| | |
|---|---|
| **Titre:** Title: | Mécanismes de sécurisation pour environnements répartis sur des grappes interopérables |
| **Auteur:** Author: | Alain Patrick Medenou |
| **Date:** | 2003 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Medenou, A. P. (2003). Mécanismes de sécurisation pour environnements répartis sur des grappes interopérables [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7291/ |

UNIVERSITÉ DE MONTRÉAL

# MÉCANISMES DE SÉCURISATION POUR ENVIRONNEMENTS RÉPARTIS SUR DES GRAPPES INTEROPÉRABLES

ALAIN PATRICK MEDENOU
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)

SEPTEMBRE 2003

*Your file Votre référence*
*ISBN: 0-612-90847-X*
*Our file Notre référence*
*ISBN: 0-612-90847-X*

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

# MÉCANISMES DE SÉCURISATION POUR ENVIRONNEMENTS RÉPARTIS SUR DES GRAPPES INTEROPÉRABLES

est présenté par : Alain Patrick MEDENOU

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

Jury d'examen composé de :

M. DAGENAIS Michel, Ph.D., Président

M. PIERRE Samuel, Ph.D., Directeur de recherche et Membre

M. QUINTERO Alejandro, Ph.D., Membre

*À mes parents*

# REMERCIEMENTS

Mes remerciements vont à l'endroit de mon directeur de recherche, le Professeur Samuel Pierre pour son soutien et son encadrement tout au long de ma maîtrise. Je remercie aussi Makan Pourzandi de Ericsson Recherche Canada pour la validation de mes implémentations et Alejandro Quintero du Laboratoire de Recherche en Réseautique et Informatique Mobile (LARIM) pour ses conseils.

Ma gratitude va également à l'endroit de tous mes collègues du projet DSI (Distributed Security Infrastructure) à Ericsson Recherche Canada pour leur collaboration enrichissante et complémentaire à la réalisation de mes travaux. Je voudrais aussi remercier mes collègues du LARIM pour leurs critiques constructives très appréciées lors de nos échanges sur mon sujet de recherche.

Je ne saurais oublier de dire merci à mes parents et mes frères qui m'ont accompagné de leurs prières et encouragements. Je rends grâce à Dieu pour sa bénédiction et son assistance sous toutes formes lors de ce parcours.

Merci enfin à toutes les personnes sympathiques qui, par leur fraternité, amitié, et soutien ont contribué de façon insoupçonnée à l'achèvement de mes études de premier et deuxième cycle universitaire. Je me garde de les nommer ici de peur de ne pas être exhaustif.

# RÉSUMÉ

Le développement fulgurant de l'Internet et des systèmes répartis en général cette dernière décennie, s'est accompagné de nombreux problèmes dont celui de la sécurité. Les approches de sécurisation basées sur des périmètres topologiques satisfont de moins en moins les besoins de protection des systèmes comme les grappes d'ordinateurs; il devient impératif de songer à mettre en place, au sein même des systèmes, des mécanismes cohérents de sécurité. D'autre part, des combinaisons complexes d'outils de sécurisation sont utilisées de façon inefficiente, parfois au détriment de la performance des systèmes protégés.

Le présent mémoire fait partie des travaux qui se penchent sur ces questions. Nous y faisons un état de la situation en présentant des approches de sécurisation actuellement utilisées ou proposées dans la littérature. Le mémoire aborde la question de la définition de la politique de sécurité : elle a valeur de loi régissant les systèmes répartis à protéger. Plus spécifiquement, nous proposons un langage de description de politique de sécurité baptisé XDSPL, qui est basé sur XML, et qui permet d'exprimer des règles de sécurité couvrant le contrôle des accès, la confidentialité et l'intégrité. Nous décrivons un environnement mettant en jeu un Serveur de Sécurité répliqué communiquant avec des Gestionnaires de Sécurité sur tous les autres nœuds, via un canal sécurisé interopérable, afin de déployer la politique de sécurité et la mettre en application. Les mécanismes de sécurisation proposés touchent les applications, les processus et les sockets, ce qui représente une bonne granularité. Par ailleurs, dans le souci de sécuriser de façon efficiente, nous proposons une architecture étroitement liée à l'environnement de déploiement de la politique de sécurité, afin de gérer la qualité de protection (QoP) : concrètement, elle s'assure d'offrir différents niveaux de sécurisation disponibles, sur demande, lorsque les ressources du système le permettent.

Nous avons implémenté l'interpréteur de la politique de sécurité écrite dans le langage XDSPL, le canal sécurisé de communication, le module (DSM) pour la mise en

application des règles de sécurité ainsi que le module de gestion de la qualité de protection.

Pour évaluer le langage XDSPL, nous avons vérifié s'il respecte ou non dix critères que nous avons définis en nous inspirant de standards XML. Nous avons ensuite vérifié la fonctionnalité de la mise en œuvre de la politique de sécurité par un jeu d'appels systèmes de sockets entre un client et un serveur. En termes de performance, le module de gestion de qualité de protection induit une surcharge en temps système de l'ordre de quelques microsecondes; ce module n'est sollicité que lors de l'application des règles de sécurité ayant trait à la confidentialité et l'intégrité. Notre évaluation de performance a également porté sur la surcharge en temps système que génère l'utilisation du module de mise en application des règles de sécurité. Pour divers appels système, nous avons comparé nos pourcentages de surcharge avec ceux d'une approche similaire de la littérature (SELinux). Nos résultats demeurent satisfaisants malgré le pourcentage élevé de surcharge pour les communications TCP. Nos travaux n'étant pas exempts de limites, nous les exposons, tout en proposant de nouvelles avenues de recherche.

# ABSTRACT

The fulgurating development of the Internet and distributed systems in general, this last decade, was accompanied by many problems of which that of security. The security approaches based on topological perimeters no more satisfy the needs for protection of systems such as computers' clusters; it becomes imperative to define coherent security frameworks within those systems. In addition, complex combinations of security tools are used in an inefficient way that leads to performance degradation of the protected systems.

This thesis takes part in the works that leans on those questions. We present a state of the art in the field of security in distributed systems by presenting approaches of security currently used or suggested in the literature. This thesis tackles the question of the definition of the security policy: it has value of law in the distributed systems to be protected. More specifically, we propose a language of description of security policy named XDSPL which is based on XML, and which makes it possible to express security rules covering access control, confidentiality and integrity. We describe an environment that involves a replicated Security Server communicating with Security Managers on all the other nodes, via an interoperable secure channel: they work together in order to deploy the security policy and apply it. The proposed security mechanisms act on applications, processes and sockets, which represent a good granularity in the control. In addition, with the aim of offering an efficient way to secure, we propose an architecture closely related to the environment of deployment of the security policy: it manages the quality of protection (QoP) by concretely offering various security levels on request, when the system's resources allow it.

We implement the parser of the security policy written in XDSPL language, the secure communication channel. We also present the module (DSM) that implements the security checks as well as the quality of protection management module.

To evaluate XDSPL language, we check if it respects or not ten criteria that are inspired by some XML standards. We then evaluate the functionality of the security policy checks with a set of sockets system calls between a client and a server. In terms of performance, the management module of quality of protection induces about a few microseconds of overhead; this module is only called for security checks that are related to confidentiality and integrity. Our performance evaluation also relates to the overhead generated by the use of the module that applies security rules in the operating system kernel. For various system calls, we compared our overhead percentages with those of a similar approach of the literature (SELinux). Our results remain satisfactory in spite of the high overhead percentage for TCP communications. Our work not being free from limits, we expose them, while proposing new research tracks.

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# LISTE DES SIGLES ET ABRÉVIATIONS

**AH** : Authentication Header
**DAC** : Discretionary Access Control
**DSM** : Distributed Security Module
**DSP** : Distributed Security Policy
**EACL** : Extended Access Control List
**ESP** : Encrypted Security Payload
**GS** : Gestionnaire de Sécurité
**IDL** : Interface Definition Language
**LSM** : Linux Security Module
**MAC** : Mandatory Access Control
**NSA** : National Security Agency
**OMG** : Object Management Group
**QoP** : Quality of Protection
**QoSS** : Quality of Security Service
**RBAC** : Role Based Access Control
**RFC** : Requests for Comments
**ScID** : Security Context Identification
**SnID** : Security Node Identification
**SS** : Serveur de Sécurité
**SSL** : Secure Socket Layer
**TLS** : Transport Layer Security
**XACL** : eXtensible Access Control Language
**XACML** : eXtensible Access Control Markup Language
**XDSPL** : eXtensible Distributed Security Policy Language

# CHAPITRE 1

# INTRODUCTION

L'utilisation des réseaux d'ordinateurs s'est avérée au fil des années une nécessité; aujourd'hui, il est devenu obsolète d'envisager un environnement de travail constitué d'ordinateurs et autres ressources matérielles informatiques qui soient physiquement et logiquement isolés les uns des autres. Le partage de ressources logicielles et matérielles ainsi que la communication élargie sont les principales motivations de la mise en réseau des machines. Les systèmes répartis qui naissent de ces environnements doivent relever plusieurs défis. L'un des défis majeurs auxquels sont confrontés ces systèmes dits *répartis* est la sécurité. Le plus populaire des systèmes répartis qu'est le vaste réseau Internet est sujet à des attaques de tout genre et d'envergures considérables. La traditionnelle réponse des organisations est d'installer des pare-feux qui sont des murailles de protection tout autour de leurs systèmes; cette approche définit des périmètres de sécurité rigides pendant que peu d'efforts sont effectués pour assurer la sécurité à l'intérieur même de ces réseaux. Plusieurs efforts sont faits pour améliorer la sécurité des systèmes à l'interne par la proposition d'architectures et d'environnements fiables et cohérents ainsi que le développement de mécanismes et d'outils de sécurisation. Le présent mémoire se place dans ce contexte. Cette introduction précisera tout d'abord certains concepts et terminologies clefs; elle présentera par la suite les éléments de problématique avant d'énoncer les objectifs de recherche visés. La méthodologie à adopter sera présentée, suivie par une annonce du plan du mémoire.

## 1.1 Définitions et concepts de base

Une *attaque* est une tentative pour outrepasser la politique de sécurité qui régit un système. La *politique de sécurité* est l'expression des privilèges qu'un administrateur

ou une organisation accorde aux entités constituantes de son environnement; c'est elle qui indique explicitement comment, contre quoi et contre qui un système est protégé. La protection d'un système passe donc par la définition claire d'une politique qui peut prendre en considération des paramètres sociaux, techniques, juridiques et organisationnels.

Dans les environnements répartis, le problème de la sécurité est souvent abordé en termes de services. Les services de sécurité ont pour but d'améliorer la sécurité des systèmes informatiques par des mécanismes spécifiques. Les services de sécurité les plus usuels sont :

- *Le service de confidentialité* : pour s'assurer que les informations ne sont dévoilées qu'aux entités autorisées ;
- *Le service d'intégrité* : pour empêcher qu'une entité non habilitée modifie une information ;
- *Le service d'audit* : pour l'analyse du journal des opérations effectuées dans le système afin d'y détecter des attaques éventuelles ;
- *Le service d'authentification* : pour s'assurer de prouver qu'une entité est bien celle qu'elle prétend être ;
- *Le service de contrôle des accès* : responsable des vérifications d'accès aux ressources du système. Il vérifie que seules les entités autorisées ont accès aux ressources.

De façon similaire au concept de qualité de service dans les réseaux, la *qualité de protection* – QoP (ou *qualité de service de sécurité* - QoSS) permet d'offrir des niveaux de sécurité variables selon les besoins d'un usager ou d'une application.

## 1.2 Éléments de problématique

La sécurité dans un système passe d'abord par la définition d'une politique cohérente. La définition ou la représentation des politiques de sécurité n'est pas toujours une tâche aisée. Plusieurs propositions ont été faites en la matière: c'est l'exemple de

KeyNote (RFC 2704), EACL [5] et bien d'autres qui utilisent des formats de représentation spécifiques. Ces solutions ne signalent pas de mécanismes de protection de l'information que représente la politique de sécurité. Par ailleurs, les syntaxes utilisées ne sont pas très flexibles et faciles d'extension. De plus, elles ne permettent pas d'exprimer des règles de contrôle de sécurité avec une granularité assez fine.

Ces insuffisances ont amené d'autres auteurs à proposer des modèles basés sur l'utilisation de métalangages comme XML qui offrent déjà des mécanismes de sécurisation d'information ainsi qu'une structuration poussée et une grande flexibilité. C'est le cas de la proposition de Nathan N. Vuong, Geoffrey S. Smith et Yi Deng [6], de XACML (*eXtensible Access Control Markup Language*) et de XACL (*eXtensible Access Control Language*). Il s'agit essentiellement de langages pour exprimer des règles de contrôle des accès; mais une politique de sécurité devrait permettre d'exprimer davantage. En particulier, le XACML est un langage de contrôle d'accès à de l'information sur Internet. Le XACL, quant à lui, sert à décrire les droits d'accès aux informations contenues dans un fichier XML : c'est donc un outil de protection de l'accès à des méta données et non à des ressources d'un système informatique réparti. La proposition des auteurs Nathan N. Vuong, Geoffrey S. Smith et Yi Deng [6] est spécifiquement basée sur le modèle RBAC (*Role Based Access Control*) dans lequel un usager ne peut accéder qu'aux ressources auxquelles le rôle qu'il occupe dans une organisation lui donne droit. Bien que ce modèle puisse être généralisé et adapté, il oblige d'abord à traduire les traditionnelles politiques dans le modèle RBAC.

La sécurité a longtemps été considérée comme une variable binaire : un système est sécurisé ou ne l'est pas. Mais de plus en plus, la tendance est d'offrir des services de sécurité dont le niveau est réglable, tout en demeurant cohérent avec la politique de sécurité qui a été définie, d'où l'émergence de la notion de Qualité de Protection (QoP) ou Qualité de Service de Sécurité (QoSS). La sécurité a en effet un coût en terme d'utilisation des ressources d'un système; par exemple, un utilisateur peut requérir plus ou moins de sévérité dans la sécurité, selon le niveau de sensibilité des informations qu'il échange lors d'une communication ou des opérations qu'il accomplit dans le

système. Par exemple, dans un environnement mobile, le besoin d'offrir des services de sécurité réglables peut se justifier par les capacités limitées des terminaux utilisés; un Palm n'a, par exemple, pas les mêmes capacités de calculs cryptographiques qu'un puissant ordinateur de bureau.

Parmi les propositions faites en matière de qualité de protection figure l'architecture ASE-COM [16] qui définit des classes de sécurité dans un contexte de commerce électronique; mais elle utilise certaines heuristiques de classification, qui dégradent considérablement le niveau de sécurité dans le cas des réseaux à trafic élevé. De nombreuses publications ont été faites par C. Irvine et T. Levin ([15], [17], [18]). Ces derniers ont défini des prémices et un cadre de travail pour l'application du concept de Qualité de Service de Sécurité. Leurs travaux peuvent être considérés comme une base intéressante pour proposer une architecture générique de gestion de qualité de protection dans un système réparti sur des grappes (*cluster*).

## 1.3 Objectifs de la recherche

Ce mémoire vise à contribuer à la sécurisation des systèmes répartis notamment en matière de **politique de sécurité** et de **qualité de protection**. Plus précisément, nous visons les objectifs ci-après :

- Proposer un langage (syntaxe et grammaire) pour définir la politique de sécurité dans un système réparti sur grappes; nous envisageons aussi de proposer un mécanisme de diffusion des règles de la politique de sécurité à tous les nœuds. Le langage proposé devra être flexible, facile de compréhension et d'administration et tenir compte d'autres services de sécurité en plus du contrôle des accès.
- Proposer et décrire une architecture de gestion de la qualité de protection dans les systèmes répartis sur grappes interopérables.

- Implémenter un interpréteur pour le format de représentation de politique de sécurité précédemment proposée, afin d'extraire les informations d'intérêt pour les besoins des applications ou systèmes d'exploitation.
- Implémenter pour les communications par sockets des mécanismes de diffusion et de mise en application effective des règles de politique de sécurité interprétées, afin de valider l'utilisation concrète du langage proposé.
- Proposer une implémentation simplifiée de l'instance principale de gestion de la qualité de protection dans l'architecture proposée.
- Enfin, évaluer notre format de représentation de politique de sécurité, puis présenter quelques résultats de fonctionnalité et de performance des mécanismes d'application des règles de politique de sécurité et de l'instance principale de gestion de la qualité de protection.

## 1.4 Plan du mémoire

Le présent mémoire est structuré en cinq chapitres. La présente introduction en constitue le premier chapitre. Elle sera suivie par le chapitre 2 qui fait une revue de la littérature sur la sécurité dans les systèmes répartis abordée selon plusieurs points de vue. Dans le chapitre 3, nous présenterons l'architecture, les outils et les mécanismes de sécurisation que nous proposons. Le chapitre 4 présentera l'implémentation de certains mécanismes et composants architecturaux proposés. Une étude des mécanismes implémentés sera effectuée, en tentant d'évaluer leur fonctionnalité et/ou impact sur le système. Le chapitre 5, quant à lui, conclura le mémoire en présentant une synthèse des travaux et de leurs limitations, assortie d'indications de recherches futures.

# CHAPITRE 2

# LA SÉCURITÉ DANS LES ENVIRONNEMENTS RÉPARTIS INTEROPÉRABLES

Le présent chapitre décrit plusieurs aspects et travaux en matière de sécurité dans les environnements répartis. Nous proposons tout d'abord une introduction aux environnements répartis interopérables puis un survol du cas particulier des grappes d'ordinateurs. Nous présentons ensuite quelques outils et mécanismes de sécurisation des systèmes répartis, du protocole IPSec jusqu'aux formes évoluées de pare-feux. Nous abordons par la suite la question des langages de représentation de politiques de sécurité. Enfin, dans la dernière section du chapitre, nous nous intéressons à la problématique de la qualité de protection (ou qualité du service de sécurité).

## 2.1 Environnements répartis interopérables

Un système réparti est un réseau dans lequel des composantes logicielles et matérielles communiquent et cordonnent leurs actions par des échanges de messages. Parmi les composantes logicielles, les intergiciels (ou *middleware*) ont pour but d'assurer l'interopérabilité en masquant les hétérogénéités. Dans cette section, nous présentons les systèmes répartis de façon générale et décrivons un exemple d'intergiciel.

### 2.1.1 Généralités sur les systèmes répartis

La motivation du déploiement des systèmes répartis est essentiellement le partage de ressources, le terme ressource étant une abstraction pour désigner autant les composantes matérielles (disques, imprimantes, etc.) que logicielles (fichiers, bases de données, programmes, etc.). L'Internet est le système réparti le plus connu. Pour

déployer de tels systèmes, les concepteurs doivent faire face à plusieurs défis importants [1] :

- L'*hétérogénéité* : il réfère à la capacité du système de prendre en considération et de gérer la diversité des réseaux interconnectés, des machines, des systèmes d'exploitation, des langages de programmation, des différences d'implémentations, etc. ;
- L'*ouverture du système* (*openness*) : le système réparti doit être extensible et permettre l'ajout et le partage aisé de nouvelles ressources. Ceci passe par la publication des interfaces des composants, l'établissement de protocoles uniformes et publiés pour accéder à ces interfaces. Il s'agit d'un défi de taille car il n'est pas aisé d'intégrer les composants implémentés par différents programmeurs ;
- La *sécurité* : cette dimension touche non seulement le système matériel et logiciel mais aussi les usagers dans leur intimité. Elle revêt donc un caractère tout particulier et peut être vu sous trois angles [1] : la confidentialité (protection contre l'accès au contenu par des personnes non autorisées), l'intégrité (protection contre l'altération ou la corruption) et la disponibilité (protection contre les troubles d'accès) ;
- L'*évolutivité* : un système réparti dont les performances sont peu ou pas affectées suite à l'ajout graduel de nouveaux composants ou ressources est dit évolutif. Aujourd'hui, le système d'adressage utilisé par la version 4 du protocole IP s'avère inefficient face à la croissance exponentielle des usagers de l'Internet. Il s'agit là d'un problème d'évolutivité ;
- La *tolérance aux fautes* : les composants du système réparti doivent pouvoir continuer à communiquer ou accomplir leur tâches lorsque l'un d'eux tombe en panne. Plusieurs mécanismes sont utilisés pour cela : le masquage des fautes, la redondance, le recouvrement, etc. ;
- La *concurrence* : les services et applications fournissent des ressources auxquelles les clients dans les systèmes répartis peuvent vouloir accéder

simultanément. Dans un tel environnement, les opérations d'accès à une ressource doivent donc être synchronisées de façon à garantir leur consistance. Ceci est souvent réalisé par des mécanismes tels que les sémaphores utilisés dans la plupart des systèmes d'exploitation ;

- La *transparence* : elle permet aux usagers de voir le système réparti comme un ensemble homogène et non un assemblage de composants indépendants. Il s'agit essentiellement de masquer certains aspects du système aux utilisateurs. Ces derniers n'ont par exemple pas besoin de connaître les mécanismes de réplication qui permettent d'accéder à certaines ressources en cas de pannes.

Dans les systèmes répartis, on distingue les couches de services logicielles et matérielles (Cf. Figure 2.1) suivantes :

- La *plateforme* : elle est constituée des ordinateurs, matériels de réseaux et du système d'exploitation; ces couches de bas niveau fournissent des services aux couches de haut niveau. Intel x86/Windows, Sun SPARC/SunOS, Intel x86 Solaris, PowerPC/MacOS, Intel x86/Linux sont les exemples les plus connus.
- L'*intergiciel (middleware)* : c'est la couche qui a pour objectif de masquer l'hétérogénéité des plates-formes en fournissant un modèle de programmation à utiliser par les programmeurs. Sun RPC, CORBA, Java RMI sont des exemples d'intergiciels.

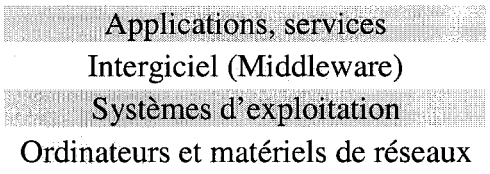



**Figure 2.1 Couches de services logicielles et matérielles dans les systèmes répartis**

### 2.1.2 Des intergiciels ou middleware : l'exemple de CORBA

Parmi les intergiciels les plus utilisés figure CORBA (Common Object Request Broker Architecture) issu du groupe OMG (Object Management Group). CORBA

permet de définir pour tout objet une interface IDL (Interface Definition Langage) qui est une forme générique des opérations et méthodes qui peuvent être invoquées sur lui; ceci permet de faire abstraction de la réelle implémentation car il existe des compilateurs adaptés capables de faire correspondre à cette interface leur équivalent dans un langage de programmation standard (C, C++, Smalltalk, etc.) D'autre part, il existe une couche ORB (Object Request Broker) qui gère les requêtes (dynamiques ou statiques) à un objet, et ce, de façon uniforme. Inspiré de [11], nous proposons ci-après une description du modèle de référence de CORBA.

Le schéma global du modèle de référence de CORBA est présenté à la Figure 2.2 [11]. Les composantes essentielles de cette structure sont: le servant, le client, l'ORB, l'interface de l'ORB, les stubs IDL et skeletons IDL, le compilateur IDL, le DII, le DSI, l'adaptateur d'objets, l'archive d'interfaces et l'archive d'implémentations.

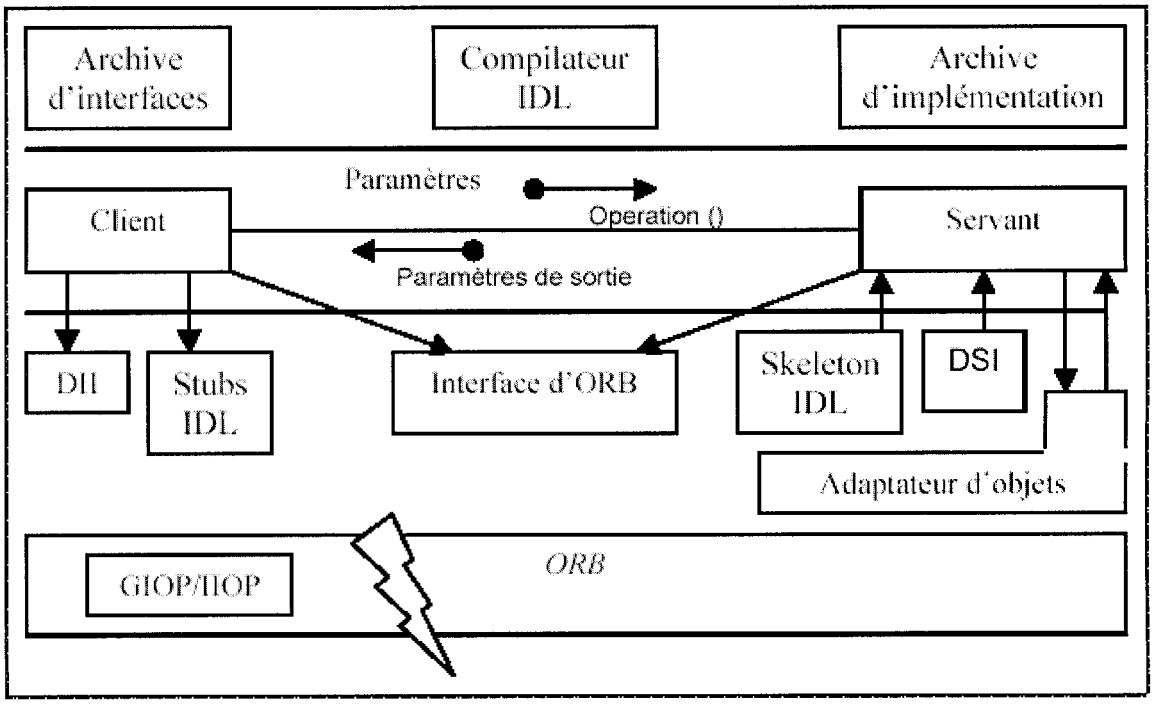



**Figure 2.2 Modèle de référence de CORBA**

- Le **servant** représente le corps d'un objet CORBA, c'est-à-dire l'implémentation dans un langage informatique de ses méthodes auparavant définies en IDL. Une référence unique identifie chaque servant ;
- Le **client** est celui qui invoque des opérations sur l'objet dont la référence est connue. Il peut également récupérer les résultats de cette invocation ;
- L'**ORB** est l'élément de base des communications CORBA car c'est lui qui permet de localiser un objet, de lui livrer les requêtes clients et leur retourner les réponses au cas où elles existent ;
- **L'interface de l'ORB** permet de distinguer les applications des détails d'implémentation des structures de l'ORB. Elle permet entre autres d'initier et de suspendre l'ORB, de convertir les références d'objets ou IOR (*Interoperable Object Reference*) en chaînes de caractères et inversement (processus appelé *marshalling*) et de mettre en forme des requêtes dynamiques et leurs paramètres à travers l'interface dynamique DII (Dynamic Invocation Interface) ;
- Les **stubs** et **skeletons IDL** *(Interface Definition Language)* sont des programmes qui séparent les clients et servants de l'ORB. Un *stub* joue le rôle d'interface statique capable d'assigner aux données de l'utilisateur, des représentations communes au niveau des paquets véhiculés sur le réseau. Le *skeleton,* quant à lui, reformate adéquatement les données reçues avant de les remettre aux servants relatifs à l'application serveur ;
- Le **compilateur IDL** permet non seulement d'assurer la transparente de la correspondance entre les interfaces IDL et les langages informatiques usuels mais élimine aussi des erreurs classiques en programmation réseau, ce qui optimise la compilation ;
- Le **DII** (*Dynamic Invocation Interface*) permet à un client d'invoquer des opérations sur des objets répartis dont l'interface n'était pas connue préalablement lors de la compilation ;
- Le **DSI** (*Dynamic Skeleton Interface*) est pour le serveur ce que le DII est pour le client ;

- L'**adaptateur d'objets** (*Object Adapter*) assure la médiation entre les objets CORBA et leurs implémentations en langages quelconques ou servants; il se charge d'activer un objet inerte, d'associer un servant à un ORB, de démultiplexer les requêtes qui lui sont destinées et d'initier les opérations requises ;
- L'**archive d'interfaces** (*Interface Repository*) stocke de façon dynamique les informations relatives aux interfaces IDL et celles correspondant aux interfaces ORB ;
- L'**archive d'implémentations** (*Implementation Repository*) stocke des informations qui permettent à l'ORB de localiser et d'activer les servants ainsi que d'autres détails sur ces derniers (contrôle administratif, allocation de ressources, sécurité, etc.).

L'existence d'une référence unique à un objet désigné serveur permet à deux objets répartis d'interagir. À chaque objet correspond une référence unique ou IOR (*Interoperable Object Reference*) retournée par un adaptateur d'objets. Cette référence est inconnue des applications et contribue donc à l'amélioration de l'indépendance des opérations vis-à-vis des langages, des protocoles et des plates-formes : c'est ce qui permet à CORBA de relever le défi de l'interopérabilité.

CORBA connaît un grand succès technologique dans le monde des développeurs dans les environnements répartis. DCOM est la réplique de Microsoft à CORBA; il exige cependant un protocole unique pour l'implantation des ORBs. Ces intergiciels sont déployés dans plusieurs systèmes répartis donc les grappes ou *clusters.*

## 2.2 Les grappes d'ordinateurs

Une grappe d'ordinateurs ou *cluster* est un exemple de système réparti. C'est un regroupement de machines (ordinateurs, composants de stockage de données, interconnexions redondantes, etc.) mises en réseau et qui se comportent comme un

système unique pour assurer de la haute disponibilité et du partage de charge [10]. Cette section présente quelques généralités des grappes et certaines de leurs spécificités par rapport aux traditionnels systèmes répartis.

### 2.2.1 Généralités sur les grappes

L'usage évident des grappes est de coordonner la réalisation de tâches complexes par un groupe d'ordinateurs, ces derniers étant rassemblés comme une machine de performance élevée utilisant souvent le parallélisme de tâches. C'est l'exemple des applications en prédiction, astronomie ou recherches cryptographiques qui exigent un niveau de performance que peut rarement atteindre une machine unique isolée [12].

Une seconde application de ces systèmes est le partage de charges par des serveurs. Ceci améliore grandement la fiabilité et le temps de réponse du système sollicité par des clients multiples. De tels systèmes sont également plus tolérants aux fautes. C'est d'ailleurs l'une des solutions pour lesquelles les entreprises gérant un trafic Web élevé optent (Cf. Figure 2.3).

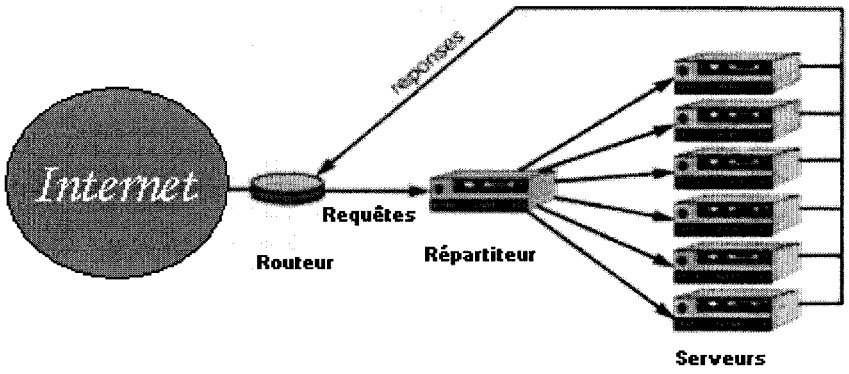



**Figure 2.3 Exemple de grappe simple pour le commerce électronique**

Les grappes d'ordinateurs peuvent également être utilisées pour assurer de la haute disponibilité ou de la redondance. Certains serveurs de la grappe agissent alors comme serveurs de rechange pour d'autres en cas de panne ou de perte de performance (Cf. Figure 2.4). Ceci est très crucial dans les environnements de commerce électronique

où le trafic est très élevé ou encore ceux dans lesquels l'indisponibilité des serveurs est critique (cas des serveurs de télécommunications). Les exigences en termes de haute disponibilité s'expriment souvent en termes du nombre de chiffres 9 décimaux dans le pourcentage indiquant le temps de disponibilité du système (Cf. Tableau 2.1) et varient selon le secteur d'activités (Cf. Tableau 2.2).

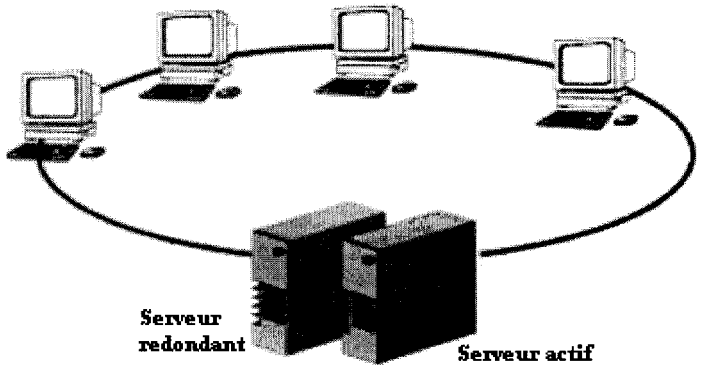



**Figure 2.4 Exemple de grappe simple avec un serveur redondant**

**Tableau 2.1 Exigences en terme de haute disponibilité**

| Disponibilité (%) | Durée de panne | Classe (nombre de 9 s) |
|---|---|---|
| 90% | Plus qu'un mois | 0/1 |
| 99% | Un peu moins de 4 jours | 1/2 |
| 99.9% | Un peu moins de 9 heures | 2/3 |
| 99.99% | Environ une heure | 3/4 |
| 99.999% | Un peu plus de 5 minutes | 4/5 |
| 99.9999% | Environ une demi-minute | 5/6 |
| 99.99999% | Environ 3 secondes | 6 |

**Tableau 2.2 Les classes de haute disponibilité par secteur d'activités**

| Classe | Secteurs d'activités |
|---|---|
| 1 | Réseaux universitaires |
| 2 | Machines usuelles |
| 3/4 | Grappes ordinaires |
| 5 | Commutateurs de téléphonie |
| 6 | Ordinateurs embarqués (avionique) |

### 2.2.2 Quelques spécificités des grappes

Dans la famille des systèmes répartis, une grappe d'ordinateurs se distingue en particulier par :

- sa *structure* qui est plus homogène, étant donné qu'elle est destinée à une tâche spécifique, contrairement aux systèmes répartis traditionnels qui sont constitués de composants très hétérogènes ;
- sa *taille* qui est relativement petite ou moyenne par opposition à d'autres systèmes répartis dont la grandeur est à l'échelle planétaire (Internet) ;
- le *type de tâches* qu'elle est capable d'effectuer, en général une ou des tâches dédiées ou spécialisées ;
- le *niveau de fiabilité* qui est relativement meilleur et dépend des exigences des tâches auxquelles concourent les machines de la grappe.

Beowulf, MOSIX, Linux Virtual Server (LVS), LifeKeeper, Evolocity, etc. sont des solutions de grappes. Beowulf est le plus connu et le plus utilisé dans les environnements de calculs performants où les tâches sont parallélisées par passage de messages (MPI).

## 2.3 Des outils et mécanismes de sécurisation

La sécurité est une préoccupation cruciale dans les environnements répartis, chaque composant du système étant ouvert aux autres pour des fins de partage de ressources, d'informations et de communication. Les objectifs essentiels que vise la sécurité dans de tels environnements sont [9] :

- La préservation de la confidentialité et de l'intégrité des informations. Il s'agit non seulement d'empêcher la divulgation d'informations à des entités non autorisées à les connaître (confidentialité), mais aussi d'empêcher toute altération intentionnelle ou non des informations (intégrité) ;

- La garantie de l'origine de l'information, de l'identité d'une personne ou organisation : c'est l'authentification. Elle consiste à prouver qu'une information provient de la source annoncée ou que l'identité de la personne émettrice est bien celle annoncée ;
- La protection de l'accès aux services : seules les entités autorisées peuvent accéder à un service et les autres doivent se les faire interdire ;
- La fourniture d'éléments de preuve sur la réalité de certaines actions et sur les tentatives d'actions non autorisées.

Les attaques dans les environnements répartis se font par le biais des canaux de communications disponibles. Coulouris et al. [1] nous proposent une classification des attaques basée sur l'utilisation non conventionnelle des canaux de communication :

- l'*écoute clandestine* : c'est la réception de l'information par des entités non autorisées ;
- la *mascarade* : c'est l'usurpation d'identité, une entité se fait passer pour une autre sans l'autorisation de cette dernière ;
- l'interception et la modification de messages ;
- le *rejeu*: c'est le fait d'intercepter des messages et de les rejouer ultérieurement; cette attaque peut avoir lieu même lorsque les messages sont encryptés et authentifiés ;
- le *déni de service* : c'est commettre des actes dans le but d'empêcher l'accès à certaines ressources.

Ces attaques sont les dangers théoriques qui existent. Sur le plan pratique, on peut s'attendre à plusieurs formes dont les succès dépendent surtout de la découverte de trous de sécurité dans les systèmes. En réponse aux attaques, plusieurs mécanismes comme IPSec (utilisé dans les réseaux IP), le contrôle des accès et les pare-feux ont été développés au fil du temps. Nous les présentons respectivement ci-après.

### 2.3.1 Survol de IPSec

À l'instar de IP, IPSec est un protocole de la couche Réseau. Il est défini dans le RFC 2401 de l'IETF. Nous décrirons ce qu'il apporte de plus que IP au niveau de la sécurité; nous présenterons sommairement ses types de paquets et modes de fonctionnement et finirons en abordant la question des associations de sécurité et la gestion des clefs cryptographiques.

#### 2.3.3.1 Plus de sécurité dans le protocole Internet

IPSec offre le service d'*authentification* : cette authentification mutuelle permet à chacun de s'assurer de l'identité de son interlocuteur. IPSec étant un protocole de la couche 3, il s'agit d'une authentification des machines mettant en œuvre le protocole plutôt que des usagers. Le protocole assure également la *confidentialité des données échangées* (chiffrement du contenu de chaque paquet IP pour éviter que quiconque ne le lise). Par ailleurs, IPSec permet d'assurer l'*authenticité des données* : pour chaque paquet échangé, le protocole vérifie qu'il a bien été émis par la bonne machine et qu'il est bien à destination de la seconde machine. En veillant à l'*intégrité des données échangées,* IPSec s'assure qu'aucun paquet n'a subi de modification quelconque durant son transport.

La *protection contre les écoutes et analyses de trafic* est assurée par le chiffrement des adresses IP réelles de la source et de la destination, ainsi que de toute l'entête IP correspondante; ceci est très utile lors de la création de tunnels. De plus, IPSec permet de se prémunir contre les attaques consistant à capturer un ou plusieurs paquets dans le but de les envoyer à nouveau ultérieurement (*protection contre le rejeu*).

#### 2.3.1.2 Les types de paquets et modes de fonctionnement de IPSec

IPSec assure essentiellement les services *d'authentification et d'intégrité* grâce au protocole AH (Authentication Header, RFC 2402) et de *confidentialité* grâce au protocole ESP (Encapsulating Security Payload, RFC 2406). La mise en œuvre des

protocoles AH et ESP consiste à insérer dans un paquet IP ordinaire des entêtes supplémentaires et/ou des éléments de fin de paquet (*trailer*) afin de permettre à IPSec d'offrir ces services de sécurité. L'emplacement de ces éléments dans un paquet IP dépend de la version de IP utilisée ainsi que du mode de fonctionnement choisi pour IPSec. IPSec distingue deux modes de fonctionnement :

- Le mode *Transport* : le paquet IP originel garde son entête à laquelle sont ajoutés les entêtes et/ou éléments de fin de paquet selon le protocole utilisé (AH ou ESP). Ce mode est souvent utilisé pour des communications à l'intérieur d'un même réseau et un analyseur de trafic peut déterminer les types de protocoles et ports utilisés.
- Le mode *Tunnel* : le paquet IP authentifié et/ou confidentiel est encapsulé dans un autre paquet IP (avec donc une nouvelle entête). Ce mode est souvent utilisé pour les communications entre réseaux et un analyseur de trafic ne peut que se rendre compte de la circulation de données encryptées.

Selon le mode utilisé et le protocole (AH ou ESP), on distingue plusieurs types de paquets IPSec. Nous limitons l'illustration qui en est faite (Figure 2.5 et Figure 2.6 extraites de [20]) à la version 4 de IP.

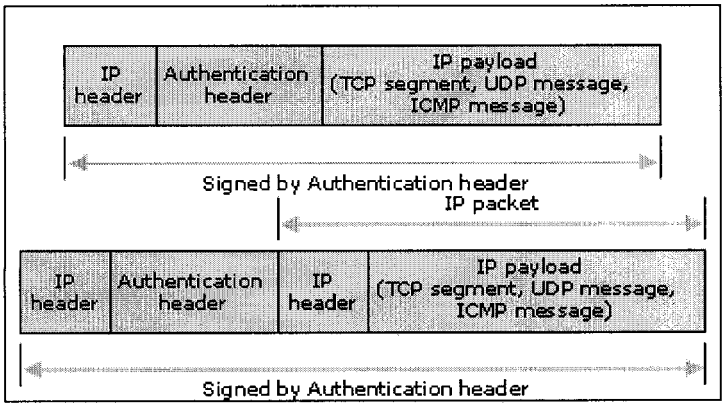



**Figure 2.5 Format de paquets AH (a) Mode transport (b) Mode tunnel**

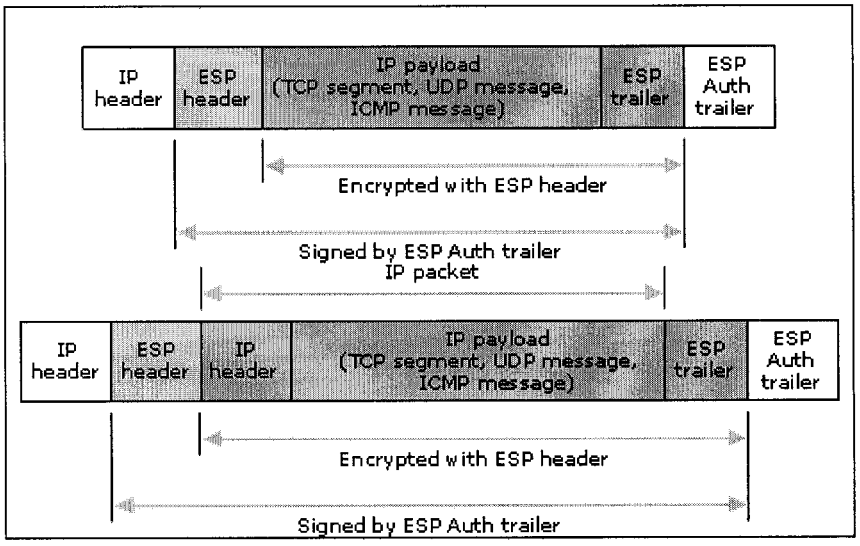



**Figure 2.6 Format de paquets ESP (a) Mode transport (b) Mode tunnel**

#### 2.3.1.3 Les associations de sécurité et la gestion des clefs

Une *association de sécurité* ou SA est une connexion simplexe qui offre des services de sécurité au trafic qu'elle supporte. Une SA contient toutes les informations requises pour caractériser et échanger des données à protéger et des éléments permettant de déterminer à quel type de trafic ou paquet elle s'applique. Les SA sont souvent gérées de façon automatique. L'enjeu principal est la gestion des clefs cryptographiques à manipuler en fonction des informations de la SA, selon que l'on utilise AH ou ESP comme protocole et selon les algorithmes utilisés. La cryptographie à clefs symétriques étant utilisée, il faudrait faire partager les clefs secrètes aux entités communicantes. Le protocole IKE (RFC 2409) est utilisé pour automatiser la génération et les mécanismes de gestion des clefs cryptographiques.

### 2.3.2 Contrôle des accès

Le contrôle des accès dans un système réparti se fait par des listes d'accès (Access Control List ou ACL) ou par des listes de capacités (Capability Lists). [9] définit comme *agents* les entités qui désirent accéder aux ressources du système; ces

ressources sont désignées comme des *objets* accessibles. Une liste d'accès définit pour chaque *objet* les droits qui lui sont associés tandis que la liste de capacités définit pour chaque *objet*, les *agents* qui peuvent y accéder.

Il existe en général deux politiques de contrôle des accès aux ressources :

- ***Le contrôle discrétionnaire des accès (Discretionary Access Control ou DAC)*** : le contrôle des accès s'effectue sur la base de l'identité des agents; c'est l'exemple du système de fichiers sous UNIX où l'accès au fichiers dépend de l'identité de l'usager. Le contrôle des accès dans Windows NT suit aussi le même principe [2]. Ainsi, on peut accéder à certains fichiers ou à d'autres, selon que l'on soit administrateur du système ou un simple usager.
- ***Le contrôle « régulé » des accès (Mandatoy Access Control ou MAC)*** : il offre une meilleure granularité du contrôle et dépend d'une politique de sécurité. Ce type de contrôle procède souvent par étiquetage des ressources afin de les identifier; il indique alors spécifiquement quelle opération particulière un agent peut effectuer sur une ressource donnée, selon les spécifications de la politique de sécurité. Par contraste à la politique de contrôle d'accès précédente, même un administrateur peut se voir refuser des accès à certaines ressources si la politique de sécurité est formulée comme telle. La notion d'agent ici s'étend et peut désigner non seulement un usager, mais aussi un processus ou une ressource matérielle.

### 2.3.3 Les pare-feux

Les pare-feux font partie des mécanismes de protection de réseaux les plus utilisés dans les organisations. Nous présenterons ici les pare-feux traditionnels ainsi que leurs versions améliorées que sont les pare-feux répartis et les micros pare-feux.

#### 2.3.3.1 Les pare-feux traditionnels

Dans les temps anciens, pour protéger les constructions contre les feux, des murs élevés étaient érigés. Le parallèle est fait avec certaines machines d'organisations dans le

vaste réseau Internet devenu au fil des années de plus en plus sujet à insécurité : les pare-feux ou coupe-feux ou *firewalls*. Un pare-feu est une collection de composants interposés entre deux réseaux et qui permet de filtrer le trafic entre eux conformément à une politique de sécurité [3]. Les pare-feux sont habituellement installés dans des routeurs multi ports ou les serveurs proxy.

Comme mentionné dans [8], dans un routeur, ils appliquent un certain nombre de règles sur chaque paquet entrant et décide s'il faut le transmettre ou pas; ce filtrage est effectué sur la base du contenu de l'entête de chaque paquet (par exemple, le numéro du protocole, les adresses IP de source et de destination, les numéros de ports, les drapeaux de connexion et éventuellement d'autres options IP). Ces routeurs offrent l'avantage de la simplicité et du coût matériel bas. Leurs désavantages résident dans la difficulté de configurer adéquatement les règles de filtrages des paquets et le manque de service d'authentification des usagers. Des efforts sont entrain d'être faits dans ce sens par les compagnies qui proposent ces solutions.

Quant aux serveurs proxy, ils agissent comme des passerelles logicielles entre un réseau et un autre pour une application spécifique; un usager contacte le serveur proxy en utilisant une application TCP/IP (comme Telnet, ftp, etc.); le serveur lui demande le nom de la destination à atteindre; lorsque l'usager répond avec en fournissant une identification et une authentification valide, le proxy contacte la destination et assure le relais des paquets de façon transparente à l'usager. Les serveurs proxys utilisés en guise de pare-feux ont l'avantage d'offrir, contrairement aux routeurs, des mécanismes d'authentification d'usagers, de traces, etc.

En général les routeurs et les serveurs proxys sont combinés pour réaliser les pare-feux afin d'offrir une protection meilleure. Les pare-feux offrent des solutions de sécurité bon marché et simples. Mais Sotiris Ioannidis et al [3] identifient plusieurs insuffisances aux pare-feux traditionnels :

- Les pare-feux sont devenus des goulots d'étranglements à cause de l'augmentation du débit sur les lignes et de l'intensité de plus en plus élevée des

calculs en réseaux; Cette croissance en débit et besoin de calculs rend considérable les données à filtrer par un pare-feux traditionnel;

- Il existe des protocoles qui sont difficiles à gérer au niveau des pare-feux car ces derniers ignorent certains paramètres qui ne sont disponibles qu'aux points terminaux dans le réseau protégé (exemples de RealAudio, ftp);
- Considérer que l'intérieur d'un réseau protégé par un pare-feux est une utopie même en présence de plusieurs couches de protection par pare-feux;
- De plus en plus, il est aisé d'établir de nouveaux points d'entrées dans les réseaux sans l'autorisation des administrateurs, et ce, par l'utilisation de formes variées de tunnels, d'infrastructures sans-fil et autres moyens qui outrepassent les mécanismes de sécurité des pare-feux;
- De plus le besoin grandissant de contrôles de sécurité ayant une granularité plus fine ne peut être satisfait par les pare-feux traditionnels sans les rendre trop complexes.

Ces raisons et bien d'autres encore ont amené les auteurs de [3] à proposer le concept de pare-feux répartis.

### 2.3.3.2 Les pare-feux répartis

Dans les pare-feux répartis, une politique de sécurité centrale est définie mais renforcée aux terminaux. Le système propage cette politique aux différents nœuds. Une telle infrastructure suppose la définition d'un langage pour exprimer la politique de sécurité, un mécanisme pour répartir cette politique et enfin un mécanisme pour appliquer ladite politique aux paquets entrants. Les auteurs ont proposés une architecture qui ne se base plus sur la topologie du réseau (nœuds intérieurs à protéger des nœuds extérieurs) : des droits sont assignés à des machines disposant de clefs privées correspondant à certaines clefs publiques. Ainsi donc par exemple, un ordinateur portable appartenant à une organisation protégée et qui est connecté à Internet, a le même niveau de protection qu'une machine à l'intérieur de ladite organisation. Par

contre, un ordinateur portable connecté au réseau de l'organisation à l'intérieur par un visiteur, n'aura pas les bonnes lettres de créances, et donc se verra refuser des accès même s'il est, du point de vue topologique, situé à l'intérieur de l'organisation protégée. Le mécanisme d'authentification des usagers est basé sur IPSec. Un module reçoit les requêtes de connexion et s'assure d'une part d'authentifier l'usager et de vérifier que sa requête de connexion est permise par la politique de sécurité en vigueur. Le fait de se départir de la notion de périmètre topologique de sécurité des pare-feux traditionnels est l'amélioration fondamentale de cette proposition. La granularité du contrôle de sécurité offerte est cependant limitée à la possibilité ou non d'accéder à un terminal donné ou de recevoir d'un autre terminal via un protocole de communication : les droits accordés ou refusés se rapportent donc à une connexion. En effet, un contrôle plus fin des opérations effectuées sur la machine hôte par l'usager une fois connecté devrait être effectué, même lorsque celui-ci détient les droits d'accès à ladite machine par son identité et que la politique en vigueur l'y autorise.

#### 2.3.3.3 Les micros pare-feux

Kai Hwang et Muralidaran ont proposé dans [7], une alternative aux pare-feux répartis qui améliore le temps système (*overhead*) et qui n'utilise pas d'authentification basée sur IPSec. Il s'agit d'une défense en profondeur à 3 niveaux (Cf. Figure 2.7) :

- Le premier niveau de protection est une passerelle qui agit à titre de filtre général entre le réseau à protéger et l'extérieur;
- Le deuxième niveau est un Administrateur de politique de sécurité (*Policy Manager*) : il met à jour la politique de sécurité et les bases de données d'intrusions et arrête la propagation des intrusions d'un nœud à l'autre;
- Le troisième niveau est le micro pare-feux implémenté au niveau du noyau du système d'exploitation de chaque nœud du réseau à protéger : il assure le filtrage local des paquets, détecte les incidents locaux, tient des journaux d'audit.

Cette solution vise essentiellement la détection d'intrusions dans le réseau protégé; les requis actuels en matière de sécurité dépassent la détection d'intrusions et exigent une granularité encore plus fine.

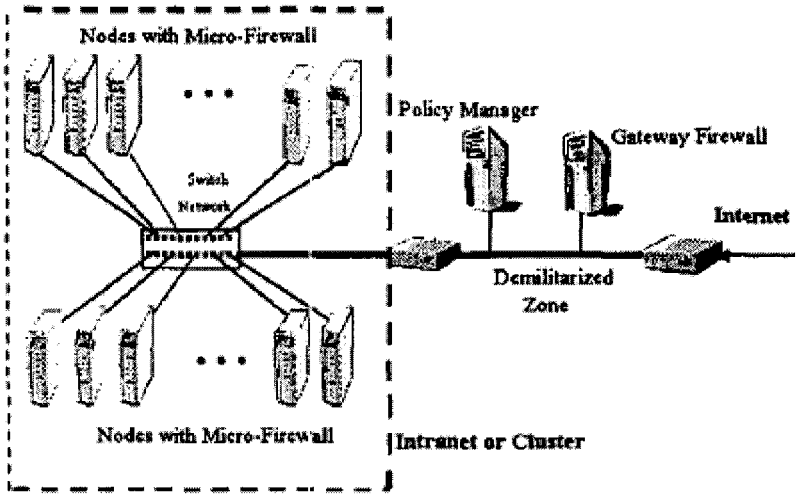



**Figure 2.7 Architecture exploitant les micro pare-feux**

## 2.4 Description des politiques de sécurité

La politique de sécurité est l'expression des privilèges qu'un administrateur ou une organisation accorde aux entités constituantes de son environnement. Ces entités peuvent être des usagers, des composants matériels, des applications, des processus, des abstractions logicielles, en fonction du niveau de granularité de ladite politique. La définition d'une politique de sécurité peut dépendre de considérations sociales, organisationnelles, juridiques, réglementaires ou techniques. La représentation des politiques de sécurité est une problématique qui a été abordée de différentes manières. Nous présenterons le langage EACL, les langages basés sur XML et enfin KeyNote.

### 2.4.1 Le langage EACL

Dans [5], Tatyana Ryutov et Clifford Neuman ont implémenté un langage du nom de **EACL** (*Extended Access Control List*) pour décrire une politique d'authentification des usagers d'un système. Plus particulièrement, une liste EACL est associée à un objet ou un groupe d'objets à protéger en spécifiant les droits permis ou non avec des conditions optionnelles. Les conditions sont évaluées dans l'ordre d'apparition dans la syntaxe. Une règle en EACL est typiquement constituée d'une permission ou refus de droit d'accès suivi de quatre blocs de conditions : un ensemble de conditions préliminaires à la requête d'accès qui doivent être vraies, des conditions à activer si le droit d'accès est accordé, les conditions qui doivent être vraies lorsque la requête d'accès est en cours d'exécution et enfin les conditions qui doivent être vraies après l'exécution de la requête d'accès. Toutes les conditions ne sont pas obligatoirement spécifiées. Par exemple, la politique suivante : « Tom ou Joe peut lire le fichier FIC seulement s'il se connecte à partir du domaine *.isi.edu» peut être représentée par les deux règles EACL suivantes attachées au fichier FIC:

*"positive access right : read, pre-conditions : Tom, *.isi.edu"*

*"positive access right : read, pre-conditions : Joe, *.isi.edu"*

La règle suivante spécifie que l'arrêt du système est permis en utilisant une authentification basée sur le protocole Kerberos; si la requête réussit, alors l'identifiant de l'usager sera stocké et dans le cas contraire, une notification sera envoyée à l'administrateur par courrier électronique.

*pos_access_right test host_shut_down*

*pre_cond_access_id KerberosV.5 trusted@ORGA.EDU*

*rr_cond_audit local on:success/info:userID*

*post_cond_notify local email/to:sysadmin/on:failure*

L'évaluation d'une liste de règles commence par la première et les cas d'inconsistances dans les politiques exprimées sont résolus en accordant la précédence sur les règles déjà examinées. Ceci demande beaucoup d'attention à l'administrateur qui rédige la politique.

La compréhension de la syntaxe n'est cependant pas intuitive. Une autre limitation réside dans le fait que les auteurs ne signalent aucun mécanisme de protection et d'authenticité de l'information que l'EACL contient; elle pourrait donc être modifiée par une personne non autorisée. Les auteurs eux-mêmes signalent des problèmes d'évolutivité des objets à contrôler car ils font l'hypothèse que les conditions sont évaluées une à la fois et que les requêtes ne se chevauchent pas. Les éléments de syntaxe EACL sont répertoriés dans la Figure 2.8.

```
eacl ::= {eacl_entry}
eacl_entry ::= pos_access_right conditions | neg_access_right conditions
pos_access_right ::= "pos_access_right" def_auth value
neg_access_right ::= "neg_access_right" def_auth value
conditions ::= pre_conds mid_conds rr_conds post_conds
pre_conds ::= {condition}
mid_conds ::= {condition}
rr_conds ::= {condition}
post_conds ::= {condition}
condition ::= cond_type def_auth value
cond_type ::= alphanumeric_string
def_auth ::= alphanumeric_string
value ::= alphanumeric_string
```

**Figure 2.8 Éléments de syntaxe EACL**

### 2.4.2 Travaux utilisant des métalangages

Pour représenter des politiques de sécurité, certains auteurs se sont servis de métalangages. Le plus utilisé dans la littérature est XML. Nous présenterons sommairement XML avant d'exposer les langages existants qui l'utilisent.

#### 2.4.2.1 Survol de XML

XML est un méta-langage qui a été proposé par le World Wide Web Consortium (W3C) [34] pour la structuration des documents pour le Web. C'est un ensemble de règles, de lignes directrices, de conventions pour la conception de formats textes permettant de structurer des données. XML est un langage balisé comme le HTML. Mais alors que HTML utilise des balises prédéfinies; XML utilise les balises seulement

pour délimiter les éléments de données et laisse l'entière interprétation des données à l'application qui les lit. Un document XML est globalement structuré en éléments avec ou non des attributs (paire nom/valeur), des déclarations de types d'éléments et d'attributs. Cette structure peut se représenter par un diagramme en arbre.

- **Quelques avantages et inconvénients**

  XML est *extensible* et *évolutif* puisqu'il permet à l'utilisateur de définir ses propres balises et de modifier aisément le schéma de son document quand besoin est. XML est *sous forme de fichier texte*; donc l'encodage est adapté pour toute plateforme, ce qui représente un atout en termes d'*interopérabilité* dans le contexte des systèmes répartis. Il permet aussi de valider le contenu d'un document par rapport à une *grammaire standardisée* et définie selon les besoins du concepteur de langage.

  Comme XML utilise des balises pour délimiter les données, les fichiers XML sont la plupart du temps volumineux; mais l'espace disque n'est plus aussi coûteux qu'autrefois et les protocoles de communication peuvent compresser des données à la volée permettant de faire transiter efficacement du texte.

- **Grammaire des documents XML**

  La grammaire décrit les balises valides, leur ordre, les règles d'imbrication, les attributs et d'autres informations pour le document XML. Un document XML *bien formé* respecte la syntaxe du langage de balisage tandis qu'un document *valide* respecte en plus les spécifications de la grammaire. L'auteur du document est responsable de sa conformité à la grammaire. La grammaire est souvent définie par une *DTD* ou un *schéma XML*. Une DTD permet de définir de façon simple la structure : l'agencement autorisé des éléments, les attributs possibles, les emplacements pour textes, etc. Un schéma XML offre une grammaire plus rigoureuse et plus riche : *typage des données* (chaîne de caractères, booléen, nombre, date, etc.), *spécification des occurrences autorisées d'éléments ou d'attributs*, *contraintes structurales* (définitions de types, classes et sous-classes, modèles de

contenus, importation, inclusion, etc.). Contrairement à la DTD, le schéma XML est écrit dans la syntaxe même du langage de balisage.

- **Les espaces de noms et la sécurité**

  Les espaces de noms (*namespace*) permettent de regrouper des éléments XML autour d'un nom unique. Ainsi, des éléments portant un même nom peuvent faire partie de sources différentes. Ils peuvent cohabiter au sein d'un même document, tout en évitant des collisions car les éléments appartenant à un espace de noms se distinguent des autres par l'ajout d'un préfixe symbolisant cette singularité.

  Il existe des mécanismes de sécurité pour les documents XML. Il est par exemple possible de signer ou d'encrypter les parties critiques d'un document XML. Des travaux plus poussés permettent de définir des règles pour l'accès régulé à l'information contenu dans un fichier XML.

#### 2.4.2.2 Langages de représentation de politique de sécurité basés sur XML

Une solution de représentation de politique de sécurité utilisant le métalangage XML et décrite dans [6] exprime le contrôle des accès sur la base des rôles (RBAC). Le RBAC (Role Based Access Control) assigne des permissions aux rôles et des rôles aux usagers; c'est un modèle abstrait où un usager ne peut accéder qu'aux ressources que son rôle dans l'organisation lui donne droit. Le modèle de référence du RBAC est montré à la Figure 2.9 et un composant RBAC en XML utilise les éléments du Tableau 2.3.

Il existe une famille de modèles RBAC : le $RBAC_0$ (modèle de base), le $RBAC_1$ (modèle hiérarchique), le $RBAC_2$ (modèle de contraintes) et le $RBAC_3$ (inclusion de tous les modèles). En RBAC, à chaque usager est assigné un rôle et à chaque rôle est assignée une permission avec la possibilité que certains rôles héritent des permissions ou droits d'autres rôles par hiérarchisation.

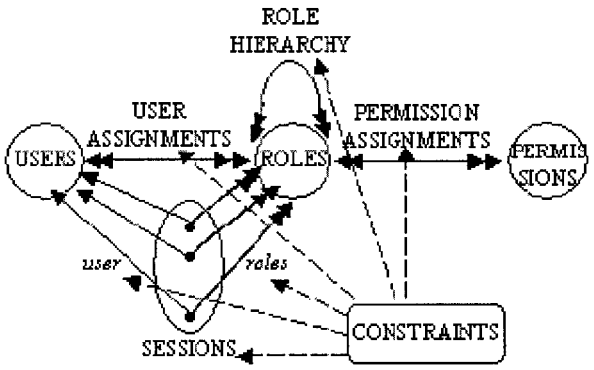



**Figure 2.9 Modèle de référence du RBAC**

**Tableau 2.3 Composants RBAC en XML**

| | |
|---|---|
| <! ELEMENT USER EMPTY><br><! ATTLIST USER<br>NAME ID #REQUIRED> | <! ELEMENT PERMISSION EMPTY><br><! ATTLIST PERMISSION %DEFINITION> |
| **Un usager** | **Un rôle** |
| | |
| <! ELEMENT ROLE EMPTY><br><! ATTLIST ROLE<br>TITLE ID #REQUIRED> | <! ELEMENT PERMISSION_ASSIGNMENT EMPTY><br><! ATTLIST PERMISSION_ASSIGNMENT<br>ROLE IDREF #REQUIRED<br>PERMISSIONS IDREFS #REQUIRED> |
| **Une permission** | **Une assignation de permissions** |
| | |
| <! ELEMENT ROLE_ASSIGNMENT EMPTY><br><! ATTLIST ROLE_ASSIGNMENT<br>ROLE IDREF #REQUIRED<br>USERS IDREFS #REQUIRED> | <! ELEMENT INHERITS EMPTY><br><! ATTLIST INHERITS<br>FROM IDREFS #REQUIRED<br>TO IDREF #REQUIRED> |
| **Une assignation de rôles** | **L'expression de la hiérarchisation des rôles** |
| | |
| <! ELEMENT RBAC1_MODEL (USER+, ROLE+,<br>INHERITS*, PERMISSION+,<br>PERMISSION_ASSIGNMENT*,<br>ROLE_ASSIGNMENT*)> | |
| **Modèle d'élément $RBAC_1$** | |

D'autres efforts de représentation de politiques de sécurité en XML ont été faits par OASIS (*Organization for the Advancement of Structured Information Standards*). Il s'agit de **XACML** (*eXtensible Access Control Markup Language*) qui permet d'exprimer des règles de contrôle d'accès à de l'information sur Internet mais aussi des requêtes/réponses pour savoir si une entité a des droits d'accès ou pas à une information.

**XACL** (*eXtensible Access Control Language*) quant à lui est un langage utilisant XML pour décrire la politique d'accès aux éléments contenus dans un document XML. Il a été développé dans les laboratoires IBM au Japon.

### 2.4.3 Le langage KeyNote

**KeyNote** est un autre outil pour exprimer des politiques de sécurité et des lettres de créances (*credentials*). Il dérive d'une architecture décrite dans le RFC 2704. Une assertion en langage KeyNote est une séquence de champs commençant par le nom du champ, suivi de deux points et du contenu du champ. Une assertion en KeyNote utilise un champ obligatoire (*Authorizer*) et 6 champs optionnels (*Comment, Conditions, KeyNote-Version, Licensees, Local-Constants, Signature*).

Le champ obligatoire spécifie l'instance qui a émis la politique de sécurité ou la lettre de créance tandis que *Licensees* indique les entités cibles de la politique ou celles déléguées par l'instance émettrice de l'assertion. Le champ *Conditions* spécifie les conditions sous lesquelles les entités désignées par le champ *Licensees* bénéficient des droits ou refus accordés par l'autorité *Authorizer*; Le champ *Comment* permet d'introduire des commentaires et *KeyNote-Version* indique la version utilisée. *Local-Constants* est un champ qui permet de surcharger certains attributs de l'assertion dans laquelle il est spécifié : il offre par exemple des mécanismes pour définir des noms plus courts pour les entités à énumérer dans le champ *Licensees* afin de rendre ce dernier champ plus facile de lecture. Le champ *Signature* représente la signature de l'autorité *Authorizer.*

Un évaluateur KeyNote est chargé de retourner les résultats des requêtes faites par des applications suivant la politique de sécurité en vigueur. Il reçoit l'identité de l'entité qui désire agir ainsi que les actions qu'elle veut poser, puis l'expression de la politique de sécurité exprimée en langage KeyNote et enfin le type de réponse voulu ( par exemple {Rejeté, Approuvé} ou {Pas d'accès, Accès limité, Accès total}).
Un exemple de politique de sécurité est la suivante :

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: "rsa-hex:1023abcd"
Comment: Allow Licensee to connect to local port 23 (telnet) from
         internal addresses only, or to port 22 (ssh) from anywhere.
         Since this is a policy, no signature field is required.
Conditions: (local_port == "23" && protocol == "tcp" &&
             remote_address > "158.130.006.000" &&
             remote_address < "158.130.007.255) -> "true";
            local_port == "22" && protocol == "tcp" -> "true";
```

Un concept important que mentionne les auteurs est le fait que KeyNote soit *monotonique* : lorsqu'une requête est formulée avec plusieurs lettres de créances ou politiques, lorsque l'une d'elles approuve la requête, les autres ne peuvent la révoquer. Ceci résout les conflits dans en présence de politiques contradictoires.

Tout en demeurant respectueux des politiques de sécurité définies au moyen des langages précédemment présentés, il peut être utile de proposer des mécanismes de variation du niveau de sécurité.

## 2.5 Gestion de la qualité du service de sécurité

La sécurité est de plus en plus vue comme un service au même titre que les autres dans les systèmes répartis. De ce fait, des usagers peuvent choisir différents niveaux de sécurité en fonction de leurs moyens et/ou besoins, ce qui peut s'assimiler à des requêtes de qualité de service. La sécurité n'est donc pas toujours binaire. D'où la notion de qualité de service de sécurité (QoSS) ou qualité de protection (QoP).

Dans [13], les auteurs utilisent le concept de *classes de sécurité* pour aborder la question de la qualité de protection, mais plus spécifiquement dans un contexte de commerce électronique. Ces classes de sécurité représentent les niveaux de sécurité possibles ainsi que les technologies cryptographiques utilisées pour en satisfaire les exigences. Ils proposent un protocole qui varie de façon dynamique le niveau de sécurité lors d'une transaction électronique, tout en demeurant dans les contraintes ou marges de manœuvres exigées par l'usager et le domaine de sécurité dans lequel il opère. Ils

classent les messages des participants en *quatre grandes classes* par ordre croissant de niveau de sécurité (respectivement décroissant en terme de latence) : SC1, SC2, SC3 et SC4. Les classes sont définies par les techniques cryptographiques qu'elles utilisent pour la confidentialité des messages et l'authentification de l'origine. La classe de niveau de sécurité le plus bas SC1 utilise par exemple RSA (pour la signature numérique des messages hachés), SHA-1 comme méthode de hachage et DES pour encrypter et décrypter les messages.

L'architecture ASE-COM proposée pour ce protocole inclut un *classificateur* : son rôle est de sélectionner dynamiquement la classe de sécurité appropriée pour un message dans une situation donnée. Le niveau de sécurité de la transaction électronique est donc ajustée en fonction de la décision de ce classificateur, mais tout en demeurant dans des limites spécifiées par l'usager. Le classificateur prend sa décision au moyen d'algorithmes heuristiques basés sur cinq (5) règles qui décident de la classe appropriée en fonction des aspects suivants :

- les aspects dépendants du domaine de sécurité dans lequel se déroulent les transactions (par exemple, le degré de protection des informations sensibles);
- les aspects dépendants du système comme la rapidité des calculs cryptographiques;
- les aspects dépendants du réseau comme le taux de transmission de messages entre un expéditeur et un récepteur;
- les aspects dépendants de l'usager : par exemple les limites de sécurité tolérables; ils sont facultatifs mais peuvent servir à orienter le processus de classification.

Leurs résultats indiquent des performances 100 fois meilleures en temps d'exécution par rapport à une situation où le protocole n'est pas utilisé. Ils signalent cependant que deux des 5 règles de leur heuristique conduisent à des choix qui dégradent considérablement le niveau de sécurité dans un réseau à trafic élevé.

Phyllis Schneck, Karsten Schwan ont spécifiquement travaillé sur l'authentification dynamique et flexible [14]. Ils ont introduit la notion de **thermostat de**

**sécurité**. Ce thermostat utilise également des heuristiques pour sélectionner le niveau d'authentification, dans le respect des limites spécifiées par l'usager. Leurs heuristiques sont basées sur le pourcentage d'authentification des paquets, la technique de retardement d'authentification (qui a pour but de diminuer la fréquence des vérifications [14]), l'usage de clefs secrètes partagées, le changement d'algorithme (RSA ou DSA).

Pour C. Irvine et T. Levin, offrir de la qualité de protection passe d'abord par l'identification des services de sécurité ainsi que les niveaux de variations possibles qu'on peut offrir pour chacun d'eux. Dans [18], ces auteurs nous proposent donc comme bases de travail, une taxonomie organisée sous la forme d'un tableau non exhaustif mais qui résume un certain nombre de services de sécurité : *Confidentialité des données, Confidentialité du trafic, Intégrité des données, Authentification, Non répudiation, Garantie de service, Disponibilité, Audit et détection d'intrusion, Contrôle aux limites.* À chacun de ces services est associé des exemples de mécanismes de sécurité selon la zone topologique du réseau où ce service est déployé - IN pour les nœuds intermédiaires (routeurs, commutateurs, ...), NC pour les connexions réseau (connexions filaires, nœuds, ...), ES pour les systèmes terminaux (client, serveurs, ...) et TS pour tous les autres cas). Un exemple est donné dans le Tableau 2.4 pour le service de confidentialité des données.

Au nombre des paramètres de sécurité qui peuvent être sujets à variation, ils citent par exemple :

- La force des algorithmes de cryptographie (par exemple RSA, DES, etc. ) mesurée en termes de réaction à une attaque de force brutale;
- La longueur de la clef cryptographique;
- Fonctions de sécurité présentes dans l'environnement distant d'exécution d'une tâche;
- Pourcentage de paquets authentifiés; etc.

**Tableau 2.4 Taxonomie partielle: cas du service de confidentialité des données**

| Service de sécurité | Zone | Exemples de mécanismes de sécurité |
|---|---|---|
| Confidentialité des données | IN | Contrôle des accès OS, Lettres de créances cryptographiques |
| | NC | 40 bits DES, 128 bits Blowfish |
| | ES | Contrôle des accès OS, Lettres de créances cryptographiques |

Dans [17], les mêmes auteurs indiquent que la motivation fondamentale de l'introduction du concept de QoSS est la gestion efficiente et plus prévisible des ressources d'un système réparti. Ils définissent un *composant de sécurité* S.c. comme une choix booléen sur une variable unique de sécurité [15]. Par exemple :

*S.1. : Longueur de la clef d'encryptions >=64, <=256*
*S.2. : % des paquets authentifiés >=50, <=90*

Un *vecteur de sécurité*, quant à lui, est composé d'un ou de plusieurs *composants de sécurité.* L'état du réseau peut imposer des restrictions sur les limites acceptables pour une variable de sécurité (par exemple, le taux d'authentification des paquets), parce que sous certaines conditions, l'usager ou l'administrateur peut exiger plus (ou respectivement moins) de sécurité pour une application. Les auteurs illustrent leur idée par 3 états ou modes possibles : *État NORMAL – État AFFECTÉ – État d'URGENCE.* Les plages de valeurs acceptables pour une variable de sécurité sont définies par la politique de sécurité pour chacun des états possibles du système. Et pour chaque plage acceptable, on peut encore définir des échelles de variations pour la variable (bas/moyen/élevé). La Figure 2.10 illustre les niveaux de la variable de sécurité « taux d'authentification de paquets » (0 à 100%) dans l'état NORMAL du réseau. Des niveaux similaires sont à définir pour les deux autres états possibles du réseau.

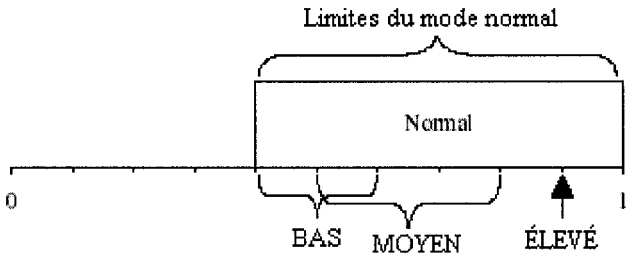



**Figure 2.10 Niveaux d'une variable de sécurité dans l'état NORMAL**

À une sécurité variable correspond une surcharge en temps système (*overhead*) variable. Irvine et Levin [15] ont calculé des formules de coûts pour le cas de l'application FTP en choisissant comme variables de sécurité : le taux d'intégrité des paquets, la longueur de la clef symétrique pour l'authentification et le type de l'autorisation d'accès. Les résultats de coûts sont des expressions linéaires des variables de sécurité, sauf dans le cas du type de l'autorisation d'accès où des fonctions non linéaires apparaissent.

Des travaux similaires ont été effectués par Zhaoyu Liu, Roy H. Campbell et M. Dennis Mickunas [19], mais adaptés au concept de réseaux actifs (*Active Networks*). Ils définissent huit niveaux de protection et utilisent l'interface GSS API (Generic Security Service) pour implémenter leurs mésscanismes. GSS API offrent aux programmeurs des outils pour un accès uniforme à certains services de sécurité (Version en langage C décrite par le RFC 1509).

# CHAPITRE 3

# MÉCANISMES ET ARCHITECTURE DE SÉCURISATION

Dans ce chapitre, nous décrirons le langage de représentation de politique de sécurité répartie XDSPL que nous proposons pour les grappes d'ordinateurs. Nous présenterons également un mécanisme pour assurer la diffusion des règles de politique de sécurité écrites par l'administrateur du système. Nous proposerons enfin une architecture de qualité de protection (QoP) ainsi qu'un modèle de logique de décision et un protocole pour la gestion de la QoP sur les grappes.

## 3.1 Langage de représentation de politique de sécurité répartie : XDSPL

Dans cette section, nous évoquerons d'abord certaines hypothèses et choix qui permettront de définir le cadre de validité de nos propositions. Nous présenterons par la suite les éléments de syntaxe ainsi que la sémantique qui leur est respectivement associée; cette sémantique peut être certes modifiée en fonction du système cible. Nous finirons par une présentation de la grammaire formelle de XDSPL, un langage de représentation de politique de sécurité répartie.

### 3.1.1 Hypothèses et choix

XDSPL permet de définir une politique de sécurité globale dans une grappe. Les services de sécurité pris en compte dans XDSPL sont : le *contrôle des accès*, la *confidentialité* et l'*intégrité*.

#### 3.1.1.1 Le contrôle des accès

Notre choix s'est porté sur le ***contrôle régulé des accès*** ou MAC (Mandatory Access Control). Dans un système basé sur le *contrôle discrétionnaire des accès* ou

DAC (Discretionary Access Control), les autorisations d'accès aux ressources ou objets sont à la discrétion des propriétaires respectifs. Dans un système basé sur le MAC, les autorisations dépendent en plus d'autres informations comme le contexte de sécurité. Par exemple, la création d'un processus fils à partir d'un processus parent sur un système Unix utilisant le MAC requiert non seulement le droit d'exécution « x » sur le processus parent mais aussi que le contexte de sécurité autorise précisément ledit processus parent à créer de nouveaux processus.

Le précédent exemple évoque la nécessité d'***attribuer des identifiants de sécurité (Security ID ou ScID) aux différentes ressources du système*** par un mécanisme dont le choix d'implémentation est libre; à chacun de ces identifiants de sécurité est associé un ***contexte de sécurité***. Sur un même nœud, toutes les entités étiquetées par un même identifiant de sécurité se voient appliquer les mêmes règles de sécurité : on dit qu'elles appartiennent au même *contexte de sécurité*; ceci autorise donc l'utilisation d'un même identifiant pour plusieurs ressources (processus, binaires, etc.).

Les approches traditionnelles du MAC ([21], [22], [23]) sont souvent basées sur un seul nœud du système, ce qui ne permet pas de définir de règles de MAC entre des entités se trouvant sur deux machines différentes. XDSPL vise une politique de sécurité globale pour toute la grappe; pour cela, notre approche considère aussi l'***utilisation d'identifiants de sécurité pour les nœuds (SnID)***. Ces derniers doivent par contre être uniques, contrairement à ceux utilisés pour les ressources.

#### 3.1.1.2 La confidentialité et l'intégrité des données

Tout comme le service de contrôle des accès, les services de confidentialité et d'intégrité nécessitent aussi l'utilisation d'identifiants de sécurité afin de permettre de signifier avec précision les entités sur lesquelles s'appliquent les règles de sécurité. Nous considérons des réseaux tout IP et proposons l'utilisation de ***IPSec comme l'un des moyens pour assurer la confidentialité et l'intégrité dans notre modèle***. Il est certes possible d'étendre les règles de sécurité concernant la confidentialité et l'intégrité par d'autres mécanismes comme le permet la grammaire de XDSPL.

### 3.1.2 Structure syntaxique, sémantique et grammaire de XDSPL

Cette sous-section décrit l'organisation syntaxique de XDSPL. Elle propose une sémantique associée aux éléments syntaxiques et aussi une grammaire pour définir des règles de validité d'un document écrit en XDSPL.

#### 3.1.2.1 Structure syntaxique globale de XDSPL

Un fichier écrit en XDSPL peut contenir plusieurs politiques de sécurité, chacune d'elles étant essentiellement constituée de l'expression de *règles de sécurité* et d'autres éléments dont la *version* et le *mode de fonctionnement* de ladite politique :

- **La version de XDSPL :** elle est exprimée par la balise XML <version> qui en contient trois autres : <major>, <minor> et <date> qui permettent respectivement d'identifier la version de XDSPL ainsi que sa date de création.
- **Le mode de fonctionnement** : il est utopique de penser à exprimer sous forme de règles de sécurité toutes les situations auxquelles sera confronté le système. Les règles exprimées ne sont donc pas exhaustives et le système doit adopter une attitude lorsqu'il se trouve dans une situation pour laquelle aucune règle de sécurité n'a été définie dans XDSPL. Le mode de fonctionnement est spécifié au moyen de la balise <mode>. On distingue trois modes de fonctionnement dont le choix est à la discrétion de l'administrateur du système:
    - Le mode *permissif* : signalé par le texte PERMISSIVE à l'intérieur des balises définissant le mode, il permet de signifier que tout ce qui n'est pas explicité sous forme de règles de sécurité dans XDSPL est permis.
    - Le mode *restrictif* : signalé par le texte PARANOID à l'intérieur des balises définissant le mode, il permet de signifier que tout ce qui n'est pas explicité sous forme de règles de sécurité dans XDSPL est interdit.
    - Le mode *intermédiaire* : signalé par le texte ENFORCING à l'intérieur des balises définissant le mode, il permet de définir un niveau intermédiaire entre les deux précédents modes; l'administrateur a la liberté d'y associer la sémantique de son choix à l'implémentation.

- **Les règles de sécurité** : ce sont elles qui expriment si un sujet *S* donné est autorisé à accéder ou effectuer des opérations spécifiques sur une ressource *T*. Signalons ici que *"sujet"* désigne l'entité du système qui fait une requête d'accès ou d'opération tandis que *"ressource"* désigne la cible de ladite requête. Les entités du système sont identifiées par la paire d'identifiants de sécurité **{SnID, ScID}** où SnID désigne le nœud où réside l'entité qui appartient au contexte de sécurité ScID. Dans XDSPL, les balises des identifiants de *sujet* et de *ressource* sont respectivement précédées des lettres « s » et « t », soient <sSnID>, <sScID> et <tSnID>, <tScID>. Ainsi, pour chaque règle de sécurité, on utilise les balises **<allow>** ou **<deny>** pour respectivement signifier que le droit est accordé ou refusé. Nous avons défini six (6) classes de règles de sécurité expliquées dans la sous-section suivante et identifiées par des balises regroupées sous la balise principale des règles de sécurité <securityRules> (Cf. aperçu graphique de la structure globale de XDSPL à la Figure 3.1)
- **L'identifiant de sécurité par défaut** : Étant donné qu'il serait impossible pour l'administrateur du système d'attribuer des identifiants de contexte de sécurité à toutes les entités, la balise <default_ScIDs> permet de spécifier la valeur par défaut à leur attribuer.

### 3.1.2.2 Sémantique associée aux règles de sécurité

Chacune des règles de sécurité est généralement structurée en quatre parties. La première partie identifie le sujet par la paire {sSnID, sScID}. La seconde partie identifie la ressource à laquelle ledit sujet désire accéder {tSnID, tScID} lorsque cela est applicable. La troisième partie indique le type de règle de sécurité (classe de règles). La quatrième partie spécifie la ou les permissions accordées (respectivement refusées) à l'intérieur de la balise <*allow*> (respectivement <*deny*>). Cette structuration dépend cependant de la classe de règles : les exceptions et cas particuliers sont spécifiés dans la grammaire de XDSPL. On distingue dans XDSPL six (6) classes de règles de sécurité qui sont :

- **La classe des règles de création de processus**

Une règle de création de processus permet d'autoriser ou non un processus d'un nœud donné à créer d'autres processus. Elle est identifiée par la balise <class_PROCESS_rule>. La seule permission que l'on peut spécifier à l'intérieur des balises <allow> ou <deny> est CREATE. La règle (a) de l'exemple de la Figure 3.2 interdit aux processus appartenant au contexte de sécurité 8 sur le nœud 1 de créer d'autres processus.

- **La classe des règles de migration de contexte de sécurité**

Une règle de migration de contexte de sécurité permet d'autoriser ou non un processus d'un nœud donné, appartenant au contexte de sécurité <parent_ScID> à migrer dans un autre contexte de sécurité <binary_ScID>. Cette règle est identifiée par la balise <class_TRANSITION_rule>. L'autorisation de migration ne s'exprime pas au moyen des balises <allow> ou <deny>; elle consiste plutôt à définir, pour le processus en migration, la valeur finale du contexte de sécurité grâce à la balise <new_ScID>. Autrement dit, pour accorder la permission de migration, on assigne à *new_ScID* la même valeur que *binary_ScID*; tandis que pour refuser la permission *new_ScID* vaut *parent_ScID* ou toute autre valeur que l'administrateur de la politique de sécurité juge adéquate. La règle (e) de l'exemple de la Figure 3.2 interdit aux processus appartenant au contexte de sécurité 2 sur le nœud 2 de migrer dans le contexte de sécurité 1.

- **La classe des règles de la couche réseau**

Une règle de la couche réseau permet d'autoriser ou non un processus (ou un binaire) identifié par la paire {sSnID, sScID} à recevoir des paquets IP d'un autre identifié par la paire {tSnID, tScID}. Cette règle est identifiée par la balise <class_NETWORK_rule>. La seule permission que l'on peut spécifier à l'intérieur des balises <allow> ou <deny> est NETWORK_RECEIVE. La règle (d) de l'exemple de la Figure 3.2 autorise les processus appartenant au contexte de sécurité

5 sur le nœud 4, à recevoir des paquets des processus du contexte de sécurité 2 sur le nœud 2.

- **La classe des règles de sockets**

Une règle de socket cible la couche transport. Elle permet d'autoriser ou non des opérations spécifiques de la couche transport entre un *socket* identifié par la paire {sSnID, sScID} et un autre identifié par la paire {tSnID, tScID}. Cette classe de règles est identifiée par la balise <class_SOCKET_rule>. Les permissions que l'on peut spécifier à l'intérieur des balises <allow> ou <deny> réfèrent aux opérations possibles sur les sockets : CREATE, CONNECT, SEND, LISTEN, RECEIVE, SHUTDOWN, GET_OPTIONS, SET_OPTIONS, GET_PEERNAME, GET_SOCKNAME. La règle (b) de l'exemple de la Figure 3.2 autorise les sockets ou processus appartenant au contexte de sécurité 1 sur le nœud 3 à se connecter et à envoyer des paquets à tous les processus, quel que soit leur contexte de sécurité (utilisation du joker ALL) sur le nœud 2.

- **La classe des règles d'initialisation de sockets**

Une règle d'initialisation de sockets est un complément à la classe des règles de sockets. Elle permet d'initialiser le contexte de sécurité <ScID> d'un socket résidant sur le nœud <SnID> en fonction du protocole utilisé à la couche transport (<protocol>) et du port correspondant (<port>). Cette classe de règles est identifiée par la balise <class_SOCKET_INIT_rule>. XDSPL propose les protocoles UDP, TCP et RAW (ce dernier, rarement utilisé, sert surtout pour des communications entre processus). La règle (c) de l'exemple de la Figure 3.2 initialise à 1 le contexte de sécurité des sockets UDP du port 80 sur le nœud 1.

- **La classe des règles de confidentialité et d'intégrité**

Dans cette classe de règles, nous proposons IPSec comme protocole pour assurer la confidentialité et l'intégrité au moyen de la balise <IPSec_mode>. Ainsi donc, pour

deux entités communicantes (processus, socket, binaire, etc.) respectivement identifiées par les paires {sSnID, sScID} et {tSnID, tScID}, on spécifie à l'intérieur des balises ouvrante <allow> et fermante </allow> le protocole d'encryptage à utiliser pour IPSec; les possibilités offertes sont : AH pour le protocole AH, ESP pour le protocole ESP et NO_SEC pour indiquer l'absence d'encryptage. Une règle de confidentialité et d'intégrité est identifiée par la balise <class_DisCI_rule>. La règle (f) de l'exemple de la Figure 3.2 spécifie l'utilisation du protocole ESP de IPSec lors de toute communication entre des entités appartenant au contexte de sécurité 1 sur le nœud 3, et celles du contexte de sécurité 2 sur le nœud 2.

#### 3.1.2.3 Grammaire de XDSPL

XDSPL étant écrit dans le métalangage XML, nous utilisons un schéma XML pour décrire sa grammaire. En particulier, il décrit le type des éléments du langage, leurs occurrence et agencements autorisés. Cette grammaire comprend la définition de plusieurs types pour faciliter sa représentation, notamment pour les éléments redondants. Le schéma est présenté de façon exhaustive à l'annexe B du présent mémoire. Contentons-nous ici de mentionner quelques règles grammaticales clefs :

- Tous les identifiants de sécurité utilisés (*sSnID, sScID, tSnID, tScID*) de même que les ports de protocoles (*port*) sont des entiers pouvant prendre des valeurs entre 0 et 65535 ou bien le joker « ALL » qui est une chaîne de caractères;
- L'identification de sécurité de toute entité du système est obligatoirement constituée à la fois de son contexte de sécurité et de son nœud; on ne peut donc avoir que des paires {sSnID, sScID} ou {tSnID, tScID} pour désigner un processus, un socket ou un binaire;
- On ne peut spécifier qu'un type de permission par règle de sécurité : soit <allow> ou soit <deny>, mais pas simultanément les deux. Autrement dit, une règle permet de façon exclusive d'accorder une autorisation ou de la refuser;

- On ne peut spécifier qu'un seul protocole à la fois dans une règle de confidentialité et d'intégrité. Il en est de même pour les protocoles de transport dans une règle d'initialisation de socket.

Une politique de sécurité répartie écrite en XDSPL doit être diffusée au travers de toutes les machines de la grappe pour se faire appliquer. Pour cela nous proposons dans la section suivante un mécanisme pour propager la politique à tous les nœuds.

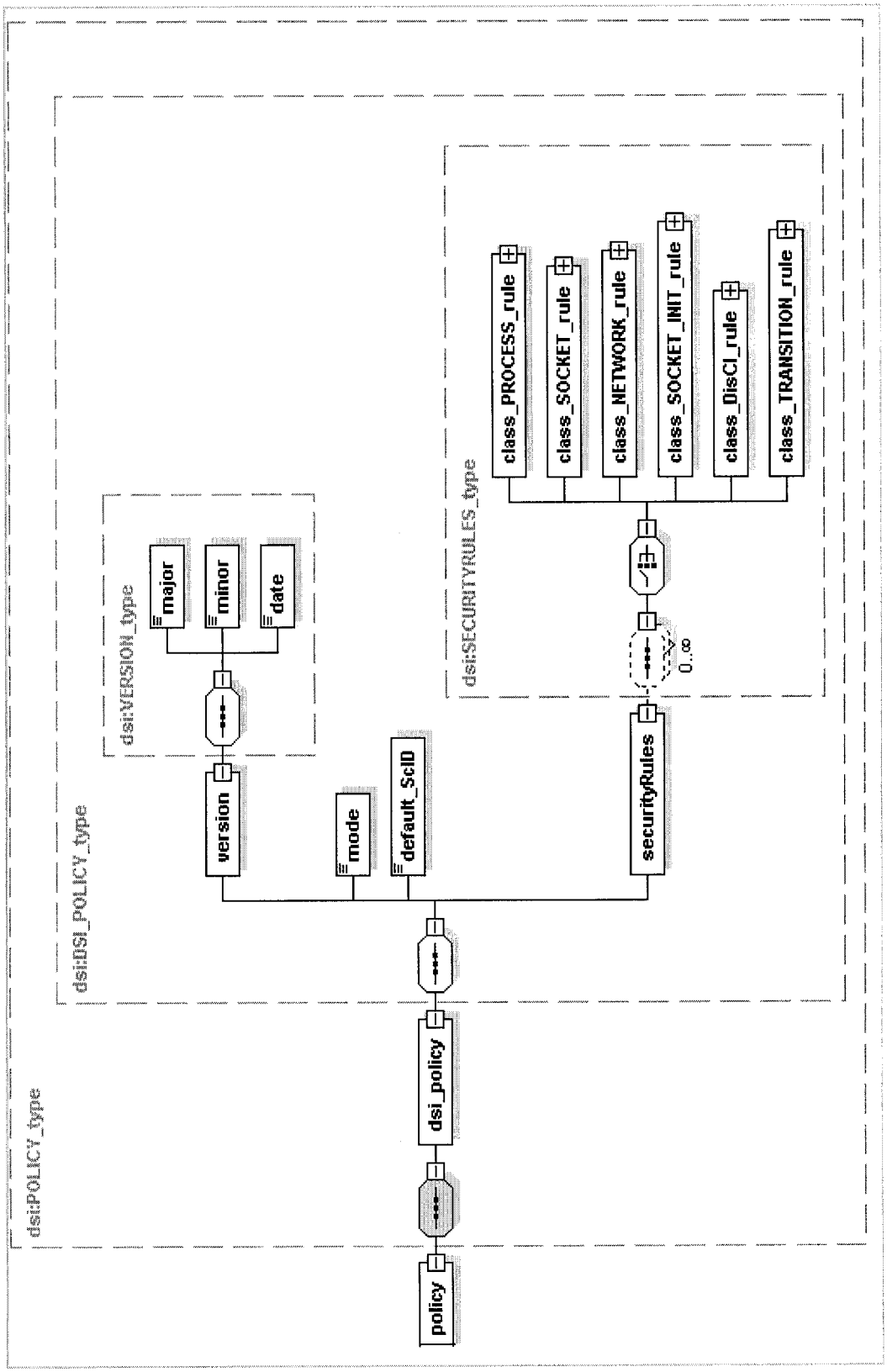



**Figure 3.1 Aperçu graphique de la structure globale de XDSPL**

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://sourceforge.net/projects/disec/DSP"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <dsi_policy>
        <version>
            <major> 1 </major>
            <minor> 0 </minor>
            <date> 2002-02-28 </date>
        </version>
        <mode> PARANOID </mode>
        <default_ScID> 2 </default_ScID>
        <securityRules>
            <class_PROCESS_rule>
                <sScID> 1 </sScID>
                <sSnID> 8 </sSnID>                          Règle (a)
                <deny>CREATE</deny>
            </class_PROCESS_rule>
            <class_SOCKET_rule>
                <sScID> 1 </sScID>
                <sSnID> 3 </sSnID>
                <tScID> ALL </tScID>                        Règle (b)
                <tSnID> 2 </tSnID>
                <allow> CONNECT SEND </allow>
            </class_SOCKET_rule>
            <class_SOCKET_INIT_rule>
                <protocol>UDP</protocol>
                <port> 80 </port>                           Règle (c)
                <SnID> 1 </SnID>
                <ScID> 1 </ScID>
            </class_SOCKET_INIT_rule>
            <class_NETWORK_rule>
                <sScID> 5 </sScID>
                <sSnID> 4 </sSnID>
                <tScID> 2 </tScID>                          Règle (d)
                <tSnID> 2 </tSnID>
                <allow>NETWORK_RECEIVE</allow>
            </class_NETWORK_rule>
            <class_TRANSITION_rule>
                <parent_ScID> 2 </parent_ScID>
                <SnID> 2 </SnID>                            Règle (e)
                <binary_ScID> 1 </binary_ScID>
                <new_ScID> 2 </new_ScID>
            </class_TRANSITION_rule>
            <class_DisCI_rule>
                <sScID> 1 </sScID>
                <sSnID> 3 </sSnID>
                <tScID> 2 </tScID>
                <tSnID> 2 </tSnID>                          Règle (f)
                <allow>
                    <IPSec_mode> ESP </IPSec_mode>
                </allow>
            </class_DisCI_rule>
        </securityRules>
  </dsi_policy>
</policy>
```

**Figure 3.2 Exemple de politique de sécurité en XDSPL**

## 3.2 Mécanisme de diffusion de XDSP

La politique de sécurité formellement écrite en XDSPL est hébergée sur un **Serveur de Sécurité (SS)** de la grappe. Une fois écrite ou mise à jour, elle est interprétée et formatée en événements de règles brutes (format compréhensible par les machines) et diffusée à tous les autres nœuds par un canal sécurisé tel que décrit à la Figure 3.3. Chacun des nœuds recevant ces règles héberge un **Gestionnaire de Sécurité (GS)** qui les reçoit, les comprend et les stocke dans une mémoire cache afin de s'en servir pour effectuer les contrôles de sécurité.

Typiquement, un événement de règle est formaté en utilisant uniquement des entiers de 32 bits, des entiers non signés ou des chaînes de caractères simples afin de garantir la portabilité. Un formatage non brut des données aurait induit une phase d'interprétation supplémentaire des règles sur chacun des nœuds destinataires, ce qui aurait été une perte en termes de performance; de plus, il n'est point utile à ce niveau du système d'assurer la lisibilité ou la facilité de compréhension requise lors de l'expression de la politique.

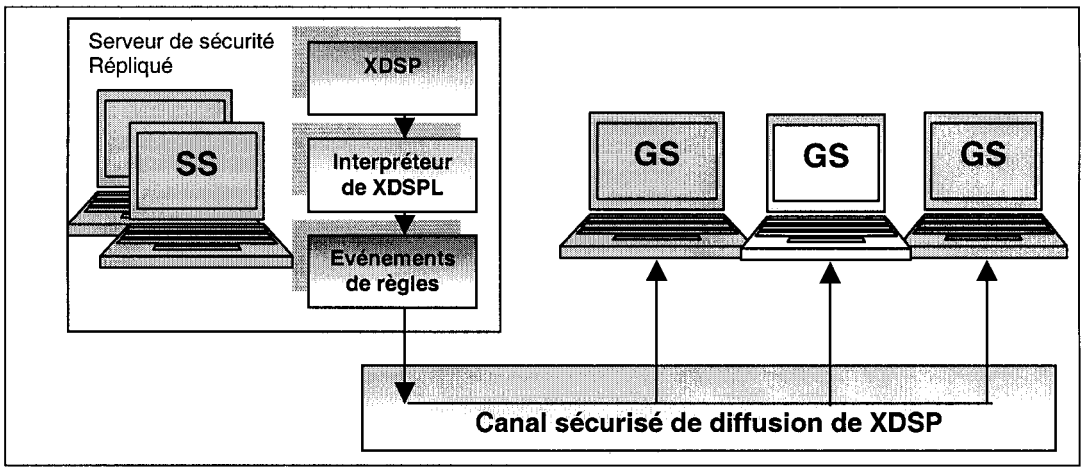



**Figure 3.3 Mécanisme de diffusion de la politique de sécurité**

Le **canal sécurisé de diffusion de la politique de sécurité** doit garantir *l'intégrité des données* qui circulent : une entité malveillante pourrait en effet modifier

les règles lors de leur diffusion, ce qui résulterait en la mise en vigueur effective d'une politique contraire à celle exprimée par l'administrateur de sécurité. Par ailleurs, le canal doit garantir l'*interopérabilité* et offrir un *service de diffusion d'événements*. Pour cela, nous proposons une approche sous forme de couches de services et protocoles pour ce canal, tel qu'illustré à la Figure 3.4.

En effet, le canal comprend d'abord **un service de noms** : il permet au Serveur de Sécurité de retrouver le canal par son nom au lieu d'une référence. À ce service de noms est associé un **service d'événements** pour assurer la diffusion des règles. Ces deux services sont basés sur **CORBA** qui assure l'interopérabilité. La dernière couche du canal utilise les **mécanismes d'encryptage de SSL/TLS** pour assurer la sécurité des règles transmises. SSL et TLS sont les mécanismes de sécurisation les plus utilisés actuellement dans les fureteurs, notamment pour le commerce électronique.

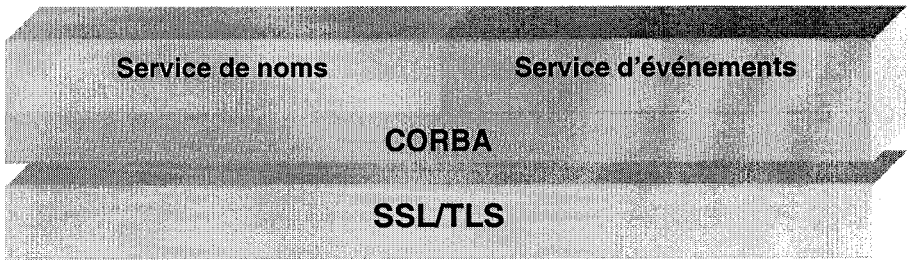



**Figure 3.4 Couches du canal sécurisé de diffusion de XDSP**

Autant il est crucial de sécuriser les systèmes par une politique bien définie, autant il devient important d'offrir des mécanismes d'ajustement du niveau de sécurité selon les besoins des applications ou des usagers. Nous nous penchons sur la question dans la section suivante en proposant un cadre pour la prise en compte de la qualité de protection offerte dans une grappe par une architecture qui s'inspire du mécanisme de diffusion de politique de sécurité ci-dessus décrit.

## 3.3 Architecture de gestion de la qualité de protection (QoP) pour grappes d'ordinateurs

La qualité de protection (QoP), tel que mentionné dans les chapitres précédents, permet essentiellement d'offrir des mécanismes pour ajuster le niveau de sévérité de la sécurité par rapport aux besoins. On peut en effet vouloir augmenter le niveau de protection d'une application qui a des requis de confidentialité ou d'intégrité élevés, tout comme on peut par exemple vouloir baisser son niveau de sécurité par souci de gestion efficiente des ressources système ou en présence d'une machine de faible capacité de calculs cryptographiques. Nous présentons ici les composants de l'architecture de gestion de QoP que nous proposons et décrirons par la suite son déploiement dans une grappe d'ordinateurs.

### 3.3.1 Description des composants de l'architecture

Dans l'architecture que nous proposons, chaque nœud du réseau héberge un **Système de Gestion de Qualité de Protection (SGQoP)** qui utilise des informations locales au nœud et des informations distantes provenant du Serveur de Sécurité (SS) afin de donner suite aux requêtes de qualité de protection. Nous décrivons ci-après ces éléments représentés à la Figure 3.5.

#### 3.3.1.1 Les inputs du Système de Gestion de Qualité de Protection (SGQoP)

Les inputs locaux du SGQoP sur un nœud sont :

- **L'application usager** : c'est elle qui désire effectuer une opération et qui requiert un niveau de sécurité donné.
- **Le gestionnaire de ressources système** : il est chargé de fournir des informations sur la disponibilité des ressources du système. Aucune requête

de niveau de sécurité ne pourra être exécutée si les ressources du système ne le permettent pas, à moins d'une souscription formelle préalable.

Les inputs distants du SGQoP sont :

- **Le contexte de sécurité (CS)**: c'est l'état du système au moment de la requête de qualité de protection. Il indique si le système est en état de fonctionnement normal, ou s'il est présentement l'objet d'attaque, ou si des alarmes signalent une intrusion ou toute autre atteinte à la sécurité. Ce paramètre entre en ligne de compte dans le choix du niveau de protection à offrir, car on peut ne pas vouloir tolérer une requête pour un niveau bas de QoP alors que le système est en état d'alerte.
- **La politique de sécurité répartie (XDSP)**: c'est elle qui décrit dans le système comment la sécurité est assurée : ce qui peut être fait, par qui, comment, etc. L'administrateur de sécurité y définit également une qualité de protection par défaut pour différentes applications. Il est important que le niveau de sécurité accordé à une entité soit conforme à la politique de sécurité en vigueur.
- **Les souscriptions aux niveaux de qualité de protection (SNQoP)**: les entités qui demandent des niveaux de sécurité doivent y avoir souscrit si elles désirent y prévaloir à tout moment, dans le respect de la politique de sécurité du système; à défaut de cela, le demandeur sera soumis à des tests additionnels pour déterminer la recevabilité de sa requête.
- **Niveaux de QoP offerts (NQoPO)**: c'est la spécification des différents niveaux de sécurité possibles indiquant, dans un format donné, la valeur de différents paramètres ou variable de sécurité. Ces paramètres de sécurité peuvent par exemple être : la longueur des clefs cryptographiques, l'algorithme de cryptographie, etc. Typiquement, il pourrait s'agir d'un fichier descriptif qui contient les $N$ niveaux possibles, soient $L_1, L_2, \ldots, L_N$.

### 3.3.1.2 Principaux composants du Système de Gestion de Qualité de Protection

Parmi les principaux composants du système de gestion de qualité de protection, on retrouve :

- **L'interface de QoP pour applications** : cette interface permet de transmettre à l'*Agent de Qualité de Protection* la requête de QoP formulée par l'usager pour une application. En effet, les applications doivent garder leur indépendance vis-à-vis de l'architecture; elles n'ont donc pas à intégrer la logique de qualité de protection. Cette interface agit donc comme mandataire de l'application.
- **L'Agent de Qualité de Protection** : cette instance est chargée de collecter et de transmettre toutes les informations dont le *module de gestion de qualité de protection* a besoin pour prendre sa décision par rapport à la requête formulée. Il s'agit essentiellement des informations des intrants du système et des niveaux de QoP offerts.
- **Le module de gestion de QoP** : c'est ce module qui gère réellement la qualité de protection (Cf. Figure 3.6). Il est composé des éléments ci-après :
  - ***Le tableau de bord de QoP*** : c'est l'interface entre l'agent de QoP et le module de gestion de QoP. C'est ce tableau qui compile toutes les informations obtenues de l'agent de QoP sous une forme facilement accessible à l'instance de décision du module de gestion de qualité de protection. Il s'agit essentiellement de la politique de sécurité en vigueur, du contexte de sécurité, de la disponibilité de ressources système, des souscriptions aux niveaux de sécurité et des niveaux de QoP offerts.
  - ***L'instance de décision de QoP*** : c'est le cœur du gestionnaire de qualité de protection. C'est l'instance qui décide du niveau de protection pour toute requête formulée. Elle prend sa décision en fonction des informations disponibles sur le tableau de bord, en fonction d'une logique de décision.

- ***La logique de décision*** : comme l'indique son nom, elle signifie à l'instance de décision de QoP les règles à appliquer pour traiter les requêtes.
- ***L'exécutant*** : pour chaque requête traitée, il formule les ***ordres de QoP*** à répartir aux services de sécurité; plus spécifiquement, il s'agit des valeurs à attribuer à certains paramètres de sécurité comme par exemple l'algorithme cryptographique à utiliser, la longueur des clefs cryptographiques, etc. L'exécutant sert donc d'interface entre le module de gestion de QoP et les différents services de sécurité.

Nous désignons par **services de sécurité** les différents services pour lesquels le système offre des mécanismes de sécurisation. Nous pouvons par exemple citer le service d'authentification, le service d'intégrité, le service d'audit, le service de contrôle des accès, etc.

### 3.3.2 Déploiement de l'architecture de Qualité de Protection

Le déploiement de l'architecture de qualité de protection s'effectue selon le même principe que les mécanismes de diffusion et de mise en application de la politique de sécurité écrite en XDSPL. L'architecture se déploie comme suit (Cf. Figure 3.7) :

- La **politique de sécurité** (XDSP), les **Souscriptions aux Niveaux de QoP** (SNQoP) et les **Niveaux de QoP Offerts** (NQoPO) sont des spécifications qui résident sur le **Serveur de Sécurité** (SS) de la grappe. À l'instar de la politique de sécurité, ces spécifications peuvent être formalisées en XML et interprétées sur le SS, puis diffusées à tous les autres nœuds par des canaux sécurisés qui leur sont dédiés. Le Contexte de Sécurité (CS) est typiquement représenté par des messages d'alarmes ou d'avertissement à transmettre à tous les nœuds, selon l'état du système (situation d'attaque par exemple).

- Sur chacun des nœuds de la grappe réside un **Gestionnaire de Sécurité** auquel est intégré le **Système de Gestion de QoP** (incluant : **Interface de QoP pour applications**, **Agent de QoP**, **Module de Gestion de QoP**). Le GS d'un nœud donné reçoit les informations d'intérêt du SS (les inputs distants) et les stocke dans une mémoire cache dont se servira le Système de Gestion de QoP (SGQoP) pour donner suite aux requêtes de QoP localement formulées sur le nœud considéré. Rappelons que le SGQoP utilise des informations locales sur le GS pour fonctionner : il s'agit d'abord de la requête de QoP puis des informations provenant du gestionnaire de ressources système.

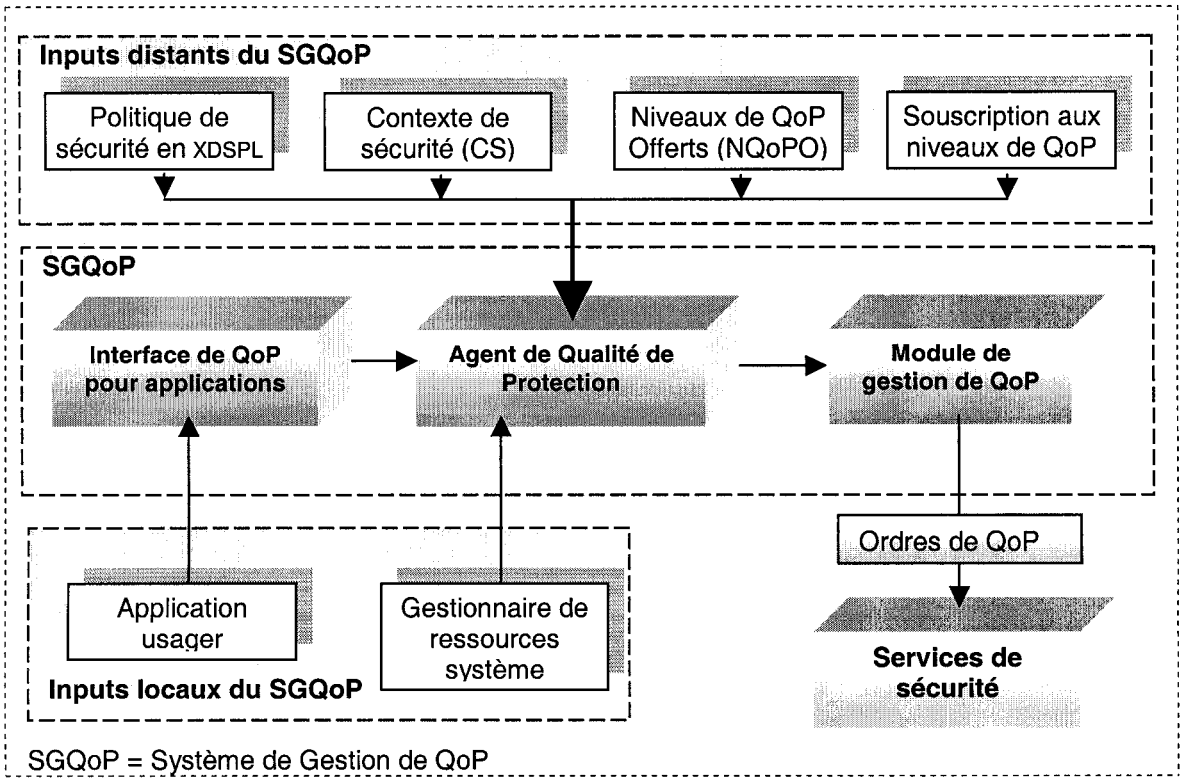



**Figure 3.5 Composants de l'architecture de qualité de protection**

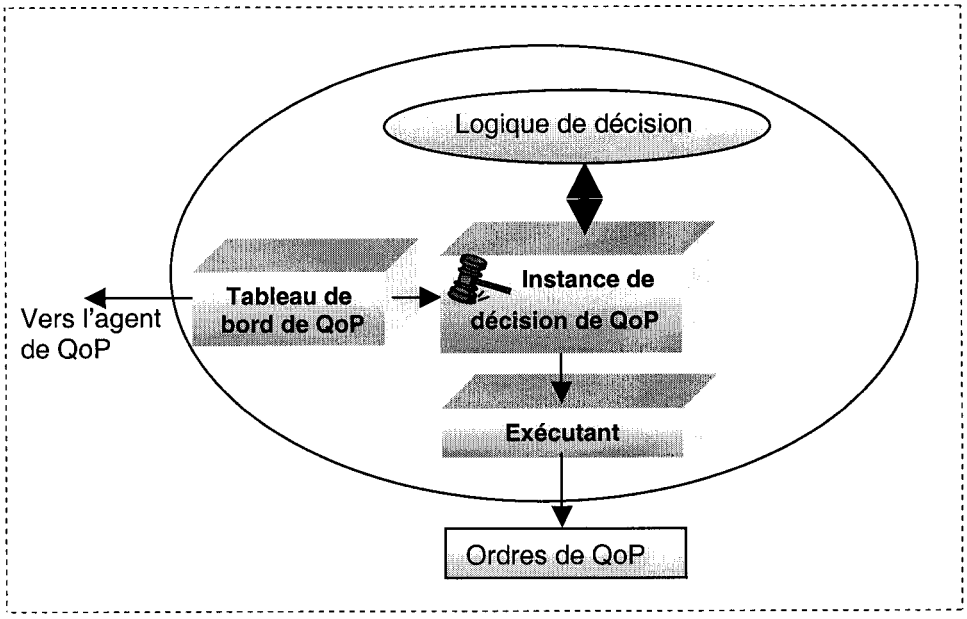
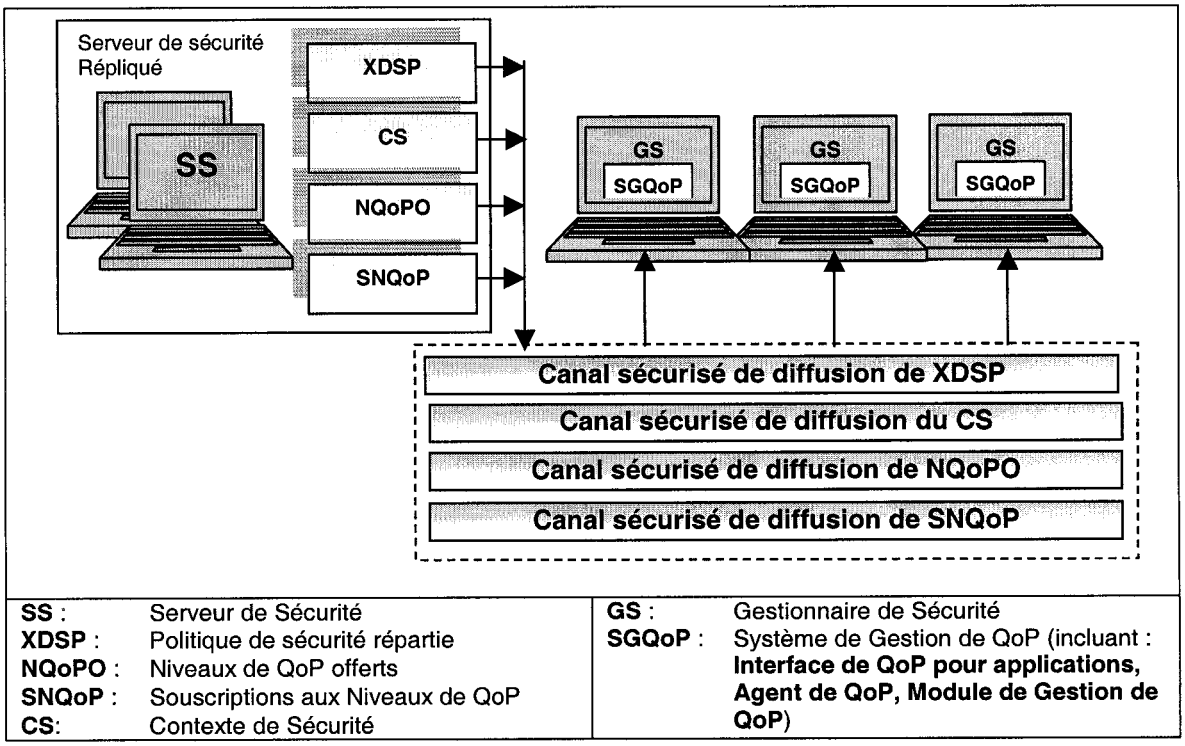



**Figure 3.6 Module de gestion de qualité de protection**




| | | | |
|---|---|---|---|
| **SS** : | Serveur de Sécurité | **GS** : | Gestionnaire de Sécurité |
| **XDSP** : | Politique de sécurité répartie | **SGQoP** : | Système de Gestion de QoP (incluant : **Interface de QoP pour applications, Agent de QoP, Module de Gestion de QoP**) |
| **NQoPO** : | Niveaux de QoP offerts | | |
| **SNQoP** : | Souscriptions aux Niveaux de QoP | | |
| **CS**: | Contexte de Sécurité | | |

**Figure 3.7 Schéma de déploiement de l'architecture de Qualité de Protection**

## 3.4 Logique de décision et protocole pour la gestion de la qualité de protection

En fonction de la logique de décision implantée dans le module de gestion de qualité de protection, des requêtes de QoP d'applications usager peuvent être rejetées ou acceptées. Nous proposons dans cette section un modèle de logique de décision ainsi qu'un protocole pour régir les requêtes de qualité de protection.

### 3.4.1 Modèle de logique de décision du module de gestion de QoP

La logique de décision considère un ordre de précédence parmi les paramètres du tableau de bord de QoP : la politique de sécurité possède la plus grande priorité car rien ne devrait l'outrepasser; le second paramètre est la souscription aux différents niveaux de sécurité : cette souscription est une forme de garantie de qualité de protection; ensuite vient le contexte de sécurité qui décrit l'état du système et finalement la disponibilité des ressources. Cet ordre a conduit à l'algorithme proposé et décrit à la Figure 3.8.

### 3.4.2 Protocole de gestion de la qualité de protection

Une entité *D* (usager, binaire, processus, etc.) qui désire accéder à une ressource *R* adresse une requête de niveau de sécurité *Nreq* au Système de Gestion de la Qualité de Protection. Ce dernier accepte d'offrir le niveau de protection demandé si cela est possible; autrement, il refuse la requête et renvoie au demandeur les niveaux possibles auquel il a droit. Ce protocole simple est décrit à la Figure 3.9. La simplicité de ce protocole permet d'éviter une dégradation significative de performances dues aux interactions multiples.

**SI** la requête respecte la *politique de sécurité*, **alors**
    **SI** le demandeur *a souscrit au niveau de sécurité demandé*, **alors**
        **SI** les *Ressources nécessaires* < *Ressources disponibles*, **alors**
            Ordonner l'exécution de la requête
        **SINON**
            Rejeter la requête
            Informer le demandeur de l'insuffisance de ressources
        **FinSI**
    **SINON**
        **SI** *le contexte de sécurité* le permet, **alors**
            **SI** les *Ressources nécessaires* < *Ressources disponibles*, **alors**
                Ordonner l'exécution de la requête
            **SINON**
                Rejeter la requête
                Informer le demandeur de l'insuffisance de ressources
            **FinSI**
        **SINON**
            Rejeter la requête
            Informer le demandeur que le contexte de sécurité ne permet pas l'exécution de cette requête.
        **FinSI**
    **FinSI**
**SINON**
    Rejeter la requête
    Informer le demandeur de la non-conformité à la politique de sécurité
**FinSI**

**Figure 3.8 Algorithme de décision du niveau de QoP**

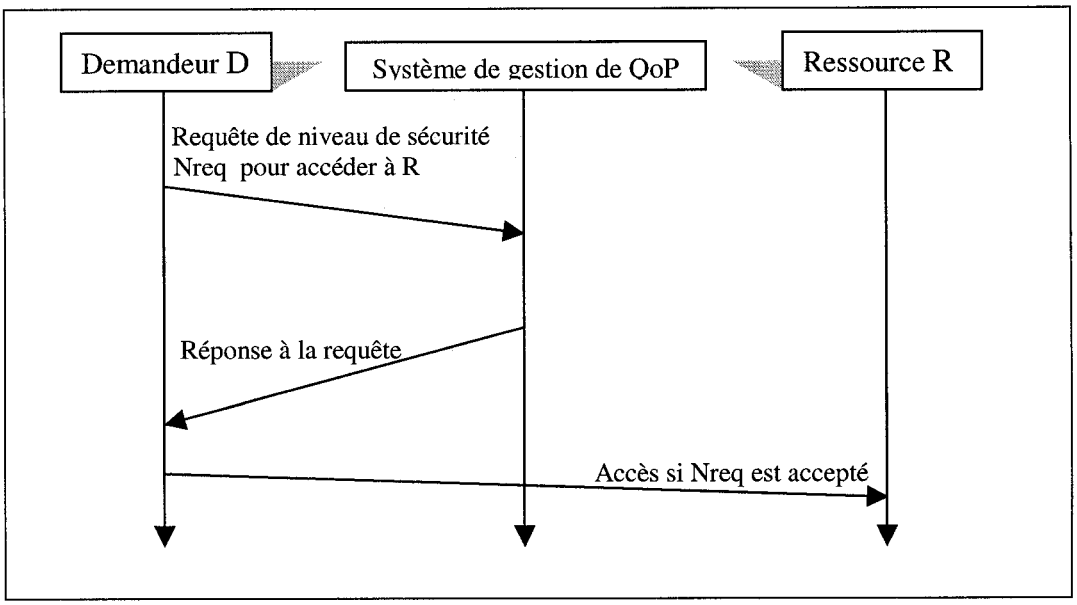



**Figure 3.9 Protocole de gestion de QoP**

# CHAPITRE 4

# IMPLÉMENTATIONS, RÉSULTATS ET PERFORMANCES

Dans ce chapitre, nous présenterons des éléments d'implémentation, des résultats de fonctionnalité ainsi que des mesures de performance. L'architecture de base pour les développements identifie trois éléments clef (Serveur de Sécurité – Gestionnaires de Sécurité – Canal Sécurisé de Communication) qui sont le siège des implémentations que nous présenterons. Par la suite, nous exposerons des résultats de fonctionnalités pour la mise en œuvre de la politique de sécurité pour les communications réseau ainsi que pour le module de qualité de protection. Pour terminer, nous présenterons les performances du module de vérification des règles de politique de sécurité comparées avec l'approche similaire la plus connue dans la littérature et une mention sur les performances du module de qualité de protection.

## 4.1 Implémentations

Signalons ici que certaines implémentations ont été faites en langage C++ et d'autres en langage C. L'utilisation stricte du langage C pour les implémentations au sein du noyau du système d'exploitation se justifie par le choix du système d'exploitation Linux qui lui-même est développé en C. L'utilisation de Linux est motivée par la possibilité d'accéder et d'apporter des modifications au système d'exploitation; elle permet aussi de pouvoir faire de la réutilisation et de l'adaptation des ressources que la communauté des programmeurs d'applications à code source ouvert rend disponible. Les différents programmes ont été compilés avec ***gcc (version 2.95 et 2.96)***.

Dans cette section, nous indiquerons tout d'abord comment sont étiquetées les ressources de l'environnement réparti (binaires, processus et sockets) pour rendre possible le déploiement des mécanismes proposés. Par la suite, nous décrirons

l'implémentation des composants au sein du Serveur de Sécurité, la réalisation pratique du canal sécurisé de communication et enfin les composants d'intérêt du Gestionnaire de Sécurité présent sur chaque nœud.

### 4.1.1 Assignation des identifiants de sécurité

Il existe deux identifiants de sécurité : l'identifiant de nœud *SnID* et l'identifiant de contexte de sécurité *ScID*. Nous décrivons ici comment ils sont respectivement assignés aux nœuds et aux ressources (binaires, processus et sockets).

#### 4.1.1.1 Assignation du SnID

L'assignation de l'identifiant de nœud *SnID* se fait au moyen de l'application ***SetNodeID*** que l'on exécute sur le nœud à étiqueter et prend comme unique paramètre le numéro entier à attribuer (Exemple : `SetNodeID 2`). La valeur en paramètre (2 dans l'exemple précédent) est passée à la fonction ***dsi_sys_security (unsigned int id, unsigned int call, unsigned long *args)*** qui se charge de la stocker dans une variable ***dsi_node_id*** au sein du noyau Linux de la machine étiquetée. Mentionnons ici que ***dsi_sys_security()*** est la fonction maîtresse qui transmet au noyau Linux les informations utiles à y stocker; ces informations demeurent persistantes aussi longtemps que le nœud est fonctionnel et n'est pas redémarré.

#### 4.1.1.2 Assignation du ScID

L'assignation du contexte de sécurité *ScID* peut être effectuée sur un fichier binaire (représentant une application), un processus ou un socket.

- **Cas des fichiers binaires** : nous assignons un identifiant *ScID* aux fichiers binaires qui possèdent une entête libre d'écriture au début (*ELF header*). L'assignation s'effectue au moyen de l'application ***SetSID*** que l'on exécute sur le nœud hébergeant le binaire à étiqueter. Cette application prend respectivement

en paramètres le nom du fichier binaire et le numéro entier d'identification à lui attribuer (Exemple : `SetSID emacs 3`). L'application *SetSID* vérifie la disponibilité du tampon en début de fichier, puis y écrit la valeur du *ScID* à une position *DSI_ELF_SID_POS* préalablement définie (sa valeur est fixée à 7).

- **Cas des processus** : l'application ***ChangeProcSID*** permet d'assigner et de changer l'identifiant de sécurité *ScID* d'un processus. Il prend deux paramètres qui sont : l'identifiant Linux *pid* du processus et le numéro *ScID* à assigner. Dans le fichier Linux <*linux/sched.h*>, une structure ***task_struct*** est associée à chaque processus; cette structure a un pointeur ***void *security*** qui dirige vers une autre structure ***task_security_t*** expressément définie pour stocker l'identifiant de sécurité du processus concerné (*int sid*). La structure ***task_security_t*** possède à son tour un pointeur ***void *task*** vers ***task_struct*** tel qu'illustré à la Figure 4.1 (a).

- **Cas des sockets** : la fonction ***internal_sk_buff_alloc_security (struct sk_buff *skb)*** permet d'assigner des identifiants de sécurité ScID aux *sockets* d'une façon similaire à ce qui est fait dans le cas des processus (Cf. Figure 4.1 (b)). La structure de base ***sk_buff*** est définie dans le fichier Linux <*linux/skbuff.h*>.

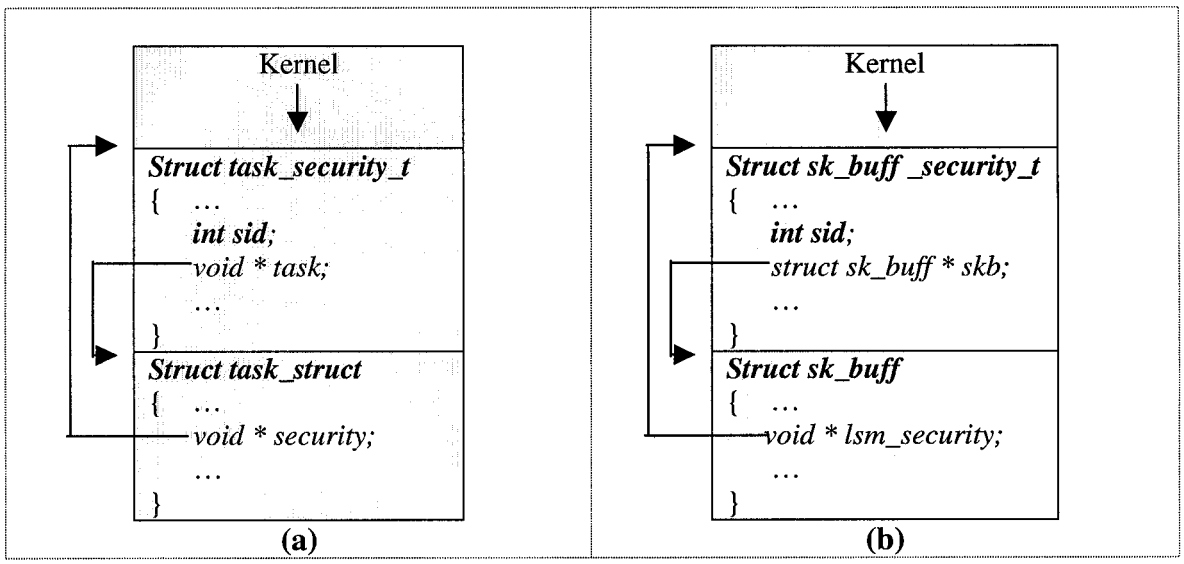



**Figure 4.1 Assignation des identifiants de sécurité pour les processus et les sockets**

### 4.1.2 Le Serveur de Sécurité

Le Serveur de Sécurité qui a été le siège de nos implémentations utilise le concept des « *threads* » afin de recevoir plusieurs connexions. Il est implémenté par une classe ***dsiSecServer*** dont le constructeur procède, entre autres, aux opérations suivantes : les initialisations pour le transport SSL, la création et la souscription à plusieurs canaux d'événements dont le canal de diffusion de la politique de sécurité aux Gestionnaires de Sécurité, de même que les canaux de communication de messages d'alarmes et d'avertissement. Les éléments d'implémentation liés à SSL et aux canaux CORBA seront présentés dans la section 4.1.3. La fonction principale du serveur initialise l'interpréteur du langage XDSPL et fait fonctionner le serveur grâce à sa méthode ***run()***. Nous présenterons dans cette sous-section la génération du fichier décrivant la politique de sécurité en XDSPL et l'implémentation de l'interpréteur associé.

#### 4.1.2.1 Interface de génération de la politique de sécurité en XDSPL

Une interface graphique (Cf. Figure 4.2) a été développée dans le langage TCL/TK [24] pour faciliter l'écriture et l'édition des fichiers de politique de sécurité en XDSPL, notamment pour les administrateurs de sécurité qui ne sont pas familiers avec le métalangage XML. Cette interface permet de spécifier le nom du fichier de politique de sécurité à créer (*'DSP file name'*). Elle rend actifs les champs à combler ainsi que les permissions possibles (*'Permissions'*), selon la classe de la règle de sécurité à écrire (*'Object class'*) .

- La fonction de rappel du bouton *'Save'* se charge de créer le fichier s'il n'existe pas et d'y écrire les règles de sécurité selon le formatage de XDSPL ;
- La fonction de rappel du bouton *'Assign ScIDs'* ne participe pas à l'écriture de la politique de sécurité mais fait plutôt appel à l'application ***SetSID*** (évoquée plus haut) pour assigner des identifiants de sécurité à des fichiers binaires (*'Binary'*) ;
- La fonction de rappel du bouton (*'Load DSP'*), quant à lui, fait appel à la fonction de diffusion de la politique de sécurité à tous les nœuds du réseau.

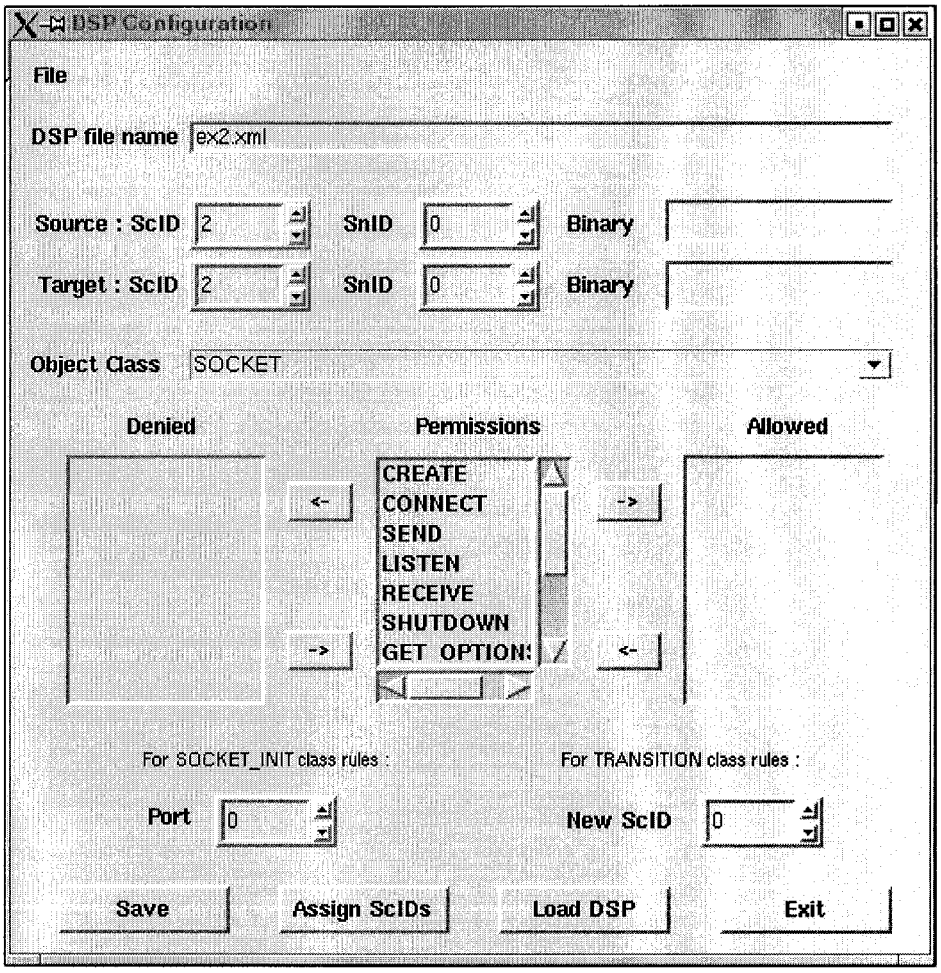



**Figure 4.2 Interface de génération de la politique de sécurité en XDSPL**

### 4.1.2.2 Interpréteur des règles de politique de sécurité

Une fois le fichier de politique de sécurité écrit ou généré via l'interface graphique, nous en assurons l'interprétation afin de le transformer dans une forme de représentation interne au réseau. Les classes qui implémentent cet interpréteur utilisent à la base, l'interface de programmation (API) *Xerces-C (version 2.1.0)* [25] développée par IBM pour le parcours de l'arborescence des fichiers XML, ainsi que l'extraction d'informations d'intérêt.

L'interface de programmation *Xerces-C* offre deux mécanismes d'accès aux informations contenues dans un fichier XML : DOM et SAX. Nous avons choisi de travailler avec DOM parce qu'il permet de construire une représentation de la structure des fichiers XML sous la forme d'un arbre, ce qui facilite beaucoup le parcours du fichier de politique de sécurité dans notre cas. SAX, par contre, est événementiel.

Plusieurs classes ont été utilisées pour l'implémentation de l'interpréteur de XDSPL :

- **La classe *dsiXMLDOMParser*** : il s'agit d'une classe générique indépendante de XDSPL. Son constructeur permet essentiellement une configuration de base de l'interpréteur (*parser*) : la spécification d'un espace de noms, d'un schéma XML servant de grammaire de validation, etc. Elle dispose d'une méthode abstraite ***parse(const *char)*** qui sera surchargée par toute classe qui hériterait d'elle. Les autres méthodes permettent d'afficher la structure arborescente des documents ou tampons XML qui seront traités par cette classe.
- **La classe *dsiXMLDOMFileParser*** dérive de la classe ***dsiXMLDOMParser*** et permet plus spécifiquement de traiter des fichiers XML, alors que la classe ***dsiXMLDOMParser*** permet en plus le traitement de simples tampons contenant des informations XML. Les classes sont représentées à la Figure 4.3. Leurs méthodes ***getDocument()*** permettent de récupérer l'arborescence XML sous la forme d'un objet de type ***DOMDocument***.
- **Les classes *dsiXMLException* et *dsiXMLErrorHandler*** représentées à la Figure 4.4 permettent respectivement de gérer les exceptions et les erreurs interceptées.
- **La classe *dsiXMLDSP*** est celle qui implémente de façon effective l'interprétation des fichiers de politique de sécurité écrite en XDSPL. Son rôle essentiel est de traduire toutes les règles de sécurité écrites en XDSPL sous la forme d'un tableau *m_RawDSP[][]* de plusieurs lignes de six (6) entiers; sur une ligne, chacune des colonnes de ce tableau est l'équivalent entier d'une valeur comprise entre deux balises d'une règle de sécurité. La classe est composée de plusieurs attributs et méthodes (Cf. Figure 4.5) dont les plus importantes sont :

- *m_Doc* : c'est l'objet de type *DOMDocument* qui représente le fichier de politique de sécurité sous la forme d'une arborescence ;
- *m_NbRules* représente le nombre de règles de sécurité calculé dans le document *m_Doc* grâce à la méthode *getNbRules()* ;
- *m_RawDSP[][][]* est un tableau qui contient autant de lignes qu'il y a de règles de sécurité, et 6 colonnes. Ces colonnes peuvent être remplies ou non d'entiers en fonction de la classe de règle traduite. Le Tableau 4.1 décrit les colonnes remplies en fonction des classes de règles de sécurité; à titre d'exemple, ce tableau contiendrait la séquence {5 4 2 2 1} pour stocker la règle (d) de l'exemple 3.3 ;
- *fillRawDSP(const char *rulename)* est une fonction qui prend en paramètre le numéro identifiant la classe de règle à traduire et se charge de parcourir toute l'arborescence de *m_Doc* afin de remplir le tableau *m_RawDSP[][][]* en conséquence; pour ce faire, cette fonction utilise les méthodes privées que sont : *setClass(), setID(), setSocket()* et *setPermission()* pour stocker respectivement l'entier identifiant la classe traitée, les identifiants de sécurité (sScID, sSnID, etc.), les informations d'initialisation de sockets et les permissions accordées ou non ;
- *read()* est une fonction qui fait appel à la fonction *fillRawDSP(const char *rulename)* pour chacune des classes de règles de sécurité afin de compléter l'écriture du tableau *m_RawDSP[][][] ;*
- *makeFirstDSMRule()* et *makeNextDSMRule()* sont des méthodes qui permettent chacune de fabriquer une chaîne de caractères contenant la séquence des entiers stockés sur une ligne du tableau *m_RawDSP[][][]*. Cette chaîne de caractères sera alors diffusée à tous les nœuds du réseau.

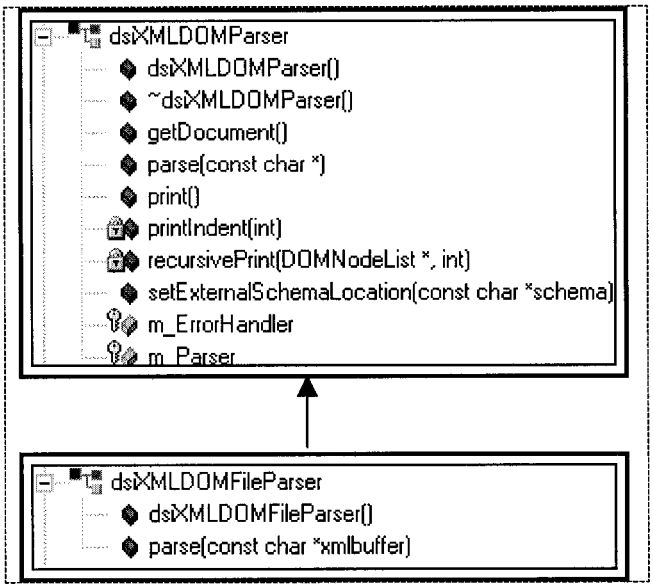



**Figure 4.3 Classes dsiXMLDOMParser et dsiXMLDOMFileParser**

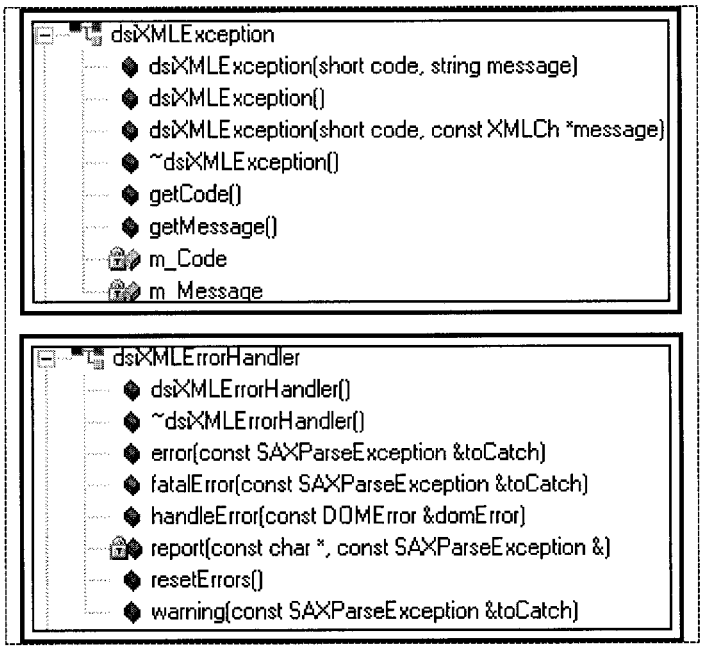



**Figure 4.4 Classes pour la gestion des exceptions et erreurs**

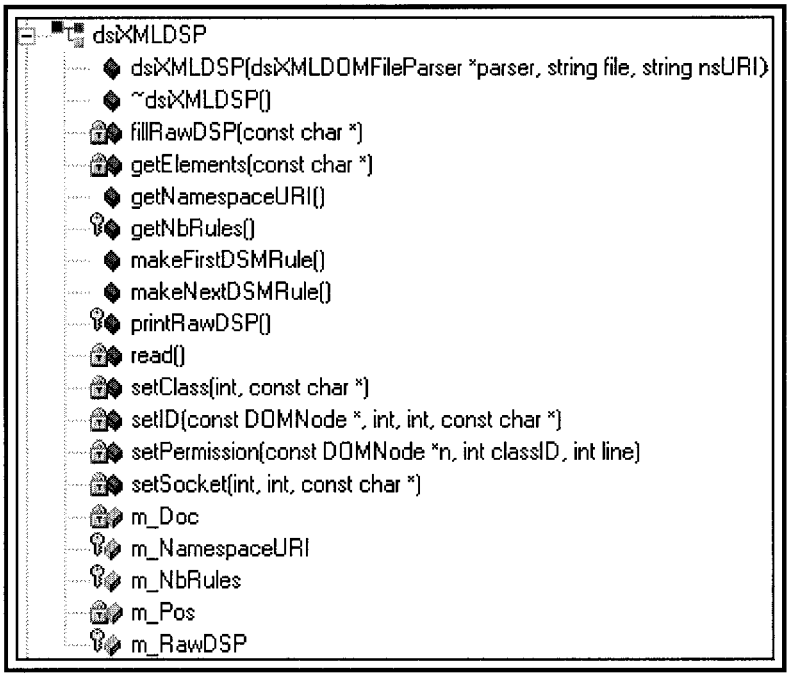



**Figure 4.5 Classe principale pour l'interprétation des fichiers écrits en XDSPL**

**Tableau 4.1 Remplissage des colonnes du tableau *m_RawDSP* selon la classe de règle de sécurité**

| Classe de règle de sécurité | Formatage brut des règles de sécurité | | | | | |
|---|---|---|---|---|---|---|
| *class_PROCESS_rule* | sScID | sSnID | | | N° classe | allow |
| *class_SOCKET_rule* | sScID | sSnID | tScID | tSnID | N° classe | allow |
| *class_SOCKET_INIT_rule* | sScID | sSnID | protocol | port | N° classe | |
| *class_NETWORK_rule* | sScID | sSnID | tScID | tSnID | N° classe | allow |
| *class_TRANSITION_rule* | parent_ScID | SnID | binary_ScID | new_ScID | N° classe | |
| *class_DisCl_rule* | sScID | sSnID | tScID | tSnID | N° classe | allow |

Note : Dans ce tableau, les noms des balises représentent la valeur que ces balises encadrent dans le fichier en XDSPL. Pour les balises qui encadrent des valeurs non entières, on a associé des entiers à ces valeurs lors de l'implémentation.

### 4.1.3 Le canal sécurisé de communication

Le canal de communication est réalisé grâce à *OmniORB-4.0.0* [26], une implémentation CORBA en C++. *OmniORB-4.0.0* inclut le service de noms (*OmniNames*) utilisé par le canal de communication. Le service d'événements, quant à lui, est basé sur l'utilisation de *OmniEvents* [27] qui est une implémentation de la spécification v1.0 du groupe *OMG*. Nous décrivons ci-après, du point de vue pratique, le mécanisme de diffusion de la politique de sécurité ainsi que la sécurisation du canal.

#### 4.1.3.1 Diffusion de la politique de sécurité via le canal CORBA

Dans le contexte de notre implémentation, un canal d'événements est en réalité un objet CORBA référencé par un IOR et sur lequel des opérations spécifiques peuvent être invoquées : typiquement, l'opération de diffusion (*push*) ou de retrait (*pull*). Des canaux d'événements sont créés (par une méthode ***CreateChannel (char *a_ChannelName)***) sur le Serveur de Sécurité (SS), parmi lesquels un canal dédié à la politique de sécurité (*DSP Channel*). Les Gestionnaires de Sécurité (GS) disponibles sur tous les nœuds du réseau souscrivent au canal *DSP Channel* (par une méthode ***Subscribe (char * a_ChannelName, Consumer_impl * a_Subscriber)***).

Le service de noms *OmniNames* maintient une table d'association entre les noms de canaux et leurs références (IOR) correspondantes; ceci permet de retrouver le canal de politique de sécurité par son nom *(DSP Channel)* plutôt que par sa référence. Le service d'événements *OmniEvents* contacte donc le service de noms lorsqu'il reçoit une invocation de méthode ***push()*** sur l'objet nommé *DSP Channel*.

La diffusion de la politique de sécurité aux Gestionnaires de Sécurité est effectuée sur le Serveur de Sécurité grâce à l'application ***dsiUpdatePolicy*** qui prend en paramètre le nom du fichier XML décrivant la politique de sécurité en XDSPL (Exemple : `dsiUpdatePolicy myXDSPLfile.xml`). Cette application fait appel à la méthode ***push()*** qui envoie à tous les nœuds, les règles de sécurité après interprétation, dans leur format brut décrit dans la sous-section 4.1.2.2.

#### 4.1.3.2 Sécurisation du canal d'événements

La sécurité du canal est assurée grâce à l'utilisation de *OpenSSL* [27] qui est une implémentation de SSL v2/v3 et de TLS v1. Nous avons créé un point terminal SSL en modifiant l'implémentation du service d'événements *OmniEvents* afin que ce dernier ne fonctionne que lorsque les instances qui le sollicitent sont configurées pour supporter le transport sécurisé SSL. Ainsi donc, tous les canaux d'événements (dont le canal de diffusion de la politique de sécurité) sont protégés. Tout ceci requiert la création de paires de certificats X509 autant sur le Serveur de Sécurité (SS) que sur tous les Gestionnaires de Sécurité (GS). Les certificats suivants ont été donc créés :

- *dsi_root.pem* : certificat de l'autorité « *root* » créé sur le SS et sur tous les GS et signé par l'autorité elle-même; dans un contexte réel, le certificat serait signé par une autre autorité de confiance connue (Exemple : *VeriSign*) ;
- *dsi_server.pem* : clef privée du SS et son certificat signé par l'autorité « *root* » ;
- *dsi_client.pem* : clef privée d'un GS et son certificat signé par l'autorité « *root* ».

Lors du démarrage du SS, nous spécifions dans la fonction principale, l'utilisation des certificats *dsi_root.pem* et *dsi_server.pem* tandis que dans le GS, ce sont les certificats *dsi_root.pem* et *dsi_client.pem* qui sont utilisés.

### 4.1.4 Les Gestionnaires de Sécurité

À l'instar du Serveur de Sécurité, un Gestionnaire de Sécurité utilise le concept des « *threads* ». Il est implémenté par la classe ***dsiSecManager***. Dans cette sous-section, nous nous intéresserons en particulier au module de gestion de la qualité de protection et au module de mise en application des règles de sécurité reçues du Serveur de Sécurité.

#### 4.1.4.1 Le module de gestion de QoP

Tel que décrit au chapitre 3, l'architecture de qualité de protection comprend, entre autres, un module de gestion de qualité de protection. Nous mentionnerons ici le

contexte d'implémentation et certains choix effectués, puis décrirons la classe principale qui réalise cette gestion.

- **Contexte et choix d'implémentation**

Nous avons considéré 3 niveaux de qualité de protection dans notre implémentation; ces niveaux correspondent aux différents protocoles d'encryptage (ou modes de *class_DisCI_rule* dans XDSPL) à utiliser par IPSec pour la communication entre deux entités du réseau : ESP et AH, puis NO_SEC pour indiquer l'absence d'encryptage. Les équivalences de niveau sont résumées au tableau 4.2.

Dans le Système de Gestion de Qualité de Protection (SGQoP) décrit au chapitre 3, l'*interface de QoP* pour applications ainsi que *l'Agent de QoP* sont des éléments qui collectent des informations (locales et distantes) à passer au module de qualité de protection qui siège dans le Gestionnaire de Sécurité. Nous avons choisi d'implémenter seul le module de gestion de QoP qui intègre en son sein une logique de décision dont l'algorithme a été présenté au chapitre 3. Ce faisant, nous avons rendu disponibles de façon locale à chaque GS toutes les informations utiles pour la prise de décision et qui devraient être récupérées du SS via des canaux sécurisés, sauf la politique de sécurité déjà diffusée et les requêtes de qualité de protection pour applications. En particulier, nous considérons comme requête de qualité de protection, toute modification effectuée dans la politique de sécurité au niveau des règles de confidentialité et d'intégrité identifiées par la balise *<class_DisCI_rule>*. Supposons par exemple que la politique de sécurité stipule qu'un socket [SnID= 3, ScID=5] doive communiquer avec un autre socket [SnID= 3, ScID=6] en utilisant le protocole AH; dès lors que l'administrateur de sécurité change la politique et demande dorénavant d'utiliser le protocole ESP, nous considérons la mise à jour de cette règle de politique de sécurité reçue du SS comme une requête pour passer du niveau de qualité de protection *MEDIUM* à *HIGH*. Pour ce faire, chaque fois qu'un GS reçoit une mise à jour de règle de sécurité du SS où il identifie que le numéro de la classe de règle correspond à celui de *class_DisCI_rule,* il passe cette règle au

module de gestion QoP afin que celui-ci décide si le passage à ce niveau de protection est autorisé ou pas, en fonction de la logique de décision. Le module de gestion de QoP est implémenté sous la forme d'une classe : ***qopModule.***

**Tableau 4.2 Niveaux de qualité de protection considérés**

| Niveaux de QoP | Abrégé des niveaux | Correspondance IPSec |
|---|---|---|
| LOW | L | NO_SEC |
| MEDIUM | M | AH |
| HIGH | H | ESP |

Note : LOW= basse sécurité / MEDIUM= sécurité intermédiaire / HIGH= sécurité élevée

- **La classe *qopModule* :**

Les éléments constitutifs de cette classe représentée à la Figure 4.6 sont des représentations des composants décrits au chapitre 3 :

- Le constructeur *qopModule(unsigned int dspRule[], int SC[], char qopFile[], char subscriptionsFile[])* prend en paramètre la nouvelle règle de politique de sécurité (*dspRule[]*) qui tient lieu de requête de QoP, ainsi que l'état du système représenté par le tableau d'entiers *SC[]* ; ce tableau contient 2 entiers : le premier signale s'il y a (valeur 1 ou 0) une alarme déclenchée dans le système par une tentative de violation de la politique de sécurité, et le second est un indicateur de message d'avertissement. Ces informations sont obtenues du module DSM (Cf. sous-section 4.1.4.2) qui met en œuvre la politique de sécurité dans le noyau Linux. Le constructeur reçoit aussi le nom de 2 fichiers : le premier (*qopFile*) mentionne les niveaux de QoP offerts avec les limites d'utilisation de ressources système associées (valeurs en %), tandis que le deuxième (*subscriptionsFile*) mentionne les souscriptions de certaines entités identifiées par la paire {*ScID, SnID*} à des niveaux de qualité de protection spécifiques (Cf. exemples de fichiers à la Figure 4.7).

- Nous avons choisi la mémoire comme ressource dont l'utilisation est limitative. La méthode *getMemUsage()* récupère l'utilisation courante de la mémoire en pourcentage et la stocke dans l'attribut *memUsageInfo*.
- La méthode *checkSubscription()* consulte le fichier des souscriptions (*subscriptionsFile*) pour déterminer si l'entité pour laquelle on demande un niveau de qualité de protection donné a souscrit ou non à ce niveau. Le résultat est stocké dans l'attribut *subscriptionsInfo*.
- La méthode *getResourceLimit()* consulte le fichier de spécification des niveaux de QoP (*qopFile*) pour connaître la limitation d'utilisation de mémoire associée au niveau de QoP demandé.
- Les attributs privés constituent essentiellement les informations du *Tableau de bord de QoP* décrit au chapitre 3 : la règle de politique de sécurité (*dspInfo*), le contexte de sécurité ou état du réseau (*securityContextInfo*), les souscriptions aux niveaux de QoP (*subscriptionsInfo*), les niveaux de QoP offerts (*availableQoPLevels*) et l'information sur les ressources systèmes (*memUsageInfo*).
- La requête de QoP est extraite de la règle de politique de sécurité par la méthode *getRequestedQoPLevel()*.
- La logique de décision du module est implémentée par la méthode *decide()* qui octroie ou non le niveau de QoP demandé.

Lorsqu'une décision positive est prise, le Gestionnaire de Sécurité envoie un ordre au module DSM dans le noyau Linux grâce à la fonction ***dsi_sys_security (unsigned int id, unsigned int call, unsigned long *args)***, afin de rendre effective la demande. Nous décrivons ci-après l'implémentation de ce module.

| | |
|---|---|
| L 10<br>M 15<br>H 99 | 1 1 H<br>4 3 H<br>2 3 M<br>3 6 H<br>5 8 L |
| # Exemple de fichier « *qopFile* » | # Exemple de fichier « *subscriptionsFile* » |

**Figure 4.6 Exemple de fichiers utilisés par le module de qualité de protection**

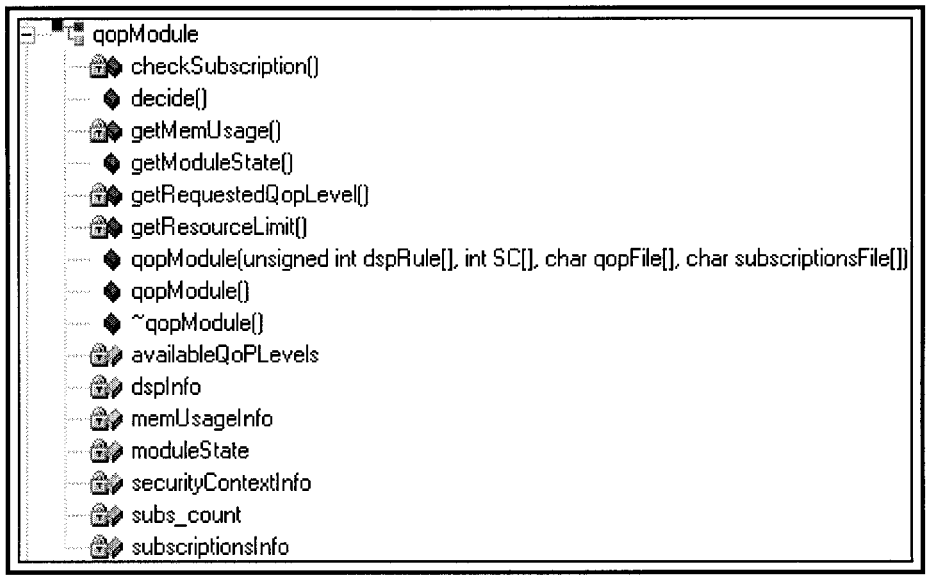



**Figure 4.7 Classe représentant le module de gestion de qualité de protection**

### 4.1.4.2 Le module de mise en œuvre de la politique de sécurité : DSM

Lorsqu'un Gestionnaire de Sécurité reçoit des règles de politique de sécurité provenant du Serveur de Sécurité, il les stocke dans une mémoire cache au sein du noyau Linux. Cette mémoire cache est implémentée sous la forme d'une liste chaînée (*struct dsi_cache_node*) qui stocke le contenu de chaque règle de sécurité dans le format natif (séquence de 6 entiers ou moins).

Afin de mettre en œuvre la politique de sécurité pour certains appels système, il est utile de procéder à des vérifications systématiques de sécurité avant de donner suite

ou non aux appels en question. Notre implémentation a touché en particulier les appels systèmes et opérations liées aux *sockets*, étant donné que dans l'architecture qui a été le siège de nos implémentations, les vérifications concernant les processus et binaires avaient déjà été prises en considération.

Nous avons travaillé avec la ***version 2.4.17 du noyau Linux***. Nous avons appliqué à ce noyau le ***patch LSM*** [28]. *LSM* est l'implémentation d'une série d'indirections appliquées à un noyau Linux pour des fins de vérification de sécurité ou de contrôle d'accès. Ainsi donc, lors de chaque appel de fonction dans l'espace usager, le système communique avec la table des appels systèmes afin de l'exécuter; mais avant même que le corps du programme représentant l'appel système soit exécuté, une indirection est effectuée au sein du module DSM (notre adaptation de LSM) afin de procéder à des vérifications de sécurité qui permettront d'autoriser ou non l'exécution des instructions réelles de l'appel système.

Le module DSM consulte le contenu de la mémoire cache (*dsi_cache*) afin de s'assurer que l'appel système est autorisé; rappelons ici que cette mémoire cache n'est rien d'autre qu'une représentation de la politique de sécurité dans un format natif compréhensible par tous les Gestionnaires de Sécurité. La Figure 4.8 illustre le cas de la fonction *send()* exécutée sur un *socket.* Un socket est identifié par la paire {ScID, SnID}, la valeur du ScID étant lue dans la structure ***sk_buff _security_t*** tel qu'indiqué à la Figure 4.1. Avant même que les instructions constituant l'appel *send()* soit exécutées ou non, le module DSM consulte la mémoire cache pour vérifier les droits accordés au socket qui procède; si le socket n'a pas le droit de procéder, le instructions correspondant à la fonction *send()* ne sont pas exécutées alors que dans le cas contraire, l'appel est exécuté comme d'habitude.

Les fonctions socket pour lesquelles des mécanismes de vérification de sécurité (contrôle d'accès) ont été implémentés sont les suivantes :

- **Fonction** ***create()*** : cette fonction permet de créer un socket spécifique d'un certain type, appartenant à une famille et capable d'échanger des paquets d'un

protocole donné. Les vérifications de sécurité préalables sont implémentées par la fonction : *dsi_socket_create (int family, int type, int protocol).*

- **Fonction *connect()*** : cette fonction permet à un socket créé de se connecter à une adresse distance ou locale. La fonction *dsi_socket_connect (struct socket *sock, struct sockaddr *address, int addrlen)* implémente les contrôles d'accès pour cette opération.
- **Fonction *listen()*** : elle permet à un socket d'écouter sur un port; les vérifications de sécurité associées sont réalisées par la fonction *dsi_socket_listen (struct socket *sock, int backlog)* où *backlog* indique la longueur maximale de la file d'attente.
- **Fonction *sendmsg()*** : cette fonction permet à un socket de transmettre des messages à un autre. Son exécution est postérieure à celle de la fonction *dsi_socket_connect (struct socket *sock, struct sockaddr *address, int addrlen)* qui vérifie si le socket est autorisé à opérer.
- **Fonction *getsockname()*** : elle permet simplement de récupérer le nom d'un socket local et l'indirection pour des fins de sécurité est effectuée vers la fonction *dsi_socket_getsockname (struct socket *sock).*
- **Fonction *getpeername()*** : similaire à la fonction précédente, elle permet de récupérer le nom d'un socket distant et sa fonction de contrôle des accès est *dsi_socket_getpeername (struct socket *sock).*
- **Fonction *setoptions()*** : cette fonction permet de spécifier les options associées à un socket après sa création et est soumise au contrôle préalable de la fonction *dsi_socket_setsockopt (struct socket *sock, int level, int optname).*
- **Fonction *getoptions()*** : elle est contrôlée par la fonction *dsi_socket_getsockopt (struct socket *sock, int level, int optname)* qui fait des vérifications de sécurité avant de récupérer les options d'un socket.
- **Fonction *receive()*** : c'est la function qui permet de recevoir des paquets d'un autre socket; elle est contrôlée par *dsi_socket_recvmsg(struct socket *sock, struct msghdr *msg, int size, int flags)* au niveau du module DSM.

- **Fonction *shutdown()*** : elle est contrôlée par la fonction *dsi_socket_shutdown (struct socket *sock, int how)* qui s'assure que le socket est autorisé à fermer sa connexion.

Outre le contrôle des accès, la confidentialité et l'intégrité des données échangées entre sockets sont assurées grâce au mécanisme de translation d'adresses réseau NAT (*Network Adress Translation*). Cette fonctionnalité est réalisée en assignant 3 adresses IP à chaque nœud du réseau. Ces adresses correspondent respectivement à des bouts de tunnels IPSec : une adresse AH, une adresse ESP et une adresse NO_SEC (sans sécurité). Ainsi donc, par exemple, lorsque la politique de sécurité stipule l'utilisation de ESP pour faire communiquer deux sockets sur deux nœuds différents, des modifications d'adresses IP sont effectuées au besoin afin de s'assurer que les adresses des deux nœuds communicants soient celles dédiées à l'établissement du tunnel ESP. Ce mécanisme est illustré à la Figure 4.9 et permet de mettre en œuvre les règles de sécurité encadrées par la balise <*class_DisCI_rule*> dans le langage XDSPL.

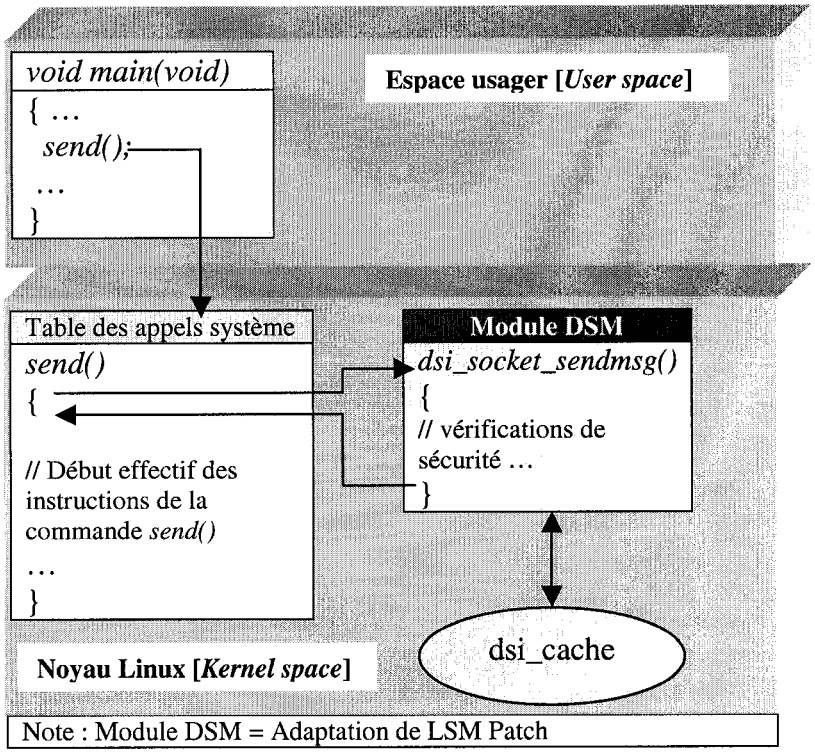



**Figure 4.8 Mécanisme des indirections pour les vérifications de sécurité lors d'un appel système**

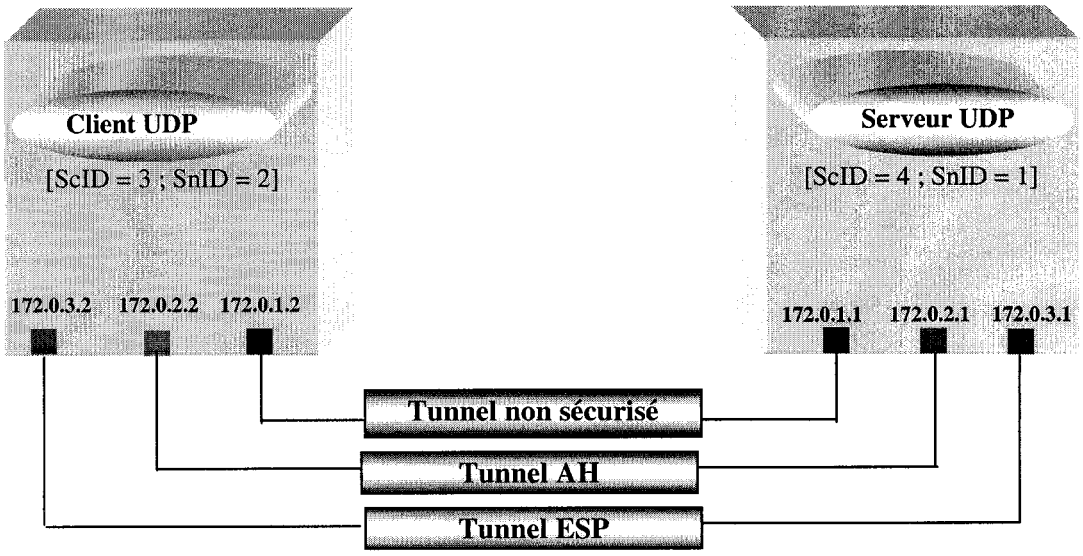



**Figure 4.9 Mécanisme de translation d'adresses réseau pour assurer la confidentialité et l'intégrité**

## 4.2 Évaluation, résultats et performances

Nous présentons dans cette section du chapitre, une évaluation du langage XDSPL au regard des objectifs fixés au départ et d'autres requis. Nous présenterons par la suite des résultats expérimentaux de la mise en œuvre de politiques de sécurité écrites en XDSPL et du module de gestion de la qualité de protection. Nous allons enfin comparer les performances du module de vérification des règles de sécurité (DSM) avec une approche similaire de la littérature.

### 4.2.1 Évaluation de XDSPL

*OASIS (Organization for the Advancement of Structured Information Standards)* est une organisation pour l'adoption de formats basés sur les standards *SGML, XML, HTML* et toute autre technologie de traitement d'information structurée [29]. Ce groupe a travaillé à l'élaboration de plusieurs standards (*XACML, XACL, SAML*, etc.), notamment dans le domaine des langages de contrôle d'accès utilisant XML. Nous avons choisi d'évaluer XDSPL en choisissant des critères qui sont inspirés des requis spécifiés par certains comités techniques de OASIS ([30], [31]) pour des langages d'expression de politiques de sécurité (essentiellement dédiés aux services Web) mais en nous référant aussi aux objectifs de départ fixés. Nous avons retenu les critères ci-après :

- **C1 –Définition du langage** : le langage doit être défini avec un schéma XML (à la différence d'une grammaire DTD) et les noms utilisés doivent émaner d'un espace de noms choisi ;
- **C2 –Définition précise de sémantique** : chaque expression écrite dans le langage doit avoir une sémantique précise et claire ;
- **C3 –Possibilité d'exprimer des conditions** : possibilité de définir des conditions de validité des assertions ;
- **C4 –Extensibilité** : possibilité de créer facilement de nouvelles assertions ;
- **C5 –Version** : possibilité de spécifier la version du langage utilisé ;
- **C6 –Cas imprévus** : interprétation en absence de certains attributs ou assertions ;

- **C7 – Facilité de compréhension** : la syntaxe doit être intuitive; la sémantique demeure celle précisée dans la définition du langage ;
- **C8 –Prise en compte de plus d'un service de sécurité** : possibilité de décrire des politiques de sécurité prenant en considération plusieurs services de sécurité ;
- **C9 –Prise en compte d'opérateurs booléens** : possibilité d'utilisations d'opérateurs booléens pour l'expression d'assertions complexes ;
- **C10 –Associations multiples (*bindings*)**: facilité d'encapsulation des informations extraites du langage, dans les protocoles de communication.

Une évaluation de XDSPL par rapport à ces critères est compilée au Tableau 4.3. La spécificité majeure de XDSPL par rapport aux formats d'expression de politique de sécurité existants est liée au critère C8.

**Tableau 4.3 Grille d'évaluation de XDSPL**

| Critères | Respect | Explications |
|---|---|---|
| C1 | ✓ | Cf. schéma à l'annexe B du mémoire. |
| C2 | ✓ | Clairement défini au chapitre 3 |
| C3 | ✗ | Non défini dans XDSPL |
| C4 | ✓ | Intrinsèque à l'utilisation de XML |
| C5 | ✓ | Pris en compte dans la balise de version |
| C6 | ✓ | Utilisation de comportements définis par les valeurs par défaut |
| C7 | ✓ | Syntaxe utilisant des balises dont les noms sont significatifs |
| C8 | ✓ | Services inclus : contrôle d'accès, confidentialité, intégrité |
| C9 | ✗ | Non défini dans XDSPL |
| C10 | ✓ | Les informations extraites sont des entiers |

**Légende :** ✗ = non respecté; ✓ = respecté

### 4.2.2 Résultats expérimentaux

Nous présentons ici des tests de fonctionnalité du module DSM (*Distributed Security Module*) qui permet de mettre en œuvre la politique de sécurité écrite en XDSPL, ainsi que du module de gestion de qualité de protection implémenté.

#### 4.2.2.1 Fonctionnalité de la politique de sécurité écrite en XDSPL

Nos implémentations ayant particulièrement porté sur les opérations liées aux sockets, nous avons choisi d'utiliser une application client sur une machine (*Colby*) communiquant avec une application serveur résidant sur une autre machine (*Munster*). Dans les communications socket, les applications serveurs et clients peuvent respectivement effectuer des séquences d'appels de fonctions bien définies. Ainsi donc, nous avons choisi ce qui suit :

- l'application **serveur** fait séquentiellement appel aux fonctions *create(), bind(), listen(), accept(), receive() et sendmsg()* ;
- l'application **client,** quant à elle, fait séquentiellement appel aux fonctions *create(), connect(), setsockopt(), getsockopt(), getsockname(), getpeername(), sendmsg(), receive(), shutdown().*

Nous avons par la suite généré un fichier écrit en XDSPL qui donne aux deux applications toutes les permissions pour exécuter tous les appels de fonctions. Nous avons par la suite édité ce même fichier en extrayant de façon séquentielle une permission à la fois pour chacune des applications; nous avons effectivement constaté que l'appel à la fonction pour laquelle la permission a été enlevée échoue. Pour certaines fonctions (disons $f_i$) , l'échec de leur exécution dû au retrait de droit d'accès se répercute de façon logique sur d'autres fonctions (disons $f_j$) dont l'exécution doit toujours être postérieure à celle des fonctions $f_i$; c'est le cas de la fonction *create()* sans laquelle aucune des autres fonctions ne peut s'exécuter.

Les deux applications ont également servi pour tester le mécanisme de translation d'adresses réseau décrit dans la sous-section 4.1.4.2. Nous avons associé à la machine *Colby* qui héberge l'application client, trois adresses IP sur des interfaces associées à la carte Ethernet: 142.133.**1.1** (*eth0 :1*), 142.133.**2.1** (*eth0 :2*), 142.133.**3.1** (*eth0 :3*); ces interfaces correspondent respectivement à des bouts de tunnels : non sécurisé (NO_SEC), sécurisé avec AH et sécurisé avec ESP. Il en a été de même pour la machine Munster qui héberge l'application serveur : ses trois adresses sont respectivement 142.133.**1.2** (*eth0 :1*), 142.133.**2.2** (*eth0 :2*), 142.133.**3.2** (*eth0 :3*). Ainsi donc, par exemple, faisons

remarquer que le couple d'adresses {142.133.**3.1**, 142.133.**3.2**} correspond à un tunnel ESP. Nous avons édité le fichier XML représentant la politique de sécurité de façon itérative pour autoriser d'abord une communication non sécurisée (NO_SEC), puis une communication avec authentification (AH) et enfin une communication où l'intégrité est assurée (ESP). Pour chaque type de fichier, nous avons considéré tous les cas de figures et constaté que l'adresse du client ou celui du serveur change au besoin afin de correspondre au tunnel dont l'utilisation a été définie par la politique de sécurité. Ces changements présentés au Tableau 4.4 sont constatés par un **plan d'expériences factoriel** sur *l'adresse du serveur*, *celle du client* et la *politique de sécurité choisie*, en utilisant un analyseur de trafic comme *Ethereal*.

**Tableau 4.4 Mécanisme de translation d'adresses IP : constat des changements**

| Adresses initiales | | Politique de sécurité | Changements d'adresses | |
|---|---|---|---|---|
| **Serveur** | **Client** | | **Serveur** | **Client** |
| 142.133.**1.2** | 142.133.**1.1** | NO_SEC | | |
| | | AH | 142.133.**2.2** | 142.133.**2.1** |
| | | ESP | 142.133.**3.2** | 142.133.**3.1** |
| | 142.133.**2.1** | NO_SEC | | 142.133.**1.1** |
| | | AH | 142.133.**2.2** | |
| | | ESP | 142.133.**3.2** | 142.133.**3.1** |
| | 142.133.**3.1** | NO_SEC | | 142.133.**1.1** |
| | | AH | 142.133.**2.2** | 142.133.**2.1** |
| | | ESP | 142.133.**3.2** | |
| 142.133.**2.2** | 142.133.**1.1** | NO_SEC | 142.133.**1.2** | |
| | | AH | | 142.133.**2.1** |
| | | ESP | 142.133.**3.2** | 142.133.**3.1** |
| | 142.133.**2.1** | NO_SEC | 142.133.**1.2** | 142.133.**1.1** |
| | | AH | | |
| | | ESP | 142.133.**3.2** | 142.133.**3.1** |
| | 142.133.**3.1** | NO_SEC | 142.133.**1.2** | 142.133.**1.1** |
| | | AH | | 142.133.**2.1** |
| | | ESP | 142.133.**3.2** | |
| 142.133.**3.2** | 142.133.**1.1** | NO_SEC | 142.133.**1.2** | |
| | | AH | 142.133.**2.2** | 142.133.**2.1** |
| | | ESP | | 142.133.**3.1** |
| | 142.133.**2.1** | NO_SEC | 142.133.**1.2** | 142.133.**1.1** |
| | | AH | 142.133.**2.2** | |
| | | ESP | | 142.133.**3.1** |
| | 142.133.**3.1** | NO_SEC | 142.133.**1.2** | 142.133.**1.1** |
| | | AH | 142.133.**2.2** | 142.133.**2.1** |
| | | ESP | | |

#### 4.2.2.2 Fonctionnalité et performance du module de gestion de QoP

L'implémentation faite du module de gestion de qualité de protection correspond essentiellement à l'application d'une logique de décision sur toute règle de sécurité ayant trait à la confidentialité et à l'intégrité (choix des tunnels IPSec). Vu la multitude des facteurs qui entrent en ligne de compte dans la prise de décision (limites en termes d'utilisation des ressources systèmes pour chaque niveau de QoP, toutes les souscriptions aux niveaux de QoP, l'état du réseau, la règle de politique de sécurité, etc.) et la simplicité de l'algorithme implémenté, nous avons opté pour un plan d'expériences aléatoire. Ce plan a consisté à définir, respectivement pour les niveaux de sécurité *H* et *L* des limites d'utilisation de la mémoire que nous sommes quasiment certains de dépasser et de ne pas atteindre; ceci a résulté au rejet de toutes les requêtes pour le niveau *H*. Par ailleurs, nous avons changé (de façon forcée) l'état du réseau en simulant un état d'alarme; dès lors, toutes les requêtes pour le niveau *L* qui étaient tantôt acceptées ont été rejetées en l'absence de la spécification d'une souscription dans le fichier *«subscriptionsFile»*.

L'implémentation de ce module sous la forme d'une simple classe C++ a conduit à un impact minime en terme de performance. Ce module induit une surcharge en temps système de l'ordre de quelques microsecondes et n'est sollicité uniquement qu'en présence d'une règle de confidentialité et d'intégrité de la politique de sécurité.

### 4.2.3 Performances comparées du module de vérification des règles de politique de sécurité

Nous avons mesuré les performances du module de vérification de sécurité DSM ajouté au noyau Linux avec l'outil *LMBench 3.0* [32] qui induit une interférence insignifiante. Les tests ont été localement effectués sur une machine Intel Pentium IV cadencée à 2.4 GHz.

Rappelons ici que le module DSM est en réalité une modification de *LSM Patch* auquel nous avons ajouté nos vérifications de sécurité. Cela nous a alors amené à

prendre des mesures pour deux configurations : une configuration de base comprenant juste *LSM Patch* appliqué au noyau, et une configuration DSM (*LSM Patch* combiné avec nos vérifications de sécurité). Nous avons par la suite mesuré le pourcentage de surcharge en temps système (*overhead*) réellement induit par nos mécanismes de sécurité pour différents types d'appels système et réseau. Les résultats sont compilés au Tableau 4.5 et correspondent à la moyenne de l'exécution de 10 sessions de tests *LMBench*.

**Tableau 4.5 Résultats de performance du module DSM**

| Type de test | Base (µsec) | DSM (µsec) | Surcharge (%) |
|---|---|---|---|
| **Fork** | 92,81 | 93,58 | **0,82%** |
| **Exec** | 322,56 | 328,33 | **1,78%** |
| **Sh proc** | 2150,00 | 2140,75 | **-0,43%** |
| **UDP** | 9,68 | 10,61 | **9,60%** |
| **RPC/UDP** | 17,66 | 18,70 | **5,90%** |
| **TCP** | 11,08 | 12,68 | **14,40%** |
| **RPC/TCP** | 23,42 | 24,30 | **3,75%** |

Les tests *Fork, Exec* et *Sh proc* sont des formes de création de processus qui sont assez coûteuses en temps d'exécution; les tests *UDP, RPC/UDP, TCP* et *RPC/TCP* correspondent à des échanges locaux de paquets de petite taille (4 octets) de ces protocoles respectifs. Les surcharges *TCP* et *UDP* sont plus considérables puisque des tests de sécurité sont effectués d'abord avant que les processus ne soient autorisés à communiquer, puis à la réception et à l'expédition de chaque paquet IP. Pour les tests utilisant *Sun RPC* au-dessus de *TCP* et *UDP*, étant donné que les temps absolus de communication sont plus significatifs, le pourcentage de surcharge est moins élevé.

Pour des fins de comparaison, nous avons recueilli des résultats d'un projet de l'*Agence de Sécurité Américaine (NSA)* dénommé *SELinux* (*Security-Enhanced Linux*) [33]. À l'instar de DSM, *SELinux* implémente au niveau du noyau Linux des mécanismes de contrôle d'accès en se basant aussi sur l'utilisation de *LSM Patch*. Des mesures de performance de *SELinux* (recueillis de [23]) réalisées avec *LMBench* sont

compilées au Tableau 4.6. Les tests de *SELinux* ont également été effectués pour deux configurations mais sur une machine Intel Pentium II à 333 MHz, ce qui justifie la valeur relativement élevée des temps absolus d'exécution (Tableau 4.6).

**Tableau 4.6 Résultats de performance de SELinux**

| Type de test | Base (μsec) | SELinux (μsec) | Surcharge (%) |
|---|---|---|---|
| **Fork** | 499 | 505 | **1%** |
| **Exec** | 2730 | 2820 | **3%** |
| **Sh proc** | 10000 | 11000 | **10%** |
| **UDP** | 310 | 356 | **15%** |
| **RPC/UDP** | 441 | 519 | **18%** |
| **TCP** | 389 | 425 | **9%** |
| **RPC/TCP** | 667 | 726 | **9%** |

Étant donné les différences significatives de performance entre les machines de tests respectivement utilisées pour DSM et *SELinux*, nous avons tracé des histogrammes comparés des pourcentages de surcharge à la Figure 4.10. Nous pouvons y constater que les performances de DSM sont acceptables.

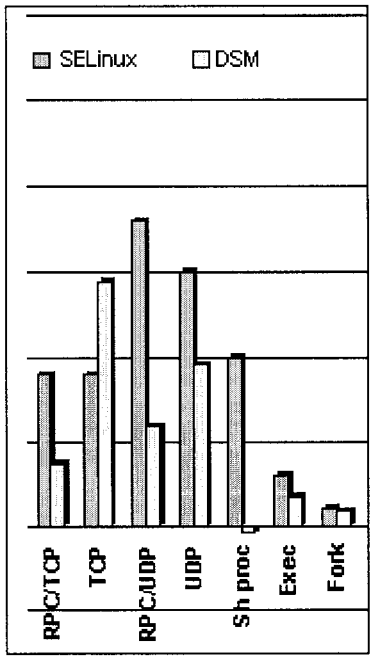



**Figure 4.10 Comparaison des pourcentages de surcharge en temps système entre DSM et SELinux**

# CHAPITRE 5

# CONCLUSION

Ce mémoire a essentiellement traité de la question de la représentation et de la mise en œuvre d'une politique de sécurité dans les grappes d'ordinateurs ainsi que des mécanismes d'ajustement du niveau de sécurité selon les besoins. Le présent chapitre résume l'ensemble des travaux et contributions, en signale les limitations et propose de nouvelles pistes de recherche et améliorations.

## 5.1 Synthèse des travaux

Nous avons tout d'abord présenté quelques solutions de représentation de politique de sécurité. Parmi les formats proposés, certains souffraient du manque d'extensibilité et la plupart se limitaient essentiellement à gérer le contrôle des accès qui n'est qu'un service de sécurité parmi une large gamme. Par ailleurs, nous avons présenté des travaux sur la possibilité d'offrir des niveaux variables de sécurité selon les besoins des applications ou des usagers : le concept de qualité de protection (QoP). Nous nous sommes inspirés de ces travaux pour proposer une architecture qui prenne en compte la qualité de protection dans un environnement réparti sur des grappes.

Après avoir sommairement décrit les technologies XML et IPSec qui ont servi de base à nos travaux, nous avons choisi d'utiliser le MAC comme mécanisme de contrôle des accès et d'offrir en plus, la possibilité d'exprimer des règles de sécurité ayant trait à la confidentialité et à l'intégrité. Le langage de représentation de politique de sécurité qui en a émané a été baptisé XDSPL (*eXtensible Distributed Security Policy Language*). Nous avons entièrement décrit sa grammaire au moyen d'un schéma XML formel, et la sémantique associée à sa syntaxe a été présentée pour chacune des balises qu'il utilise. Tous ces éléments sont graphiquement représentés en annexe à ce mémoire afin d'en faciliter la compréhension. Nous avons ensuite décrit les éléments constitutifs de

l'environnement de déploiement de la politique de sécurité ainsi que le mécanisme de diffusion, à proprement parler. La politique de sécurité est hébergée sur un Serveur de Sécurité qui l'interprète et l'envoie à tous les autres nœuds (hôtes de Gestionnaires de Sécurité) sous la forme d'événements de règles de sécurité via un canal sécurisé assurant l'interopérabilité avec CORBA.

Nous avons ensuite décrit les composants qui entrent en jeu dans la mise en place d'une architecture de gestion de la qualité de protection. L'architecture elle-même a été présentée ainsi que son déploiement dans l'environnement préalablement décrit pour la diffusion de la politique de sécurité. La logique de décision que cette architecture intègre a été présentée, de même qu'un protocole simplifié pour les requêtes de QoP.

Ces différentes propositions ont fait l'objet de plusieurs implémentations que nous avons présentées. Nous avons notamment décrit la manière dont les identifiants de sécurité sont attribués aux processus, binaires et sockets du système réparti. Nous avons ensuite présenté l'interface de génération de la politique de sécurité en XDSPL. Les classes qui implémentent l'interprétation des fichiers de politique de sécurité sur le Serveur de Sécurité ont été présentées, de même que la mise en place du canal sécurisé de communication d'un point de vue pratique. Au niveau de chaque Gestionnaire de Sécurité, nous avons implémenté un module de gestion de la qualité de protection auquel nous avons rendu disponible, de façon locale, toutes les informations utiles pour son fonctionnement. Le module DSM implémenté dans le noyau Linux et qui joue le rôle de « police de sécurité » a été présenté de même que les fonctions de sockets qu'il surveille. Un mécanisme de translations d'adresses IP pour offrir les services de confidentialité et d'intégrité au moyen de IPSec a été décrit.

Nous avons enfin évalué le langage XDSPL par rapport à des critères inspirés des requis de certains standards, mais aussi par rapport aux objectifs de départ fixés. Cette évaluation nous permet de proposer des améliorations dans la section 5.2. La présentation de nos résultats expérimentaux nous a permis de montrer la fonctionnalité du déploiement réel de la politique de sécurité ainsi que du module de gestion de QoP. Nous avons finalement évalué les performances du module DSM et l'avons comparé

avec une approche similaire dans la littérature afin de constater que les pourcentages de surcharge (en temps système) générée par ce module restent acceptables. Nos travaux ne sont cependant pas exempts de limites.

## 5.2 Limitations des travaux

Tel que le révèle l'évaluation de XDSPL effectué au chapitre 4 de ce mémoire, le langage que nous avons proposé n'offre pas de possibilités d'utilisation d'expressions booléennes; cela aurait permis d'exprimer des règles de politique de sécurité plus complexes, diminuant ainsi la longueur des fichiers XDSPL, mais probablement au détriment de la facilité de compréhension de la politique de sécurité. Par ailleurs, XDSPL ne permet pas d'exprimer des règles de sécurité conditionnées : par exemple, une règle pourrait être appliquée à la condition de vérifier un état donné ou l'horloge du système, etc.

Dans l'environnement de déploiement de la politique de sécurité, nous avons prévu de répliquer le Serveur de Sécurité; mais nos implémentations se sont effectuées sur un serveur non répliqué. Par ailleurs, nous n'avons pas abordé la question de la gestion des cas de pannes d'un nœud, d'un ou de plusieurs des liens du réseau.
La mise en œuvre des mécanismes de vérification de sécurité au coeur du système d'exploitation nous assure que toute application ou intergiciel qui fonctionne au dessus de lui bénéficie de la protection offerte. Mais cela suppose que ces mécanismes ne peuvent être déployés que sur une plate-forme ouverte aux modifications par l'administrateur principal (exemple de Linux).

Le mécanisme de translation d'adresses, décrit au chapitre 4, procède parfois à des changements d'adresses IP de la source et/ou de la destination qui communiquent, afin de les faire correspondre aux bouts de tunnel IPSec adéquats, conformément à la politique de sécurité. Mais cette modification d'adresses IP au niveau du noyau, totalement transparente aux applications, pourrait causer des problèmes d'incohérence lors de l'utilisation de protocoles de couches supérieures qui manipulent des adresses IP;

c'est typiquement le cas du protocole SIP (*Session Initiation Protocol*) qui utiliserait dans ses messages une adresse IP qui ne serait pas forcément celle que la machine a réellement au moment de la communication, ayant subi un changement. Le mécanisme a cependant été implémenté avec succès pour le protocole UDP.

## 5.3 Indications de travaux futurs

Un travail d'extension du langage XDSPL est envisageable afin de le pourvoir d'opérateurs de logique et de conditionnement des règles. L'extension peut également élargir davantage les services de sécurité pris en compte par la définition de règles pour l'audit, l'authentification, etc. Il est également possible de définir plus de règles de sécurité pour les services actuellement pris en compte par XDSPL.

Il serait intéressant de doter le module de gestion de la qualité de protection décrit au chapitre 3, d'un mécanisme pour garantir la qualité de protection pour laquelle certaines applications ou processus ont souscrit. Ceci pourrait être réalisé en s'inspirant des mécanismes existants pour garantir la qualité de service dans les réseaux.

Les performances du module DSM sont sujets à amélioration : d'autres approches pourraient être envisagées afin de minimiser davantage la surcharge en temps système induite par ce module de vérification dans le noyau du système d'exploitation. Enfin, afin d'être exhaustif, tous les mécanismes de vérification de sécurité pourraient être implémentés pour tous les appels système possibles.

# BIBLIOGRAPHIE

[1] G. Coulouris, J. Dollimore, et T. Kindberg, *Distributed Systems: Concepts and Design*, Third Edition, Addison-Wesley Publishers, Wokingham UK, 2001, ISBN 0-201-619-180, chapitre 7.

[2] M. Swift, P. Brundrett, C. Van Dyke, P. Garg, A. Hopkins, S. Chan, M. Goertzel, et G. Jensenworth, "Improving the granularity of access control in Windows NT", In *Proceedings of the 6th ACM Symposium On Access Control Models and Technologies*, Mai 2001, pp. 87-96.

[3] S. Ioannidis, A. Keromytis, S. Bellovin, et J. Smith, "Implementing a Distributed Firewall" In *Proceedings of Computer and Communications Security (CCS) 2000*, Novembre 2000, pp. 190-199.

[4] E. Bertino, S. Castano, E. Ferrari, et M. Mesiti, "On specifying security policies for web documents with an xml-based language", In *Proceedings of the 6th ACM Symposium On Access Control Models and Technologies*, Mai 2001, pp. 57-65.

[5] T. Ryutov, C. Neuman, "The specification and enforcement of advanced security policies", In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Juin 2002, pp. 128-138.

[6] N. Vuong, G. Smith, et Y. Deng, "Managing security policies in a distributed environment using extensible markup language", In *Proceedings of the 2001 ACM symposium on Applied computing*, 2001, pp. 405-411.

[7] K. Hwang, M. Gangadharan, "Micro-Firewalls for Dynamic Network Security with Distributed Intrusion Detection", In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'01)*, Octobre 2001, pp. 68-78.

[8] R. Oppliger, "Internet security: Firewalls and beyond", *Communications of the ACM*, 40(5):92--102, Mai 1997.

[9] S. Krakowiak, *Sécurité des systèmes répartis*, Laboratoire Sirac (INPG – INRIA-UJF) – Université Joseph Fourier, France. http://sardes.inrialpes.fr/people/krakowia/Enseignement/dess-gi/Flips/10b-securite.pdf, Septembre 2002

[10] SearchWebServices.com Definitions, *cluster computing*, http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci762034,00.html, 31 Dec. 2002.

[11] B. LENOU, *Services sur réseaux mobiles : architectures et maintenance*, Mémoire de Maîtrise, École Polytechnique de Montréal, Canada, Décembre 2000.

[12] K. Railsback, "Linux Clustering in Depth", In *Linux Magazine*, Octobre 2000. (http://www.linux-mag.com/2000-10/clustering_01.html).

[13] S. Taks, Y. Lee, E. K. Park, et J. Stach, 2001, "Design and Evaluation of Adaptive Secure Protocol for E-Commerce", *IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2001, pp. 32-39.

[14] P. Schneck, K. Schwan, "Dynamic Authentication for High-Performance Networked Applications", *1998 Sixth IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, Napa, CA., Mai 1998, pp. 127-136.

[15] E. Spyropoulou, T. Levin, et C. Irvine, "Calculating costs for quality of security service", *16th Annual Computer Security Applications Conference (ACSAC'00)*, Décembre 2000, pp. 334-342.

[16] C. Irvine, et T. Levin, "Toward Quality of Security Service in A Resource Management System Benefit Function", In *Proceedings of the 2000 Heterogeneous Computing Workshop*, Mai 2000, pp. 133-139.

[17] C. Irvine, et T. Levin, "Quality of Security Service", In *Proceedings of the New Security Paradigms Workshop, Cork, Ireland*, Septembre 2000, pp. 18-22.

[18] C. Irvine, et T. Levin, "Toward a Taxonomy and Costing Method for Security Services", In *the Proceedings of the 15th Computer Security Application Conference, Phoenix, AZ*, December 1999, pp. 183-187.

[19] Z. Liu, R. Campbell et M. Mickunas, "Security as Services in Active Networks", In *the Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, 2002, pp. 883-889.

[20] http://www.microsoft.com

[21] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, J. Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies", in *the Proceedings of the USENIX Security Symposium*, 1999, pp. 123-139.

[22] Dragovic, B. LinSec - *Linux Security Protection System University College London*, April 2002, http://www.linsec.org/doc/final.

[23] P. Loscocco, S. Smalley, "Integrating Flexible Support for Security Policies in the Linux Operating System", In *the Proceedings of the FREENIX track of the 2001 USENIX Annual Technical Conference*, 2001, pp. 29-42.
http://www.nsa.gov/selinux

[24] Langage TCL, http://www.tcl.tk

[25] Xerces-C++ Parser, http://xml.apache.org/xerces-c/index.html

[26] Free High Performance ORB, http://omniorb.sourceforge.net

[27] OpenSSL, http://www.openssl.org

[28] Linux Security Module (LSM), http://lsm.immunix.org

[29] OASIS (*Organization for the Advancement of Structured Information Standards*), http://www.oasis-open.org/home/index.php, Juin 2003

[30] Rights Language Technical Committee (RLTC), *Requirements*, http://www.oasis-open.org/committees/documents.php?wg_abbrev=rights, 27 Août 2002.

[31] OASIS Technical Committee, *Web-services policy language use-cases and requirements*, working draft, http://www.oasis-open.org/committees/xacml/docs, 16 Avril 2003.

[32] Mc Voy L., Staelin C. *LmBench, portable tools for performance analysis*, In *the Proceedings of the 1996 USENIX Annual Technical Conference*, 1996, pp. 279-294 http://www.bitmover.com/lmbench

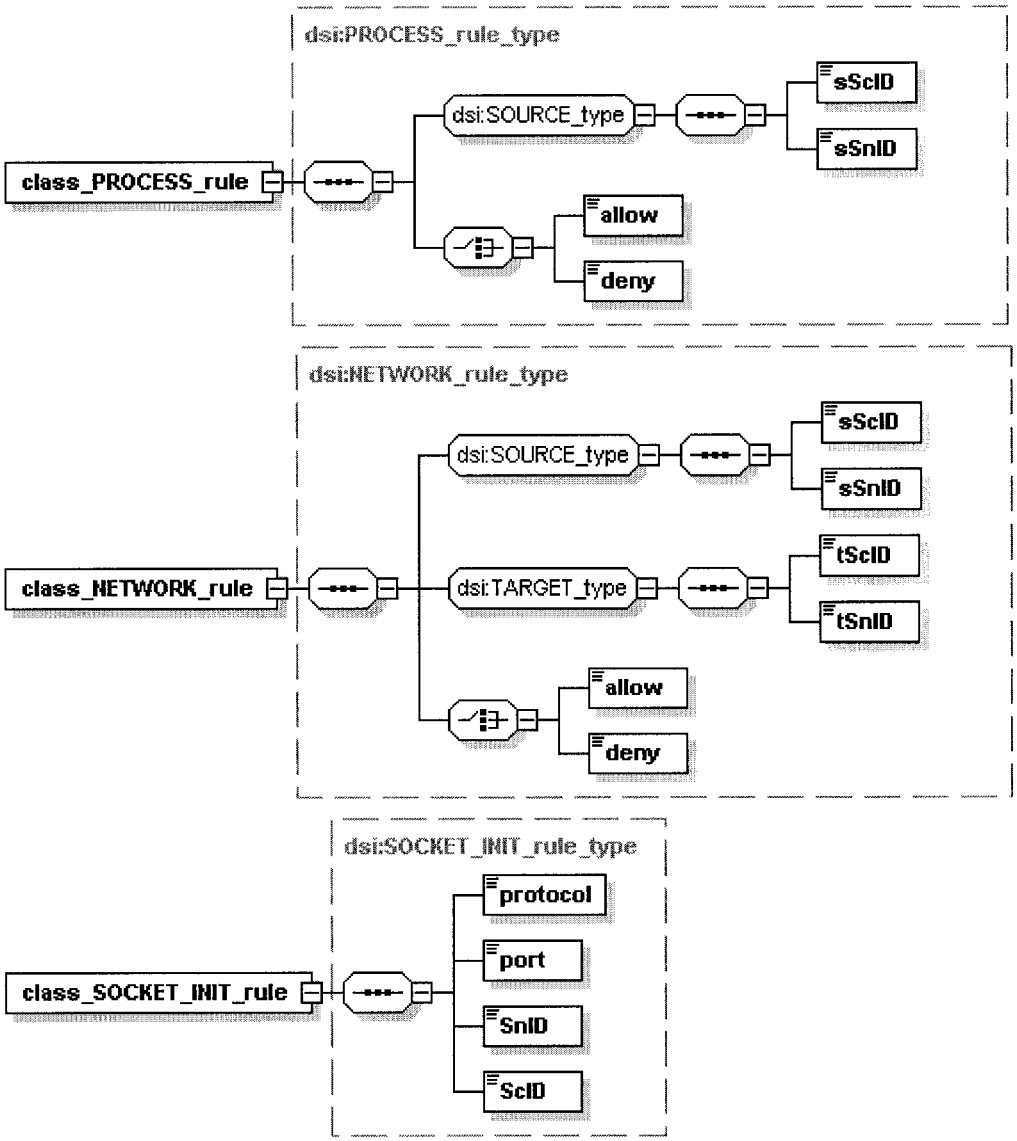
[33] Security-Enhanced Linux, http://www.nsa.gov/selinux

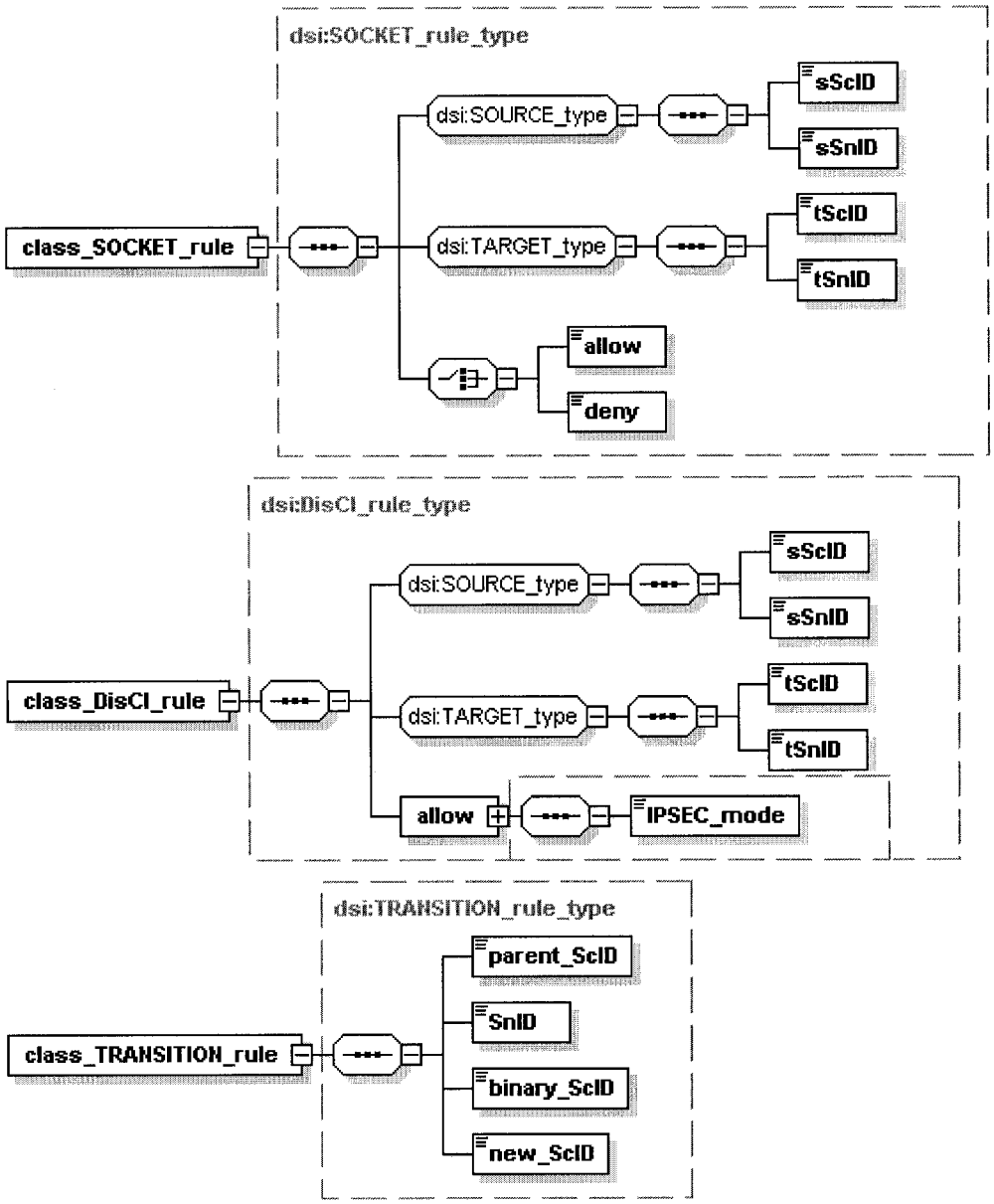
[34] World Wide Web Consortium, http://www.w3.org

# ANNEXE A

## Aperçu graphique de la structure des classes de règles de sécurité

Seuls les éléments en texte gras représentent des balises de XDSPL, les autres étant des éléments intermédiaires et modulaires utiles lors de la définition de la grammaire.




dsi:SOCKET_rule_type
class_SOCKET_rule
dsi:SOURCE_type
sScID
sSnID
dsi:TARGET_type
tScID
tSnID
allow
deny
dsi:DisCI_rule_type
class_DisCI_rule
dsi:SOURCE_type
sScID
sSnID
dsi:TARGET_type
tScID
tSnID
allow
IPSEC_mode
dsi:TRANSITION_rule_type
class_TRANSITION_rule
parent_ScID
SnID
binary_ScID
new_ScID

# ANNEXE B

## Schéma XML définissant la grammaire de XDSPL

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://sourceforge.net/projects/disec/DSP"
xmlns:dsi="http://sourceforge.net/projects/disec/DSP" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <import namespace="http://www.w3.org/2001/XMLSchema"/>
    <!-- ############################################################ -->
    <!-- ################# S_ID types ############################### -->
    <simpleType name="INTEGER_ID_type">
        <restriction base="nonNegativeInteger">
            <minInclusive value="0"/>
            <maxInclusive value="65535"/>
        </restriction>
    </simpleType>
    <simpleType name="ID_type">
        <union>
            <simpleType>
                <restriction base="nonNegativeInteger">
                    <minInclusive value="0"/>
                    <maxInclusive value="65535"/>
                </restriction>
            </simpleType>
            <simpleType>
                <restriction base="string">
                    <enumeration value="ALL"/>
                </restriction>
            </simpleType>
        </union>
    </simpleType>
    <group name="SOURCE_type">
        <sequence>
            <element name="sScID" type="dsi:ID_type"/>
            <element name="sSnID" type="dsi:ID_type"/>
        </sequence>
    </group>
    <group name="TARGET_type">
        <sequence>
            <element name="tScID" type="dsi:ID_type"/>
            <element name="tSnID" type="dsi:ID_type"/>
        </sequence>
    </group>
    <!-- ############################################################ -->
    <!-- ################## process rules types ##################### -->
    <simpleType name="PROCESS_permissions">
        <restriction base="string">
            <enumeration value="CREATE"/>
        </restriction>
    </simpleType>
    <simpleType name="PROCESS_permissions_type">
        <list itemType="PROCESS_permissions"/>
    </simpleType>
    <complexType name="PROCESS_rule_type">
```

```
    <sequence>
        <group ref="dsi:SOURCE_type"/>
        <choice>
            <element name="allow" type="dsi:PROCESS_permissions_type"/>
            <element name="deny" type="dsi:PROCESS_permissions_type"/>
        </choice>
    </sequence>
</complexType>
<!-- ############################################################ -->
<!-- ############### socket rules types ######################### -->
<simpleType name="SOCKET_permissions">
    <restriction base="string">
        <enumeration value="CREATE"/>
        <enumeration value="CONNECT"/>
        <enumeration value="SEND"/>
        <enumeration value="LISTEN"/>
        <enumeration value="RECEIVE"/>
        <enumeration value="SHUTDOWN"/>
        <enumeration value="GET_OPTIONS"/>
        <enumeration value="SET_OPTIONS"/>
        <enumeration value="GET_SOCKNAME"/>
        <enumeration value="GET_PEERNAME"/>
    </restriction>
</simpleType>
<simpleType name="SOCKET_permissions_type">
    <list itemType="SOCKET_permissions"/>
</simpleType>
<complexType name="SOCKET_rule_type">
    <sequence>
        <group ref="dsi:SOURCE_type"/>
        <group ref="dsi:TARGET_type"/>
        <choice>
            <element name="allow" type="dsi:SOCKET_permissions_type"/>
            <element name="deny" type="dsi:SOCKET_permissions_type"/>
        </choice>
    </sequence>
</complexType>
<!-- ############################################################ -->
<!-- ################### network rules types ################## -->
<simpleType name="NETWORK_permissions">
    <restriction base="string">
        <enumeration value="NETWORK_RECEIVE"/>
    </restriction>
</simpleType>
<simpleType name="NETWORK_permissions_type">
    <list itemType="NETWORK_permissions"/>
</simpleType>
<complexType name="NETWORK_rule_type">
    <sequence>
        <group ref="dsi:SOURCE_type"/>
        <group ref="dsi:TARGET_type"/>
        <choice>
            <element name="allow" type="dsi:NETWORK_permissions_type"/>
            <element name="deny" type="dsi:NETWORK_permissions_type"/>
        </choice>
    </sequence>
</complexType>
<!-- ############################################################ -->
<!-- ################### socket_init rules types ############## -->
<simpleType name="SOCKET_protocols_type">
    <restriction base="string">
        <enumeration value="TCP"/>
```

```
            <enumeration value="UDP"/>
            <enumeration value="RAW"/>
        </restriction>
    </simpleType>
    <simpleType name="SOCKET_ports_type">
        <restriction base="integer">
            <minInclusive value="1"/>
            <maxInclusive value="65536"/>
        </restriction>
    </simpleType>
    <complexType name="SOCKET_INIT_rule_type">
        <sequence>
            <element name="protocol" type="dsi:SOCKET_protocols_type"/>
            <element name="port" type="dsi:SOCKET_ports_type"/>
            <element name="SnID" type="dsi:INTEGER_ID_type"/>
            <element name="ScID" type="dsi:INTEGER_ID_type"/>
        </sequence>
    </complexType>
    <!-- ################################################################ -->
    <!-- ##################### DisCI rule types ##################### -->
    <simpleType name="IPSec_modes_type">
        <restriction base="string">
            <enumeration value="ESP"/>
            <enumeration value="AH"/>
            <enumeration value="NO_SEC"/>
        </restriction>
    </simpleType>
    <complexType name="DisCI_modes_type">
        <sequence>
            <element name="IPSec_mode" type="dsi:IPSec_modes_type"/>
        </sequence>
    </complexType>
    <complexType name="DisCI_rule_type">
        <sequence>
            <group ref="dsi:SOURCE_type"/>
            <group ref="dsi:TARGET_type"/>
            <element name="allow" type="dsi:DisCI_modes_type"/>
        </sequence>
    </complexType>
    <!-- ################################################################ -->
    <!-- ################### transition rules types ################### -->
    <complexType name="TRANSITION_rule_type">
        <sequence>
            <element name="parent_ScID" type="dsi:ID_type"/>
            <element name="SnID" type="dsi:ID_type"/>
            <element name="binary_ScID" type="dsi:ID_type"/>
            <element name="new_ScID" type="dsi:INTEGER_ID_type"/>
        </sequence>
    </complexType>
    <!-- ################################################################ -->
    <!-- ################### DSP elements types ###################### -->
    <complexType name="VERSION_type">
        <sequence>
            <element name="major" type="integer"/>
            <element name="minor" type="integer"/>
            <element name="date" type="date"/>
        </sequence>
    </complexType>
    <simpleType name="DSI_MODE_type">
        <restriction base="string">
            <enumeration value="PERMISSIVE"/>
            <enumeration value="ENFORCING"/>
```

```
            <enumeration value="PARANOID"/>
        </restriction>
    </simpleType>
    <complexType name="SECURITYRULES_type">
        <sequence minOccurs="0" maxOccurs="unbounded">
            <choice>
                <element name="class_PROCESS_rule" type="dsi:PROCESS_rule_type"/>
                <element name="class_SOCKET_rule" type="dsi:SOCKET_rule_type"/>
                <element name="class_NETWORK_rule" type="dsi:NETWORK_rule_type"/>
                <element name="class_SOCKET_INIT_rule" type="dsi:SOCKET_INIT_rule_type"/>
                <element name="class_DisCI_rule" type="dsi:DisCI_rule_type"/>
                <element name="class_TRANSITION_rule" type="dsi:TRANSITION_rule_type"/>
            </choice>
        </sequence>
    </complexType>
    <complexType name="DSI_POLICY_type">
        <sequence>
            <element name="version" type="dsi:VERSION_type"/>
            <element name="mode" type="dsi:DSI_MODE_type"/>
            <element name="default_ScID" type="dsi:INTEGER_ID_type"/>
            <element name="securityRules" type="dsi:SECURITYRULES_type"/>
        </sequence>
    </complexType>
    <complexType name="POLICY_type">
        <sequence>
            <element name="dsi_policy" type="dsi:DSI_POLICY_type"/>
        </sequence>
    </complexType>
    <element name="policy" type="dsi:POLICY_type"/>
    <!-- ################################################################ -->
</schema>
```