opérations de la spécification abstraite de départ. Le but de cette technique est de passer progressivement d'une spécification non-déterministe de haut niveau à une spécification concrète déterministe. La machine raffinée conserve la même interface et le même comportement que celle abstraite mais qui permet de reformuler la machine en une expression de plus en plus concrète et de l'enrichir en y introduisant de nouvelles données et invariants. C'est-à-dire, la machine raffinée constitue des étapes pour aboutir à une implantation [21]. Il y a deux types de raffinements qui sont établis entre la machine abstraite et celle raffinée :

- *Le raffinement de données* où les données ou variables abstraites définies dans la clause VARIABLES peuvent être conservées. Le raffinement de données peut consister en l'introduction de nouvelles variables. Dans ce cas, un invariant de liason est défini en vue de spécifier la relation entre données abstraites et données concrètes (celles du raffinement).
- *Le raffinement des opérations* où chaque raffinée doit réaliser ce qui est spécifié dans l'abstraction, à l'aide des données du raffinement et de substitutions plus concrètes et plus déterministes. Le nom de l'opération et les paramètres formels doivent être identiques. Les définitions des opérations concrètes doivent être données entièrement en fonction des variables d'état concrètes. Toute invocation d'une opération depuis un état valide doit avoir le même effet que l'invocation de l'opération abstraite qui préserve l'invariant de liaison.

Notons qu'il faut évoquer des obligations de preuve de l'initialisation et celles des opérations de la machine raffinée comme pour les obligations de preuve d'une machine abstraite.

3.4 Event-B

La méthode B classique est moins appropriée pour modéliser des systèmes réactifs. Par exemple, il n'y a pas un moyen d'exprimer certaines contraintes dynamiques telles que des contraintes de *liveness* "*if a holds, eventually the property b will hold*" ou plus généralement des contraintes exprimées dans une logique temporelle. Cette lacune est une des raisons de la création d'Event-B dont l'objectif est de permettre la spécification des systèmes réactifs sans négliger la notion de raffinement. Un grand atout d'Event-B est la plate-forme RODIN (http://www.event-b.org/platform.html), qui est basée sur Eclipse. RODIN fournit de nouveaux prouveurs puissants qui peuvent être manipulés à l'aide d'une interface graphique. Les différences principales entre Event-B et

B classique sont les suivantes [22]:

- La notion d'opération est remplacée par celle d'événement. Un événement a une garde (au lieu de *précondition*) et il peut être déclenché si la garde est vraie.
- Les notions de structuration de machines (**USES**, **SEES**, …) sont remplacées par la seule notion de contexte qui regroupe les constantes et les ensembles de bases ainsi que des axiomes associés.
- Certains types de données et leurs opérateurs associés , notamment les séquences (**seq**), les structures (**struct**) et les arbres (**btree**) ont disparu.
- Certaines substitutions ont également disparu. Chaque événement en Event-B a en fait une forme très simple. La forme la plus compliquée peut-être exprimée par un seul **ANY** contenant des assignations (déterministes et non-déterministes) parallèles.
- Une notion adaptée du raffinement permet d'introduire les nouveaux événements (qui doivent raffiner *skip*) de séparer un événement en plusieurs événements dans la machine raffinée, et inversement, de fusionner plusieurs événements.
- Certains aspects permettent une preuve plus dirigée comme des labels sur les invariants et les gardes, ainsi que des témoins (**WITNESS**) pour le raffinement.
- Certains nouveaux opérateurs comme la relation totale ont apparu.

En bref, B classique est appropriée pour le développement de logiciels et Event-B pour le développement de systèmes réactifs. Nous renvoyons le lecteur au détail du document *RODIN Deliverable 3.2* [http://rodin.cs.ncl.ac.uk] pour plus d'informations.

4 Modélisation des politiques de sécurité

Comme déjà énoncé, le modèle Or-BAC a été créé pour modéliser des politiques de sécurité. À cet égard, il faut donner une approche systématique et générale pour modéliser des politiques de sécurité basées sur le modèle Or-BAC. Actuellement, il y a quelques outils qui aident l'administrateur à modéliser des politiques de sécurité et à les traduire dans un langage de configuration d'un équipement concret (par exemple, Firewall Builder [8]). Ces outils suivent l'approche *bottom-up*, c'est-à-dire, ils relient à une topologie de réseau avec des équipements concrets pour produire directement des règles de sécurité dans le langage de la configuration d'un équipement (par exemple, les commandes *iptable* du logiciel *netfilter*, ou bien, la configuration du firewall *Pix* de *Cisco*). Par conséquent, le processus du raisonnement et de la vérification des politiques de sécurité générées n'est pas été considéré. C'est pourquoi, l'approche *bottom-up* manque d'une sémantique précise et claire qui permet à l'administrateur de prévenir des fausses configurations généré et des conflits entre des règles de sécurité générées. De plus, ces règles de sécurité peuvent être inconsistantes et inconformes avec des politiques de sécurité

globales et initiales. Cela entraîne les failles de sécurité dans le réseau. Pour surmonter ces inconvénients de l'approche *bottom-up*, l'approche *top-down* est crée comme dans la figure suivante :

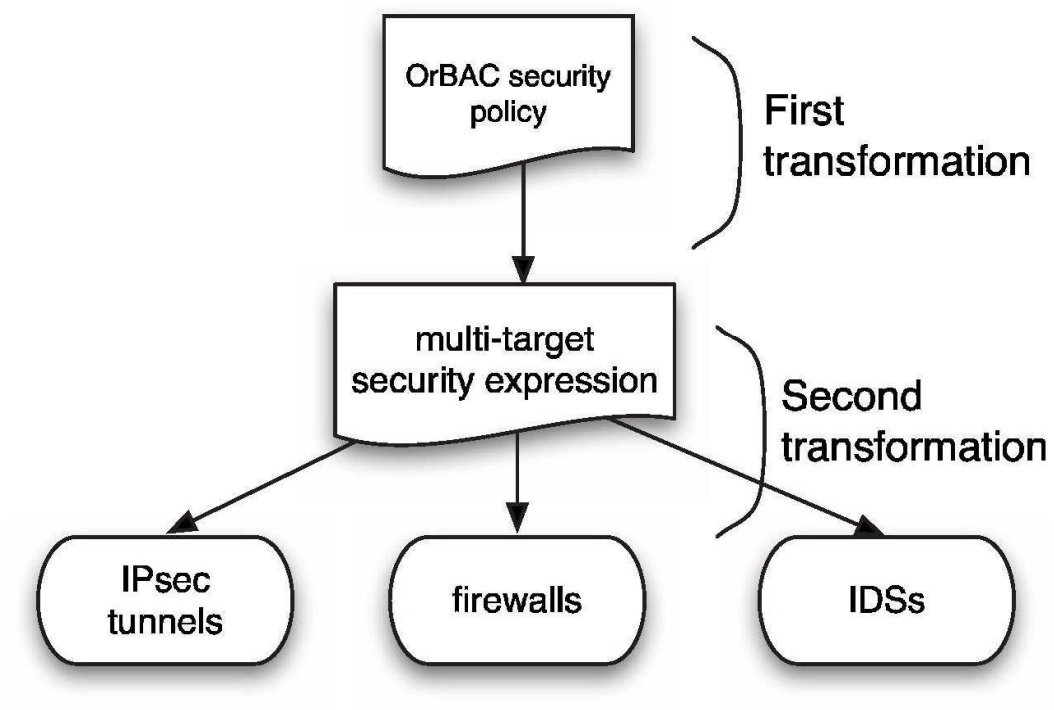


Figure 5: L'approche top-down [13]

Il y a deux transformations principales dans l'approche *top-down* :

- La première transformation: les politiques de sécurité d'Or-BAC sont transformées en expressions de sécurité *multi-target* (c'est-à-dire, des expression intermédiaires conviennent à tous les équipements de sécurité).
- La deuxième transformation: les règles concrètes de configuration d'un composant de sécurité sont générées à partir des expressions de sécurité *multi-target* en ajoutant des informations qui décrivent le type des équipements de sécurité (par exemple, firewall *Pix* de *Cisco* ou bien firewall *netfilter*).

Selon l'approche *top-down*, les politiques de sécurité ont été spécifiées du niveau plus abstrait (la politique de sécurité d'Or-BAC) au niveau plus concret (la règle de sécurité de configuration d'un équipement de réseau). Cela ressemble bien le raffinement en méthode B. Du point de vue, nous tenterons de traduire des politiques de sécurité d'Or-BAC en Event-B pour utiliser les capacités de Event-B (par exemple, obligations de preuves,…) en raisonnant et validant formellement ces politiques de sécurité.

4.1 Spécification des politiques de sécurité en Event-B

Le but principal du processus de la spécification en Envent-B est de vérifier formellement des propriétés des politiques de sécurité, de détecter des

conflits (*voir la section 4.7.2 du chapitre 1*) et de les corriger dès les phases amont de spécification. Le processus de traduction des politiques de sécurité d'Or-BAC en Event-B comporte quelques étapes successives qui sont décrites dans la figure suivante :

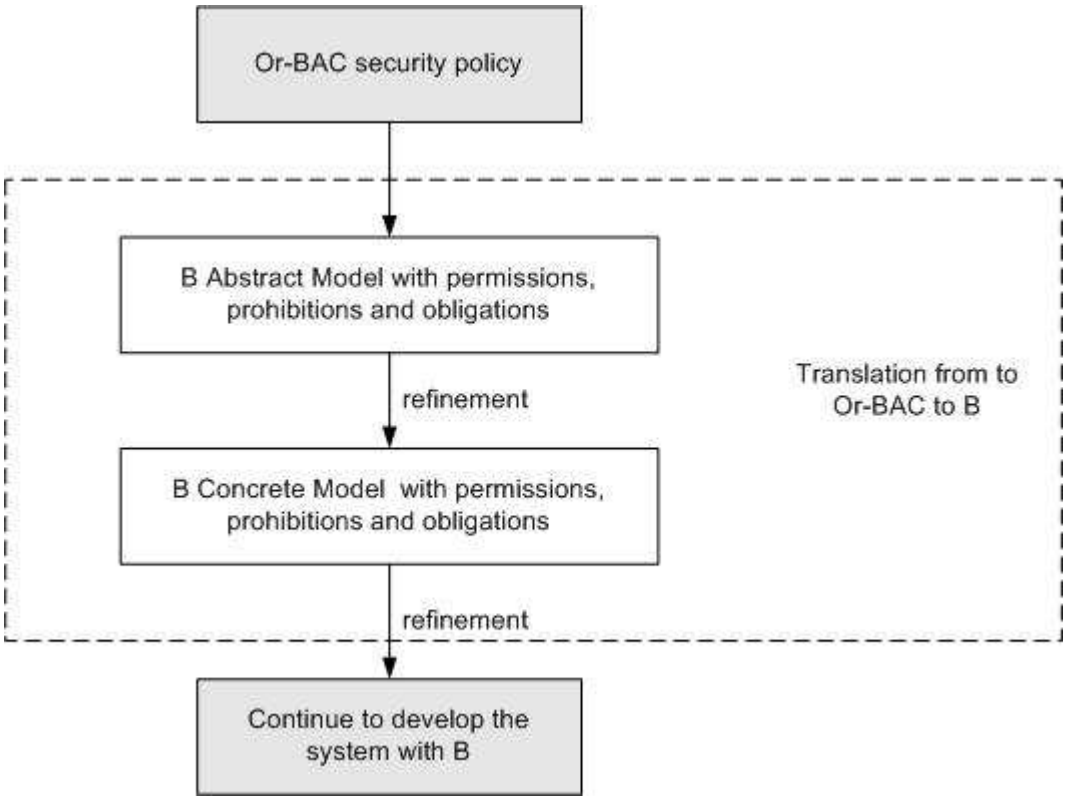


Figure 6 : Étapes de transformation de Or-BAC au modèle en Event-B [23]

4.1.1 Modèle abstrait avec permissions, interdictions et obligations

Le première étape crée une machine B ce qui décrit le modèle abstrait des politiques de sécurité, c'est-à-dire qu'on ne considère que les concepts de l'organisation, du rôle, de la vue , de l'activité et du contexte :

- Les entités *Organisation, Rôle, Vue, Activité, Contexte* sont spécifiées en B par les ensembles *ORGS, ROLES, ACTIVITIES, VIEWS, CONTEXTS* en utilisant la clause **SETS** .[23]
- Les permissions, interdictions et obligations sont spécifiées en B comme constantes *permission, prohibition* et *obligation* en utilisant les clauses **CONSTANTS** et **PROPERTIES** (**AXIOMS** en Event-B) .[23]
- Les concepts hiérarchiques *sub-organisation, sub-role, sub-view, sub-activity* sont également spécifiés en B comme constantes *sub_org, sub_role, sub_view, sub_activity* en utilisant les clauses **CONSTANTS** et **PROPERTIES** (**AXIOMS** en Event-B). [23]

Les variables sont utilisées pour modéliser l'état du système à l'égard de la permission, l'interdiction ou l'obligation. On a les variables suivantes :

- *hist_abst_permission* contient les politiques de sécurité de la *permission* au niveau abstrait appliquées sur le système.

- *hist_abst_prohibition* contient les politiques de sécurité de l' *interdiction* au niveau abstrait appliquées sur le système.

CONTEXT ORBAC_C0 **SETS** ORGS // Organisations ROLES // Roles ACTIVITIES // Activities VIEWS // Views CONTEXTS // Contexts **CONSTANTS** permission prohibition obligation default sub_org sub_role sub_activity sub_view	**AXIOMS** axm1: permission ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS axm2: prohibition ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS axm3: obligation ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS axm4: default ∈ CONTEXTS axm5: sub_org ⊆ ORGS × ORGS axm6: sub_role ⊆ ROLES × ROLES axm7: sub_activity ⊆ ACTIVITIES × ACTIVITIES axm8: sub_view ⊆ VIEWS × VIEWS

Tableau 3 : Spécification au niveau abstrait en Event-B

- *hist_abst_obligation* contient les politiques de sécurité de l'obligation au niveau abstrait appliquées sur le système.
- *context* détermine le context courant du système. [23]

VARIABLES context hist_abst_permission hist_abst_prohibition hist_abst_obligation **INITIALISATION** ≙ **BEGIN** act1: context≔ default act2: hist_abst_permission ≔ ∅ act3: hist_abst_prohibition ≔ ∅ act4 : hist_abst_obligation ≔ ∅ **END**	**INVARIANTS** inv1: context ∈ CONTEXTS inv2: hist_abst_permission ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS inv3: hist_abst_permission ⊆ permission inv4: hist_abst_prohibition ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS inv5: hist_abst_prohibition ⊆ prohibition inv6: hist_abst_obligation ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS inv7: hist_abst_obligation ⊆ obligation

Tableau 4 : Spécification des varibales et invariants

Les événements seront déclenchés quand une demande du contrôle d'accès sera générée par un sujet du système (par exemple, l'événement *permission_access_to_Internet* sera déclenché quand une demande du contrôle d'accès sera générée par un *subnet 111.222.2.0/24*). On a des événements suivants :

- *permission_action*: Cet événement sera déclenché quand une exigence du contrôle d'accès est générée à l'égard de la permission [23].
- *prohibition_action*: Cet événement sera déclenché quand une exigence du contrôle d'accès est générée à l'égard de l'interdiction.

- *obligation_action*: Cet événement sera déclenché quand une exigence du contrôle d'accès sera générée à l'égard de l'obligation.

Ces trois suffixes *action* seront remplacés par les noms concrets (par exemple, *permission_access_to_Internet*) dans un cas concret (par exemple, le réseau d'une organisation). En dehors de ces trois événements, on a deux événements concernant le contexte *set_context_default* et *set_context_value* [23].

```
set_context_default   ≙
BEGIN
act1   :    context ≔ default
END
set_context_value   ≙
ANY
c
WHERE
grd1   :    c ∈ CONTEXTS
grd2   :    c ≠ default
THEN
act1   :    context ≔ c
END
permission_action   ≙
ANY
org ,r ,v ,av
WHERE
grd1   :    org ∈ ORGS
grd2   :    r ∈ ROLES
grd3   :    v ∈ VIEWS
grd4   :    av ∈ ACTIVITIES
grd5   :    (org ↦ r ↦ av ↦ v ↦
context) ∈ permission
THEN
act1   :    hist_abst_permission ≔
hist_abst_permission ∪{(org ↦ r ↦
av ↦ v ↦ context)}
END
```

```
prohibition_action   ≙
ANY
org ,r ,v ,av
WHERE
grd1   :    org ∈ ORGS
grd2   :    r ∈ ROLES
grd3   :    v ∈ VIEWS
grd4   :    av ∈ ACTIVITIES
grd5   :    (org ↦ r ↦ av ↦ v ↦
context) ∈ prohibition
THEN
act1   :    hist_abst_prohibition
≔ hist_abst_prohibition ∪ {(org
↦ r ↦ av ↦ v ↦ context)}
END
obligation_action   ≙
ANY
org ,r ,v ,av
WHERE
grd1   :    org ∈ ORGS
grd2   :    r ∈ ROLES
grd3   :    v ∈ VIEWS
grd4   :    av ∈ ACTIVITIES
grd5   :    (org ↦ r ↦ av ↦ v ↦
context) ∈ obligation
THEN
act1   :    hist_abst_obligation
≔ hist_abst_obligation ∪ {(org
↦ r ↦ av ↦ v ↦ context)}
END
```

Tableau 5 : Spécification des événements.

4.1.2 Premier raffinement: Modèle concrète avec permissions, interdictions et obligations

Le modèle Or-BAC permet de modéliser des politiques de sécurité à deux niveaux : le niveau concret et le niveau abstrait. Dans ce premier raffinement, nous tenterons de spécifier le niveau concret du modèle Or-BAC (c'est-à-dire, le triplet *sujet*, *action*, *objet*) en Event-B. Pour ce faire, ces entités sont spécifiées en Event-B par les ensembles *SUBJECTS*, *ACTIONS*, *OBJECTS* en utilisant la clause **SETS**. Ensuite, les relations *empower* (assignant *sujets* à *rôles*), *use* (assignant *objets* à *vues*), *consider* (assignant *actions* à *activités*) et *hold* (par laquelle les entités *organisation*, *subject*, *action*, *object*, *context* sont liées) dans Or-BAC sont spécifiées comme constantes en Event-B [23].

CONTEXT ORBAC_C1 **EXTENDS** ORBAC_C0 **SETS** SUBJECTS ACTIONS OBJECTS	**CONSTANTS** empower , use ,consider, hold **AXIOMS** axm1 : empower ⊆ ORGS × ROLES × SUBJECTS axm2 : use ⊆ ORGS × VIEWS × OBJECTS axm3 : consider ⊆ ORGS × ACTIVITIES × ACTIONS axm4 : hold ⊆ ORGS × SUBJECTS × ACTIONS × OBJECTS × CONTEXTS **END**

Tableau 6 : Spécification du contexte au niveau du premier raffinement

MACHINE ORBAC_1 **REFINES** ORBAC_0 **SEES** ORBAC_C1 **VARIABLES** hist_conc_permission hist_conc_prohibition hist_conc_obligation context **INVARIANTS** inv1 : hist_conc_permission ⊆ SUBJECTS × ACTIONS × OBJECTS inv2 : hist_conc_prohibition ⊆ SUBJECTS × ACTIONS × OBJECTS inv3: hist_conc_obligation ⊆ SUBJECTS × ACTIONS × OBJECTS inv7 : context ∈ CONTEXTS	inv4: ∀ s· ∀ a· ∀ o·((s ∈ SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s↦a↦o ∈ hist_conc_permission) ⇒ (∃ org· ∃ r· ∃ av· ∃ v·(org ∈ ORGS ∧ r∈ ROLES ∧ av∈ ACTIVITIES ∧ v∈ VIEWS ∧ (org↦r↦s∈ empower ∧ (org↦v↦o)∈ use ∧ (org↦av↦a) ∈ consider ∧ (org↦s↦a↦o↦context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_permission))) inv5 : ∀ s· ∀ a· ∀ o·((s ∈ SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s↦a↦o ∈ hist_conc_prohibition) ⇒ (∃ org· ∃ r· ∃ av· ∃ v·(org ∈ ORGS ∧ r∈ ROLES ∧ av∈ ACTIVITIES ∧ v∈ VIEWS ∧ (org↦r↦s∈ empower ∧ (org↦v↦o)∈ use ∧ (org↦av↦a) ∈ consider ∧ (org↦s↦a↦o↦context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_prohibition))) inv6 : ∀ s· ∀ a· ∀ o·((s ∈ SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s↦a↦o ∈ hist_conc_obligation) ⇒ (∃ org· ∃ r· ∃ av· ∃ v·(org ∈ ORGS ∧ r∈ ROLES ∧ av∈ ACTIVITIES ∧ v∈ VIEWS ∧ (org↦r↦s∈ empower ∧ (org↦v↦o)∈ use ∧ (org↦av↦a) ∈ consider ∧ (org↦s↦a↦o↦context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_obligation)))

Tableau 7 : Spécification de la machine au niveau du premier raffinement

Les trois nouvelles variables *hist_conc_permission*, *hist_conc_prohibition* et *hist_conc_obligation* sont introduites pour modéliser l'état du système à l'égard de la politique de sécurité. Ces trois variables contiennent les journaux des actions qui sont effectuées par les sujets du système sur ces actions. Les invariants de liaisons entre les trois variables concrètes et celles abstraites sont

spécifiés par *inv4,inv5,inv6* dans le tableau 6 . Les nouveaux événements sont raffinés à partir des événéments abstraits avec les mêmes noms, mais les gardes sont plus complexes (*voir l'Annexes*). On a donc spécifié des politiques concrètes du modèle Or-BAC (c'est-à-dire, un triplet *sujet*, *vue*, *action*) en utilisant le raffinement en Event-B. Dans cette spécification, nous n'avons envisagé que des conflits simples entre des politiques de sécurité du type *permission* et *obligation* contre *prohibition*.

4.2 Expression des politiques de sécurité en XML

Comme déjà énoncé, XML est un standard de facto pour l'échange et la représentation de données dans l'environnement distribué. La spécification d'une politique de sécurité du modèle Or-BAC en utilisant la syntaxe du langage XML est indispensable à modéliser la politique de sécurité, avant de la déployer aux composants dédiés ou non-dédiés à la sécurité, surtout dans le cas où le processus du déploiement se fonctionne sur des protocoles de communications utilisant XML pour la configuration de données et la formulation de messages comme NETCONF. Le schéma XML correspondant au modèle Or-BAC de base (hors de contexte) est présenté dans la figure suivante:

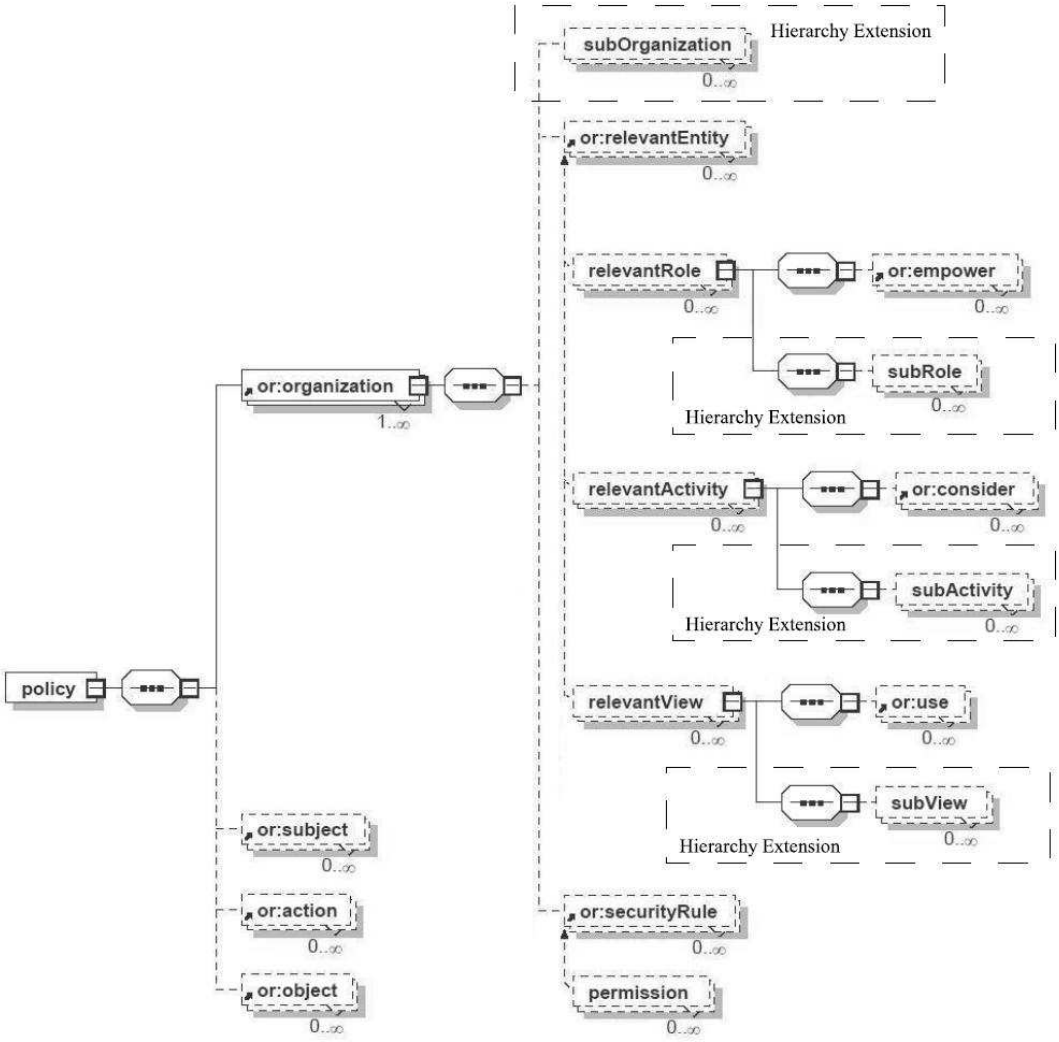


Figure 7: Le modèle Or-BAC de base en XML [6]

Chaque organisation spécifie respectivement les rôles, les activités et les vues qui sont “relatives” dans cette organisation. Pour chaque rôle relatif, l’organisation spécifie les sujets qui sont assignés à ce rôle en utilisant l’élément XML *empower*. Similairement, pour chaque activité relative, les actions sont assignées à cette activité en utilisant l’élément XML *consider* et, chaque vue relative, les objets sont assignés à cette vue en utilisant l’élément XML *use*. Dans le schéma XML, l’hiérarchie d’une organisations, d’un rôle, d’une activité et d’une vues sont respectivement spécifiées en utilisant les élément *subOrganization*, *subRole*, *subActivity*, *subView* [6].

Afin d’illustrer le processus de modélisation, nous prenons la topologie de réseau suivante d’une organisation qui s’appelle ***H***:

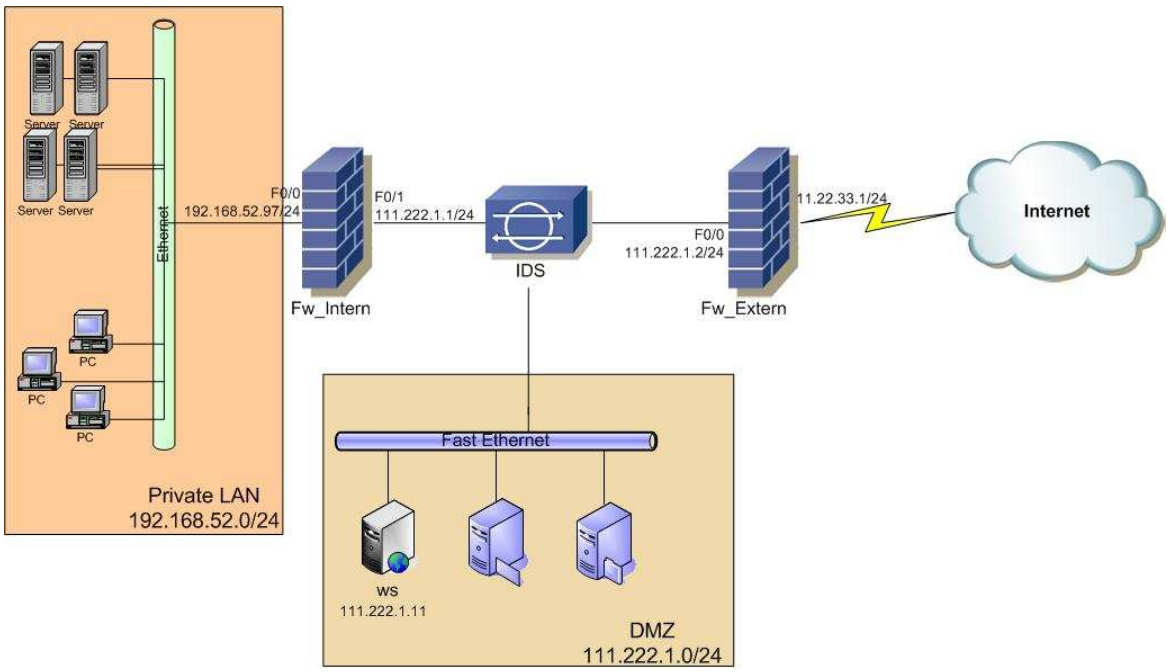


Figure 8: Réseau d’application

Dans le modèle Or-BAC, l’organisation ***H*** qui définit un ensemble des politiques de sécurité (nous nous intéressons aux celles de contrôle d’accès) est un réseau. Ce réseau comporte un sous-réseau local ***H_LAN***. Nous tenterons de traduire en XML une politique de sécurité qui permet aux machines de ***H_LAN*** d’accéder à l’Internet.

4.2.1 Modélisation de rôle

Afin de déterminer les rôles, il faut détecter les sujets car le sujet est abstrait par le *rôle* en utilisant le prédicat *empower* dans le modèle Or-BAC. En général, un sujet est un ordinateur, un équipement de réseau ou bien un ensemble des ordinateurs. Ils sont représentés par l’adresse IP et le masque de réseau. Selon l’article [6], la définition d’un rôle en XML a deux parties:

- **hostIclusion**: correspond à la condition positive qui comporte un ou plusieurs sujets qui sont habilités à ce rôle à travers le prédicat

empower(org,s,r).

- **hostExclusion**: correspond à la condition négative qui comporte un ou plusieurs sujets qui ne sont pas habilités à ce rôle.

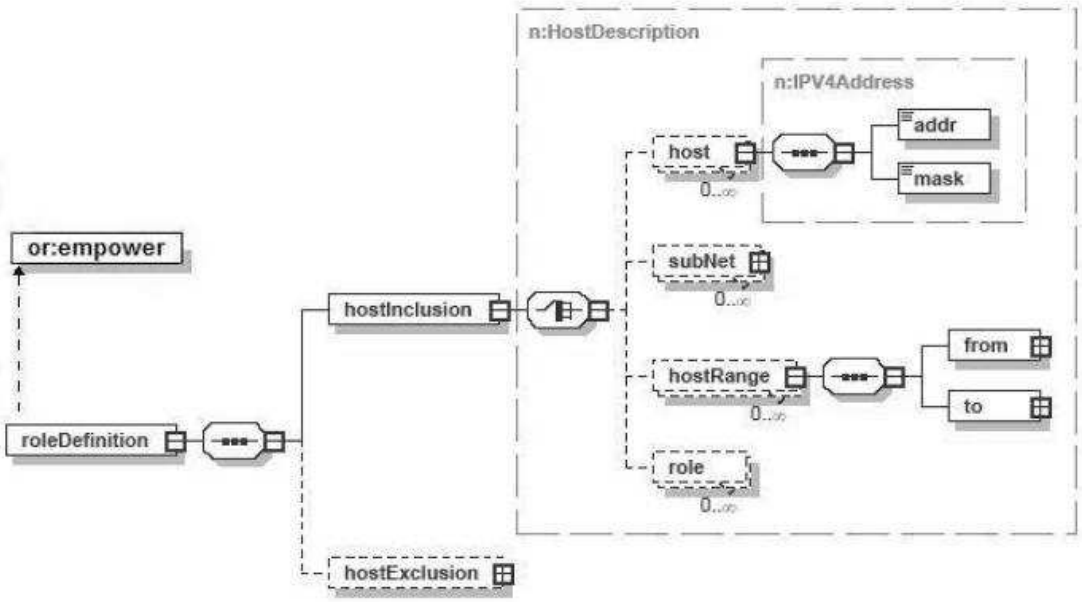


Figure 9: Définition de rôle [6]

Pour l'organisation ***H***, on a un rôle *Private_LAN* qui est une abstraction d'un sujet représentant un ensemble des machines de hôte ce qui sont intérieurs du firewall internal *FW_Intern*. La structure XML du rôle *Private_LAN* est suivante [6]:

```
<relevantRole name="Private_LAN">
   <roleDefinition>
      <hostInclusion>
            <subNet>
            <addr>192.168.52.0</addr>
            <mask>24</mask>
            </subNet>
      </hostInclusion>
      <hostExclusion>
            <host name="FW_Intern"/>
      </hostExclusion>
   </roleDefinition>
</relevantRole>
```

Cette structure dit qu'un sujet qui est habilité au rôle *Private_LAN* est subnet 192.168.52.0/24 (la condition *hostInclusion*) sauf *FW_Intern* (la condition *hostExclusion*).

4.2.2 Modélisation d'activité

Pour déterminer des activités, il faut considérer des services de réseau comme *ping*, *snmp*, *http*, etc. En général, chaque service se compose de trois éléments: *protocol*, *source port* et *destination port*. Dans le modèle Or-BAC, l'activité est une abstraction d'un service de réseau. Afin de représenter une

activité correspondant à un service de réseau, on utilise l'élément XML *activityDefinition*; puis choisit un protocole tel que *tcp, udp* or *icmp* sur lequel ce service se fonctionne. Ensuite, dans le cas des protocoles TCP et UDP, deux éléments supplémentaires peuvent être utilisés pour spécifier en détail l'activité: *srcPort* et *destPort* qui représentent respectivement les conditions du porte de l'expéditeur et du porte de destinataire. Enfin, il est possible d'utiliser l'élément *portRange* pour spécifier un intervalle des portes or *singlePort* pour indiquer un porte unique. Pour le protocole ICMP, la structure est différente, il faut utiliser deux éléments *type* et *code* .

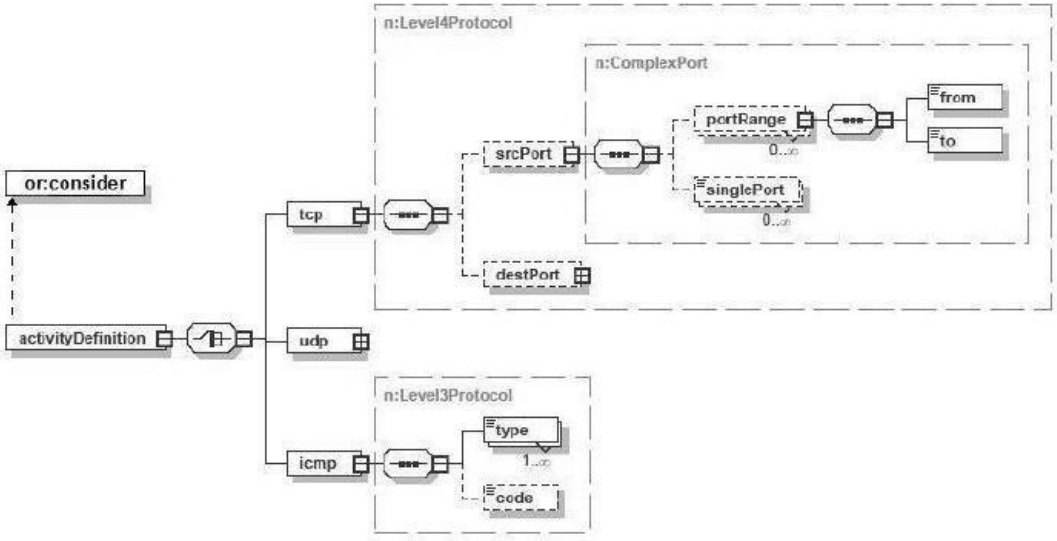


Figure 10: Définition d'activité [6]

Pour l'organisation ***H***, on détermine une activité *Web_HTPP* qui représente une abstraction d'un service *Web* qu'un serveur de la zone Internet fournit. La structure XML suivante représente l'activité *Web_HTTP*[6]:

```
<relevantActivity name="Web_HTTP">
   <activityDefinition>
      <tcp>
         <destPort>
            <singlePort>80</singlePort>
         </destPort>
      </tcp>
   </activityDefinition>
</relevantActivity>
```

Cette représentation dit que le service *Web* correspond à l'activité *Web_HTTP*, si le protocole est TCP et le numéro du port de destinataire est égal à 80. L'élément *srcPort* n'est pas utilisé, ce qui veut dire que tous les ports source sont acceptables.

4.2.3 Modélisation de vue

Dans le modèle Or-BAC, une vue est une abstraction d'un objet en utilisant le prédicat *use(org,o,v)*. La définition de vue peut être spécifiée en XML en utilisant l'élément *toTarget* qui doit être attribué au nom d'un rôle.

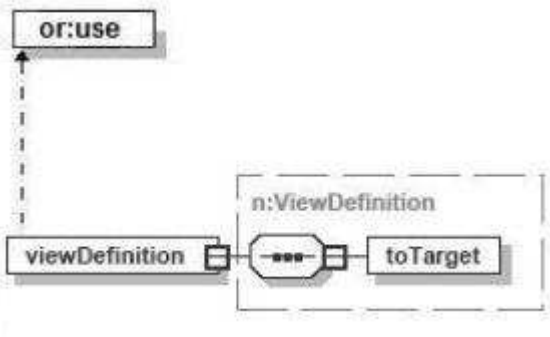


Figure 11: Définition de vue [6]

Pour l'organisation ***H***, on a une vue *To_Internet* pour décrire l'accès à Internet. La structure XML suivante définit cette vue qui correspond au rôle *Internet* (c'est-à-dire, le rôle représente des machines dans l'Internet) [6]:

```
<relevantView name="To_Internet">
   <n:viewDefinition>
      <n:toTarget roleName="Internet"/>
   </n:viewDefinition>
</relevantView>
```

4.2.4 Modélisation de contexte

Dans cette partie, nous tentons de modéliser le contexte des politiques de sécurité dans le modèle Or-BAC. Dans le cadre de notre travail, nous distinguons deux types de poilitque : l'un est appliqué par défaut et l'autre est appliqué pour répondre aux menaces ou attaques (politique de réaction). Le contexte est souvent intégré aux politiques de sécurité du type *réaction*. La politique de réaction est lancée quand une menace est détectée. Plus concrètement, de nouvelles permissions, interdictions ou obligations d'une politique de réaction sont lancées et transmises vers les composants de sécurité appropriés en fonction des violations (ou les essais de violations) et de leur impacts dans le système d'information ciblé. Dans le modèle Or-BAC, la politique de réaction est modélisée en utilisant un contexte spécial qui s'appelle le contexte de menace (*threat context*, en anglais). Le contexte de menace sera activé quand des menaces auront lieu (par exemple, *syn_flooding* est considéré comme un contexte de menace).

Dans le domaine de la détection d'intrusion, le contexte peut être classifié en trois catégories: *opérationnel* (*operational*), *menace* (*threat*) et *minimal* [7] :

- Le contexte opérationnel (un autre nom est contexte *nominal*) a pour but de spécifier la politique opérationnelle. Il est actif, quand aucune attaque, ni intrusion ne se produit.
- Le contexte de menace se réfère des contextes qui sont utilisées pour caractériser des menaces et à fournir des réponses contre menaces. Le contexte de menace sera actif, quand une telle menace est détectée.

Dans la section suivante, nous n'envisagerons que le contexte de menace. L'activation du contexte de menace lance deux problèmes principaux:

activation de contexte et désactivation de contexte

- ***Activation de contexte de menace*** s'adresse à l'activation de la relation *hold(org,subject,action,object,context)* dans le modèle Or-BAC. Cela veut dire que pour un contexte de menace, le sujet, l'action, et l'objet doivent correctement être mappées aux informations disponibles d'une menace (*informations d'alerte*), y compris la source de menace, la classification de menace et la cible de menace. Pour ce faire, les messages IDMEF (Intrusion Detection Message Exchange Format) sont utilisés. IDMEF est une norme définissant les formats de données qui sont utilisés pour communiquer l'information entre des composants de détection d'intrusion [RFC 4765]. *Mapper l'information d'alerte au contexte* demande de créer une transformation du contenu d'alerte au triplet instancié (*Sujet*, *Action*, *Objet*) en écrivant précisément le prédicat *hold*. Le tableau suivant décrit le lien entre les informations d'alerte et les éléments du contexte.

		Sujet	Action	Objet	Contexte	Durée de vie
CreateTime	Ntpstamp					X
Source	Node.name	X				
	Node.Address.address	X				
	Node.Address.netmask	X				
	User.Userid.name	X				
	Process.name	X			x	
	Service.name				x	
	Service.port				x	
Target	Node.name			X		
	Node.Address.address			X		
	Node.Address.netmask			x		
	User.Userid.name			x		
	Process.name			x	x	
	Service.name		X		x	
	Service.port		X		x	
Classification	Reference.name	X	x	x	X	

Assessment	Impact.severity				x	X
	Impact.type				x	X

Tableau 8: Mappage des classes IDMEF aux éléments d'Or-BAC [7]

'X' signifie que l'information considérée est très importante et valable pour être utilisé dans le mappage. 'x' qui signifie que l'information considérée est moins importante et valable que celle indiquée par 'X' ; et elle ne peut pas être utilisée beaucoup dans le mappage.

- ***Désactivation de contexte de menace*** révoque des contre-mesures quand les menaces ne sont pas présentes. Pour ce faire, on surveille la durée de vie statique du contexte calculée sur les attributs *Assessment* (*severity* et *type*) de l'alerte IDMEF pour la dériver. Quand l'alerte se produit, elle est justifiée par une certaine durée. Le contexte correspondant est également activé avec une date d'expiration. Tandis que cette alerte reste à stocker dans le système, le contexte reste actif. Quand la durée de vie expire, l'alerte est supprimée dans le système, le contexte est désactivé.

Dans le réseau de l'organisation *H* (*voir figure 8*), on suppose qu'une attaque syn-flooding vers un serveur de web est rapportée par une alerte IDMEF avec les informations suivantes:

- La référence de classification est égale à CVE-1999-0116 (correspondante à la référence CVE de l'attaque Syn-flooding).
- La cible est attaquée via un service dont le nom est http (ou le port est tcp/80).
- La cible correspond à un noeud de réseau dont le nom est *ws*.

Par conséquent, le contexte *synflooding* est actif pour l'action *http* dans l'objet *ws*. Le mappage des informations de cette alerte IDMEF aux éléments de la relation *hold* dans le modèle Or-BAC est le suivant:

- *alert*(*CreateTime, Source, Target, Classification*),

  reference(*Classification , 'CVE−1999−0116 ') ,*

  service(*Target , http*) ,

  hostname (*Target , ws*)

  → *hold*(*H,_, http, ws, synflooding*)

Notons que le sujet correspondant à l'origine de menace n'est pas présent dans la relation *hold* (représenté par "_"), parce que l'intrus a caché son adresse IP dans l'attaque *Syn-flooding*. La règle de sécurité associée à ce contexte est déclenchée pour réagir à la menace:

- **srx**: *prohibition*(*H, Internet, web_http, webserver, synflooding*)

Cette règle de sécurité dit que dans le contexte de menace *synflooding*, le

rôle *internet* est interdit à effectuer l'activité *web_http* dans la vue *webserver*. Cette règle de réaction ne fermera pas tous les services *tcp* sur tous les serveur de Web, car un seul hôte *ws* (dont le rôle est *webserver*) apparaît dans le contexte *synflooding* via le protocole *http* (dont l'activité est *web_http*). C'est pouquoi, dans ce cas-ci, cette règle de réaction sera limitée à fermer le service *http* sur l'hôte *ws* pour les machines de la zone Internet.

4.2.5 Modélisation de politique

Dans les sections précédentes, les éléments au niveau abstrait (*rôle*,*activité*,*vue*,*contexte*) d'une politique de sécurité Or-Bac ont été représentés en XML à travers un exemple *le réseau de l'organisation H*. Maintenant, nous les réunissons pour modéliser en XML des politiques de sécurité appliquées au réseau de l'organisation *H*. Dans l'organisation H, deux politiques de sécurité d'Or-BAC suivantes ont été envisagées:

- **sr1**: *permission* (*H*, *Private_LAN*, *web_http*, *To_Internet*, *default*)
- **sr2**: *prohibition*(*H*, *Internet*, *web_http*, *webserver*, *synflooding*)

La structure XML de ces deux politiques de sécurité d'OrBAC au niveau abstrait ont un formulaire commun comme suit :

```
<AbstractPolicy name=″…″ type=″…″ context=″…″>
      <relevantRole name="…">
         <roleDefinition>…</roleDefinition>
      </relevantRole>
      <relevantActivity name="…">
         <activityDefinition>…</activityDefinition>
      </relevantActivity>
      <relevantView name="…">
          <viewDefinition>…</viewDefinition>
      </relevantView>
</AbstractPolicy>
```

À partir des politiques de sécurité d'Or-BAC au niveau abstrait, leurs éléments au niveau concrèt (*sujet*,*action*,*vue*) sont crées. La structure XML des politiques de sécurité d'Or-BAC au niveau concrète ont un formulaire commun comme suit :

```
<PolicyInstance name=″…″ type=″…″>
      <Subject name="…">
         <SubjectDefinition>
               <subnet>…</subnet>
               <host>…</host>
         </SubjectDefinition>
      </Subject>
      <Action name="…">
         <actionDefinition>
               <service>
                     <name>…</name>
                     <port>…</port>
               </service>
               <command>
                     <name>…</name>
```

```
                    <path>…</path>
                    <param>…</param>
                </command>
            </actionDefinition>
        </action>
        <Object name="…">
            <objectDefinition>
                <host>…</host>
                <subnet>…</subnet>
            </objectDefinition>
        </Object>
</PolicyInstance>
```

En se basant sur les politiques de sécurité d'Or-BAC au niveau concret, les règles de sécurité concrètes sont générées pour les déployer vers les équipements de types différents. Par exemple, les règles de sécurité générées qui sont envoyées à un firewall *netfilter* sont suivantes:

```
<PEP_Message>
   <PEP_Command>
       <Deploy_Target>
            <name>Fw_intern</name>
            <address>192.168.52.97</address>
       </Deploy_Target>
       <Action type="exec">
            iptables -N Intranet-Web_HTTP-To_Internet
            iptables -A FORWARD -s 192.168.57.0/24 -p tcp
                     –dport 80 -j Intranet-Web_HTTP-
            To_Internet
            iptables -A Intranet-Web_HTTP-To_Internet
                     –s 192.168.57.1/32 -j RETURN
            iptables -A Intranet-Web_HTTP-To_Internet
                     –d 192.168.0.0/16 -j RETURN
            iptables -A Intranet-Web_HTTP-To_Internet -j
            ACCEPT
        </Action>
    </PEP_Command>
</PEP_Message>
```

5 Vérification formelle des politiques de sécurité

La technique de vérification basée sur les preuves avec Event-B permet de détecter les failles de sécurtié dès les phases amont de conception d'applications (c'est-à-dire sans exécution de celle-ci) en prouvant les invariants établis qui sont inchangeables en fonction des transformations des variables. C'est pourquoi, cette technique est appelée la vérification statique. Plus concrètement, les obligations de preuve générées, précisément celles qui se rapportent aux appels des actions sécurisées, permettent de vérifier le respect des invariants et des pré-conditions lors de l'appel. Dans le cas de failles de sécurité signalées par le non-respect des préconditions et des invariants, les propriétés de sécurité prévues ne seront pas assurées.

Dans le modèle Or-BAC, quand des politiques de sécurité contenant en même temps la permission, l'interdiction et l'obligation sont spécifiées en Event-B, le conflit sera inévitable (*voir la section 4.7 du chapitre 1*). Nous

voudrions vérifier ou bien valider ce conflit. Pour ce faire, dans Event-B, nous ajouterons des conditions tolérant ce conflit dans la section **INVARIANTS** d'une machine abstraite qui spécifie un système utilisant le modèle de contrôle d'accès Or-BAC. Par exemple, l'invariant suivant est ajouté pour prévenir le conflit "un même sujet est habilité à deux rôles différents en utilisant le prédicat *empower*" :

$$\forall s.((s \in SUBJECTS \land (org \mapsto r1 \mapsto s) \in empower) \Rightarrow (org \mapsto r2 \mapsto s) \notin empower)$$

En résultat, les obligations de preuve appliquées à cet invariant assure que le conflit entre les politiques de sécurité contenant en même temps la permission, l'interdiction et l'obligation sera validé.

6 Conclusion

Ce chapitre a présenté de manière détaillée comment des politiques de sécurité sont modélisées en utilisant une méthode formelle Event-B et une représentation sémantique du langage XML. Pour éclaircir la représentation en XML, on a illustré cette modélisation par un exemple. On trouve que des règles de sécurité de configuration qui ont été générées sont disponibles pour distribuer aux différentes équipements dédiés et non-dédiés à la sécurité. Bien qu'on ait limité les spécifications de cette règles de sécurité à *Netfilter*, cela ne dit pas que ces règles de sécurité ne sont que déployées aux composants de sécurité compatibles avec *Netfilter*. On va voir, dans les chapitres suivantes, comment ces règles de sécurité pourront être déployées sur de différentes équipements dédiés et non-dédiés à la sécurité.

Chapitre 3: Déploiement des politiques de sécurité

1 Introduction

Dans ce chapitre, nous présenterons le processus du déploiement des politiques de sécurité. D'abord, l'architecture du système répondant aux menaces et attaques dans un réseau sera présentée. Elle joue un rôle important, car toutes les politiques de sécurités, y compris des politiques de réaction sont déployées en se basant sur cette architecture. Elle comporte les modules qui permettent de recevoir des alertes, d'activer des contextes, de spécifier des politiques de réaction et de générer des règles de sécurité concrètes. PDP (*Policy Decision Point*) et PEP (*Policy Enforcement Point*) sont deux des modules qui concernent entièrement le déploiement des politiques de sécurité dans cette architecture. Ensuite, nous envisagerons trois protocoles de communication entre PDP et PEP qui sont COPS-PR (*Common Open Policy Service for policy provisioning*), SNMP (*Simple Network Management Protocol*), NETCONF (*Network Configuration*). Dans le cadre de notre travail, nous nous intéressons spécialement à deux protocoles SNMP et surtout NETCONF et à leurs implantations *HP OpenView Network Node Mananger, cfengine* et *Ensuite*. Enfin, nous tenterons de comparer ces deux protocoles pour viser lequel est meilleur pour déployer des politiques de sécurité.

2 Architecture du système répondant aux menaces et attaques

Le processus de la modélisation et du déploiement des politiques de sécurité se fonctionne sur l'architecture du système répondant aux menaces et attaques dans un réseau informatique. Cette architecture est présentée dans la figure 12. Les modules matériels et logiciels dans cette figure sont représentés par les cercles et les messages et l'information de configuration sont représentées par les diamants. Voici que ses modules principaux et leurs fonctions sont suivants:

2.1 Sondes (Sensors)

Les sondes se rapportent des événements se produisant dans le système d'information. Les données collectées par les sondes viennent des paquets de réseau, des fichiers journals sur une machines hôte et/ou un équipement de réseau, ou dans une application. Les sondes sont normalement des agents installés sur des machines hôtes, des serveurs ou bien des équipements de réseau dans un système de détection d'intrusion.

2.2 Alert Correlation Engine (ACE)

En général, l'information qui est produite par les sondes ne peut pas être directement utilisée dans ACE. En effet, cette information vient de plusieurs sources (de plusieurs sondes) et avec plusieurs formats différents (par exemple, une alerte de Snort, un fichier journal de Netfilter,…). De plus, il est indispensable de réduire le volume d'alerte, d'améliorer sa sémantique et d'enrichir la reconnaissance et le diagnostic des attaquants. La corrélation

d'alertes, un des fonctionnements principaux du module ACE,a pour but de réaliser cette tâche et permet de réduire de nombreux faux positifs générés, à cause de la déviation du comportement normal, appelé comportement sur le long terme qui ne correspond pas toujours à l'occurrence d'une attaque. Elle permet également de produire des méta-alertes offrant une meilleure sémantique et des niveaux de sévérité pour analyser plus efficacement des attaques. Dans le cadre de notre travail, nous envisageons un ACE comme un module qui reçoit tous les événements possibles générés par les sondes comme input et produit des méta-alertes sous forme des messages *IDMEF* comme output. La définition d'ACE détaillée est en dehors de l'étendue de notre travail. En général, ACE a des fonctionnements principaux suivants : Regroupement d'alertes, Fusion d'alertes, Corrélation d'alertes, Reconnaisance d'intentions et Réaction aux attaques. Une implantation du composant ACE est *CRIM* (Coopération de Reconnaissance d'Intentions Malveillantes). *CRIM* qui est développé par l'équipe SERES chez TELECOM Bretagne est un outil très puissant pour mettre en oeuvre la réaction et la corrélation des attaques [*http://www.crim-platinum.org/*].

2.3 Policy Decision Point (PDP)

PDP est un module où des décisions d'exécution des politiques de sécurité sont prises. Plus concrètement, les politiques de sécurité instanciées par PIE (*Policy Instantiation Engine, voir la section 2.5*) sont *input* de PDP. Après avoir reçu ces instances des politiques de sécurité, PDP les transforme à des règles de sécurité concrètes pour les envoyer à PEPs (*Policy Enforcement Point, voir la section 2.4*). PDP est également capable de déterminer quel type de PEP (par exemple, firewall *netfilter* ou firewall *Pix* de Cisco) qui va recevoir des règles de sécurité concrètes pour les dériver. En ce qui concerne l'envoi des règles de sécurité concrète, PDP est également capable de choisir les PEPs les plus proches en se basant sur le critère du "plus court chemin" entre PEP et PDP dans la topologie de réseau.

2.4 Policy Enforcement Point (PEP)

PEP est un module qui va exécuter des règles de sécurité concrètes envoyées par PDP (*Policy Decision Point voir la section 2.3*). PEP est responsable de recevoir des règles de sécurité concrètes transmises par PDP, puis, de les exécuter. Chaque PEP sera reçu un script de configuration considérant tous les deux informations: son type (par exemple, *firewall netfilter*) et des règles de sécurité (par exemple, les commandes *iptables*). Dans l'architecture du système répondant aux menaces et attaques, chaque PEP est aussi une sonde, c'est-à-dire, il va se rapporter des événements se produisant dans le système.

Nous présenterons la communication en détail entre PEP et PEP dans la figure 13 suivante. Nous nous intéressons aux protocoles de communication, car elle concerne directement le déploiement des politiques de sécurité.

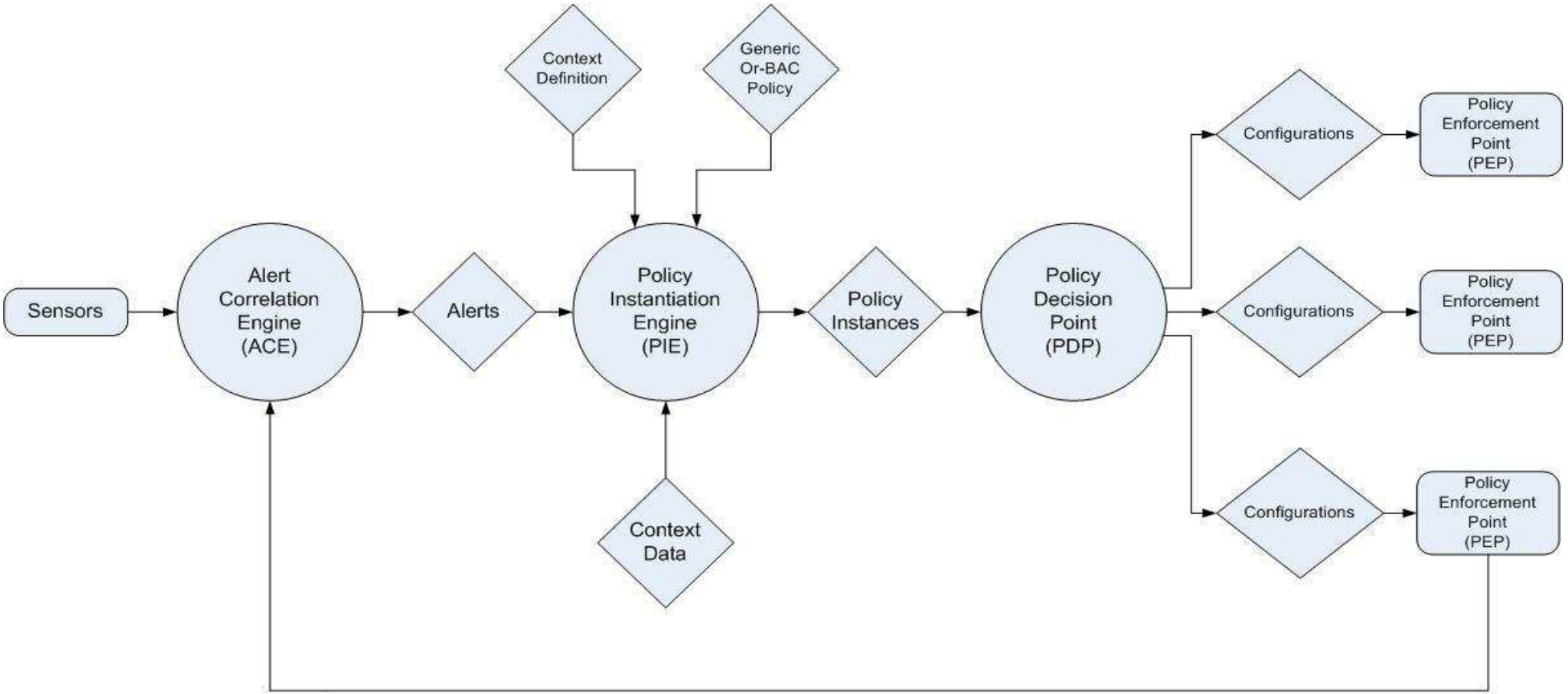


Figure 12: Architecture du système répondant aux menaces [7]

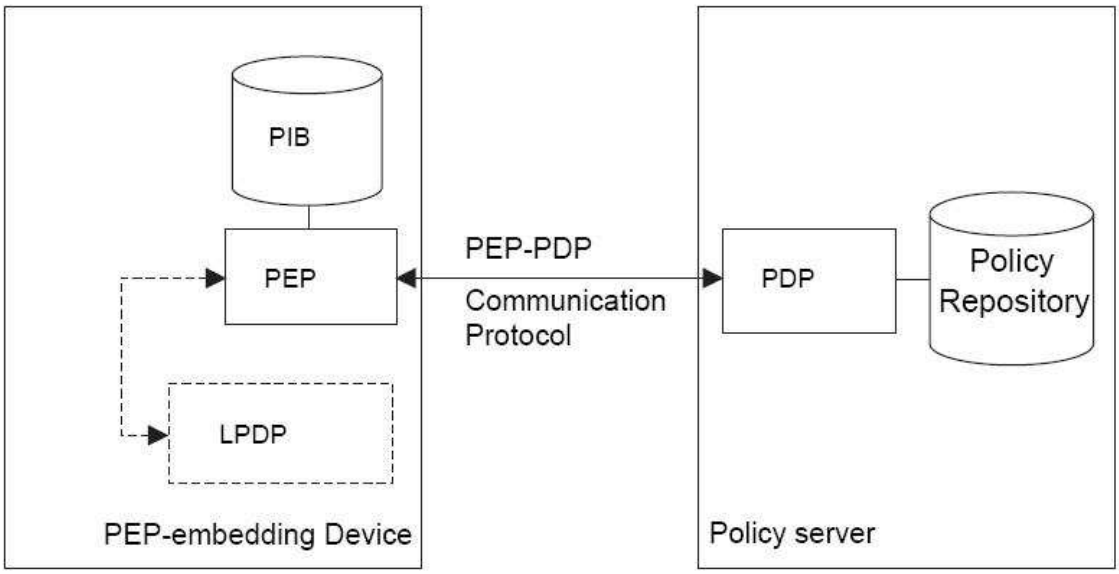


Figure 13: Policy-Based Management Model [9]

Policy Information Base (PIB): est une base de donnée locale de PEP qui stocke l'information des politiques.

PEP-PDP Communication Protocol: transmet l'information de politiques entre PEP et PDP. Quelques protocoles qui ont pour but de la transmettre sont COPS et COPS-PR [*RFC 2748, RFC 3084*], NETCONF[*RFC 4741*] et SNMP[*RFC 1157*].

2.5 Policy Instantiation Engine (PIE)

PIE est un module le plus important de cette architecture. Il a trois fonctions principales:

- Activer des contextes via la relation *hold* dans le modèle Or-BAC.
- Déclencher la re-évaluation de la politique de sécurité via l'activation des règles Or-BAC abstraites.
- Générer des politiques de sécurité au niveau concret.

En effet, quand on considère l'activation des contextes, on va envisager l'activation de la relation *hold* contenant 5 entités : *Organisation*, *Sujet*, *Action*, *Objet* et *Contexte* dans le modèle Or-BAC. PIE va recevoir des messages IDMEF transmis par ACE (*Alert Correlation Engine, voir la section 2.2*) comme input. Pour générer des politiques de sécurité au niveau concret comme output, il doit également considérer des politiques Or-BAC génériques (c'est-à-dire, des politiques de sécurité au niveau abstrait), la définition de contexte qui détermine comment la relation *hold* est activée, et la donnée supplémentaire du contexte concernant certains informations des machines hôtes, du temps, des références vulnérables, etc. Notons que PIE réalise également la déactivation de contexte et la gestion de conflit pour créer un ensemble cohérent des politiques de sécurité d'Or-BAC concrètes. Les

composants principaux de PIE seront présentés dans la figure suivante.

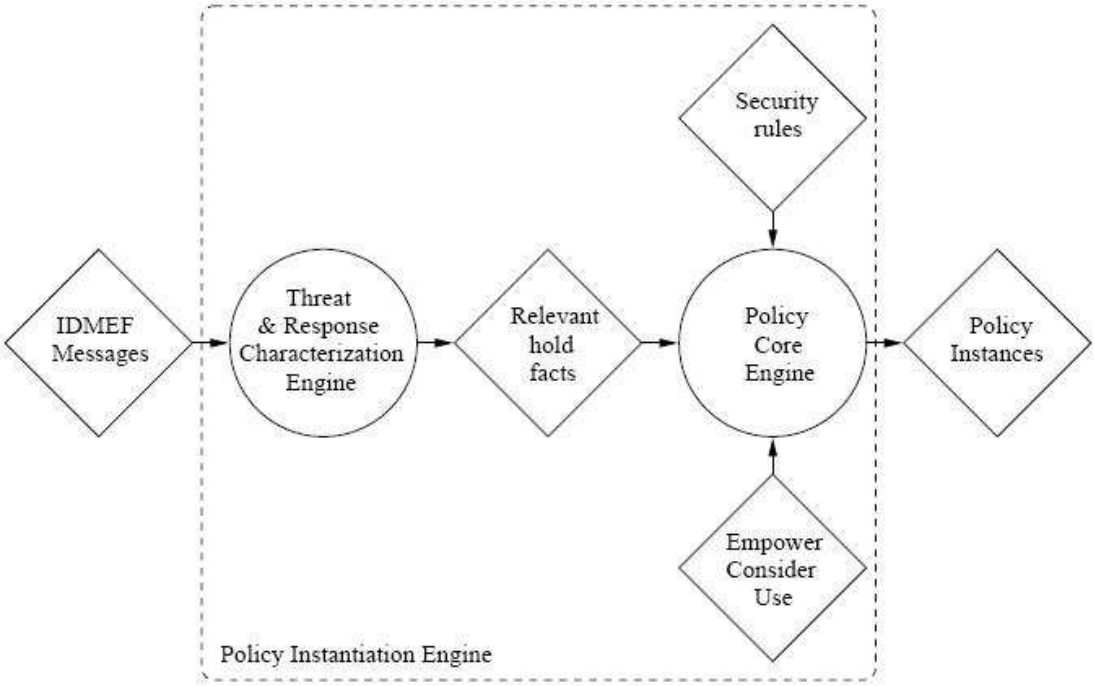


Figure 14: Les composants de PIE [7]

2.5.1 PCE (Policy Core Engine)

PCE est responsable de dériver des politiques de sécurité au niveau concrètes des éléments *sujet*, *action*, *objet* à partir des politiques de sécurité au niveau abstrait des éléments *organisation*, *rôle*, *activité*, *vue* et *context* en utilisant les prédicat *empower*, *use*, *consider* et *hold*. Il est également responsable de traiter des conflits potentiels entre les permissions, les interdictions et les obligations des politiques de sécurités générées.

2.5.2 TCE (Threat Characterization Engine)

TCE est responsable de créer la relation *hold*(*sujet*, *action*, *objet*, *context*) à partir des messages IDMEF. Il est divisé en 3 étapes décrites dans la figure 15:

- Syntactic Mapping (*mappage syntaxique*): réalise un trivial mappage syntaxique qui extrait les triplets (*sujet*, *action*, *object*) à partir d'un message IDMEF à l'aide d'un tableau de mappage (*voir tableau 2*).
- Enrichment (*enrichissement*): consiste à enrichir des sujets, des actions et des objets pour compléter les informations absentes dans les messages IDMEF.
- Strategy Application (*application stratégique*) consiste à déclencher le contexte et dériver la relation *hold(org,s,a,o,c)* pour répondre à la menace relative.

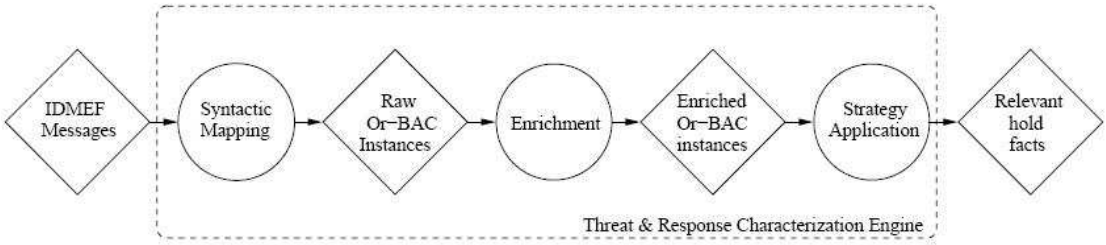


Figure 15: Threat Characterization Engine [7]

3 Sélection des protocoles à déployer des politiques de sécurité

Dans cette section, nous nous intéressons à la communication entre **PDP** (Policy Decision Point) et **PEP** (Policy Enforcement Point), surtout les protocoles de communication, car elle concerne directement le déploiement des politiques de sécurités. Nous envisagerons les protocoles de communication COPS-PR, SNMP et NETCONF.

3.1 COPS-PR

Le protocole COPS-PR (*Common Open Policy Service for policy provisioning*) est l'extension du protocole COPS [RFC2748] pour le fonctionnement en mode provisionning. COPS-PR (*Common Open Policy Service for policy provisioning*) définit un modèle de management simple où des équipements de réseau (PEPs - Policy Enforcement Points) utilisant TCP, sont généralement connectés à un seul serveur de politique (PDP - Policy Decision Point) afin de récupérer l'information de politique mise à jour. Chaque PEP a une base de données de l'information de politique locale (Policy Information Base-PIB) pour stocker l'information des politiques. Cependant, au contraire de la MIB (*Management Information Base*) du protocole SNMP (*Simple Network Management Protocol, voir la section 3.2*) qui considère la politique pour chaque équipement, la PIB se préoccupe du type d'équipement en introduisant la notion de combinaison de rôles. Lorsque le PEP se connecte au PDP, il envoie un ensemble de combinaisons de rôles qui vont caractériser les fonctionnalités des interfaces qu'il gère. Par rapport à ces informations, le PDP retourne la politique adéquate au PEP. Il faut donc définir la gestion de ces combinaisons de rôles pour la PIB. Par conséquent, la mise en oeuvre du protocole COPS-PR est plus complexe que celle du protocole SNMP, quand il nous faut sélectionner un protocole de communication pour déployer des politiques de sécurité.

Le protocole COPS-PR définit également un ensemble de messages pour réaliser la synchronization et la mise à jour de PIB entre PEPs et PDP dans un environement PBNM (*Policy-Based Network Management*) de plusieurs événements différents (c'est-à-dire, chaque PEP peut posséder plusieurs domaines de politique (par exemple, des politiques de sécurité, des politique de QoS). Au contraire des messages sous forme du langage XML du protocole NETCONF (*voir la section 3.3*), les messages du protocole COPS-PR sont encodés en forme binaire en se basant le standard BER (*Basic Encoding*

Rules) [31]. C'est un inconvénient, quand il nous faut sélectionner un protocole de communication pour déployer des politiques de sécurité dans un réseau de plusieurs équipements qui ne peuvent pas supporter un standard particulier comme BER.

De plus, dans le protocole COPS-PR, c'est le PEP qui sera activement responsable de reprendre des connections, quand une corruption entre PDP et PEP se passe. Ce sera un petit contraire car l'architecture du système répondant aux menaces et attaques réglemente que c'est PDP qui dérive des règles de sécurité concrètes, puis qui les déploie vers les PEPs, et ainsi, qui doit garder la position active pour reprendre des connections quand une corrution entre PDP et PEP se produit.

3.2 SNMP

SNMP (*Simple Network Management Protocol*) est l'un des protocoles les plus utilisés pour la gestion des réseaux. Il est basé sur le principe du manager et de l'agent. Un manager a sous sa responsabilité un certain nombre d'agents qu'il interroge via des requête SNMP. Un agent maintient une base de données appelée MIB (*Management Information Base*) qui stocke deux grandes classes de données: la première correspond aux données d'état de la machine (par exemple, nombre de paquets reçus sur une interface,…) ; la seconde correspond à la configuration de la machine (par exemple, politique de sécurité,…). Les données de la MIB sont modélisées de façon arborescente et peuvent être adressées par un OID (*Object IDentifier*) qui correspond au chemin depuis la racine de l'arborescence des données de gestion jusqu'à l'objet, chaque nœud étant marqué par un identifiant (nombre et nom).

La hausse continuelle des attaques par le réseau a fait apparaître le besoin de sécurité du protocole SNMP. C'est pourquoi, IETF a proposé une nouvelle version SNMPv3 qui vise à apporter des outils pour sécuriser les échanges SNMP. Deux des buts principaux étaient de conserver un protocole simple d'utilisation et de rester compatible avec les anciennes versions de SNMPv1 et SNMPv2. VACM (*View-based Access Control Model*) [24] est le système de contrôle d'accès proposé par défaut dans SNMPv3. Ce modèle a la charge de définir quels sont les droits des managers sur les MIBs par type d'opération (lecture, écriture, notification).

3.2.1 Présentation de HP OpenView Network Node Manager

HP OpenView Network Node Manager est une plate-forme basée sur SNMP pour le management de réseau IP en large échelle d'entreprise. Le modèle du management de réseau du HP OpenView Network Node Manager est décrit dans la figure 16 suivante.

Il existe deux versions différentes de HP OpenView Network Node Manager: *Network Node Manager Starter Edition* et *Network Node Manager Advanced Edition*. *Network Node Manager Starter Edition* a pour objectif de filtrer et mettre en corrélation le grand volume d'événements du réseau, avant de présenter l'information à l'opérateur, de manière à ce qu'il sout possible d'identifier la source de la cause du problème en moins de temps avec les

caractéristiques comme suit:

- Facile à déployer et opérer
- Actions automatiques et guidées
- Efficience des ressources et coûts

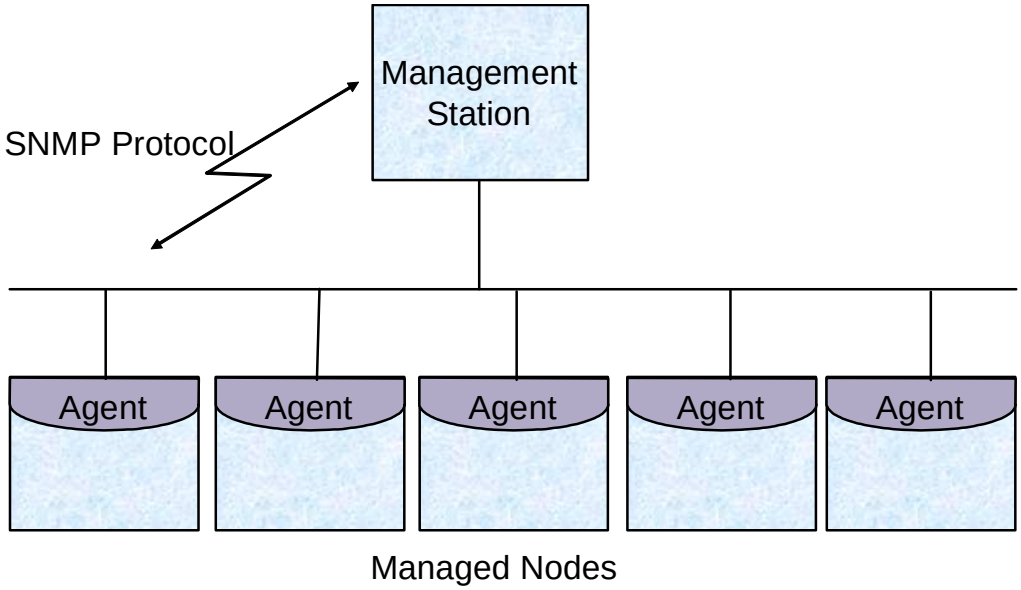


Figure 16 : Modèle du management de réseau

Network Node Manager Advanced Edition supporte plus de caractéristiques (par exemple, la capacité d'analyse de routes pour environnements de réseau complexes , le support des services de réseau tels que Téléphonie IP, Multicast , l'optimisation des ressources de réseau , la réduction du coût total et des temps de réparation,…). HP OpenView Network Node Manager permet de déployer des politiques y compris celles de sécurité d'un gestionnaire aux agents SNMP pour réaliser une tâche de gestion quelconque. Pourtant, il est un produit commercial et fermé. C'est pourquoi, il n'est pas convenable pour étudier dans le cadre d'un travail académique. Pour le remplacer, nous avons utilisé *cfengine*, un moteur de configuration du type *gestionnaire/agent* comme une implémentation du protocole SNMP pour déployer des politiques de sécurité entre PDP(*Policy Decision Point*) et PEP(*Policy Enforcement Point*) .

3.2.2 Présentation de *cfengine*

La gestion de la configuration et des variations de configuration sur un ensemble de systèmes distribués constitue un défi permanent pour les administrateurs système. Pour faciliter cette tâche, il y a plusieurs outils permettant de traiter différents aspects de la gestion de configuration multi-systèmes comme le moteur de configuration (*cfengine*). *Cfengine* est un outil *Open Source* suivant la paradigme gestionnaire/agent. Il permet de définir une stratégie de gestion de configuration grâce à lesquels l'administrateur peut déterminer des actions de gestion qui seront appliquées à des groupes de systèmes en vue d'obtenir l'état désiré. *Cfengine* utilise un modèle *Pull*, c'est-

à-dire, il ne force pas de télécharger une image d'un fichier dans un serveur aux clients, il peut les signaler et leur demande de collecter les fichiers que ces clients veulent recevoir. De plus, *cfengine* peut également simuler un modèle *Push* en sondant (*polling*) chaque client et puis, il exécute un local *script* de configuration qui donne au client la capacité pour télécharger des fichiers mises à jour du serveur distant, mais laisser le client décider de mettre à jour ces fichiers.

Cfengine fonctionne selon le modèle client/serveur. Pour cela, le serveur de règles central (le serveur maître) héberge un fichier de stratégie de configuration qui définit les actions de gestion devant être effectuées sur chaque client géré. La communication entre serveur et client se base sur un *socket* TCP du port 5308. L'administrateur peut utiliser *cfengine* pour effectuer automatiquement les tâches suivantes [25]:

- Vérifier les systèmes, le côté des clients, qui utilisent les bons fichiers de configuration, en copiant les fichiers ou les répertoires de référence.
- Désactiver les fichiers mal configurés sur le client.
- Vérifier les autorisations de fichiers, les propriétaires et contrôler les modifications de somme de contrôle.
- Modifier des fichiers de configuration.
- Exécuter des commandes de shell spécifiées sur chaque client.

La mise en oeuvre de *cfengine* se comporte en deux étapes, comme suit:

- L'administrateur doit tout d'abord définir un gestionnaire qui hébergera les fichiers maîtres de stratégie (par exemple, *cfagent.conf*), qui définit les règles de configuration souhaitées, ainsi que les copies de référence ou les copies originales des fichiers avant d'être distribués aux clients gérés. Le serveur maître *cfengine* peut être un système autonome desservant des groupes de clients distribués.
- Chaque agent (le client géré) récupère les copies originales des fichiers de stratégie à partir du gestionnaire de configuration et évalue l'état actuel par rapport à celui souhaité, tel que défini par le fichier de stratégie. Toute différence entraîne l'exécution des règles de configuration en vue de resynchroniser le client. L'administrateur peut lancer les opérations de synchronisation sur les clients gérés de deux manières, à l'aide d'une opération de diffusion ou d'extraction:
    - À l'aide de la commande *cfrun* exécutée sur le serveur maître de configuration, l'administrateur peut diffuser les modifications.
    - Les opérations d'extraction sont effectuées à l'aide du démon *cfexecd* (similaire à *cron* du système Unix) de *cfengine*. Ensuite, il appelle la commande *cfagent* à intervalles fixes, afin d'effectuer la synchronisation de configuration lancée par le client.

Cfengine utilise différents démons et commandes pour effectuer les opérations de synchronisation de configuration. La liste suivante décrit les composants principaux de *cfengine* [25]:

- ***cfagent***: est exécutée sur chaque client géré et s'amorce elle-même à l'aide du fichier *update.conf*, qui décrit l'ensemble de fichiers à transférer du serveur maître vers le client local géré. Les fichiers transférés comprennent le fichier principal de stratégie, *cfagent.conf*, et tous les fichiers liés à cette stratégie. Après le transfert des fichiers de configuration, *cfagent* évalue les instructions de configuration contenues dans ceux-ci. Si la configuration actuelle du système client est différente de la configuration souhaitée, *cfagent* exécute les actions définies afin de restaurer l'état correct du client.

- ***cfservd:*** est exécuté sur le serveur maître de configuration et constitue le centre d'échange des demandes de transfert de fichiers émanant des clients gérés. Ensuite, le *cfagent* sur les clients gérés contacte le démon *cfservd* du serveur maître et demande des copies des fichiers maîtres de stratégie, ainsi que des copies de tous les fichiers de référence requis dans le cadre des opérations de synchronisation de configuration définies. Le *démon cfservd* est chargé de l'authentification des clients distants, à l'aide d'un mécanisme d'échange de clés privées et publiques avec cryptage optionnel des fichiers transférés vers les clients gérés.

- ***cfexecd***: est un outil de planification et de rapport. Si l'administrateur utilise *cron* pour effectuer des sessions de synchronisation à intervalles fixes, *cfexecd* est la commande placée dans le fichier *crontab* en vue de conditionner l'appel de *cfagent*. Elle stocke le résultat de l'exécution de *cfagent* dans le répertoire de sortie et envoie éventuellement un e-mail. *cfexecd* possède ses propres fonctionnalités de type *cron* basées sur les classes temporelles de *cfengine*. L'administrateur peut exécuter *cfexecd* en mode démon et l'utiliser pour appeler *cfagent* à intervalles définis à la place de *cron*. Le réglage par défaut est un appel de cfagent toutes les heures. On recommande d'ajouter une entrée pour *cfexecd* dans le fichier crontab pour la configuration initiale.

- ***cfrun***: contacte les clients gérés et demande à chacun d'effectuer une synchronisation immédiate. Plus particulièrement, elle se connecte au démon *cfservd* de chaque client géré qui, à son tour, lance *cfagent*.

La figure 17 suivante illustre la relation entre les commandes et les démons de *cfengine* et donne un exemple d'utilisation de *cfrun* par l'administrateur (le modèle *Push*). Les lignes en pointillés sur le schéma indiquent les séquences d'appel (par exemple, A appelle B). Les lignes pleines indiquent la lecture des données des fichiers de configuration [25].

Étape 1: L'administrateur est connecté au serveur maître de synchronisation de configuration et effectue une modification qui doit être diffusée aux clients gérés, à l'aide de la commande *cfrun*. À son tour, *cfrun*

vérifie la liste des clients gérés dans le fichier *cfrun.hosts*. Notez que le serveur maître peut être son propre client. Sur cette figure, il existe deux clients : le serveur maître et un client distant.

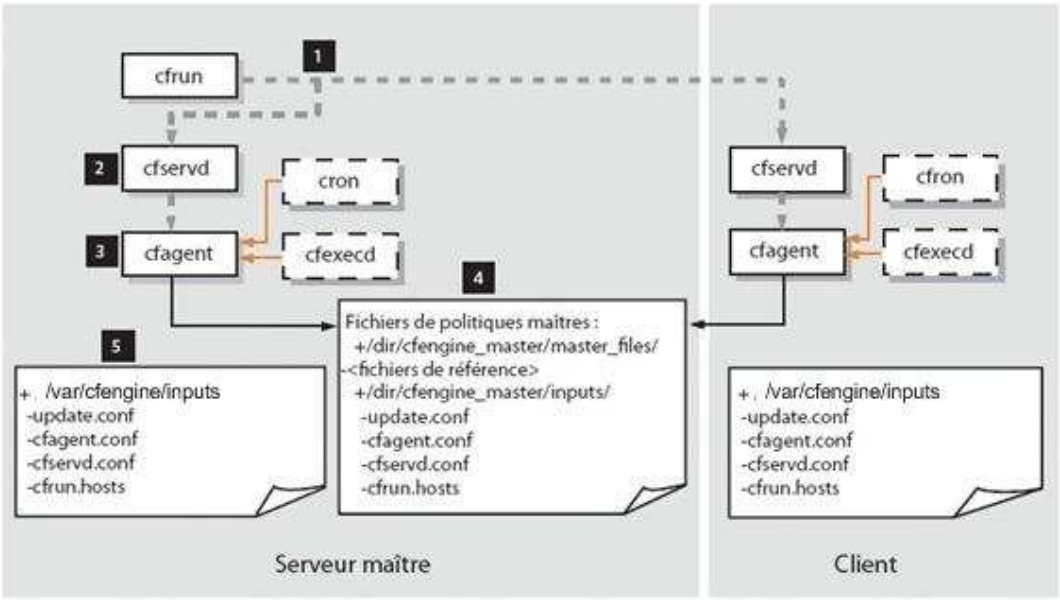


Figure 17: Exemple du modèle Push en cfengine [25]

Étape 2: *cfrun* contacte *cfservd* sur chaque client géré, qui à son tour appelle *cfagent*.

Étape 3: *cfagent* vérifie tout d'abord s'il existe une copie mise à jour du fichier *update.conf* sur le serveur maître et transfère éventuellement celle-ci sur le client.

Étape 4: Si un système autonome est utilisé comme serveur maître, la copie originale du fichier *update.conf* est stockée dans */var/cfengine/inputs/* par défaut. Les copies originales des autres fichiers de configuration, par exemple *cfagent.conf*, *cfservd.conf*, et *cfrun.hosts* figurent également dans ce répertoire.

Étape 5: Lors de la copie des fichiers de configuration sur le système local, *cfagent* stocke ceux-ci dans */var/cfengine/inputs*. *cfagent* analyse tout d'abord le contenu de *update.conf* afin de mettre à jour les données binaires modifiées de *cfengine* (le cas échéant) et d'obtenir la version la plus récente des fichiers de stratégie (*cfagent.conf* et fichiers associés). *cfagent* analyse ensuite *cfagent.conf* afin de savoir si le client se trouve dans l'état souhaité. En cas de différences, *cfagent* effectue les actions définies afin de corriger la configuration du client.

En se basant sur ces étapes, nous développerons un module qui permet de déployer des règles de sécurité concrètes vers des composant dédiés et non-dédiés à la sécurité dans un réseau (voir la section 5 du chapitre 4).

3.3 NETCONF

NETCONF [RFC 4741] est un protocole de configuration de réseau très prometteur qui reprend aussi le paradigme *gestionnaire/agent*. En se bassant sur le langage de balise XML pour la partie protocolaire mais aussi pour formuler les données, il profite des années d'expérience acquise lors de la croissance de l'Internet. En comparaison des protocoles existants, il apporte principalement une gestion plus transactionnelle des configurations, à l'aide du concept de configuration *candidate* et des opérations qui lui sont propres : *validate* et *commit*. Le fait qu'il repose sur le protocole sécurisé SSH est un point important. L'utilisation du langage XML rend également son implantation plus aisée grâce aux nombreuses librairies existantes : *parsing* de document, transformation par XPath, langage de transformation XSLT.

3.3.1 Présentation de NETCONF

NETCONF se décompose en quatre couches illustrées par la figure suivante :

- **Couche de transport** fournit les fonctions nécessaires pour établir la communication entre un gestionnaire et des agents. Cette couche est le protocole de niveau inférieur, dans la structure en couche traditionnelle, sur lequel NETCONF repose. Plusieurs protocoles sont possibles SSH (*Secure Shell*), SOAP (*Simple Object Access Protocol*) et BEEP (Blocks Extensible Exchange Protocol), mais c'est SSH qui s'impose comme le protocole obligatoire. SSH apporte un niveau très élevé de sécurité et fournit les services d'authentification, confidentialité et non-répudiation.
- **Couche de RPC (Remote Procedure Call)** est responsable de différencier les message NETCONF entre eux au sein d'une session. Elle fournit un mécanisme pour encoder et décoder ces messages sous forme du langage XML. De plus, elle définit un identifiant, à la manière d'un numéro de séquence, qui permet à un gestionnaire de savoir à quelle requête correspond à une réponse.
- **Couche d'Opérations** définit un ensemble d'opérations de base pour récupérer des données (*get*, *get-config*), modifier des données (*copy-config*, *delete-config*, *edit-config*), poser et relâcher des verrous (*lock*, *unlock*) pour acquérir un accès privilégié aux données, forcer la fermeture d'une session (*close-session*, *kill-session*). Ces opérations sont aussi appellées les méthodes RPC. L'encodage de ces opérations suit un schéma XML. En dehors de ces opérations, le protocole NETCONF permet de définir de nouvelles méthodes et leur capacités.
- **Couche de contenu** est utilisée pour désigner les données de configuration manipulées par l'utilisation des méthodes RPC. Par exemple, il peut s'agir de données relatives à une politique de firewall, une configuration de routage,…

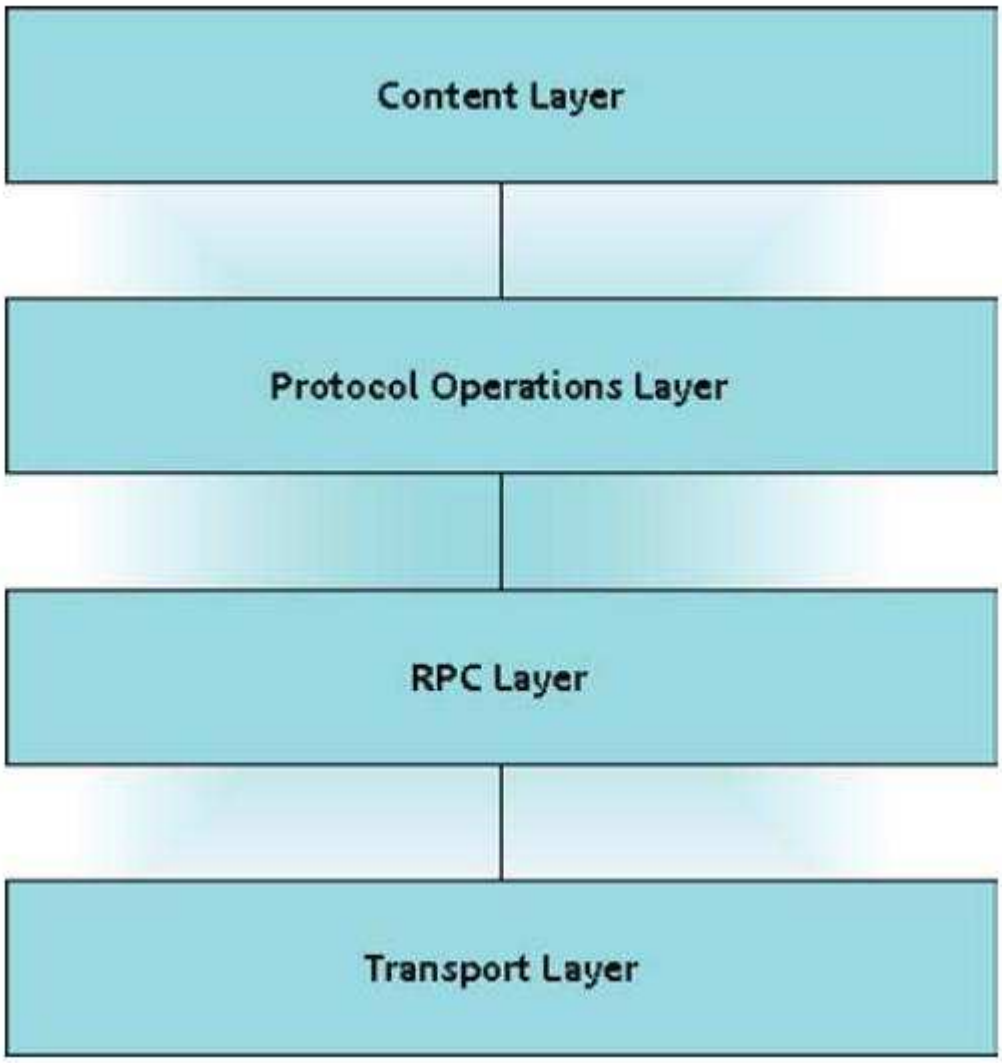


Figure 18: Les 4 couches de NETCONF [9]

3.3.2 Phases de communication

L'établissement d'une session NETCONF entre la gestionnaire et les agents est composé de 4 phases principales comme suit [9]:

- **Première phase:** Durant cette phase, les canaux de transport sont établis entre un gestionnaire et un agent. L'authentication et le processus de l'échange de clés entre le gestionnaire et l'agent sont réalisées dans cette phase.
- **Deuxième phase:** Juste après avoir établi successivement une session de transport, une paire gestionnaire/agent échangera simultanément les capacités des opérations de base en envoyants mutuellement les messages *Hello*.
- **Troisième phase:** Quand les capacités sont échangées, une paire gestionnaire/agent peut envoyer les méthodes RPC et émettre les opérations.

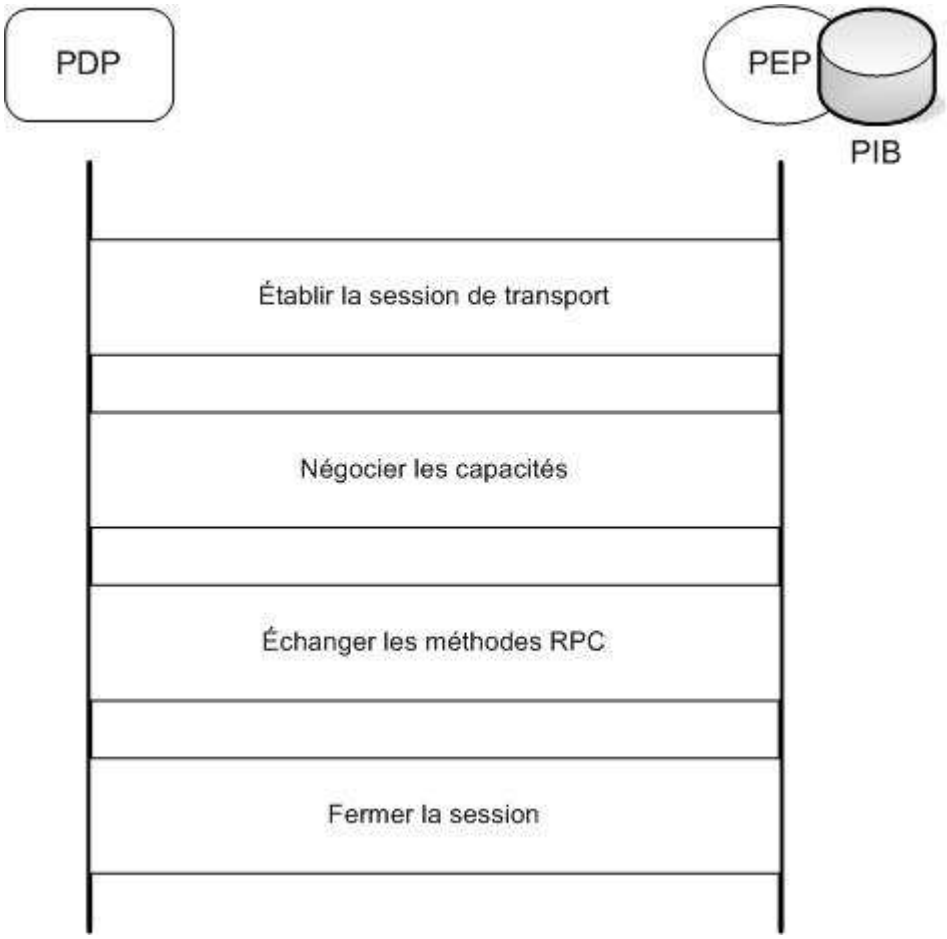


Figure 19: 4 étapes principales dans la phase de communication [9]

- **Quatrième phase:** une session NETCONF sera terminée par l'envoi d'une opération <**kill-session**> ou <**close-session**>.

3.3.3 Les points clés du protocole NETCONF

Les trois types de configurations: NETCONF distingue trois types de configuration *running*, *startup* et *candidate* [9]:

- La configuration *running* correspond à la configuration qui est en cours d'exploitation par l'équipement réseau. Il est toujours présent et est modifiable à chaud à condition que l'équipement supporte l'extension *#writable-running*. Il existe une seule configuration *running* dans chaque équipement. En ce qui concerne la représentation en XML, il est représenté par la balise **<*running*>**.
- La configuration *candidate* est une configuration temporaire de travail qui peut être utilisée au lieu de travailler directement sur la configuration en cours d'exploitation (*running*). Cela permet de faire des changements de comme si on travaillait sur la configuration *running* puis de valider ces changements (*validate*) et enfin de les appliquer (*commit*). On peut également annuler les changements (*discard-changes*). L'utilisation de la configuration *candidate* est un facteur de réduction des riques d'erreur de manipulation car les

changements sont seulement autorisés lorsque l'administrateur pense que la configuration est valable. En ce qui concerne la représentation en XML, il est représenté par la balise <***candidate***>.

- La configuration *startup* se rapporte à une configuration qui est chargée en mémoire lors du démarrage de l'équipement réseau. En cas de mise hors-tension puis de redémarrage de l'équipement, la configuration *running* est perdue et c'est cette configuration qui redevient celle courante. Il est évidemment possible de sauvegarder la configuration running dans ce stockage, grâce à l'opération ***copy-config***. En ce qui concerne la représentation en XML, il est représenté par la balise <***startup***> .

Les extensions ou bien les capacités: NETCONF est un protocole extensible par deux aspects : d'une part il est possible de définir de nouvelles opérations, d'autre part chacun peut étendre le jeu de données de configuration. Pour communiquer les capacités supportées, le gestionnaire et l'agent s'envoient mutuellement et de façon asynchrone un message *hello* contenant leurs extensions respectives lors de l'ouverture de la session NETCONF. Le protocole NETCONF utilise un espace de nom dédié définit par des URNs pour identifier de façon unique les extensions. Par exemple, si un agent annonce l'extension ***urn :ietf :params :netconf :capability :xpath :1.0***, cela signifie qu'il supporte la technologie XPath [9]. La figure 21 décrit un exemple de l'échange des capacités avec les messages *hello*.

XML Subtree filtering : Le protocole NETCONF permet à un administrateur de spécifier et sélectionner quelle partie de la donnée de configuration maintenue par un équipement réseau est ajoutée dans l'élément <**rpc-reply**>. Cette technique est nommée *XML subtree filtering* et permet de réaliser des requêtes de lecture sur une configuration XML. Cette méthode de sélection définit un ensemble des règles de filtrage qui régissent le processus de construction du document résultat. Le principal intérêt de cette technique est qu'elle permet à un administrateur de construire des sortes de templates qui correspondent à une vue partielle de la configuration d'un équipement. La figure suivante décrit le filtrage d'un routeur dont l'identificateur est égal **ASBR_157485** :

```
<filter type=``subtree''>
  <network xmlns=``http://serviceauto.org/filters/''>
    <routers>
      <router>
        <routerID>ASBR_157485<routerID>
      </router>
    </routers>
  </network>
</filter>
```

Figure 20 : Un exemple de XML Subtree filtering [9]

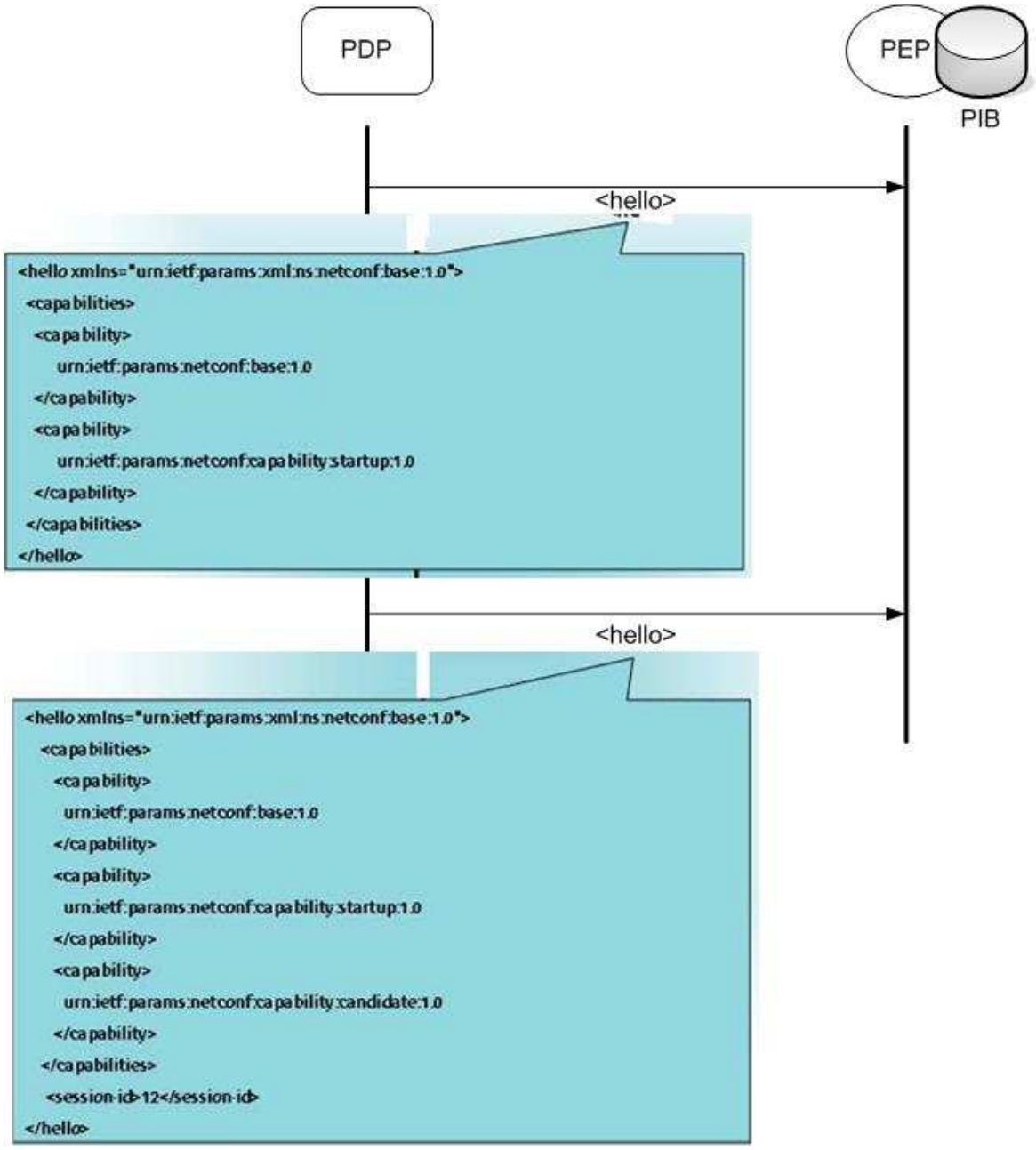


Figure 21: L'échange des capacités avec les messages hello [9]

En dehors de cette technique, il existe une autre manière de sélection basée sur XPath. Celle-ci a l'avantage de s'exprimer de façon beaucoup plus concise De plus, elle est une technologie standard et répandue dans XML. Un exemple de filtrage basé sur XPath est décrit dans la figure 22. Dans cet exemple, le client demande au serveur de fournir un fichier *sip.conf* contenant la configuration d'un serveur VoIP nommé Asterisk.

L'indépendance protocolaire et la sécurité: Le protocole NETCONF est un protocole indépendant par rapport à la couche Transport. Par conséquent, il peut fonctionner sur n'importe quel protocole de transport qui satisfait les exigences : *Connection oriented transport protocol* et un canal fiable pour transmettre les données sensibles et valider l'identité des participants. C'est pourquoi, l'utilisation sous-jacente du protocole SSH s'adapte à ces exigences

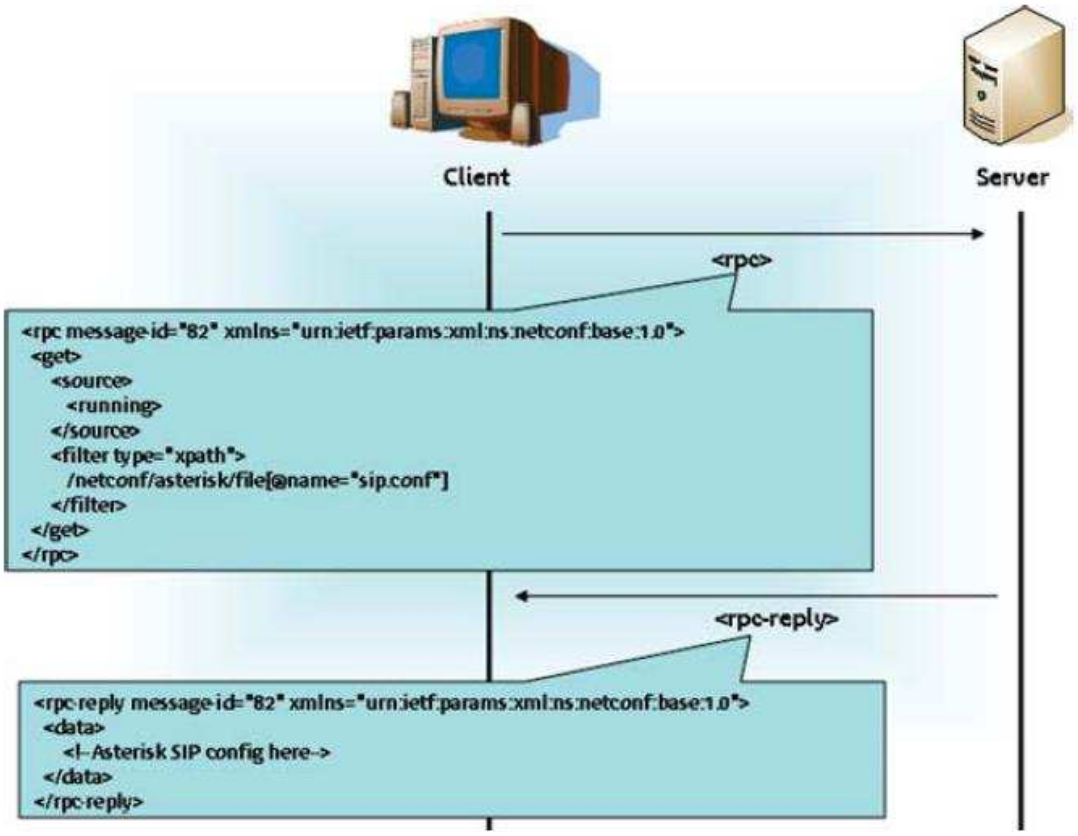


Figure 22 : Un exemple de filtrage basé sur XPath [9]

car SSH fournit un gage de sécurité en assurant l'authentification de l'agent et du gestionnaire, l'intégrité et la confidentialité. Mais la problématique du contrôle d'accès pour NETCONF est toujours en cours de discussion à l'IETF. Dans le futur, un draft sera écrit pour décrire les modèles et mécanismes de contrôle d'accès pour NETCONF.

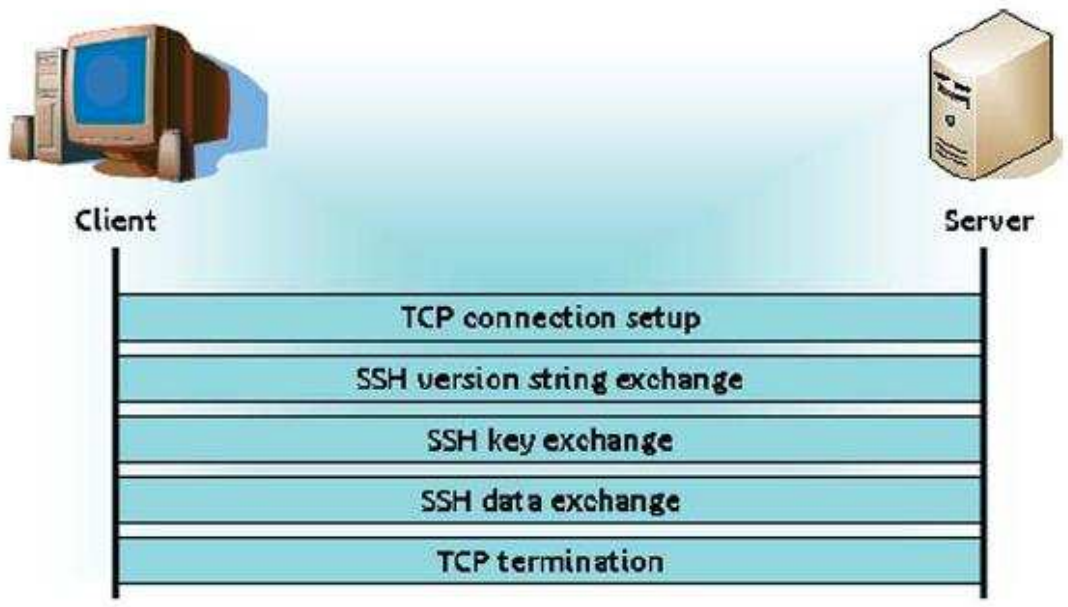


Figure 23: L'établissement de la connection basé sur SSH [9]

3.3.4 Ensuite

Ensuite (*Extended NETCONF Suite*) est une plateforme de configuration de réseau qui est développée par l'équipe MADYNES chez LORIA [12,26]. Ensuite implante l'agent (YencaP) et le gestionnaire (YencaPManager) du protocole NETCONF. Pour démarrer une communication, le gestionnaire ouvrira une session https (via un navigateur web) sur le serveur web de YencaPManager. Un gestionnaire peut évidemment ouvrir plusieurs sessions NETCONF avec des agents différents. Les connexions sont sécurisées des deux côtés de YencaPManager avec SSL pour la partie http et SSH pour NETCONF. Le modèle d'information de la plateforme est modulaire et extensible. L'ensemble des opérations peut également être étendu sans modifications du code interne.

L'architecture de l'agent (YencaP) définit une classe standard pour chacune des couches *Content*, *Operation*, *RPC* et *Transport* du protocole NETCONF. Chacune de ces classes communique par l'intermédiaire d'API bien définies (c'est-à-dire, tous les modules ont une interface de programmation commune héritée de la classe générique *Module*). De même, toutes les opérations de NETCONF sont implantées dans les classes respectives et héritent de la classe générique *Operation*. En ce qui concerne les réponses aux appels d'opération, on a défini des classes génériques qui servent à tout module et toute opération. Par exemple, les modules YencaP répondent à la couche *Operation* avec des objets *ModuleReply*. Ainsi, il est beaucoup plus simple d'implanter un nouveau module puisqu'il suffit d'utiliser les classes existantes. C'est également beaucoup plus propre pour la gestion des erreurs définies dans le protocole NETCONF car toutes les erreurs sont instanciables facilement grâce à la classe *ModuleReply* [12,26].

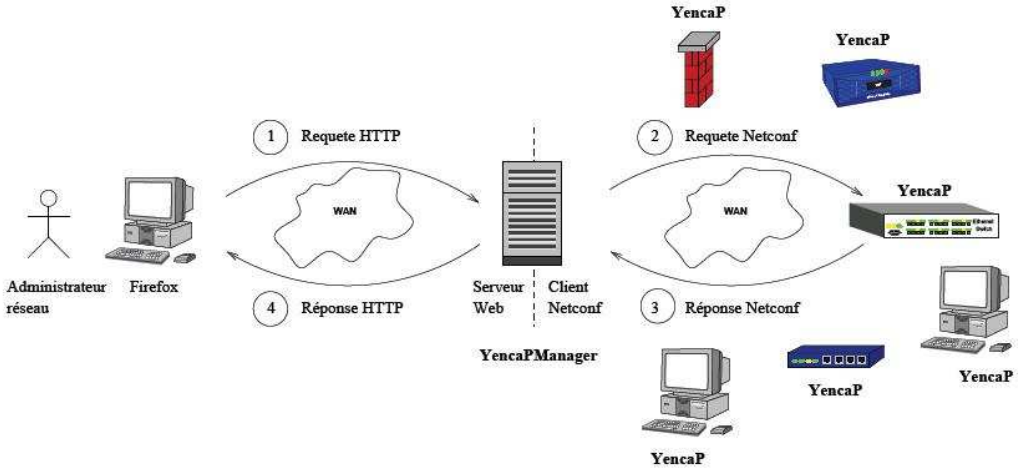


Figure 24 : Plateforme Ensuite [12,26]

YencaP fonctionne par défaut sur le protocole SSH, mais sa couche Server est suffisament générique pour être étendre aux protocoles SOAP et BEEP qui sont les deux autres protocoles de transport possibles de NETCONF. Dans Ensuite, les opérations de NETCONF sont organisées de façon à pouvoir ajouter de nouvelles commandes sans changer quoi que ce soit dans le code de

la pile NETCONF. Le design pattern *Command* répond exactement à cette problématique car il découple la logique de traitement de la classe concernée.

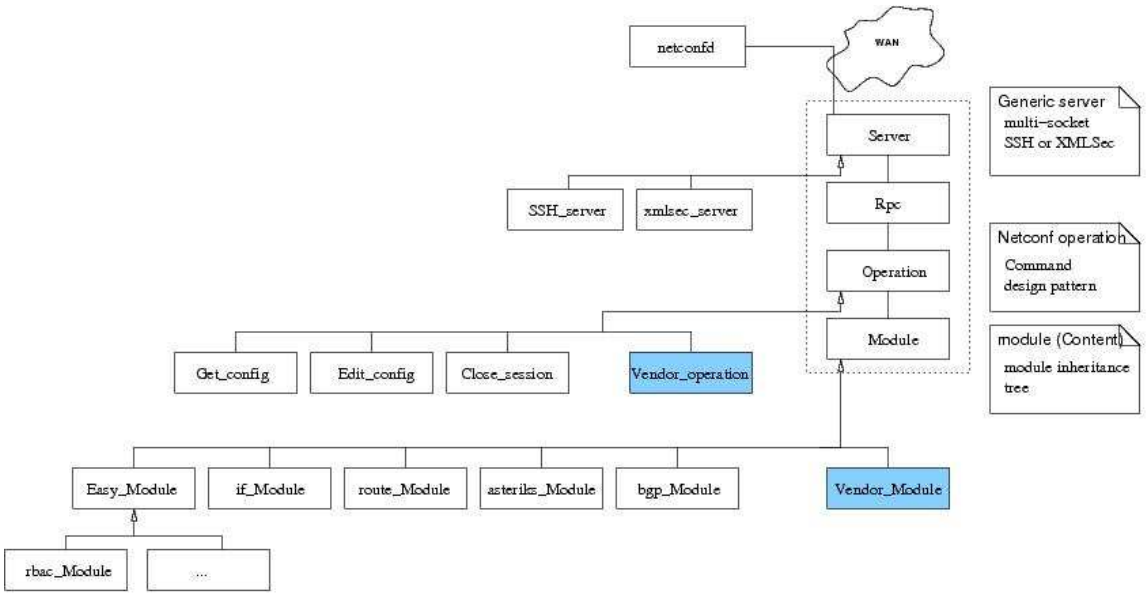


Figure 25: L'architecture multi-couches de l'agent YencaP [12,26]

Le modèle de données de YencaP est divisé en sous-modèles, chacun est géré par un module. L'extensibilité du modèle de donnée est assurée par l'ajout de nouveaux sous-modèles qui viennent se greffer dans l'arborescence générale représentant le modèle de données complet. Pour être interopérable et être intégré avec succès dans un agent YencaP, un module doit hériter de la classe *Module* qui définit un certain nombre de méthodes communes à tous les modules et implémenter un certain nombre de méthodes. Chaque module a son propre espace de nom (URN), ce qui permet d'accéder aux données sans ambiguité vis-à-vis d'un éventuel double emploi d'un nom de balise XML dans deux modules distincts. Les modules peuvent être déployés dans l'agent et chargés à chaud (c'est-à-dire, sans nécessiter de redémarrage de l'agent) [12,26]. À cet égard, afin de développer un module permettant de déployer des règles de sécurité vers des composant d'un réseau, il faut créer ce module, écrire le code pour spécifier ses fonctions et l'intégrer dans *Ensuite* (voir la section 5 du chapitre 4).

3.4 SNMP versus NETCONF

Après avoir présenté les deux protocoles NETCONF et SNMP dans les sections précédentes, nous comparons leur propriétés dans cette section pour sélectionner un protocole le plus convenable qui sera utilisé pour déployer des politiques de sécurité. Cette comparaison se base sur les critères suivants :

- **Premier critère**: la modélisation de données

SNMP définit l'information de management se basant sur SMI (*Structure of Management Information*) dont la capacité de la modélisation de données est limitée aux représentations scalaires [32]. Concrètement, SMI fournit une manière pour modéliser chaque objet géré. Par exemple, le tableau suivant est

une représentation scalaire des interfaces d'un routeur :

Interface name	ip address	mask	description
lo0	192.0.0.1	32	Loopback interface
s0-0/0/0	192.0.1.1	30	Serial interface

Cette représentation est définit par SMI comme suit:

```
...
ifTable OBJECT-TYPE
              SYNTAX  SEQUENCE OF IfEntry
              ACCESS  not-accessible
              STATUS  mandatory
              DESCRIPTION
                  "A list of interface entries"
              ::= { interfaces 2 }
ifEntry OBJECT-TYPE
              SYNTAX  IfEntry
              ACCESS  not-accessible
              STATUS  mandatory
              DESCRIPTION
                  "An interface entry containing objects
                   for a particular interface."
              INDEX   { ifIndex }
              ::= { ifTable 1 }
IfEntry ::=
              SEQUENCE {
                  ifName
                      Octet String,
                  ifDescr
                      DisplayString,
                  ifIPAddress
                      IpAddress,
                  ifMask
                      Integer
              }
ifDescr OBJECT-TYPE
              SYNTAX  DisplayString (SIZE (0..255))
              ACCESS  read-only
              STATUS  mandatory
              DESCRIPTION
                  "A textual string containing
                   information about the interface."
              ::= { ifEntry 1 }

...
```

Mais, s'il faut modéliser ces interfaces sous forme d'une représentation telle qu'un arbre, *Structure of Management Information* n'a pas de capacité pour réaliser cette modélisation. Par contre, NETCONF utilise le schéma XML ou Document Type Definition (DTD) qui permet de modéliser hiérarchiquement des données de la configuration. Par exemple, un arbre représentant les interfaces d'un routeur est suivant:

```
<interfaces>
<interface name="lo0">
<description>Loopback Interface</description>
<ip mask="32" type="ipv4">192.0.0.1</ip>
```

```
</interface>
<interface name="so-0/0/0">
<description>Serial Interface</description>
<ip mask="30" type="ipv4">192.0.1.1</ip>
</interface>
```

En pratique, les données de la configuration de réseau exigent souvent une modélisation de données hiérarchique. À cet égard, le protocole NETCONF est plus fort que le protocole SNMP.

- **Deuxième critère**: Le mécanisme d'identification d'objet

SNMP utilise OID (*Object Identifier*) pour identifier de façon unique chaque objet de MIB (*Management Information Base*). MIB est une base de données comportant des objets gérés (par exemple, une interface d'un routeur est un objet géré) [32]. Le processus de cette identification est effectué à partir d'un noeud *root* de l'arbre dont chaque noeud correspond à l'OID d'un objet géré. Par contre, NETCONF ne spécifie concrètement aucun mécanisme d'identification d'objet. Deux mécanismes, l'un nommé *Subtree filtering* et l'autre basé sur XPath sont utilisés mutuellement pour réaliser des requêtes d'identification sur une configuration XML (*voir la section 3.3.3 de ce chapitre*).

- **Troisième critère:** Les opérations de gestion de configuration

SNMP utilise une seule opération *SET* pour exécuter les opérations de mangement de configuration telles que changement, activation, et restauration d'une configuration. Par contre, NETCONF utilise le paradigme RPC (*Remote Procedure Call*) pour définir un ensemble d'opérations qui a pour but de récupérer des données (*get*, *get-config*), de modifier des données (*copy-config*, *delete-config*, *edit-config*), de poser et relâcher des verrous (*lock*, *unlock*) pour acquérir un accès privilégié aux données, de forcer la fermeture d'une session (*close-session*, *kill-session*). De plus, NETCONF apporte une gestion transactionnelle des configurations, grâce au concept de configuration *candidate* et aux opérations *validate* et *commit*. L'opération *validate* permet de valider la syntaxe et la sémantique de la configuration *candidate*, tandis que celle *commit* permet de remplacer officiellement la configuration *running* par la configuration *candidate*.

- **Quatrième critère:** Le protocole de transport

SNMP utilise le protocol de transport UDP (*User Datagram Protocol*) qui ne supporte pas de mécanismes de la retransmission et de la fiabilité. Par contre, NETCONF utilise plusieurs protocoles de transport tels que SSH (*Secure Shell*), SOAP (*Simple Object Access Protocol*) et BEEP (*Blocks Extensible Exchange Protocol*), mais c'est SSH qui s'impose comme le protocole obligatoire. SSH a apporté un niveau très élevé de sécurité et fournit les services d'authentification, confidentialité et non-répudiation sur lesquels NETCONF repose.

- **Cinquième critère :** Le modèle de contrôle d'accès

SNMPv3 utilise le modèle de contrôle d'accès VACM (*View-based Access Control Model*) [24] pour définir quels sont les droits des managers sur les MIBs (*Management Information Base*) par type d'opération (lecture, écriture, notification). Par contre, le modèle de contrôle d'accès pour NETCONF n'est pas encore spécifié et est toujours en cours de discussion à l'IETF (*Internet Engineering Task Force*) jusqu'à maintenant. Nous tentons d'envisager quel modèle de contrôle d'accès est le plus convenable pour NETCONF. En effet, de nombreux modèles de contrôle d'accès ont été adaptés pour la protection des données au format XML. La plupart reposent sur les rôles [3]. Ces modèles utilisent souvent des politiques de permission dont les ressources sont décrites à l'aide d'expressions XPath. Les auteurs de l'outil *Ensuite* ont proposé une extension au protocole NETCONF pour permettre un contrôle d'accès basé sur les rôles (RBAC) [26]. Ce mécanisme permet de définir les privilèges des utilisateurs sur les ressources de l'agent, matérialisées par sa configuration XML. Concrètement, lorsqu'une requête NETCONF se rapporte à un nœud particulier, le processus de contrôle d'accès consiste à vérifier que les opérations correspondantes sont bien dans un ACL (*Access Control List*). Le tableau suivant décrit comment une liste de contrôle d'accès (ACL) est ajoutée à un nœud XML représentant une interface [26] :

```
<interface>
 <ACL>
  <rule roleRef="myFirstRole">
   <operations><get/><replace/><merge/></operations>
  </rule>
  <rule roleRef="mySecondRole">
   <operations><get/></operations>
  </rule>
 </ACL>
 <name>Ethernet0/0</name>
 <mtu>1500</mtu>
</interface>
```

Elle définit que le rôle *myFirstRole* peut faire les opérations <*get* />, <*replace* /> et <merge /> sur ce nœud, tandis que le rôle *mySecondRole* ne peut faire que l'opération <*get* /> sur ce nœud XML.

- **Sixième critère :** Le déploiement des configurations

Comme déjà énoncé, les deux protocoles NETCONF et SNMP fonctionnent sur un paradigme *gestionnaire/agent*. Lorsqu'il faut effectuer un changement sur un agent, le gestionnaire va déployer de nouvelles configurations du changement sur cet agent. SNMP utilise une seule opération *SET* pour ce déploiement. L'opération *SET* ne permet que de changer des attribut d'un objet géré (par exemple, le changement de l'adresse IP de l'interface). En utilisant une seule opération *SET*, il est très difficile d'envoyer une configuration telle qu'un fichier *script* des commandes *iptables* vers un agent installant l'outil *Netfilter*, puis de l'effectuer sur cet agent. C'est une autre raison que nous avons choisie *cfengine* au lieu de HP OpenView

Network Node Manager, car *cfengine* fournit plus d'opérations, lorsqu'il faut déployer une configuration vers un agent (voir la section 3.2.2 de ce chapitre et la section 5 du chapitre 4). Pour le protocole NETCONF, les opération *copy-config* et *edit-config* sont utilisées pour déployer une configuration vers un agent. Plus concrètement, l'opération *copy-config* copie une configuration et l'opération *edit-config* exécute le contenu de cette configuration sur un agent.

En bref, le protocole NETCONF a plusieurs points forts par rapport au protocole SNMP. De plus, grâce au protocole NETCONF et au langage XML, il est possible d'aboutir à un niveau de sûreté du logiciel du mangement de réseau. En effet, le protocole NETCONF étant décrit par une grammaire du schéma XML, il est possible de vérifier la validité syntaxique et grammaticale de tous les messages NETCONF. Les configurations elles-mêmes peuvent être vérifiées à condition qu'un schéma XML ait été fourni avec le modèle de données. Une telle validation permet de détecter très simplement un très grand pourcentage des erreurs potentielles involontaires ou volontaires de programmation.

4 Conclusion

Ce chapitre a présenté l'architecture du système répondant aux menaces et attaques. Cette architecture est un modèle référencé pour concevoir et construire des applications qui sont capable de lutter dynamiquement contre des menaces dans un réseau informatique. Deux composants PDP et PEP de cette architecture et les protocoles de communications entre eux, surtout le protocole NETCONF, jouent les rôles importants dans le déploiement des politiques de sécurité. Dans ce chapitre, nous avons comparé ces protocoles en se basant sur les critères telles que la modélisation de données, le modèle de contrôle d'accès utilisé et le déploiement des configurations. Cette comparaison a montré leur avantages et inconvénients sur la vue théorique. La comparaison sur la vue pratique sera présentée par la mise en œuvre de leur implémentations (*cfengine* et *Ensuite*) à travers une étude de cas dans le chapitre suivant.

Chapitre 4: Etude de cas

1 Introduction

Pour justifier la capacité rélle de l'architecture du système répondant aux menaces et illustrer la méthodologie que nous avons montrée dans les chapitres précédents, nous présentons une étude de case basée sur un réseau pratique. L'étude consiste à spécifier des politiques de sécurité appliquées sur ce réseau et à mettre en oeuvre deux outils *Ensuite* et *cfengine* pour déployer ces politiques de sécurité vers des équipements dédiés et non-dédiés à la sécurité de ce réseau.

Le chapitre est débuté en décrivant la topologie d'un réseau pratique. Ensuite, des politiques de sécurité appliquées sur ce réseau sont spécifiées en Event-B. Puis, des règles de sécurité concrètes sont générées grâce au transformateur en XML. Enfin, elles sont déployées sur des composants dédiés et non-dédiés à la sécurité de ce réseau pratique.

2 Topologie de réseau

Le réseau de l'organisation *H* est décrit dans la figure suivante (la topologie est inspirée d'un exemple dans l'article [6]):

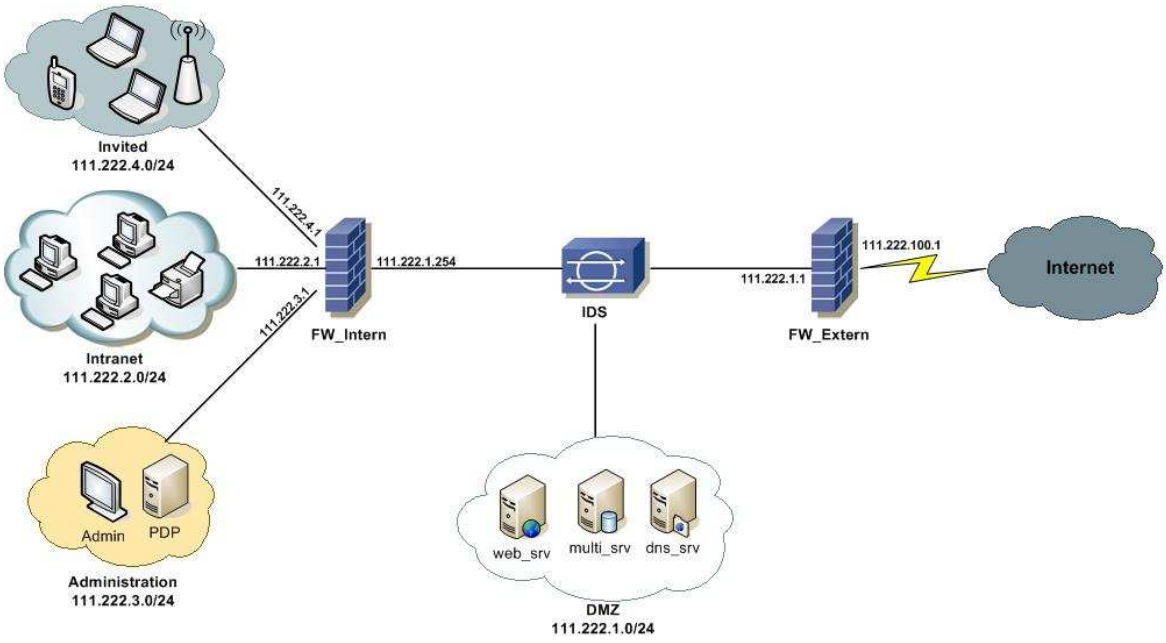


Figure 26: Topologie du réseau de l'organisation H

Le réseau est divisé par les sous-réseaux comme suit:

- La zone **DMZ** (111.222.1.0/24): contient plusieurs serveurs (serveur Web (*web_srv*), serveur DNS (*dns_srv*), multi-serveur (*multi_srv*) hébergeant plusieurs services (par exemple, email, base de donnée, DHCP,…).
- La zone **Administration** (111.222.3.0/24): prend en charge le

management du réseau. Elle contient un ordinateur (*Admin*) qui est capable d'accéder à la zone **DMZ** et aux composant dédié à la sécurité (*FW_Intern*, *FW_Extern* et *IDS*). La machine PDP est également dans cette zone.

- La zone **Intranet** (111.222.2.0/24): comporte les équipements (ordinateurs et imprimantes) de travail des employés dans l'organisation *H*.
- La zone **Invited** (111.222.4.0/24): est un sous-réseau sans fil qui a pour but de fournir des connexions aux équipements mobiles.

Le firewall *FW_Intern* sépare les trois zones **Intranet**, **Invited** et **Administration** et contrôle le flux d'information d'accès à ces trois zones. Le firewall *FW_Extern* protège le réseau contre des accès illicites en provenance de l'Internet. Le système de détection d'intrusion *IDS* envoie des alertes au ordinateur *Admin*, chaque fois qu'il détecte des signaux anormaux.

Nous nous intéressons aux politiques de sécurité de l'organisation *H* suivantes :

1.*Tous les ordinateurs dans l'organisation H peuvent accéder à l'Internet*
2. *La zone DMZ peut être accessible à l'extérieur*
3. *La zone* ***Intranet*** *peut accéder au serveur web*
4. *Toutes les zones peuvent accéder au service DNS*
5. *La zone* ***Administration*** *peut accéder à deux firewalls et aux serveurs via ssh*
6. *Le serveur Web est accessible à l'extérieur, quand aucune attaque syn-flooding est détectée*
7. *Interdire à toutes les connexions de l'Internet d'accéder au serveur web_srv, quand une attaque syn_flooding est détectée.*

Tableau 9: Les politiques de sécurité de l'organisation H

Maintenant, nous réaliserons le procecessus de modélisation de ces politiques de sécurité en utilisant le modèle Or-BAC. Pour ce faire, il faut, d'abord identifier ses éléments abstraits (*rôle*, *activité*, *vue*, *contexte*) :

- **Rôles**: *R_Intra, R_MultiServ, R_DNS, R_PEP*, *R_Corporate* qui est un rôle qui comporte toutes les entités de sous-réseau 111.222.0.0/16, *R_Internet* qui correspond au réseau Internet, *R_Admin* qui est rôle du sujet **Admin**.
- **Activités** : *WEB*(http et https), *TCP*(le service TCP), dns (le service DNS), *stop_ac* qui s'arrête le service *httpd*.
- **Vues**: La vue *Internet* où les activités (par exemple, web, email,…)

sont effectuées, la vue *web-srv* où les activités (par exemple, web, stop_ac,…) sont lancées, la vue *dnsServ* où le service *dns* est lancé, la vue *multi-serv* où le multi-service est lancé, la vue *Admin* représente une entité qui a le droit de gérer tous les équipements de l'organisation H.

- **Contextes**: le context *default* est activé dans la condition normale du réseau ; le context *synflooding* qui est un type du contexte de menace (*voir la section 4.2.4 du chapitre 2*) sera activé quand une attaque *synflooding* se produit. Dans le cadre de notre travail, nous n'envisageons pas comment la relation *hold* du contexte *synflooding* est définie . Supposons qu'on a la relation *hold* du contexte *synflooding* définie comme suit:
  - *alert*(*CreateTime, Source, Target, Classification*),
    reference(*Classification , 'CVE−1999−0116 ')* ,
    service(*Target* , *http*) ,
    hostname (*Target* , *web_srv*)
    → *hold*(*H,_,http, web_srv, synflooding*)

Les politiques de sécurité d'Or-BAC de l'organisation *H* sont décrites dans le tableau suivant:

sr1	*Permission(H, R_Corporate,WEB,Internet,default)*
sr2	*Permission(H, R_MultiServ, WEB, Internet, default)*
sr3	*Permission(H, R_Intra, WEB, multi-serv, default)*
sr4	*Permisssion(H, R_Corporate, dns, dnsServ, default)*
sr5	*Permisssion(H, R_DNS, dns, Internet, default)*
sr6	*Permisssion(H, R_PEP, ssh, Admin, default)*
sr7	*Permisssion(H, R_Internet, WEB, multi-serv, !(synflooding))*
sr8	*Prohibition(H, R_Internet, WEB, web-serv, synflooding)*
sr9	*Obligation(H, R_Admin, stop_ac, web-serv, synflooding)*

Tableau 10: Les politiques de sécurité d'Or-BAC de H

Pour illustrer la méthodologie que nous avons présentées, les trois politiques *sr1, sr8, sr9* sont choisies car elles représentent tous les trois types des politiques d'Or-BAC *permission*, *interdiction* et *obligation*. La politique *sr1* est activée dans le contexte *default* et les deux politiques *sr8*, *sr9* sont activées quand une attaque *synflooding* en provenance de l'Internet se produit.

D’abord, ces trois politiques de sécurté seront spécifiées en Event-B et en XML. Ensuite, les règles de sécurité concrètes qui correspondent à ces trois politiques de sécurité seront dérivées. Enfin, ces règles de sécurité seront déployées vers les deux firewalls et le serveur Web en utilisant les deux outils *Ensuite* et *cfengine*.

3 Spécification des politiques en Event-B

La spécification des politiques en Event-B a pour but de vérifier formellement et détecter des conflits entre des politiques de sécurité d’Or-BAC à deux niveaux abstraits et concrets. Pour cela, nous utilisons *Rodin 0.2.9.1* [*www.event-b.org*], un environnement de développement intégré pour Event-B, pour spécifier des politiques de sécurité appliquées sur le réseau de l’organisation *H*. Ces politiques sont spécifiées dans *Rodin* par deux contextes ORBAC_EDC_C0, ORBAC_EDC_C1 et deux machines abstraites ORBAC_EDC_0, ORBAC_EDC_1. Voici que leurs détails sont suivants :

Context ORBAC_EDC_C0 :

ORBAC_EDC_C0 est un composant particulier qui ne contient que les ensembles, les constantes que les machines abstraites utiliseront pour spécifier des politiques de sécurité d’Or-BAC au niveau abstrait. Par rapport au composant ORBAC_C1 (*voir la section 4.1.2 du chapitre 2*), nous ajoutons les nouvelles constantes qui représentent les rôles, les activités et les vues dans le réseau de l’organisation *H*.

CONTEXT ORBAC_EDC_C0 **SETS** ORGS ACTIVITIES VIEWS CONTEXTS ROLES	**CONSTANTS** H web stop_ac Internet web_srv default synflooding R_Corporate R_Internet R_Admin permission prohibition obligation
AXIOMS axm1: ORGS = {H} axm4: ACTIVITIES = {web, stop_ac} axm10:VIEWS = {Internet, web_srv} axm12:CONTEXTS = {default,synflooding} axm14:ROLES = {R_Corporate,R_Internet, R_Admin} axm18:permission ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS axm19:prohibition ⊆ORGS×ROLES ×ACTIVITIES × VIEWS × CONTEXTS axm20:obligation⊆ORGS × ROLES ×ACTIVITIES × VIEWS × CONTEXTS	

Machine ORBAC_EDC_0 :

ORBAC_EDC_0 est une machine abstraite qui spécifie des politiques de sécurité d’Or-BAC au niveau abstrait.

```
MACHINE
ORBAC_EDC_0
SEES
ORBAC_EDC_C0
VARIABLES
ctx    //   Context Variable
hist_abst_permission
hist_abst_prohibition
hist_abst_obligation
```

```
INVARIANTS
inv1: ctx ∈ CONTEXTS
inv2: hist_abst_permission ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS
inv3: hist_abst_permission ⊆ permission
inv4: hist_abst_prohibition ⊆ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS
inv5: hist_abst_prohibition⊆prohibition
inv6: hist_abst_obligation⊆ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS
inv7: hist_abst_obligation ⊆ obligation
```

En se basant sur les actions que les trois politiques de sécurité *sr1*, *sr8*, *sr9* décrivent (p.ex, *sr1* décrit l'action « *permettre d'accéder à l'Internet »*), on a trois événements correspondants *permission_access_to_Internet*, *obligation_stop_WebServer*, *prohibition_access_to_WebServer*.

```
EVENTS
INITIALISATION   ≙
BEGIN
act1 : ctx≔ default
act2 : hist_abst_permission ≔ ∅
act3: hist_abst_prohibition ≔ ∅
act4: hist_abst_obligation ≔ ∅
END
set_context_value   ≙
ANY
c
WHERE
grd1 :    c ∈ CONTEXTS
THEN
act1 :    ctx ≔ c
END
prohibition_access_to_WebServer ≙
ANY
org,r,v,av
WHERE
grd1: org ∈ ORGS ∧ org = H
grd2: r ∈ ROLES ∧ r = R_Internet
grd3: v ∈ VIEWS ∧ v = web_srv
grd4: av ∈ ACTIVITIES ∧ av = web
grd5:    ctx = synflooding
grd6: (org ↦ r ↦ av ↦ v ↦ ctx)
∈ prohibition
THEN
act1:hist_abst_prohibition      ≔
hist_abst_prohibition ∪ {(org ↦
r ↦ av ↦ v ↦ ctx)}
END
```

```
permission_access_to_Internet   ≙
ANY
org,r,v,av
WHERE
grd1: org ∈ ORGS ∧ org = H
grd2:  r  ∈   ROLES  ∧   r  =
R_Corporate
grd3: v ∈ VIEWS ∧ v = Internet
grd4: av ∈ ACTIVITIES ∧ av = web
grd5: ctx = default
grd6: (org ↦ r ↦ av ↦ v ↦ ctx)
∈ permission
THEN
act1:hist_abst_permission≔
hist_abst_permission ∪ {(org ↦ r
↦ av ↦ v ↦ ctx)}
END
obligation_stop_WebServer   ≙
ANY
org,r,v,av
WHERE
grd1: org ∈ ORGS ∧ org = H
grd2: r ∈ ROLES ∧ r = R_Admin
grd3: v ∈ VIEWS ∧ v = web_srv
grd4:  av  ∈   ACTIVITIES  ∧   av  =
stop_ac
grd5 : ctx = synflooding
grd6: (org ↦ r ↦ av ↦ v ↦ ctx)
∈ obligation
THEN
act1:   hist_abst_obligation   ≔
hist_abst_obligation ∪ {(org ↦ r
↦ av ↦ v ↦ ctx)}
END
END
```

Contexte ORBAC_EDC_C1 :

ORBAC_EDC_C1 raffine le composant ORBAC_EDC_C0 en ajoutant les ensembles, les constantes concernant les politiques de sécurités au niveau concret de l'organisation *H*.

CONTEXT	**CONSTANTS**
ORBAC_EDC_C1	fw_intern
EXTENDS	fw_extern
ORBAC_EDC_C0	subnet_111_222_0_0
SETS	subnet_111_222_1_0
SUBJECTS	subnet_any
ACTIONS	service_http_port80
OBJECTS	command_httpd_stop
	host_111_222_1_11
	any_subnet
	empower
	use
	consider
	hold
	admin

AXIOMS

axm1:SUBJECTS={fw_intern,fw_extern,subnet_111_222_0_0,subnet_111_222_1_0, subnet_any,admin}

axm12: ACTIONS = { service_http_port80, command_httpd_stop}

axm14: OBJECTS={any_subnet,host_111_222_1_11}

axm16: empower $\subseteq$ ORGS $\times$ ROLES $\times$ SUBJECTS

axm17: use $\subseteq$ ORGS $\times$ VIEWS $\times$ OBJECTS

axm18: consider $\subseteq$ ORGS $\times$ ACTIVITIES $\times$ ACTIONS

axm19: hold $\subseteq$ ORGS $\times$ SUBJECTS $\times$ ACTIONS $\times$ OBJECTS $\times$ CONTEXTS

Machine ORBAC_EDC_1 :

ORBAC_EDC_1 est une machine abstraite qui spécifie des politiques de sécurité d'Or-BAC au niveau concret. Elle raffine la machine ORBAC_EDC_0. Les invariants de liaison *inv3*,*inv9*,*inv10* qui contraignent les variables de la machine ORBAC_EDC_1 avec celles de la machine ORBAC_EDC_0 sont suivants :

MACHINE
ORBAC_EDC_1
REFINES
ORBAC_EDC_0
SEES
ORBAC_EDC_C1
VARIABLES
ctx,s,o,a
hist_conc_permission
hist_conc_prohibition
hist_conc_obligation
INVARIANTS

inv1 : ctx $\in$ CONTEXTS

inv4 : s $\in$ SUBJECTS

inv5 : o $\in$ OBJECTS

inv6 : a $\in$ ACTIONS

inv2: hist_conc_permission$\subseteq$ SUBJECTS $\times$ ACTIONS $\times$ OBJECTS

inv7: hist_conc_prohibition$\subseteq$ SUBJECTS $\times$ ACTIONS $\times$ OBJECTS

inv8: hist_conc_obligation $\subseteq$ SUBJECTS $\times$ ACTIONS $\times$ OBJECTS

inv3: $\forall$ sub$\cdot$ $\forall$ ac$\cdot$ $\forall$ obj$\cdot$((sub $\in$ SUBJECTS $\wedge$ ac $\in$ ACTIONS $\wedge$ obj $\in$ OBJECTS $\wedge$ (sub $\mapsto$ ac $\mapsto$ obj) $\in$ hist_conc_permission) $\Rightarrow$

(∃org·∃r·∃av·∃v·∃c·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ c ∈ CONTEXTS ∧ (org ↦ r ↦ sub) ∈ empower ∧ (org ↦ v ↦ obj) ∈ use ∧ (org ↦ av ↦ ac) ∈ consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_permission)))

inv9: ∀ sub· ∀ ac· ∀ obj·((sub ∈ SUBJECTS ∧ ac ∈ ACTIONS ∧ obj ∈ OBJECTS ∧ (sub ↦ ac ↦ obj) ∈ hist_conc_prohibition) ⇒ (∃org·∃r·∃av·∃v·∃c·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ c ∈ CONTEXTS ∧ (org ↦ r ↦ sub) ∈ empower ∧ (org ↦ v ↦ obj) ∈ use ∧ (org ↦ av ↦ ac) ∈ consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_prohibition)))

inv10: ∀ sub· ∀ ac· ∀ obj·((sub ∈ SUBJECTS ∧ ac ∈ ACTIONS ∧ obj ∈ OBJECTS ∧ (sub ↦ ac ↦ obj) ∈ hist_conc_obligation) ⇒ (∃org·∃r·∃av·∃v·∃c·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ c ∈ CONTEXTS ∧ (org ↦ r ↦ sub) ∈ empower ∧ (org ↦ v ↦ obj) ∈ use ∧ (org ↦ av ↦ ac) ∈ consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_obligation)))

Les événements de la machine ORBAC_EDC_1 qui raffine ceux de la machine ORBAC_EDC_0 sont suivants:

EVENTS

INITIALISATION ≙
BEGIN
act1: ctx ≔ default
act2: hist_conc_permission ≔ ∅
act3:s≔null1,o ≔ null3,a ≔ null2
act6: hist_conc_prohibition ≔ ∅
act7: hist_conc_obligation ≔ ∅
END

set_context_value ≙
REFINES
set_context_value
ANY
c
WHERE
grd1 : c ∈ CONTEXTS
THEN
act1 : ctx ≔ c
END

set_action_value ≙
ANY
ac
WHERE
grd1 : ac ≠ null2
grd2 : ac ∈ ACTIONS
THEN
act1 : a ≔ ac
END

permission_access_to_Internet ≙
REFINES
permission_access_to_Internet
ANY

set_subject_value ≙
ANY
sub
WHERE
grd1 : sub ∈ SUBJECTS
grd2 : sub ≠ null1
THEN
act1 : s ≔ sub
END

set_object_value ≙
ANY
obj
WHERE
grd1 : obj ∈ OBJECTS
grd2 : obj ≠ null3
THEN
act1 : o ≔ obj
END

prohibition_access_to_WebServer ≙
REFINES
prohibition_access_to_WebServer
ANY
org,r,av,v
WHERE
grd1: org ∈ ORGS ∧ org = H
grd2: r ∈ ROLES ∧ r = R_Internet
grd3: v ∈ VIEWS ∧ v = web_srv
grd4: av ∈ ACTIVITIES ∧ av = web
grd5: ctx = synflooding
grd6: (org↦r↦av↦v↦ctx) ∈ prohibition
grd7: s = subnet_any∧s≠fw_intern ∧ s ≠ fw_extern ∧ s ≠ subnet_111_222_0_0 ∧

org,r,v,av
WHERE
grd1:org ∈ ORGS ∧ org = H
grd2:r ∈ ROLES ∧ r = R_Corporate
grd3:v ∈ VIEWS ∧ v = Internet
grd4:av ∈ ACTIVITIES ∧ av = web
grd5:ctx = default
grd6:(org ↦ r ↦ av ↦ v ↦ ctx) ∈ permission
grd7: s = subnet_111_222_0_0 ∧ s ≠ fw_intern ∧ (org↦r↦s) ∈ empower
grd8:o = any_subnet ∧ (org↦v↦o) ∈ use
grd9:a = service_http_port80 ∧ (org↦av↦a) ∈ consider
grd10:(org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
THEN
act1: hist_conc_permission ≔ hist_conc_permissio∪{(s↦a↦ o)}
END

(org↦r↦s) ∈ empower
grd8:o=host_111_222_1_11(org↦v↦o) ∈ use
grd9:a=service_http_port80 ∧ (org↦av↦a) ∈ consider
grd10:(org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
THEN
act1:hist_conc_prohibition≔ hist_conc_prohibition ∪ {(s ↦ a ↦ o)}
END

obligation_stop_WebServer ≙
REFINES
obligation_stop_WebServer
ANY
org,r,v,av
WHERE
grd1: org ∈ ORGS ∧ org = H
grd2: r ∈ ROLES ∧ r = R_Admin
grd3: v ∈ VIEWS ∧ v = web_srv
grd4: av ∈ ACTIVITIES ∧ av = stop_ac
grd5: ctx = synflooding
grd6: (org↦r↦av↦v↦ctx) ∈ obligation
grd7: s = admin ∧ (org↦r↦s) ∈ empower
grd8:o=host_111_222_1_1∧(org↦v↦o)∈ use
grd9:a= command_httpd_stop ∧ (org↦av↦a) ∈ consider
grd10: (org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
THEN
act1:hist_conc_obligation≔ hist_conc_obligation ∪ {(s ↦ a ↦ o)}
END

4 Expression des politiques en XML

L'expression des politiques en XML a pour but de générer des règles de sécurité concrètes, à partir des politiques de sécurité d'Or-BAC au niveau abstrait en XML. Elle a été réalisée à la main, nous développerons un module pour la faire sur l'ordinateur dans le futur. Elle comporte deux étapes :

- Transformer des politiques de sécurité d'Or-BAC au niveau abstrait (c'est-à-dire, des politiques de sécurité d'Or-BAC sous forme d'un quadruple (*rôle*,*activité*,*vue*,*contexte*) en des politiques de sécurité d'Or-BAC en XML au niveau concret (c'est-à-dire, des politiques de sécurité d'Or-BAC sous forme d'un triplet (*sujet*,*action*,*objet*). Avant cette transformation, ces deux types des politiques de sécurité d'Or-BAC ont été formellement vérifiés en Event-B pour détecter des conflits entre eux. PIE (*Policy Instantiation Engine*) est responsable de réaliser cette étape.
- Générer des règles de sécurité conrètes qui sont prêtes à déployer sur des composants dédiés ou non-dédiés à la sécurtié correspondants (PEPs), à partir de ces politiques de sécurité d'Or-BAC au niveau

concret. PDP (*Policy Decision Point*) est responsable de réaliser cette étape.

Pour la politique de sécurité sr1 :

Permission(H,R_Corporate,WEB,Internet,default)

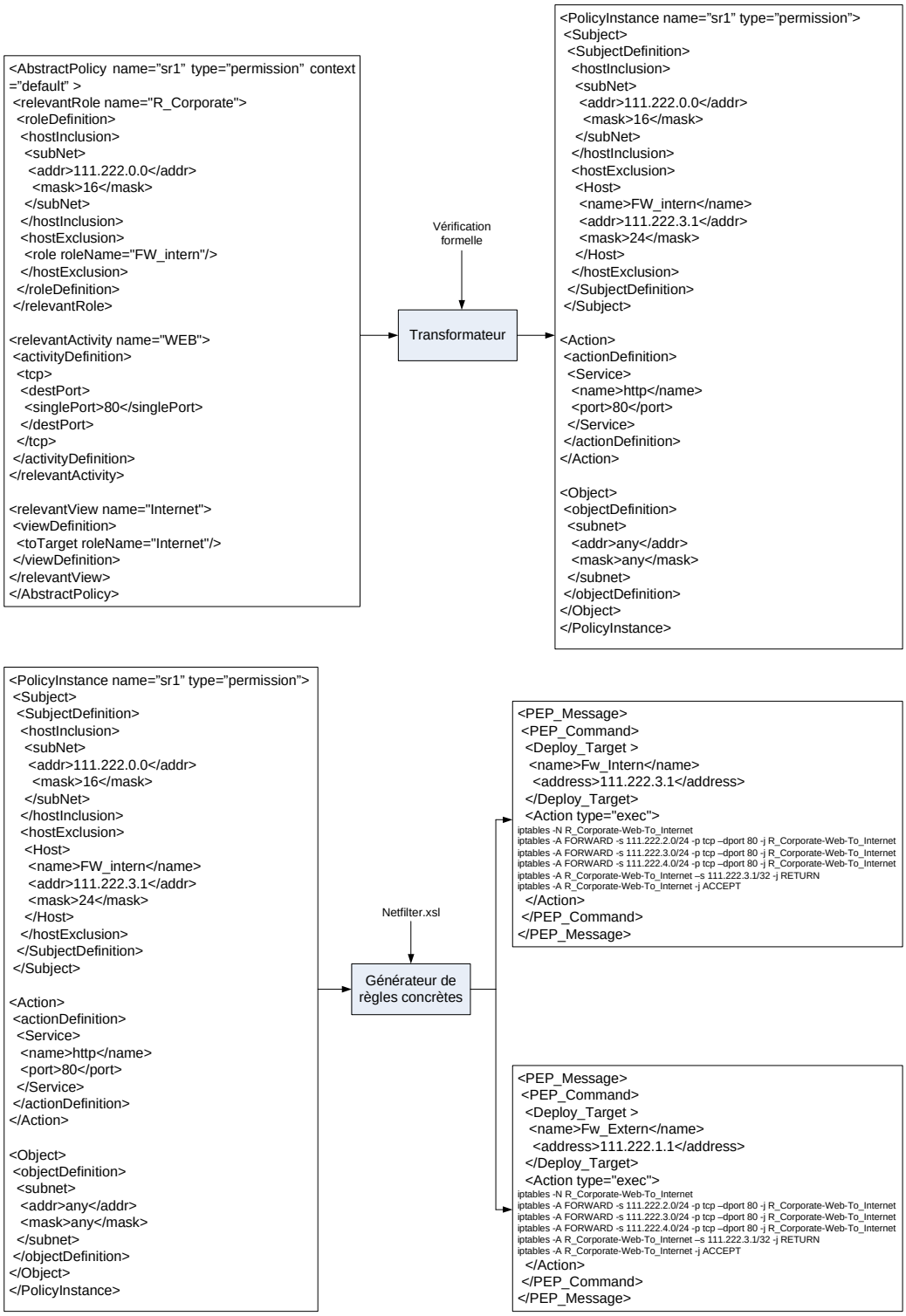


Figure 27: Génération de règles concrètes de sr1

Pour la politique de sécurité *sr8* :

Prohibition(H, R_Internet, WEB, web-serv, synflooding)

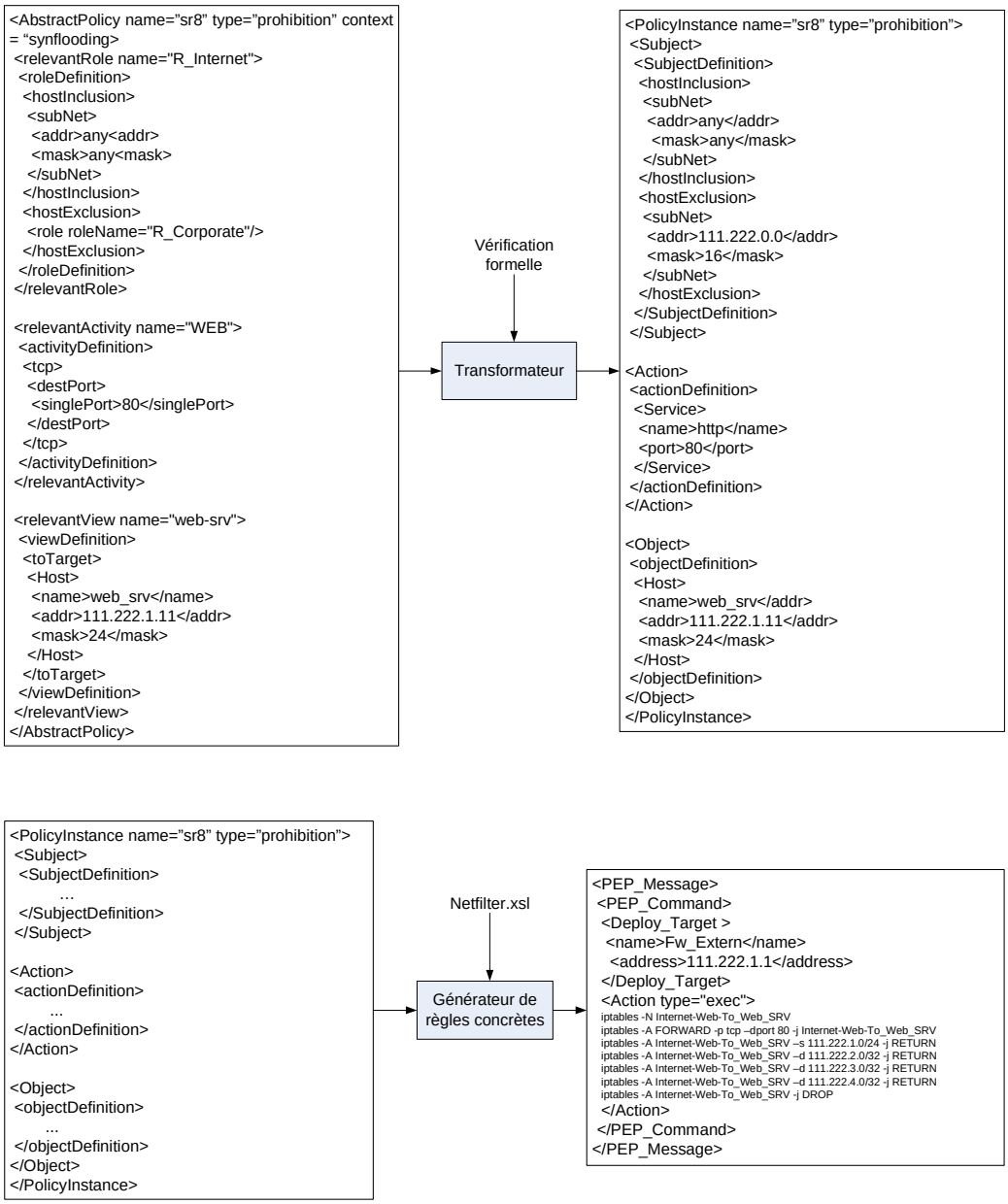


Figure 28: Génération de règles concètes de sr8

Pour la politique de sécurité *sr9* :

Obligation(H,R_Admin, stop_ac, web-serv, synflooding)

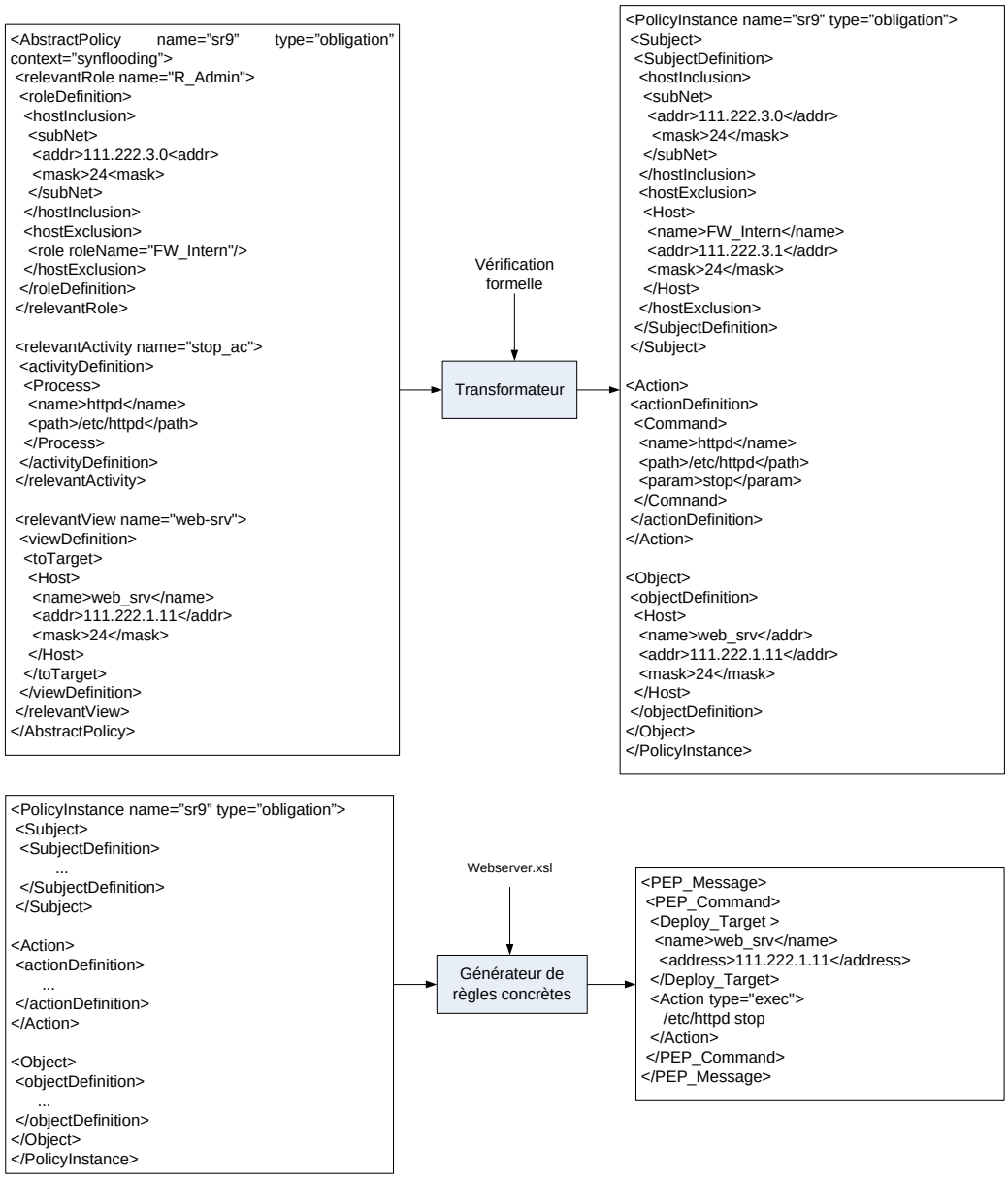


Figure 29: Génération de règles concrètes de sr9

Les fichiers *Netfilter.xsl* et *Webserver.xsl* qui sont écrits en XSLT (*Extensible Stylesheet Language for Transformations*) contiennent des informations qui permettent d'extraire des règles de sécurité concrètes. Par exemple, pour générer la commande */etc/httpd stop* qui est lancée sur le serveur Web, il faut créer la feuille de style XSLT suivante :

```
…
<xsl:template match='Command'>
 <xsl:text></xsl:text><xsl:value-of select="@path"/>
 <xsl:text> </xsl:text>
 <xsl:text></xsl:text><xsl:value-of select="@param"/>
</xsl:template>
…
```

Tableau 11 : Exemple d'une feuille de style XSLT

Basant sur cette feuille de style XSLT, la règle de sécurité concrère qui contient la commande */etc/httpd stop* est formée (*voir la figure 29*).

5 Déploiement des politiques de sécurité

Dans cette section, nous allons déployer des règles de sécurité concrètes vers les composants dédiés à la sécurité (le firewall *FW_Intern*) et non-dédiés à la sécurité (le serveur web). Pour ce faire, d'abord, nous allons décrire la configuration matérielle qui est utilisée dans notre étude de cas :

	Admin/PDP (Debian0:111.222.3.2)	**FW_Intern (Debian2:111.222.3.1)**	**web_srv (Debian1:111.222.1.11)**
CPU	AMD Athlon 64	AMD Athlon 64	AMD Athlon 64
Mémoire	1024 MB	1024 MB	512 MB
Réseau	Fast Ethernet 100 Mb/s	Fast Ethernet 100 Mb/s	Fast Ethernet 100 Mb/s
OS	Debian Linux 2.6.x kernel	Debian Linux 2.6.x kernel avec Netfilter 1.x	Debian Linux 2.6.x kernel avec http en service
Service		netfilter	httpd
cfengine	cfengine-2.2.8	cfengine-2.2.8	cfengine-2.2.8
YencaP	yencap-manager-2.1.11	yencap-2.1.11	yencap-2.1.11

Tableau 12: Configuration matérielle de notre étude de cas

Ensuite, nous allons déployer des règles de sécurité concrètes sur le firewall *FW_Intern*. Ces règles de sécurité sont contenues dans le fichier *rules.fw*. Pour ce faire, nous allons configurer simultanément *Ensuite* et *cfengine* sur les deux machines **Debian0** et **Debian2** selon la figure suivante :

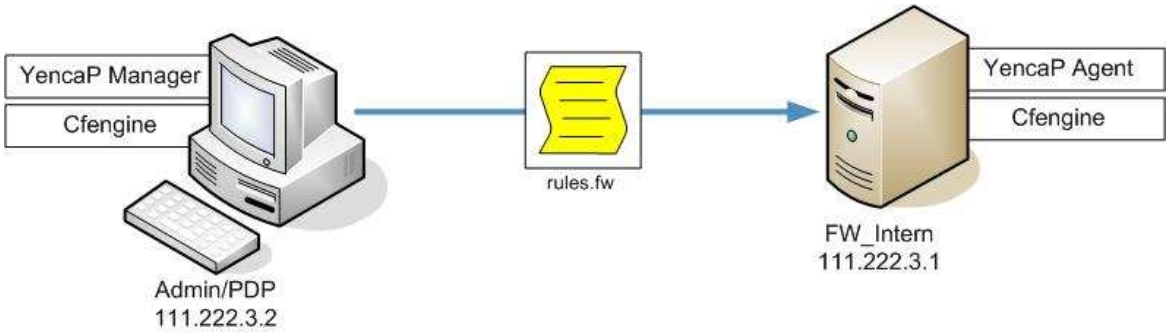


Figure 30: Modèle d'implémentation entre Admin/PDP et FW_Intern

Afin de configurer *cfengine*, il faut réaliser les étapes suivantes:

Étape 1: Dans le répertoire */var/cfengine* de la machine *Admin/PDP*, il faut créer le répertoire *masterfiles* qui contient les sous-répertoires comme suit:

```
# ls /var/cfengine/masterfiles

inputs (This will hold the main configuration files)
config (This will hold the actual files which needs to be
```

```
copied to the CFengine clients)
      |netfilter
       |rules.fw
      |server
       |rules.serv
```

Étape 2: Dans le répertoire */var/cfengine/masterfiles/inputs* de la machine *Admin/PDP*, il faut créer les fichiers *cfagent.conf*, *cfservd.conf*, *update.conf*, *cfagent.global.conf* et *cfrun.hosts* (voir le contenu de ces fichiers dans la partie Annexes).

Étape 3: Dans la machine *Admin/PDP*, il faut copier les fichier *cfagent.conf*, *cfservd.conf*, *update.conf*, *cfagent.global.conf* et *cfrun.hosts* du répertoire */var/cfengine/masterfiles/inputs* au répertoire */var/cfengine/inputs*.

Étape 4: Dans les deux machines *Admin/PDP* et *FW_Intern*, il faut lancer la commande *cfkey* pour créer les fichiers de clé. Il faut renommer le fichier *localhost.pub* en nouveau fichier sous forme de *root-Ipaddress.pub* (par exemple, *root-111.222.3.2.pub*) . Puis, il faut copier ce fichier dans le répertoire */var/cfengine/ppkeys* de l'une à l'autre machine et vice-versa.

Étape 5: Sur les deux machines *Admin/PDP* et *FW_Intern*, il faut lancer la commande *cfservd –F*.

Étape 6: Dans la machine *Admin/PDP*, il faut lancer la commande *cfrun –v* pour commencer le processus de distribuer le fichier *rules.fw***.**

Étape 7: Dans la machine *FW_Intern*, il faut lancer la commande *iptables –L* pour voir le résultat.

Afin de configurer *Ensuite,* il faut réaliser les étapes suivantes:

Étape 1: Il faut créer un squelette du module en utilisant l'outil ***YencaP Module Generator*** qui est publié dans le même site que *YencaP*. Ce squelette comprend trois fichiers: *PolicyDistr_Module.py*,*_init_.py*, *README*. Le répertoire *PolicyDistr_Module* qui contient ces trois fichier doit être copié au *${YENCAP_Agent_HOME}/Modules*.

Étape 2: Mettre à jour le fichier *modules.xml* dans les deux répertoires *${YENCAP_Agent_HOME}/modules.xml*,*${YENCAP_Manager_HOME}/modules.xml* sur les deux machines en ajoutant à ce fichier les lignes suivantes:

```
<module>
  <name>PolicyDistr</name>
    <xpath>/ycp:netconf/ycp:security/pd:PolicyDistr</xpath>
    <namespace pref="pd">
      urn:loria:madynes:ensuite:yencap:module:PolicyDistr:1.0
    </namespace>
    <cachelifetime>1000000</cachelifetime>
</module>
```

Étape 3: Il faut écrire le code en Python dans le fichier *PolicyDistr_Module.py* pour décrire les opération *get-config* et *edit-config* qui permettent de recevoir et exécuter des règles de sécurité (le fichier *PolicyDistr_Module.py* est dans l'*Annexes*).

Étape 4: Lancer la commande */etc/init.d/yencap-manager start* sur la

machine *Admin/PDP*. Il faut également lancer la commande */etc/init.d/yencap start* sur la machine *FW_Intern*.

Étape 5: Ouvrir le browser Web sur la machine *Admin/PDP* avec l'adresse *https://localhost:8888* et, il faut lancer le login *netconf* et le mot de passe *netconf* pour accéder à l'interface Web de *YencaP Manager*.

Étape 6: Dans l'interface Web de *YencaP Manager*, il faut choisir la machine *FW_Intern* (111.222.3.1) pour préparer le déploiement d'une politique.

Étape 7: Activer le rôle *Security Manager***,** choisir le module PolicyDistr_Module et donner le fichier *rules.fw* pour commencer à déployer la politique à la machine *FW_Intern*.

Nous venons de configurer *Ensuite* et *cfengine* sur une machine incarnant le PDP (*Policy Decision Point*) et sur un firewall *netfilter* pour déployer des règles de sécurtié concrètes. Pour compléter ce processus de déploiement, il nous faut configurer *Ensuite* et *cfengine* sur le serveur Web *Debian1* selon la figure suivante. Le fichier *rules.serv* est créé pour contenir des règles de sécurité réservées au serveur Web.

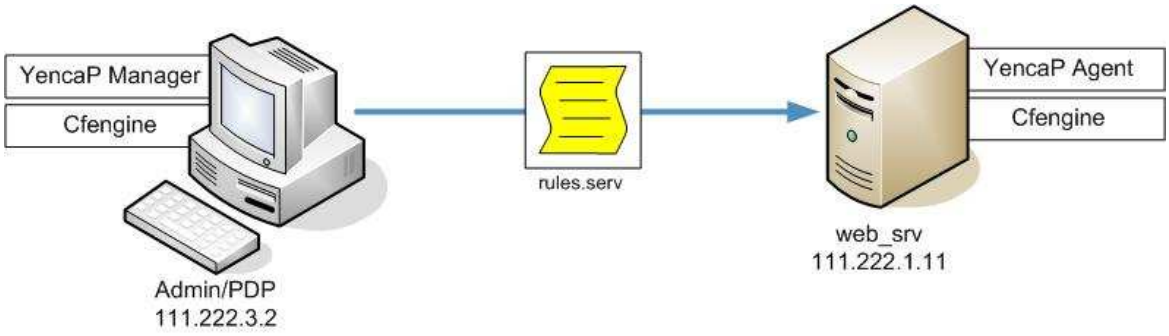


Figure 31: Modèle d'implémentation entre Admin/PDP et web_srv

Afin de distribuer cette règle de sécurité au serveur *web_srv*, on réalise les mêmes étapes que dans le paragraphe ci-dessus pour configurer *cfengine* et *YenCaP* (le fichier *rules.serv* est hébergé dans le répertoire */var/cfengine/masterfiles/server/*).

6 Évaluation et résultats

Dans cette section, nous évaluons le résultat de la modélisation et du déploiement des politiques de sécurité que nous avons fait à travers l'étude de cas. D'abord, la spécification des politiques de sécurité en Event-B permet de vérifier formellement et détecter les conflits générés entre les politiques de sécurité appliquées sur le réseau de l'organisation *H*. Pour ce faire, des invariants qui décrivent les conflits ont été ajoutés dans les machines abstraites. Les conflits qui ont été formellement vérifiés sont suivants :

- *Les contradictions* se produisent quand on peut dériver à la fois toutes les deux politiques de sécurité de la permission et de l'interdiction qui contiennent les mêmes rôles, activités et vues, ou bien les mêmes sujets, actions et objets. Par exemple, dans le réseau de l'organisation *H*, la

politique de sécurité de la permission « *Permettre à tous les ordinateurs de l'organisation H (c'est-à-dire,les ordinateurs des subnets 111.222.x.0/24 (x=1,2,3,4)) d'accéder à l'Internet*» peut être exprimée par celle de l'interdiction équivalente « *Interdire à tous les ordinateurs hors de l'organisation H (c'est-à-dire, les ordinateurs n'appartiennent pas au subnet 111.222.x.0/24 (x=1,2,3,4)) d'accéder à l'Internet* ». Par conséquent, on peut dériver à la fois les politiques de sécurité d'Or-BAC comme suit:

Permission(H, R_Corporate,WEB,Internet,default)

Is_permitted(subnet_111.222.0.0,servive_http_port_80,any_subnet)

Prohibition(H, R_Corporate,WEB,Internet,default)

Is_prohibited(subnet_111.222.0.0,servive_http_port_80,any_subnet)

Pour vérifier les contradictions, il faut ajouter les invariants suivants dans les deux machines abstraites ORBAC_EDC_0 et ORBAC_EDC_1:

- `inv8:hist_abst_permission ∩ hist_abst_prohibition = ∅`
- `inv11:hist_conc_permission ∩ hist_conc_prohibition= ∅`

- *Les incapacités* se produisent quand on peut dériver, à partir de mêmes sujets et objets, toutes les deux politiques de sécurité au niveau concret *is_obliged*(*sujet,action1,objet*) et *is_obliged*(*sujet,action2,objet*), mais il est impossible de réaliser en même temps toutes ces deux actions (par exemple, *action1* est l'action de s'arrêter le serveur Web et *action2* est l'action de démarrer le serveur Web). Pour vérifier les incapacités, il faut ajouter l'invariant suivant dans la machine abstraite ORBAC_EDC_1 :

```
inv12: ∀s1.∀s2.((s1∈SUBJECTS ∧ s2∈SUBJECTS ∧
(s1↦stop_ac↦host_111_222_1_11)∈ hist_conc_obligation) ∧
(s2↦start_ac↦host_111_222_1_11) ∈ hist_conc_obligation)
⇒ (s1 ≠ s2)
```

Cet invariant dit que le même sujet ne peut pas réaliser en même temps deux actions *stop_ac* et *start_ac* sur le même objet *host_111_222_1_11* (le serveur Web).

Les obligations de preuves qui étaient générées après cet ajout des invariants sont décrites dans les trois figures 32,33,34 suivantes. Ces obligations de preuves montre qu'il existe encore des conflits entre des politiques de sécurité appliquées sur le réseau de l'organisation *H*. Concrètement, pour l'obligation de preuve générée sur l'invariant `hist_abst_permission ∩ hist_abst_prohibition = ∅`, lorsque l'événement *permission_access_to_Internet* est déclenché, la variable `hist_abst_permission` va admettre une nouvelle politique de sécurité `{H ↦ R_Corporate ↦ web ↦ Internet ↦ default}`. Cela n'assure plus l'invariant car la variable `hist_abst_prohibition`

peut également contenir la même politique de sécurité. Pour résoudre ces conflits, il faudra suivre la stratégie de gestion des conflits présentée dans la section 4.7.2 du chapitre 1. Pour cela, les nouvelles machines abstraites ORBAC_EDC_NONCONFLIT_0 et ORBAC_EDC_NONCONFLIT_1 sont crées pour spécifier cette stratégie (voir l'Annexe).

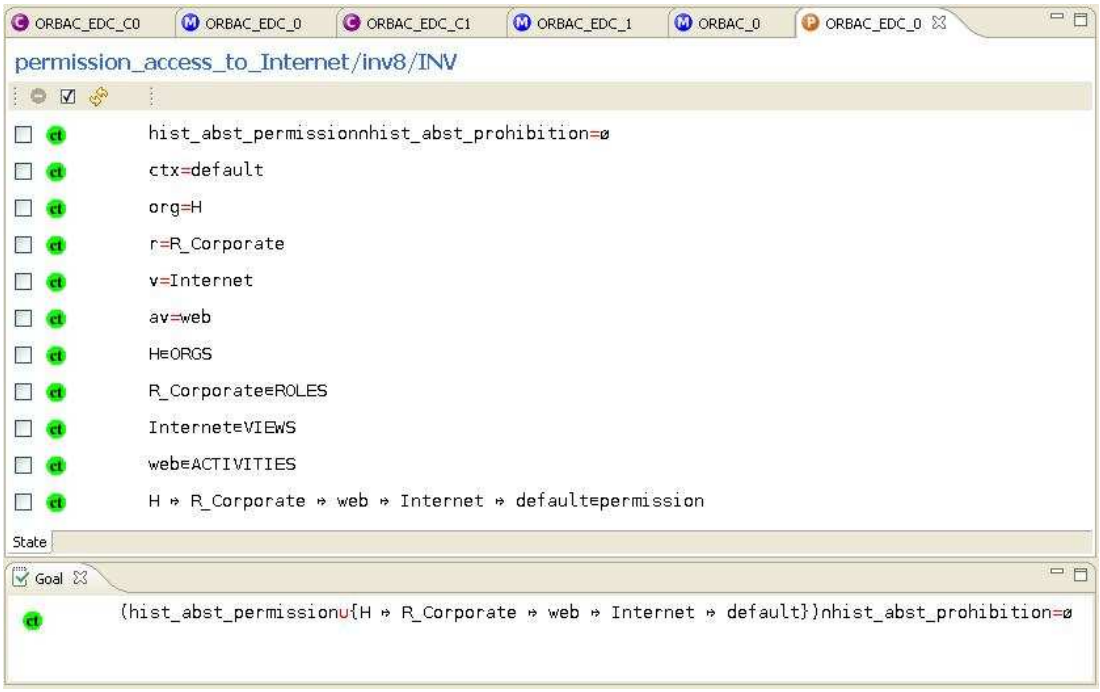


Figure 32 : Obligation de preuve générée sur l'invariant inv8

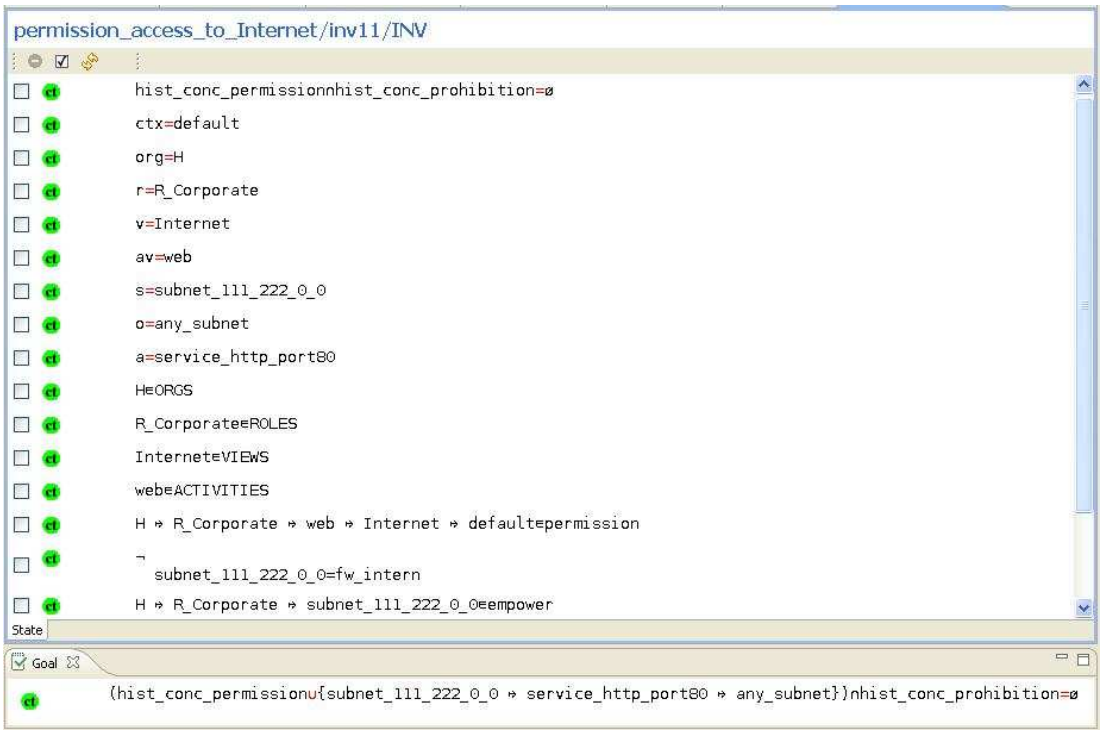


Figure 33 : Obligation de preuve générée sur l'invariant 11

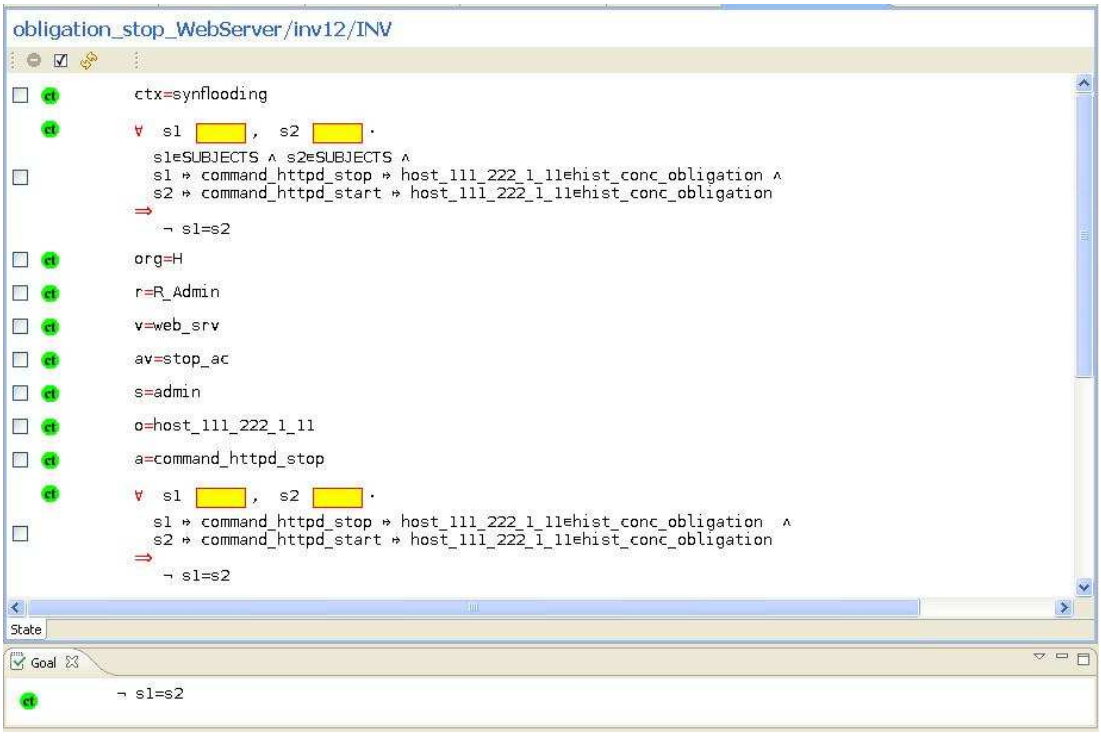


Figure 34 : Obligation de preuve générée sur l'invariant 12

Ensuite, la représentation des politiques de sécurité d'Or-BAC en XML y compris l'élément *context* qui permet de générér des règles de sécurité concrète est indispensable et valable lorsqu'il faut développer un système répondant aux menaces et attaques se basant sur les technologies XML et fonctionnant dans un environnement distribué.

Enfin, pour évaluer la faisabilité du déploiement des politiques de sécurité, nous réalisons un test sur les deux outils *Ensuite* et *cfengine*. Dans ce test, d'abord, nous mesurons le temps du déploiement des politiques de sécurité d'une machine PDP (*Policy Decision Point*) à une machine PEP (*Policy Enforcement Point*) dans le réseau de l'organisation *H*. Ensuite, nous évaluons l'efficacité ou bien la performance du déploiement de plusieurs fichiers des règles de sécurité. Pour ce faire, en premier lieu, nous créons les fichiers des règles de politiques de sécurité avec les tailles différentes dans la machine PDP. En second lieu, ces fichiers seront envoyés à la machine PEP. Enfin, la machine PEP exécutera des règles de poliques de sécurité dans ces fichiers.

Nombre de règles	1000	2000	4000	5000	6000	8000	10000
Taille de fichier	44,8 Kio	90,7 Kio	182,5 Kio	228,4 Kio	274,3 Kio	366,1 Kio	457,9 Kio

Tableau 13: Tailles des fichiers shell script et nombres des règles iptables

Les scénarios de ce test sont décrits en détail dans deux figures 35,36 suivantes :

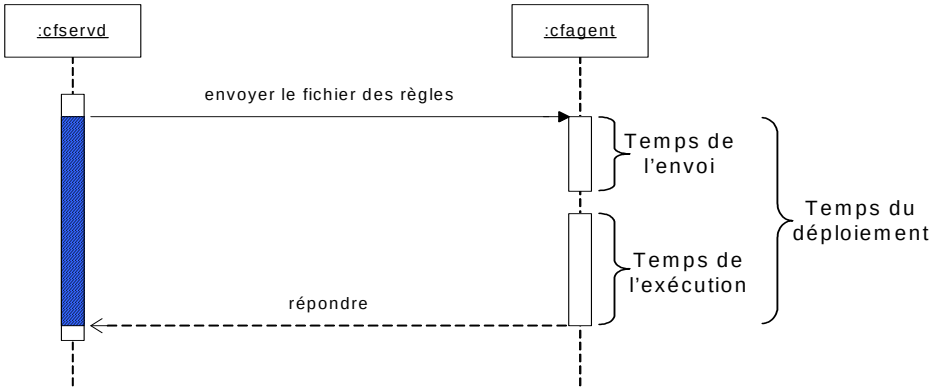


Figure 35 : Scénario du test avec cfengine

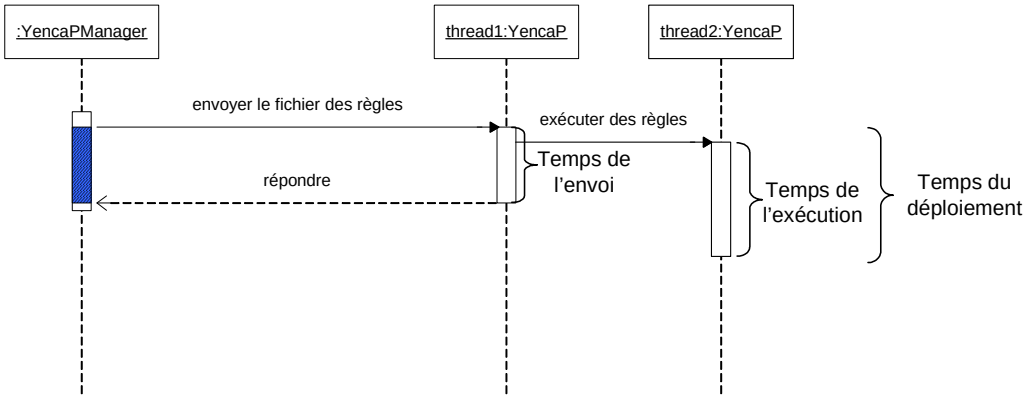


Figure 36 : Scénario du test avec Ensuite

Le temps du déploiement = Le temps de l'envoi + Le temps de l'exécution

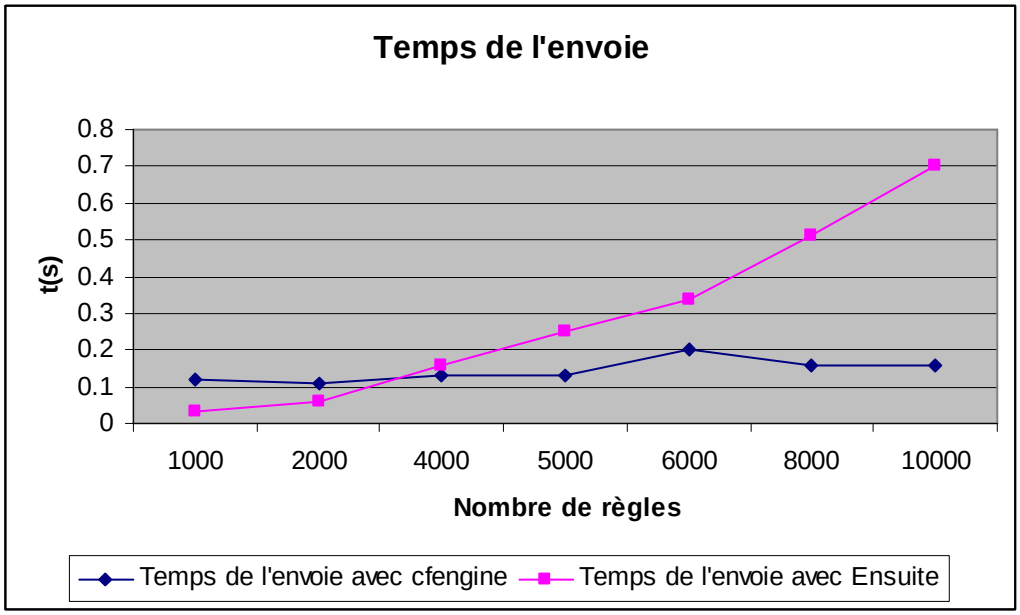


Figure 37: Comparaison de temps de l'envoi entre Ensuite et cfengine

Nombre de règles	1000	2000	4000	5000
Temp d'exécution	12,9 s	87,7 s	666,7 s	1391,1 s

Tableau 14: Temps de l'exécution des règles de sécurité avec cfengine

Selon le test, le temps de l'envoi les fichiers des règles de sécurité en utilisant *Ensuite* est plus long que celui en utilisant *cfengine*. Mais, ce temps de l'envoie est acceptable (0,7 second pour envoyer un fichier de 10000 règles) en vue du nombre de règles de sécurités à déployer. En ce qui concerne le temps de l'exécution, *cfengine* exécute des règles de sécurité en série (c'est-à-dire, après avoir reçu complètement chaque fichier des règles de sécurité, il commence à les exécuter ; tandis que *Ensuite* exécute des règles de sécurité en parallèle (c'est-à-dire, il utilise *multi-threads*, l'un pour recevoir les fichiers de règles et l'autre pour exécuter des règles dans chaque fichier). Par conséquent, le temps du déploiement des règles de sécurité en utilisant *cfengine* est plus long que celui en utilisant *Ensuite*. Pour éclaircir plus cette remarque, dans ce test, nous avons mesuré le temps de l'exécution des règles de sécurité en utilisant *cfengine*, nous trouvons que ce temps est trop long (20 minutes pour exécuter 5000 règles de sécurité) dans l'observation du côté de la machine PDP, et elle doit attendre jusqu'à ce que ce temps complet pour envoyer un autre fichier des règles de sécurité. Par contre, grâce à la technique de programmation *multi-threading*, le temps du déploiement en utilisant *Ensuite* a réduit notablement et semble être égal le temps de l'envoie des règles de sécurité, surtout dans l'observation du côté de la machine PDP, car le temps pour que la machine PDP reçoive la réponse est égal ce temps l'envoie des règles de sécurité et il ne faut pas compter le temps de l'exécution des règles, lorsque la machine PDP continuer à envoyer l'autre fichier des règles de sécurité. Cela montre que l'efficacité ou bien la performance du déploiement des règles de sécurité en utilisant *Ensuite* est meilleure que celle en utilisant *cfengine*, lorsqu'il faut choisir un outil pour déployer plusieurs fichiers des règles de sécurités de PDP vers PEPs dans un environnement distribué tel que le réseau de l'organisation *H*. De plus, *Ensuite* utilise le protocole NETCONF. Comme déjà énoncé, c'est le protocole de configuration de réseau très prometteur basé sur les technologies XML.

Pour compléter cette évaluation, nous tenterons de comparer notre approche avec les deux outils de même type *FirewallBuilder* [http://*www.fwbuilder.org*] et *MotOrBac* [*http://motorbac.sourceforge.net/*]. *FirewallBuilder* est une interface graphique orientée objet permettant de créer des politiques de sécurité pour de nombreuses plate-formes firewall, y compris *iptables*, *ipfilter*, *Cisco Pix*, *Cisco IOS ACL*; et de les déployer vers des firewalls d'un réseau. Dans *FirewallBuilder*, une politique de sécurité est composée d'un ensemble de règles. Chaque règle est un objet abstrait qui représente une entité d'un réseau ou un service. Comme déjà énoncé dans la section 4 du chapitre 2, *FirewallBuilder* manque d'un processus du raisonnement et de la vérification des politiques de sécurité générées (par exemple, vérifier et détecter les conflits entre ces politiques). *MotOrBac* est

une implémentation du modèle Or-BAC permettant de créer des politiques de sécurité abstraites et concrètes d'une organisation et de détecter les conflits entre ces politiques. Une fois des conflits détectés, *MotOrBAC* intègre des stratégies de résolution de conflits pour rétablir la cohérence. Pourtant, *MotOrBac* manque d'un processus qui permet de déployer des politiques de sécurité vers des composants dédiés et non-dédiés à la sécurité d'un réseau. Pour notre approche, bien qu'elle soit encore au stade de prototype et de description, nous avons montré qu'elle est capable de surmonter les inconvénients de deux outils *FirewallBuilder* et *MotOrBAC*. Concrètement, elle permet de modéliser et vérifier des politiques de sécurité en utilsant Event- B, puis de les déployer vers des composants dédiés et non-dédiés à la sécurité d'un réseau en utilisant un protocole de la gestion et la configuration de réseau tel que NETCONF. Actuellement, les grandes firmes dans le domaine du réseau comme Cisco et Juniper sont en train d'intégrer des agents supportant le protocole NETCONF dans leur produits. C'est pourquoi, il suffit de développer un gestionnaire commun pour interfacer avec ces agents différents, lorsqu'on veut transformer cette approche en un logiciel qui est capable de spécifier et déployer des politiques de sécurité vers des composants de *multi-vendeurs* dans un réseau.

7 Conclusion

Dans ce chapitre, nous avons montré comment l'approche proposée a été appliquée au cas du réseau d'une organisation. Pour ce faire, tout d'abord, nous avons réalisé la spécification des politiques de sécurité de cette organisation appliquiées sur son réseau en Event-B, grâce au modèle Or-BAC. Cette spécification permet de vérifier formellement et raisonner des conflits existants entre ces politiques. Ensuite, pour produire des règles de sécurité concrètes, ces politiques de sécurité ont été représentées en XML. Cette représentation est utile lorsqu'il faut déployer ces politiques dans un environnement distribué. Enfin, nous avons mis en oeuvre deux outils *Ensuite* et *cfengine* pour déployer ces règles de sécurtié concrètes vers les composants dédiés et non-dédiés à la sécurité dans le réseau. L'approche que nous avons réalisée sur le réseau peut être améliorée selon l'orientation d'un système répondant automatiquement aux menaces qui comporte quatre modules pricipaux : ACE (*Alert Correlation Engine*), PIE (*Policy Instantiation Engine*), PDP(*Policy Decision Point*) et PEP(*Policy Enforcement Point*) . À cet égard, il faut reconcevoir et reconstruire ce système basé sur un gestionnaire central et unifié qui communique avec des agents installés sur des composants des vendeurs différents.

Conclusion et perspectives

Arrivés au terme de ce mémoire, nous désirons faire le point sur le chemin parcouru. Afin de commencer, les objectifs présentés au début de ce travail ont été atteints, des politiques de sécurité d'Or-BAC sont spécifiées, vérifiées et des règles de sécurités concrètes sont générées avant de les déployer sur les composants dédiés et non-dédiés à la sécurité. Certes, ce processus est encore au stade de prototype et de description, mais nous avons montré sa faisabilité au travers de la spécification, la vérification et le déploiement des politiques de sécurité dans un réseau pratique.

La première partie de ce travail a été consacrée à une présentation des modèles et politiques de sécurité, spécialement, du modèle Or-BAC. En effet, il était utilisé dans la majorité de différentes parties du processus de la modélisation et du déploiement des politiques de sécurité.

Ensuite, nous avons commencé la deuxème partie de ce travail par la modélisation des politiques de sécurité. Pour ce faire, XML, ainsi que de ses technologies connexes (XPath, Xquery et XSLT) ont été présentées pour représenter des politiques de sécurité du modèle Or-BAC et générer des règles de sécurité concrètes. Nous avons déjà utilisé la méthode formelle Event-B dans le processus de spécification, car ses obligations de preuve permettent de vérifier formellement et de raisonner pourquoi des conflits entre des politiques du type *prohibition* et celles du type *permission* et *obligation* sont germés. Le résultat de cette spécification est un ensembles des politiques de sécurité cohérentes et non-conflictuelles.

Le chapitre suivant a été consacré à l'étude de l'architecture d'un système répondant aux menaces et les trois protocoles COPS-PR (*Common Open Policy Service for policy provisioning*), SNMP (*Simple Network Management Protocol*) et NETCONF (NETwork CONFiguration) concernant le processus de déploiement des politiques de sécurité. Les composants de cette architecture qui ont été présentés sont *PIE*, *PDP* et *PEP*. *PIE* est responsable de modéliser des politiques de sécurité abstraites (c'est-à-dire, celles de la formule (*rôle,activité,vue*)) et de les transformer en des politiques de sécurité concrètes (c'est-à-dire, celles du triplet (*sujet,action,objet*)). *PDP* reçoit ces politiques de sécurité concrètes comme input, génère des règles de sécurité concrètes et les déploie aux *PEPs*, grâce à un protocole de communication tel que NETCONF. Dans ce chapitre, nous avons comparé, du point de vue de la théorique, deux protocoles NETCONF et SNMP surtout sur les critères concernant la modélisation de données et le déploiement de configurations. À ces égards, le protocole NETCONF a plus d'avantages que le protocole SNMP.

Le dernier chapitre a été consacré à une étude de cas. C'était un réseau pratique d'une organisation. Dans la première étape, nous avons effectué la spécification formelle des politiques de sécurité appliquées sur ce réseau en Event-B. Grâce à cette spécification et aux obligations de preuve générées, nous avons pu vérifier formellement et raisonner pourquoi des conflits entre des politiques du type *prohibition* et celles du type *permission* et *obligation*

ont été germés. Les conflits qui ont été vérifiés formellement en Event-B sont les contradictions et les incapacités. Une solution de ces conflits a été spécifiée en Event-B. Puis, des politiques de sécurité d'Or-BAC non-conflictuelles sont représentées en langage XML pour produire des règles de sécurité concrètes. Ensuite, nous avons effectué le déploiement des règles de sécurité générées dans l'étape de spécification sur les composants dédiés (firewall *netfilter*) et non-dédiés (serveur *web*) à la sécurité de ce réseau en utilisant deux outils *Ensuite* (une implémentation du protocole NETCONF) et *cfengine* (une implémentation du paradigme *gestionnaire-agent*). Enfin, c'est la partie de l'évaluation de résultats. Le résultat de la spécification en Event-B est des politiques d'Or-BAC non-conflictuelles. Le résultat de la représentation en XML est des règles de sécurité cohérentes et concrètes. Le résultat du déploiement des règles de sécurité concrètes est l'évaluation du temps du déploiement et celle de l'efficacité du déploiement sur deux outils *cfengine* et *Ensuite*. Du point de vue de la pratique, *Ensuite*, une implémentation du protocole NETCONF, a plus d'avantages que *cfengine*, grâce à la capacité de l'exécution concurrente des fichiers de configuration en se basant sur la technique de programmation *multi-threading*. Les résultats obtenus nous permettent de penser au fait de développer un logiciel pour effectuer les processus de la modélisation et du déploiement sur un réseau plus large et complexe comme Internet.

En ce qui concerne les perspectives futures, il manque actuellement d'un logiciel de modéliser et de déployer des politiques de sécurité de A à Z. En effet, l'approche que nous avons présentée est une coordination de plusieurs outils (Event-B et les technologies XML pour spécifier, vérifier formellement et représenter des politiques de sécurité, et surtout *Ensuite* et *cfengine* pour les déployer). Le problème actuel est de réunir ces outils en un seul système. Une solution serait de créer un logiciel unifié basé sur l'architecture du système répondant aux menaces et sur le paradigme *gestionnaire-agent* qui est capable de, à partir des alertes de menace, générer des politiques de réaction Or-BAC, puis des règles de sécurité pour les envoyer plus rapidement possible aux composants du réseau concernés grâce au protocole NETCONF. Il serait aussi capable de spécifier et générer des politiques de sécurité dans le contexte normal pour les déployer aux composants différents du réseau et garantir l'activité correcte de ces composants et ainsi du réseau. De plus, le test de conformité des politiques de sécurité n'est pas encore abordé dans notre travail. Il serait principalement réalisé à l'aide des méthodes formelles comme B et Event-B. Actuellement, c'est un problème qui est attire encore beaucoup de recherches.

En guise de conclusion, nous espérons que cette approche est promise à un bel avenir, car les apports sont réels et les bénéfices nombreux.

Annexes

Machine abstraite ORBAC_1

MACHINE
ORBAC_1
REFINES
ORBAC_0
SEES
ORBAC_C1
VARIABLES
hist_conc_permission
hist_conc_prohibition
hist_conc_obligation
context
INVARIANTS
inv1 : hist_conc_permission ⊆ SUBJECTS × ACTIONS × OBJECTS
inv2 : hist_conc_prohibition ⊆ SUBJECTS × ACTIONS × OBJECTS
inv3 : hist_conc_obligation ⊆ SUBJECTS × ACTIONS × OBJECTS
inv4 : ∀s·∀a·∀o·((s ∈SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s ↦ a ↦ o) ∈ hist_conc_permission) ⇒ (∃org·∃r·∃av·∃v·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ (org ↦ r ↦ s) ∈ empower ∧ (org ↦ v ↦ o) ∈ use ∧ (org ↦ av ↦ a) ∈ consider ∧(org ↦ s ↦ a ↦ o ↦ context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_permission)))
inv5 : ∀s·∀a·∀o·((s ∈SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s ↦ a ↦ o) ∈ hist_conc_prohibition) ⇒ (∃org·∃r·∃av·∃v·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ (org ↦ r ↦ s) ∈ empower ∧ (org ↦ v ↦ o) ∈ use ∧ (org ↦ av ↦ a) ∈ consider ∧(org ↦ s ↦ a ↦ o ↦ context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_prohibition)))
inv6 : ∀s·∀a·∀o·((s ∈SUBJECTS ∧ a ∈ ACTIONS ∧ o ∈ OBJECTS ∧ (s ↦ a ↦ o) ∈ hist_conc_obligation) ⇒ (∃org·∃r·∃av·∃v·(org ∈ ORGS ∧ r ∈ ROLES ∧ av ∈ ACTIVITIES ∧ v ∈ VIEWS ∧ (org ↦ r ↦ s) ∈ empower ∧ (org ↦ v ↦ o) ∈ use ∧ (org ↦ av ↦ a) ∈ consider ∧(org ↦ s ↦ a ↦ o ↦ context) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ context) ∈ hist_abst_obligation)))
inv7 : context ∈ CONTEXTS

EVENTS
INITIALISATION =
BEGIN
act1 : hist_conc_permission := ∅
act2 : hist_conc_prohibition := ∅
act3 : hist_conc_obligation := ∅
act4 : context := default
END
permission_action =
REFINES
permission_action
ANY
s,a,o,org,r,v,av
WHERE
grd1 : s ∈ SUBJECTS

grd2 : a ∈ ACTIONS
grd3 : o ∈ OBJECTS
grd4 : org ∈ ORGS
grd5 : r ∈ ROLES
grd6 : av ∈ ACTIVITIES
grd7 : v ∈ VIEWS
grd8 : (org↦r↦s) ∈ empower
grd9 : (org↦v↦o) ∈ use
grd10 : (org↦av↦a) ∈ consider
grd11 : (org ↦ s ↦ a ↦ o ↦ context) ∈ hold
grd12 : (org ↦ r ↦ av ↦ v ↦ context) ∈ permission
grd13 : ∀ri·∀avi·∀vi·((ri ∈ ROLES ∧ avi ∈ ACTIVITIES ∧ vi ∈ VIEWS ∧ (org↦ri↦s) ∈ empower ∧ (org↦vi↦o) ∈ use ∧ (org↦avi↦a) ∈ consider) ⇒ ((org↦ri↦avi↦vi↦context) ∉ prohibition))
THEN
act1: hist_conc_permission := hist_conc_permission ∪ {(s↦a↦o)}
END
prohibition_action =
REFINES
prohibition_action
ANY
s,a,o,org,r,v,av
WHERE
grd1 : s ∈ SUBJECTS
grd2 : a ∈ ACTIONS
grd3 : o ∈ OBJECTS
grd4 : org ∈ ORGS
grd5 : r ∈ ROLES
grd6 : av ∈ ACTIVITIES
grd7 : v ∈ VIEWS
grd8 : (org↦r↦s) ∈ empower
grd9 : (org↦v↦o) ∈ use
grd10 : (org↦av↦a) ∈ consider
grd11 : (org ↦ s ↦ a ↦ o ↦ context) ∈ hold
grd12 : (org ↦ r ↦ av ↦ v ↦ context) ∈ prohibition
grd13 : ∀ri·∀avi·∀vi·((ri ∈ ROLES ∧ avi ∈ ACTIVITIES ∧ vi ∈ VIEWS ∧ (org↦ri↦s) ∈ empower ∧ (org↦vi↦o) ∈ use ∧ (org↦avi↦a) ∈ consider) ⇒ ((org↦ri↦avi↦vi↦context) ∉ permission) ∧ (org↦ri↦avi↦vi↦context) ∉ obligation))
THEN
act1: hist_conc_prohibition := hist_conc_prohibition ∪ {(s↦a↦o)}
END
obligation_action =
REFINES
obligation_action
ANY
s,a,o,org,r,v,av
WHERE
grd1 : s ∈ SUBJECTS
grd2 : a ∈ ACTIONS
grd3 : o ∈ OBJECTS
grd4 : org ∈ ORGS
grd5 : r ∈ ROLES
grd6 : av ∈ ACTIVITIES
grd7 : v ∈ VIEWS

grd8 : (org↦r↦s) ∈ empower

grd9 : (org↦v↦o) ∈ use

grd10 : (org↦av↦a) ∈ consider

grd11 : (org ↦ s ↦ a ↦ o ↦ context) ∈ hold

grd12 : (org ↦ r ↦ av ↦ v ↦ context) ∈ obligation

grd13 : ∀ri·∀avi·∀vi·((ri ∈ ROLES ∧ avi ∈ ACTIVITIES ∧ vi ∈ VIEWS ∧ (org↦ri↦s) ∈ empower ∧ (org↦vi↦o) ∈ use ∧ (org↦avi↦a) ∈ consider) ⇒ ((org↦ri↦avi↦vi↦context) ∉ prohibition))

THEN

act1: hist_conc_obligation := hist_conc_obligation ∪ {(s↦a↦o)}

END

END

Machine abstraite ORBAC_EDC_NONCONFLIT_0

MACHINE

ORBAC_EDC_NONCONFLIT_0

SEES

ORBAC_EDC_C0

VARIABLES

ctx // Context Variable

hist_abst_permission

hist_abst_prohibition

hist_abst_obligation

level_p // Level of permission policy

level_pr // Level of prohibition policy

level_o // Level of obligation policy

INVARIANTS

inv1 : ctx ∈ CONTEXTS

inv2 : hist_abst_permission ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS

inv3 : hist_abst_permission ⊆ permission

inv4 : hist_abst_prohibition ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS

inv5 : hist_abst_prohibition ⊆ prohibition

inv6 : hist_abst_obligation ⊆ ORGS × ROLES × ACTIVITIES × VIEWS × CONTEXTS

inv7 : hist_abst_obligation ⊆ obligation

inv8 : level_p ∈ ℕ

inv9 : level_pr ∈ ℕ

inv10 : level_o ∈ ℕ

inv11 : level_p ≠ level_pr // Invariant to verify conflict between permission and prohibition policy

inv12 : level_o ≠ level_pr // Invariant to verify conflict between obligation and prohibition policy

EVENTS

INITIALISATION ≙

BEGIN

act1 : ctx≔ default

act2 : hist_abst_permission ≔ ∅

act3 : hist_abst_prohibition ≔ ∅

act4 : hist_abst_obligation ≔ ∅

act5 : level_p ≔ 2

act6 : level_pr ≔ 1

act7 : level_o ≔ 0

END

set_context_value ≙

ANY

c

WHERE

grd1 : c ∈ CONTEXTS

THEN

act1 : ctx ≔ c

END

permission_access_to_Internet ≙

ANY

org,r,v,av,lv

WHERE

grd1 : org ∈ ORGS ∧ org = H

grd2 : r ∈ ROLES ∧ r = R_Corporate

grd3 : v ∈ VIEWS ∧ v = Internet

grd4 : av ∈ ACTIVITIES ∧ av = web

grd5 : ctx = default

grd6 : (org ↦ r ↦ av ↦ v ↦ ctx) ∈ permission

grd7 : lv ∈ ℕ ∧ lv ≠ level_pr

THEN

act1 : hist_abst_permission ≔ hist_abst_permission ∪{(org ↦ r ↦ av ↦ v ↦ ctx)}

act2 : level_p ≔ lv

END

prohibition_access_to_WebServer ≙

ANY

org,r,v,av,lv

WHERE

grd1 : org ∈ ORGS ∧ org = H

grd2 : r ∈ ROLES ∧ r = R_Internet

grd3 : v ∈ VIEWS ∧ v = web_srv

grd4 : av ∈ ACTIVITIES ∧ av = web

grd5 : ctx = synflooding

grd6 : (org ↦ r ↦ av ↦ v ↦ ctx) ∈ prohibition

grd7 : lv ∈ ℕ ∧ lv ≠ level_p ∧ lv ≠ level_o

THEN

act1 : hist_abst_prohibition ≔ hist_abst_prohibition ∪{(org ↦ r ↦ av ↦ v ↦ ctx)}

act2 : level_pr ≔ lv

END

obligation_stop_WebServer ≙

ANY

org,r,v,av,lv
WHERE
grd1 : org ∈ ORGS ∧ org = H
grd2 : r ∈ ROLES ∧ r = R_Admin
grd3 : v ∈ VIEWS ∧ v = web_srv
grd4 : av ∈ ACTIVITIES ∧ av = stop_ac
grd5 : ctx = synflooding
grd6 : (org ↦ r ↦ av ↦ v ↦ ctx) ∈ obligation
grd7 : lv ∈ ℕ ∧ lv ≠ level_pr
THEN
act1 : hist_abst_obligation ≔ hist_abst_obligation ∪{(org ↦ r ↦ av ↦ v ↦ ctx)}
act2 : level_o ≔ lv
END
END

Machine abstraite ORBAC_EDC_NONCONFLIT_1

MACHINE
ORB_EDC_NONCONFLIT_1
REFINES
ORBAC_EDC_NONCONFLIT_0
SEES
ORBAC_EDC_C1
VARIABLES
ctx
s
a
o
hist_conc_permission
hist_conc_prohibition
hist_conc_obligation
level_p // Level of permission policy
level_pr // Level of prohibition policy
level_o // Leve of obligation policy
INVARIANTS
inv1 : ctx ∈ CONTEXTS
inv2 : hist_conc_permission ⊆ SUBJECTS × ACTIONS × OBJECTS
inv3 : ∀sub·∀ac·∀obj·((sub ∈ SUBJECTS ∧ ac ∈ ACTIONS ∧ obj ∈ OBJECTS ∧ (sub ↦ ac ↦ obj) ∈ hist_conc_permission) ⇒ (∃org·∃r·∃av·∃v·∃c·(org∈ORGS ∧ r∈ROLES ∧ av∈ACTIVITIES ∧ v∈VIEWS ∧ c∈CONTEXTS ∧ (org ↦ r ↦ sub)∈empower ∧ (org ↦ v ↦ obj)∈use ∧ (org ↦ av ↦ ac)∈consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_permission)))
inv4 : s ∈ SUBJECTS
inv5 : o ∈ OBJECTS
inv6 : a ∈ ACTIONS
inv7 : hist_conc_prohibition ⊆ SUBJECTS × ACTIONS × OBJECTS
inv8 : hist_conc_obligation ⊆ SUBJECTS × ACTIONS × OBJECTS
inv9 : ∀sub·∀ac·∀obj·((sub ∈ SUBJECTS ∧ ac ∈ ACTIONS ∧

obj ∈ OBJECTS ∧ (sub ↦ ac ↦ obj) ∈ hist_conc_prohibtion) ⇒ (∃org·∃r·∃av·∃v·∃c·(org∈ORGS ∧ r∈ROLES ∧ av∈ACTIVITIES ∧ v∈VIEWS ∧ c∈CONTEXTS ∧ (org ↦ r ↦ sub)∈empower ∧ (org ↦ v ↦ obj)∈use ∧ (org ↦ av ↦ ac)∈consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_prohibition)))

inv10 : ∀sub·∀ac·∀obj·((sub ∈ SUBJECTS ∧ ac ∈ ACTIONS ∧ obj ∈ OBJECTS ∧ (sub ↦ ac ↦ obj) ∈ hist_conc_obligation) ⇒ (∃org·∃r·∃av·∃v·∃c·(org∈ORGS ∧ r∈ROLES ∧ av∈ACTIVITIES ∧ v∈VIEWS ∧ c∈CONTEXTS ∧ (org ↦ r ↦ sub)∈empower ∧ (org ↦ v ↦ obj)∈use ∧ (org ↦ av ↦ ac)∈consider ∧ (org ↦ sub ↦ ac ↦ obj ↦ c) ∈ hold ∧ (org ↦ r ↦ av ↦ v ↦ c) ∈ hist_abst_obligation)))

inv11 : level_p ∈ ℕ

inv12 : level_pr ∈ ℕ

inv13 : level_o ∈ ℕ

inv14 : level_p ≠ level_pr // Invariant to verify conflict between permission and prohibition policy

inv15 : level_o ≠ level_pr // Invariant to verify conflict between obligation and prohibition policy

EVENTS

INITIALISATION ≙

BEGIN

act1 : ctx ≔ default

act2 : hist_conc_permission ≔ ∅

act3 : s ≔ null1

act4 : o ≔ null3

act5 : a ≔ null2

act6 : hist_conc_prohibition ≔ ∅

act7 : hist_conc_obligation ≔ ∅

act8 : level_p ≔ 2

act9 : level_pr ≔ 1

act10 : level_o ≔ 0

END

set_context_value ≙

REFINES

set_context_value

ANY

c

WHERE

grd1 : c ∈ CONTEXTS

THEN

act1 : ctx ≔ c

END

permission_access_to_Internet ≙

REFINES

permission_access_to_Internet
ANY
org,r,v,av,lv
WHERE
grd1 : org ∈ ORGS ∧ org = H
grd2 : r ∈ ROLES ∧ r = R_Corporate
grd3 : v ∈ VIEWS ∧ v = Internet
grd4 : av ∈ ACTIVITIES ∧ av = web
grd5 : ctx = default
grd6 : (org ↦ r ↦ av ↦ v ↦ ctx) ∈ permission
grd7 : s = subnet_111_222_0_0 ∧ s ≠ fw_intern ∧ (org↦r↦s) ∈ empower
grd8 : o = any_subnet ∧ (org↦v↦o) ∈ use
grd9 : a = service_http_port80 ∧ (org↦av↦a) ∈ consider
grd10 : (org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
grd11 : lv ∈ ℕ ∧ lv ≠ level_pr
THEN
act1: hist_conc_permission ≔ hist_conc_permission∪{(s↦a↦o)}
act2 : level_p ≔ lv
END

prohibition_access_to_WebServer ≙
REFINES
prohibition_access_to_WebServer
ANY
org,r,v,av,lv
WHERE
grd1 : org ∈ ORGS ∧ org = H
grd2 : r ∈ ROLES ∧ r = R_Internet
grd3 : v ∈ VIEWS ∧ v = web_srv
grd4 : av ∈ ACTIVITIES ∧ av = web
grd5 : ctx = synflooding
grd6 : (org ↦ r ↦ av ↦ v ↦ ctx) ∈ prohibition
grd7 : s = subnet_any ∧ s ≠ fw_intern ∧ s ≠ fw_extern ∧ s ≠ subnet_111_222_0_0 ∧ (org↦r↦s) ∈ empower
grd8 : o = host_111_222_1_11 ∧ (org↦v↦o) ∈ use
grd9 : a = service_http_port80 ∧ (org↦av↦a) ∈ consider
grd10 : (org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
grd11 : lv ∈ ℕ ∧ lv ≠ level_p ∧ lv ≠ level_o
THEN
act1: hist_conc_prohibition≔hist_conc_prohibition∪{(s↦a↦o)}
act2 : level_pr ≔ lv
END

obligation_stop_WebServer ≙
REFINES
obligation_stop_WebServer
ANY
org,r,v,av,lv
WHERE

```
grd1   :    org ∈ ORGS ∧ org = H
grd2   :    r ∈ ROLES ∧ r = R_Admin
grd3   :    v ∈ VIEWS ∧ v = web_srv
grd4   :    av ∈ ACTIVITIES ∧ av = stop_ac
grd5   :    ctx = synflooding
grd6   :    (org ↦ r ↦ av ↦ v ↦ ctx) ∈ obligation
grd7   :    s = admin ∧ (org↦r↦s) ∈ empower
grd8   :    o = host_111_222_1_11 ∧ (org↦v↦o) ∈ use
grd9   :    a = command_httpd_stop ∧ (org↦av↦a) ∈ consider
grd10   :    (org ↦ s ↦ a ↦ o ↦ ctx) ∈ hold
grd11   :    lv ∈ ℕ ∧ lv ≠ level_pr
THEN
act1: hist_conc_obligation ≔ hist_conc_obligation∪{(s↦a↦ o)}
act2   :    level_o ≔ lv
END
END
```

Fichier *cfagent.conf*

```
###
#
# START cfagent.conf
####
import:
  # Global policy: which cfengine server or clients should include?
  # "any" is a special word, it means any server that reads this
cfagent.conf
  # will have this configuration file included.
  any::
    cfagent.global.conf
###
#
# END cfagent.conf
#
###
```

Fichier *cfagent.global.conf*

```
###
# cfagent.global.conf: included by cfagent.conf
# Control: define the variables for the cfagent service
###
control:
 actionsequence = ( copy editfiles shellcommands )
         domain = ( localdomain )
         access = ( root )
         smtpserver = ( mail.telecom-bretagne.eu )
         sysadm = ( HuuThan.VU@telecom-bretagne.eu )
         configroot = ( /var/cfengine/masterfiles/config )
         moduleroot = ( /var/cfengine/masterfiles/modules )
         schedule = ( Min20_25 )
         ChecksumUpdates = ( on )
         DefaultCopyType = ( checksum )
         policyhost = ( debian0 )
```

```
         timezone = ( MET CET )
         Syslog = ( on )

groups:
  cf_firewalls = ( debian2 )
  cf_servers = ( debian1 )

editfiles:
         { /var/spool/cron/crontabs/root
             AutoCreate
             AppendIfNoSuchLine "# make sure cfexecd runs hourly
from cron too"
             AppendIfNoSuchLine "0,15,30,45 * * *
*/usr/local/sbin/cfexecd -F"
         }

copy:
  cf_firewalls::
        ${configroot}/netfilter/rules.fw dest=/tmp/rules.fw
        server=${policyhost} type=sum owner=root group=root
mode=0644
  cf_servers::
        ${configroot}/server/rules.serv dest=/tmp/rules.serv
        server=${policyhost} type=sum owner=root group=root
mode=0644

shellcommands:
  cf_firewalls::
      "/bin/sh /tmp/rules.fw"
  cf_servers::
      "/bin/sh /tmp/rules.serv"
```

Fichier *cfservd.conf*

```
######
#
# BEGIN cfservd.conf
#
#######

control:
        domain = ( localdomain )
        sysadm = ( "HuuThan.VU@telecom-bretagne.eu" )
        workdir = ( "/var/cfengine" )
        bindir = ( "/usr/local/sbin" )
        cfrunCommand = ( "${bindir}/cfagent" )
        IfElapsed = ( 1 )
        MaxConnections = ( 50 )
        AllowUsers = ( root )
        TrustKeysFrom = ( 111.222.0.0/16 )
        LogAllConnections = ( on )
        masterfiles = ( "/var/cfengine/masterfiles" )

# This is grant read access to newly added files under
# directories that have not yet been defined. Only add
# the directory name & not the actual file.

admit: # or grant?
any::
        $(cfrunCommand)              *
        $(masterfiles)/inputs        *
```

```
        $(masterfiles)/modules       *
        $(masterfiles)/bin           *
        $(masterfiles)/config        *
######
#
# END cfservd.conf
#
#######
```

Fichier *update.conf*

```
###
#
# BEGIN cf.update
#
###

#############################################################################
###
#
# This script distributes the configuration, a simple file so that,
# if there are syntax errors in the main config, we can still
# distribute a correct configuration to the machines afterwards,
even
# though the main config won't parse. It is read and run just before
the
# main configuration is parsed.
#
#############################################################################
###

control:

   actionsequence  = ( copy processes tidy )  # Keep this simple and
constant

   domain          = ( localdomain )  # Needed for remote copy

   #
   # Which host/dir is the master for configuration roll-outs?
   #

   policyhost      = ( debian0.localdomain )
   master_cfinput  = ( /var/cfengine/masterfiles/inputs )

   AddInstallable = ( new_cfenvd new_cfservd )

   #
   # Some convenient variables
   #

   workdir         = ( /var/cfengine )

  solaris::

   cf_install_dir  = ( /iu/nexus/local/sbin )

  linux::

   cf_install_dir  = ( /usr/local/sbin )
```

```
#############################################################################
   #
   # Spread the load, make sure the servers get done first though
   #

#############################################################################

  !AllBinaryServers::

     SplayTime = ( 1 )

#############################################################################
########

 #
 # Make sure there is a local copy of the configuration and
 # the most important binaries in case we have no connectivity
 # e.g. for mobile stations or during DOS attacks
 #

copy:

     $(master_cfinput)            dest=$(workdir)/inputs
                                  r=inf
                                  mode=700
                                  type=binary
                                  exclude=*.lst
                                  exclude=*~
                                  exclude=#*
                                  server=$(policyhost)
                                  trustkey=true

 !quetzalcoatal::

     $(cf_install_dir)/cfagent    dest=$(workdir)/bin/cfagent
                                  mode=755
                                  backup=false
                                  type=checksum

     $(cf_install_dir)/cfservd    dest=$(workdir)/bin/cfservd
                                  mode=755
                                  backup=false
                                  type=checksum
                                  define=new_cfservd

     $(cf_install_dir)/cfexecd    dest=$(workdir)/bin/cfexecd
                                  mode=755
                                  backup=false
                                  type=checksum

     $(cf_install_dir)/cfenvd     dest=$(workdir)/bin/cfenvd
                                  mode=755
                                  backup=false
                                  type=checksum
                                  define=new_cfenvd

#############################################################################
#
```

```
tidy:

     #
     # Cfexecd stores output in this directory.
     # Make sure we don't build up files and choke on our own words!
     #

     $(workdir)/outputs pattern=* age=7

#############################################################################
#

processes:

  new_cfservd::

    "cfservd" signal=term restart /var/cfengine/bin/cfservd

  new_cfenvd::

    "cfenvd" signal=kill restart "/var/cfengine/bin/cfenvd -H"


###
#
# END cf.update
#
###
```

Fichier *cfrun.hosts*

```
#
# This is the host list for cfrun
# cfrun.hosts
# Only these hosts will be contacted by remote connection
#

domain=localdomain
access=root
outputdir = /var/cfengine/outputs
debian0
debian1
debian2
```

Fichier *rules.fw*

```
echo "executing update iptables ..."

iptables -N Internet-Web-To_Web_SRV
iptables -A FORWARD -p tcp –dport 80 -j Internet-Web-To_Web_SRV
iptables -A Internet-Web-To_Web_SRV –s 111.222.1.0/24 -j RETURN
iptables -A Internet-Web-To_Web_SRV –d 111.222.2.0/32 -j RETURN
iptables -A Internet-Web-To_Web_SRV –d 111.222.3.0/32 -j RETURN
iptables -A Internet-Web-To_Web_SRV –d 111.222.4.0/32 -j RETURN
iptables -A Internet-Web-To_Web_SRV -j DROP

exit 0
```

Fichier *rules.serv*

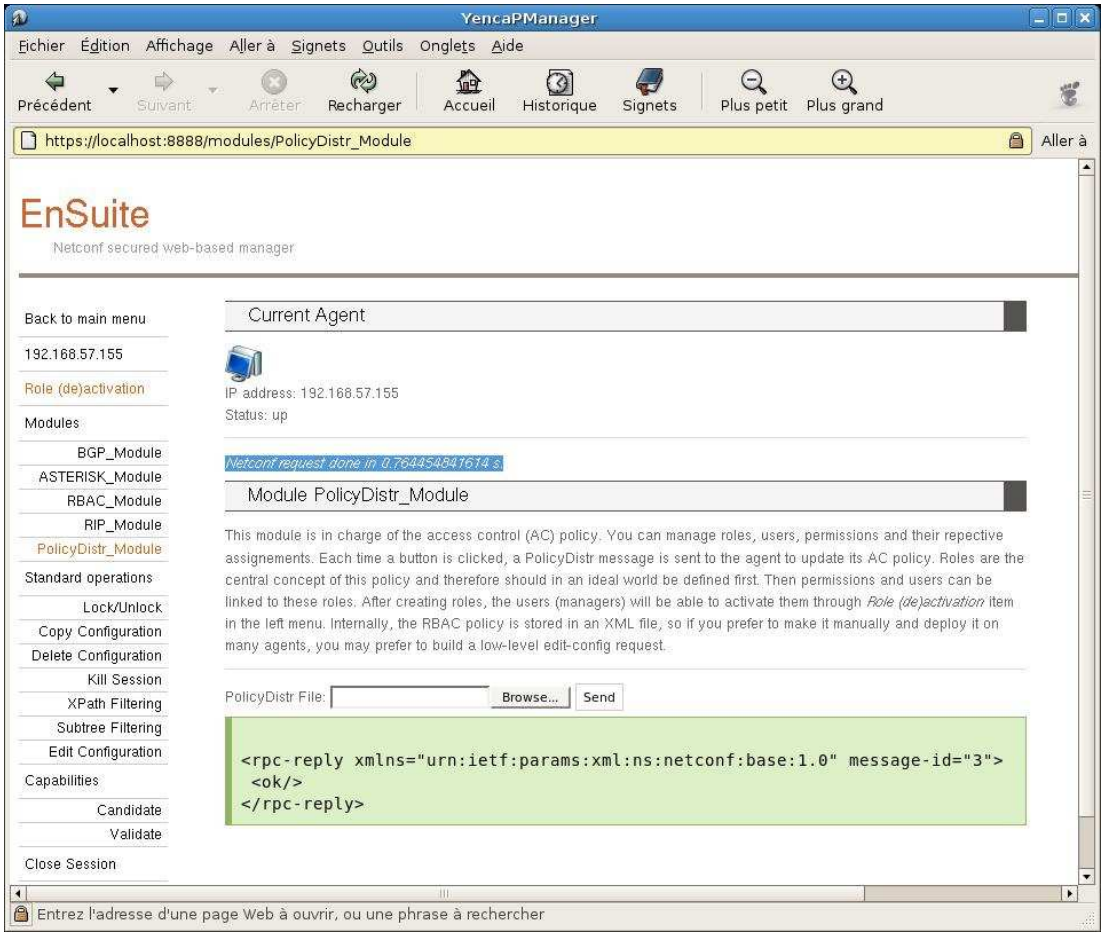
```
#!/bin/sh
#rules.serv
echo "execute stop httpd service ..."

/etc/httpd stop

exit 0
```

Fichier *PolicyDistr_Module.py*

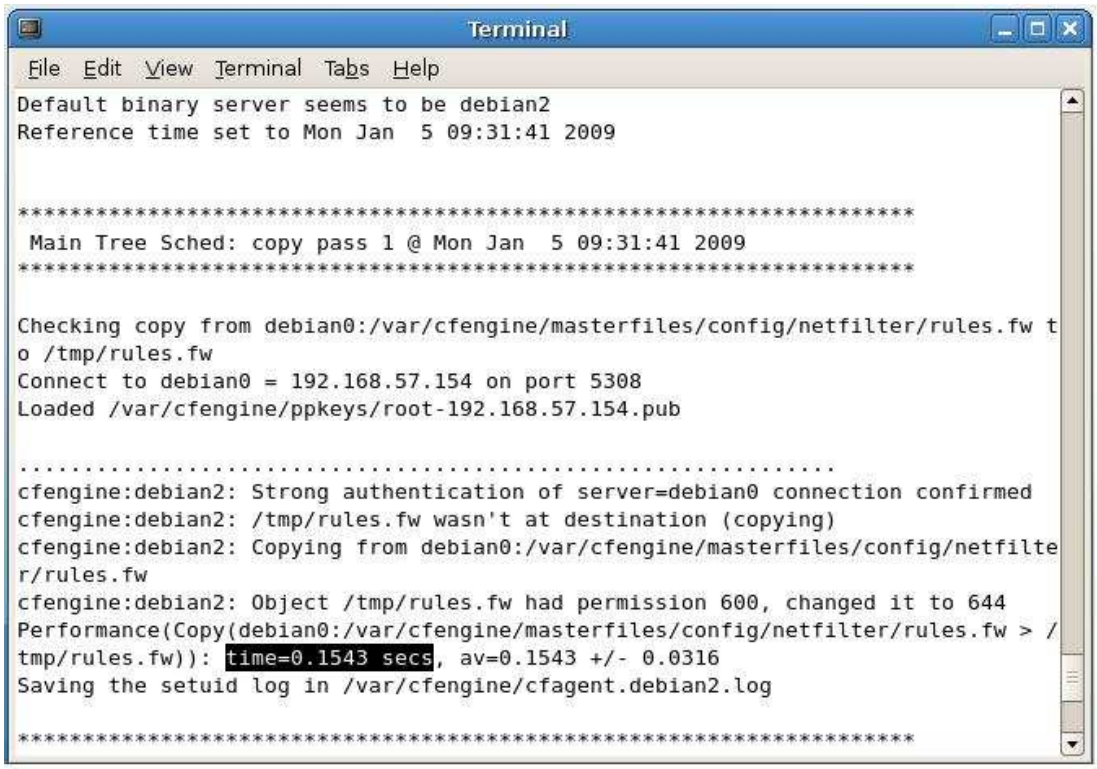
```
import os, string, threading

from Ft.Xml.Domlette import NonvalidatingReader, implementation,
PrettyPrint
from Ft.Xml import XPath, EMPTY_NAMESPACE
from xml.dom import Node
from Ft.Xml.XPath import Evaluate, Compile
from Ft.Xml.XPath.Context import Context

from Modules.modulereply import ModuleReply
from Modules.module import Module
from Modules.easyModule import EasyModule

from constants import C
import util


class PolicyDistr_Module(Module):

            PolicyDistrFile = '/etc/ensuite/yencap/rules.sh'

            def __init__(self, name, path, namespace, cacheLifetime,
parameters):
                        Module.__init__(self, name, path, namespace,
cacheLifetime)

            def getConfig(self):
                        modulereply =
ModuleReply(replynode=self.doc.documentElement)
                        return modulereply

            def editConfig(self, defaultoperation, testoption,
erroroption, target, configNode, targetnode=None):
                        confStr =
util.convertNodeToString(configNode)
                        confbsn1 = string.replace(confStr,'\$','$')
                        confbsn =
string.replace(confbsn1,'\\n','\n')
                        confoKl = confbsn.lstrip('<PolicyDistr
xmlns="urn:loria:madynes:ensuite:yencap:module:NetFilter:1.0">')
                        confoK =
string.replace(confoKl,'\n</PolicyDistr>','')

                        fileOrBAC = open(self.PolicyDistrFile,'w')
                        fileOrBAC.write(confoK)
                        fileOrBAC.close()
```

```
                        #Multithreading Program
                        t1=threading.Thread(target =
self.shell_execute)
                        t1.start();

                        modulereply = ModuleReply()
                        return modulereply

            def shell_execute(self):
                        command = "sh %s" % (self.PolicyDistrFile)
                        os.system(command)
```

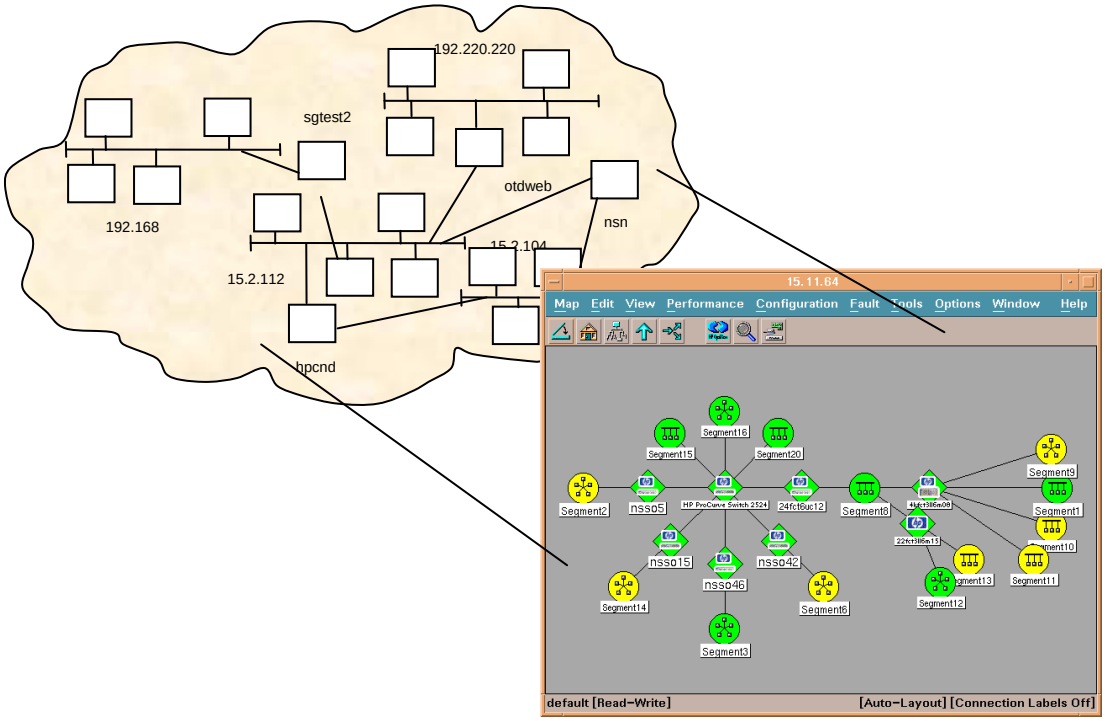
Interface de YencaP Manager



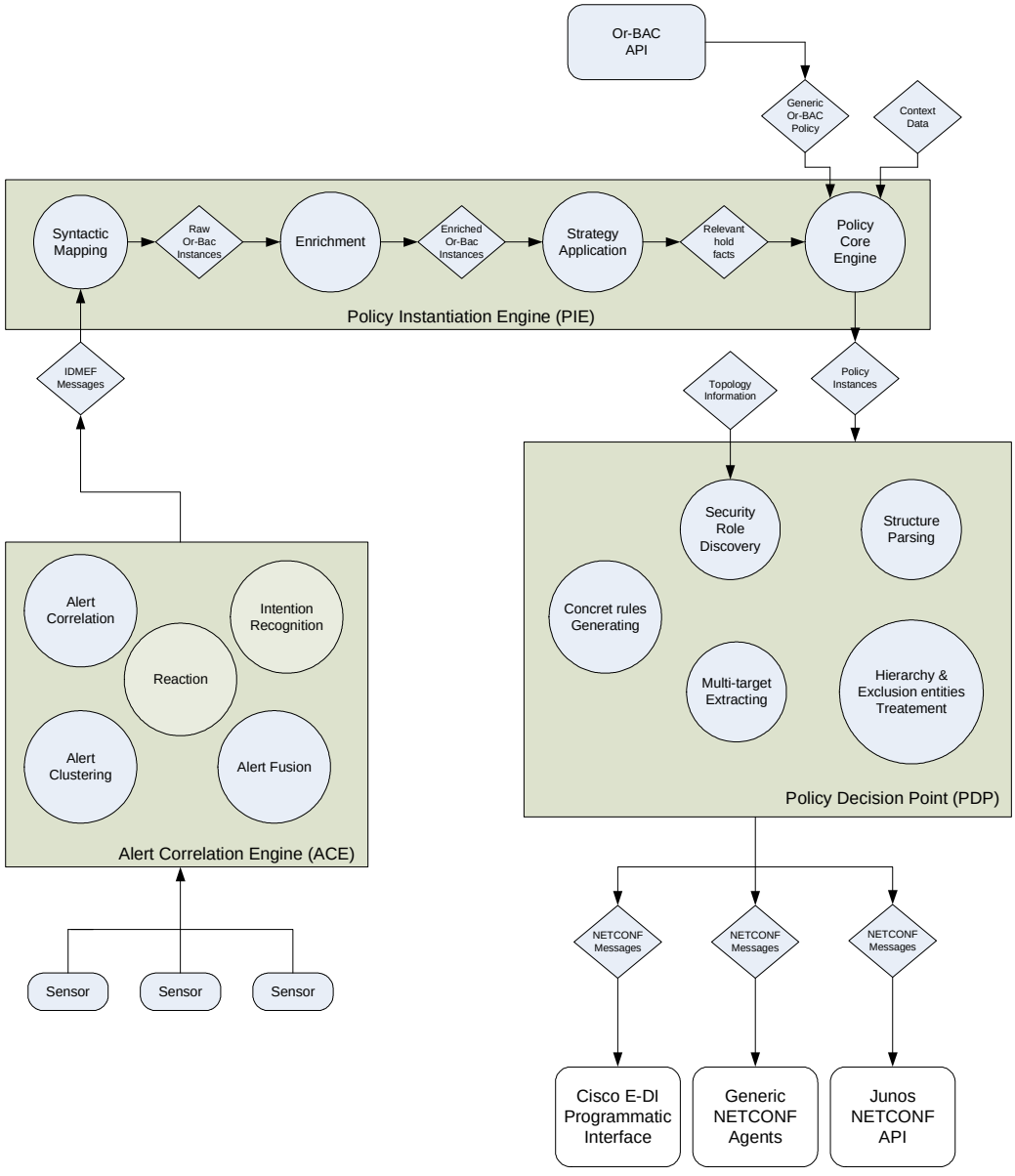
Interface de cfengine



Interface de HP OpenView Network Node Manger



Le système répondant aux menaces complet, unifié dans le futur



Bibliographie

[1] Yusuf Bhaiji - CCIE No. 9305, *CCIE Professional Development Series Network Security Technologies and Solutions* , Cisco Press, March 19, 2008.

[2] Frédéric Cuppens, *Le modèle Or-BAC,* cours FR403b @ ENST-BRETAGNE.

[3] David F.Ferraiolo, D.Richard Kuhn, Ramaswamy Chandramouli. *Role-Based Access Control,* Artech House inc, 2003.

[4] Frédéric Cuppens, Nora Cuppens-Boulahia, Alexandre A Miège. Inheritance hierachies in the Or-BAC model and their applications in a network environment. Journées SSTIC, Rennes, France, Juin 2004.

[5] Frédéric Cuppens, Alexandre A Miège. *Modelling Contexts in the Or-BAC Model,* 19th Applied Computer Security Associated Conference (ACSAC 2003), Las Vegas, Nevada, USA, Décembre 8-12, 2003.

[6] Frédéric Cuppens, Nora Cuppens-Boulahia, Thierry Sans, Alexandre A Miège. *Formal approach to specify and deploy a network security policy*, Second Workshop on Formal Aspect in Security and Trust (FAST), Toulouse France, 2004.

[7] Hevré Débar, Yohann Thomas, Frédéric Cuppens, Nora Cuppens-Boulahia. *Enabling Automated Threat Reponse through the Use of a Dynamic Security Policy*, Journal in Computer Virology, 3, 2007.

[8] Kurland, V. *Firewall Builder. White paper,* 2003.

[9] Christian Jacquenet, Gilles Bourdon, Mohamed Boucadair, *Service Automation and Dynamic Provisioning Techniques in IP MPLS Environments*, John Wiley and Sons**,** Apr 2008.

[10] Theo Ferreira Franco and all, *Substituting COPS-PR An Evaluation of NETCONF and SOAP for Policy Provisioning*, Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06), IEEE 2006.

[11] Mi-Jung Choi, Hyoun-Mi Choi, James W. Hong, Hong-Taek Ju, *XML-Based Configuration Management for IP Network Devices*, IEEE Communications Magazine, July 2004.

[12] *http://ensuite.sourceforge.net/yencap_devel_doc.html*.

[13] S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. Garcia-Alfaro, L. Toutain, *Reliable Process for Security Policy Deployment*, International Conference on Security and Cryptography (Secrypt 2007), Barcelona, Spain, July 2007.

[14] http://orbac.org/index.php?page=motorbac&lang=fr.

[15] F. Cuppens, Alexandre A Miège, *Alert correlation in a cooperative intrusion detection framework*, IEEE Symposium on Research in Security and Privacy, Oakland, May 2002.

[16] Information Technologie Security Evaluation Criteria, European Comunities, juin 1991.

[17] J. Mclean, *The specification and modeling of computer security, Computer*, 23(1): 9-16, January 1990.

[18] Elliote Rusty Harold et W. Scott Means : *XML in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[19] Allen Brown, Matthew Fuchs, Jonathan Robie et Philip Wadler : MSL A model for W3C XML Schema. *In WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 191–200, New York, NY, USA, 2001. ACM.

[20] Clearsy, *Manuel de Référence du langage B*, Version 1.8.6, 2007.

[21] Steve Schneider, *The B-Method an introduction*, Palgrave Macmillan 2001.

[22] Jean Bendisposto, Michael Leuschel, Oliver Ligot, Mireille Samia, *La validation de modèles Event-B avec le plug-in ProB pour RODIN*, AFADL 2007, pages 1065 à 1084.

[23] Nazim Benaïssa, Dominique Cansell, and Dominique Méry, *Integration of Security Policy into System Modeling*, 7th International Conference of B Users, Besançon, France, January 2007.

[24] B. Wijnen and U. Blumenthal, *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP).* STD 62, http://www.ietf.org/rfc/rfc3415.txt, December 2002.

[25] HP Company, *Distributed Systems Administration Utilities Guide d'utilisation*, T2786-90270, Edition 1.3, Mars 2008.

[26] V. Cridlig, R. State, O. Festor, An integrated security framework for XML based management, IFIP/IEEE International Symposium on Integrated Network Management, 2005.

[27] Butler W. Lampson, *Protection*, In 5th Princeton Symposium on Information Science and Systems, pages 437-443, 1971. Reprinted in ACM Operating Systems Review 8(1): 18-24, 1974.

[28] G. S. Graham and P. J. Denning, *Protection- principles and practice*, In AFIPS Press, editor, Proc. Spring Jt. Computer Conference, volume 40, pages 417- 429, Montvale, N.J., 1972.

[29] M.H. Harrison, W. L. Ruzzo and J. D. Ullman, *Protection in operating systems*, Communications of the ACM,19(8), pages 461-471, 1976.

[30] James Clark, *XSL Transformations (XSLT) Version 1.0*, W3C recommendation, W3C, novembre 1999.

[31] http://en.wikipedia.org/wiki/Basic_Encoding_Rules.

[32] Douglas Mauro, Kevin Schmidt, *Essential SNMP*, O'Reilly 2001.