République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département Informatique

Projet de fin d'étude pour l'obtention du diplôme Master

Option

Sécurité des Systèmes Informatiques

Thème

Conception et réalisation d'un système standardisé de gestion de politiques de sécurité

Sujet Proposé par :

M : BOUABID Mohamed El Amine

Présenté par :

MALEK Mohamed
SEMMAR Mehdi

Soutenu le 20/06/2015

Devant le jury composé de :

M^me	F.KHELLAF	Présidente
M^me	C.BENZAID	Membre

Binôme N° 034/2015/SSI

Remerciements

Nous tenons à remercier tout d'abord notre promoteur Dr. Mohamed El Amine BOUABID, d'avoir accepté de nous encadrer durant ce projet, de nous avoir beaucoup aidé et soutenu, pour sa disponibilité, sa patience, ses conseils, et surtout son encouragement durant les moment difficiles.

Nous remercions Dr. F. KHELLAF d'avoir accepté de présider le jury de soutenance de ce projet.

Nous adressons toute notre gratitude et reconnaissance au Dr. C. BENZAID, qui a accepté de faire parti de ce jury, qui nous a beaucoup appris en tant qu'enseignante et co-promotrice.

Nous désirons encore exprimer notre très vive reconnaissance au enseignants de l'USTHB qui nous ont aidés et encouragés, de près ou de loin.

Je remercie toute ma famille, ma mère et mon père pour leurs conseilles, leur soutien inconditionnel, leur contribution et leur patience, ma sœur Manel qui rend ma vie heureuse, mes frères Akram et Anes, en leurs souhaitant une vie pleine de bonheur. Je remercie Mr. Messaoudi qui m'a guidé pour être ce que je suis, pour ses conseils et son inquiétude pour mes études. Je remercie SEMMAR Mehdi mon binôme pour sa disponibilité et les beaux moments passés au cour de notre projet, tous mes amis Hamza, Yacine, Amine et mon frère Issam.

M.MALEK

Mes remerciements infinis à ma Mère et mon Père, ma tante, ma sœur Maya et son fils Diaa Eddine ainsi que son époux Mohamed, tata Nabila, qui sont ma raison de vivre et qui m'ont tous soutenu pour être ce que je suis aujourd'hui. Je tien aussi à remercier mes oncles, cousins ainsi que tous mes amis. Je voudrais, en particulier, exprimer mes remerciements à mes amis Mohamed, Yacine, Fethia, Meriem, Nabile, Ayoub, Amine, Akram, Malek, Sekoura, Amer et Hani, pour les moments inoubliables passés ensemble.

M.SEMMAR

Résumé

La gestion des politiques de sécurité est devenue de nos jours, un enjeu de taille pour les directions de systèmes d'informations. Avec l'évolution exponentielle des attaques, toute négligence risque de causer de grands dégâts, que ce soit sur le plan économique, politique ou même humain. Ainsi, pour garantir un maximum de sécurité, les grands leaders informatiques ne cessent de proposer et d'améliorer des nouveaux systèmes de sécurité.

Dans le cadre de notre projet, nous avons réalisé un système dédié à la gestion de politiques de sécurité au niveau d'un système d'exploitation Linux doté du module de contrôle d'accès obligatoire (MAC) SELinux en se basant sur le standard CIM/WBEM. Notre système permet de modéliser d'une manière abstraite des politiques traduites en règles SELinux simples (allow, auditallow, etc.), en modules et booléens SELinux ainsi qu'en règles de contrôle d'accès aux ports réseaux. Notre système repose sur une architecture adaptant les composants WBEM élémentaires à l'architecture standard d'un système de gestion de politiques (comme COPS). La traduction CIM/SELinux est réalisée par le *Provider* CIM-SELinux que nous avons développé et qui représente le noyau de notre travail car il assure d'un coté la traduction CIM de/vers SELinux et d'un autre coté le renforcement des politiques modélisées. Nous avons développé également un client CIM/WBEM permettant de faciliter la gestion des politiques à travers une interface graphique simple et conviviale.

Notre contribution représente une brique essentielle dans un écosystème de composants logicielles permettant d'assurer une gestion de politiques de sécurité abstraites et indépendantes des plate-formes en se basant sur un standard de gestion largement adopté et déployé (CIM/WBEM) L'efficacité d'un tel écosystème est tributaire de l'adaptation du même standard pour d'autres systèmes de sécurité (comme AppArmor, GrSecure, XACML, etc.) et ce défi représente la perspective de notre travail.

Abstract

Security policy management has become today a major challenge for the supervision of information systems. With the exponential growth of the attacks, any negligence can cause great damages, whether economical, political or even human. Thus, to ensure maximum safety, large IT leaders constantly suggest and improve new security systems.

As part of our project, we realized a system dedicated to managing security policies on operating system level with the *Mandatary Access Control* module SELinux based on CIM/WBEM standard. Our system allows abstractly modeling policies translated in various SELinux policies as simple rules (allow, auditallow, etc.) SELinux modules and booleans and control access rules over network ports. Our system is based on an architecture mapping WBEM elementary components to a standard policy management architecture (like COPS) The CIM-SELinux Provider we have been developed represents the core of our work since it ensures in one hand, CIM/SELinux translation and in the other hand policy enforcement. We have also developed a CIM/WBEM client to facilitate policy management through a simple and user-friendly graphical interface.

Our contribution is an essential building block in an ecosystem of software components ensuring the management of abstract and platform independent policies based on a widely adopted and deployed management standard (CIM/WBEM). The efficiency of such an ecosystem is dependent on the adaptation of the same standard for other security systems (like AppArmor, GrSecure, XACML, etc.) and this challenge represents a future work.

Sommaire

Liste des figures

INTRODUCTION

Contexte

Les réseaux et les systèmes informatiques prennent de plus en plus une place stratégique au sein des entreprises et des organisations. Ainsi l'atteinte à la sécurité de ses derniers est devenue quasi inacceptable. Ceci implique une prise en charge de la sécurité dès la phase de conception du système d'information par l'application de tous les principes, standards et bonnes pratiques liés à la sécurité.

Une fois que le système d'information est opérationnel, il faut définir des politiques de sécurité sous la forme de charte à (faire) respecter par tous les membres de l'organisation, ses partenaires, fournisseurs, clients et autres collaborateurs.

Ces politiques « réglementaires » doivent être traduites en termes de règles de contrôle d'accès et de traçage permettant de prévenir contre d'éventuels accès malveillants et d'auditer ces tentatives. Cette tache est rendue compliquée par l'expansion des architectures et la distribution massive des systèmes, services et applications qui sont en plus fortement hétérogènes.

Ceci rend nécessaire l'élaboration de standards ouverts dédiés à la conception, composition, distribution et application des politiques de sécurités sur tout le réseau. Différentes initiatives ont été proposées, notamment le standard XACML (consortium OASIS) et CIM Policy/CIM-SPL (DMTF) Malheureusement l'élaboration de tels standards se heurte à la réalité des systèmes de sécurité existants qui sont aussi divers qu'hétérogènes et opérant chacun à un niveau particulier de la sécurité : contrôle d'authentification, contrôle d'autorisation, contrôle de comptabilité, filtrage, etc.

Objectif du projet

L'objectif de ce projet est de contribuer à l'adaptation d'un standard largement adopté par les industriels IT, en l'occurrence CIM/WBEM (en particulier le profile CIM Policy et CIM-SPL) à un système de sécurité largement déployé dans les systèmes d'exploitation de la famille Linux, en l'occurrence SELinux.

Un tel travail devrait produire une brique dans un écosystème qui va permettre une gestion homogène et unifiée des politiques de sécurité sur toute l'infrastructure IT des organisations.

L'approche CIM-WBEM est le fruit de collaboration d'un grand nombre de constructeurs et éditeurs informatiques dont notamment Cisco, Dell, VMware, HP, IBM, Intel, Microsoft et Oracle entre autres. Ces entreprises ont fondé le **DMTF *(Distributed Management Task Force)*** une organisation qui se charge du développement et du maintien des standards pour l'administration des systèmes informatiques. Cette approche se base sur deux standards :

- **CIM *(Common Information Model)***, qui est un standard ouvert qui permet de modéliser n'importe quelle entité gérable;
- **WBEM *(Web-Based Enterprise Management)***, qui est un ensemble de techniques et de standards qui servent à unifier la gestion des systèmes distribués;

Le standard CIM-WBEM a été choisi dans le cadre de ce projet, pour son caractère orienté modèle, l'exhaustivité des modèles qu'il propose et la richesse des domaines technologiques qu'il couvre. En effet, le DMTF propose des profils bien définis qui offrent, aux développeurs souhaitant réaliser n'importe quel système de gestion standardisé, des modèles abstraits à utiliser comme squelette de départ dont il faudrait étendre et adapter aux besoins spécifiques.

Par ailleurs, le système SELinux représente une solution de renforcement de sécurité robuste et souple (au prix d'une complexité à gérer) présent nativement sur la plupart des systèmes Linux dérivés de la distribution RedHat (Centos, Fedora, ScientificLinux etc.) et pouvant être exploité sur tout système Linux

La réalisation de ce système standardisé a comme objectif la contribution à la normalisation des politiques de sécurité sur deux plans :

- Traduction de politiques exprimées en **langage SELinux** en politiques sous format **CIM**, ce qui va permettre, entre autres, de faciliter le partage et l'échange entre plusieurs administrateurs.
- Intégrer le système **SELinux**, dans un dispositif global normalisé d'application de politiques de sécurité.

Ce travail devrait permettre également la centralisation de la gestion de tous les systèmes qui mettent œuvre des politiques SELinux dans le parc informatique.

Organisation du manuscrit

Dans cette partie introductive, nous avons expliqué le contexte de notre travail avant de parler de notre objectif. Nous présentons dans le chapitre suivant le domaine la gestion des réseaux et des systèmes de manière générale, et expliquer les différentes approches et standards permettant de l'assurer. Dans le deuxième chapitre, nous introduisons le standard de gestion adopté dans notre projet, en l'occurrence CIM/WBEM. Dans le troisième chapitre, nous détaillons le principe de fonctionnement de SELinux, qui représente le système de contrôle d'accès que nous utilisons pour concrétiser nos politiques de sécurité. Le quatrième chapitre présente la conception de notre système de gestion. Nous présentons l'architecture générale de notre dispositif de gestion des politiques ainsi que des éléments de modélisation de ces politiques fondés sur des profiles spécialisés du standard CIM. Dans le dernier chapitre, nous présentons les détails de la réalisation de notre système ainsi qu'un plan de test. Nous terminons ce document par une conclusion générale ainsi que quelques perspectives.

CHAPITRE 1

GESTION DES RÉSEAUX ET DES SYSTÈMES

1.1 Introduction

Le début des années 1980 a connu une expansion remarquable dans le domaine des réseaux informatiques. Ainsi, les entreprises ont réalisé l'importance des gains rapportés grâce aux technologies réseaux. C'est ce qui les a poussées à élargir leurs réseaux et à mettre en œuvre d'autres plus performants, aussi rapidement que les nouvelles technologies apparaissaient.

Au milieu des années 1980, certaines entreprises éprouvaient de grandes difficultés d'exploitation, à cause du déploiement de plusieurs nouvelles technologies réseaux. Étant donné que chaque technologie exigeait son propre groupe d'experts, ce qui a conduit à un manque de personnel notable pour la gestion de leurs grands réseaux hétérogènes. Ces difficultés ont affecté non seulement les activités quotidiennes de gestion des réseaux, mais aussi les plans stratégiques de ces entreprises. Dans ces circonstances, l'automatisation de la gestion des réseaux et systèmes est devenue un besoin urgent.

1.2 Concepts et principe de la gestion

1.2.1 Définition

La gestion des réseaux et systèmes peut faire référence à différents concepts. Les définitions les plus significatives sont celles proposées par des organismes de standardisation, qui utilisent des terminologies spécifiques aux domaines d'applications.

En général, la gestion des réseaux et systèmes est définie comme étant **l'ensemble des activités, méthodes et techniques, permettant de mettre en œuvre, exploiter et administrer un réseau informatique**[1][23][36].

1.2.2 Les activités de la gestion des réseaux et systèmes

Les objectifs de la gestion des réseaux peuvent se résumer en deux points : **la satisfaction des utilisateurs** et **la facilitation du travail des administrateurs.** Afin d'atteindre ces objectifs, la gestion des réseaux doit intégrer trois principales activités [23][36] :

1.2.2.1 La supervision

L'activité de supervision consiste à surveiller les systèmes et à récupérer des informations sur leur état et leur comportement afin d'assurer leur bon fonctionnement. La supervision devait être capable de s'adapter à des milieux hétérogènes, d'automatiser le contrôle des réseaux et de générer un ensemble de statistiques donnant une meilleure vision du réseau et permettant d'anticiper les besoins de celui-ci.

La supervision peut ainsi se définir comme étant l'utilisation de ressources réseaux adaptées (matérielles ou logicielles) afin d'obtenir des informations sur l'utilisation et sur l'état des composants des réseaux.

La supervision est capable de diagnostiquer et bien souvent de réparer les pannes. Si ce n'est pas le cas, elle se charge d'alerter immédiatement les administrateurs. Elle est donc réactive et assure un gain important en temps. De plus, par sa vision continue du réseau, elle anticipe souvent sur des problèmes ultérieurs. On parle alors de pro-activité.

Ainsi, la supervision est à la fois réactive et proactive, c'est pourquoi, elle s'impose comme une des activités principales de la gestion des réseaux.

1.2.2.2 L'administration

L'administration réseau désigne l'ensemble des opérations de contrôle du réseau ainsi que la gestion des configurations et de la sécurité.

Selon les entités qu'elle gère, l'administration réseau peut se diviser en trois types [36]:

- L'administration des utilisateurs : elle fournit l'ensemble des mécanismes qui permettent à un utilisateur d'utiliser le réseau. Ces mécanismes doivent assurer :

- La connectivité des utilisateurs, qui assure l'accès aux différentes applications fournies par le réseau et met à disposition un ensemble d'outils leur assurant une certaine transparence au niveau des méthodes d'accès et de connexion aux applications.
- La sécurité, à travers des mécanismes qui permettent de garantir l'authentification des utilisateurs, la confidentialité des informations échangées, la sécurité de leurs environnements et la prévention contre toute perte ou altération de ces informations.
- La qualité de service, qui se traduit principalement par les performances du système et sa capacité d'assurer le niveau de service attendu (et/ou convenu).

- L'administration des serveurs : elle fournit l'ensemble des mécanismes qui permettent d'assurer :

- La connexion et la distribution des applications hébergées dans le réseau, afin d'assurer la relation entre les différents services.
- La gestion et la distribution des données, qui est le rôle des outils de transfert de fichiers, qui permettent le partage des supports de stockage entre plusieurs systèmes
- La gestion des applications, est essentiellement liée au contrôle et à la protection des accès relatifs aux applications présentes dans les serveurs.

- L'administration des équipements réseaux : elle permet de fournir :

- Les opérations pour l'intervention sur le fonctionnement et la modification du réseau.
- La (re)configuration des équipements afin d'améliorer les performances et la qualité du service.
- La gestion des performances du réseau, pour pou/voir évaluer le système par un ensemble de métriques (temps de réponse, charge du réseau, etc.).
- L'inventaire, qui a pour rôle de tenir à jour et en temps réel la liste des éléments qui constituent le réseau.

1.2.2.3 Exploitation

Au niveau d'un réseau, les activités d'exploitation sont assurées via des équipements terminaux (ordinateurs, smartphones, tablettes etc.). Ces équipements exécutent des applications qui font appel à des protocoles implémentés par les systèmes d'exploitation, afin d'exploiter les réseaux et

leurs services. De nos jours, ces systèmes gèrent tous les aspects d'exploitation par le biais d'un ensemble de protocoles, à l'instar de la pile TCP/IP qui est implémentée partout.

1.3 Les systèmes de gestion de réseaux

1.3.1 Définition

Un système de gestion de réseaux est **un ensemble d'outils informatiques [23][36]** qui permettent aux administrateurs de réaliser toutes les tâches relatives à cette fonction.

1.3.2 Les architectures des systèmes de gestion de réseaux

Les systèmes de gestion de réseaux sont construits sur deux entités fondamentales : le **manager** (gestionnaire) et **l'agent**. Un manager est un composant assurant un certain nombre de fonctions de gestion adressant des besoins du système de gestion global. Pour ce faire, le manager procède à l'émission de requêtes à l'agent. L'agent, étant en contact direct avec l'entité gérée, exécute et répond à ces requêtes.

Un système de gestion de réseaux peut être conçu selon trois différentes architectures [1][23] : centralisée, distribuée plate et distribuée hiérarchique Le choix d'une architecture dépend de plusieurs paramètres, comme par exemple le coût de l'implémentation et les besoins exprimés par les gestionnaires.

1.3.2.1 Architecture centralisée

C'est l'organisation la plus classique de la gestion, dans laquelle un seul manager contrôle toutes les ressources du réseau (en interagissant avec les agents qui y sont présents). L'avantage de cette architecture est sa facilité de conception et d'implémentation. Cependant, elle est inefficace dans le cas de réseaux étendus. La figure 1-1 présente cette architecture.

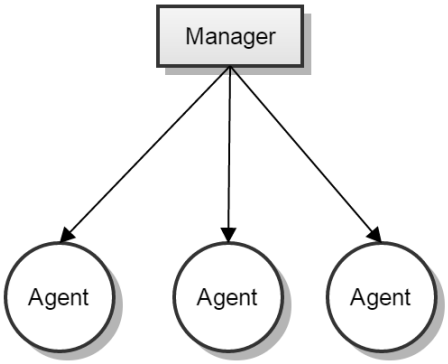

Figure 1-1. Architecture de gestion centralisée [23].

1.3.2.1 Architecture distribuée

C'est l'architecture dans laquelle plusieurs managers coopèrent pour gérer toutes les ressources du réseau. Cette architecture implique une forte interaction entre les managers.

Deux types d'architectures distribuées existent [23] :

- Architecture plate : dans laquelle les managers sont tous dans un même niveau hiérarchique. Pour gérer l'intégrité du réseau, chaque manager est responsable d'un ensemble de

- ressources. Si un manager a besoins d'informations sur des ressources qu'il ne gère pas directement, il envoie des requêtes aux managers concernés qui, à leurs tours, envoient des requêtes aux agents qu'ils gèrent et retransmet le résultat au manager qui l'à demandé.
- Architecture hiérarchique : Les managers sont dans des niveaux différents. Les managers dans un niveau N sont considérés comme des agents par les managers de niveau N+1. Autrement dit, un manager dans un niveau fait appel aux managers dont les niveaux sont inférieurs au sien.

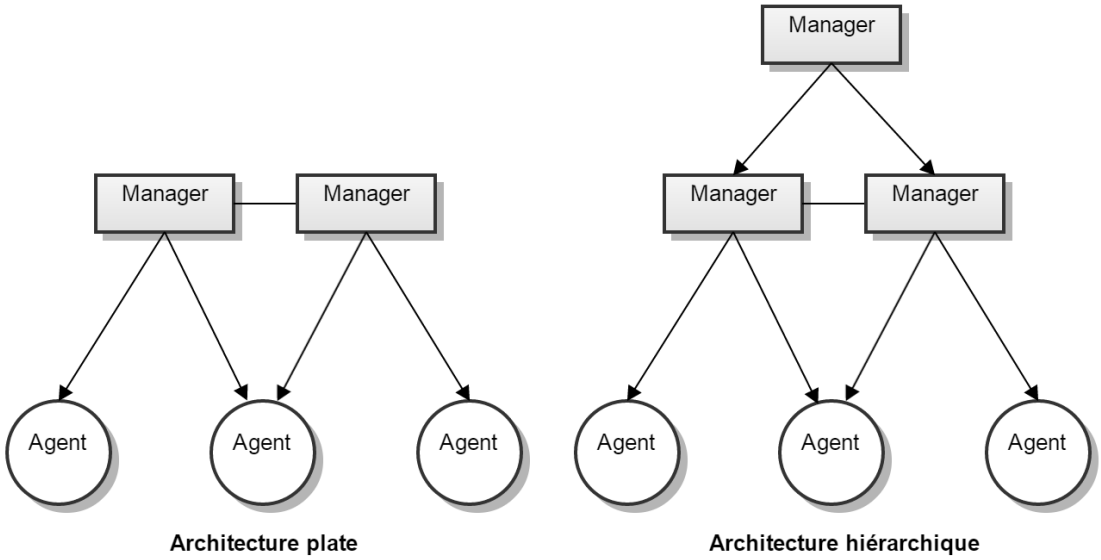

Figure 1-2. Architectures distribuées : plate et hiérarchique [23].

1.4 Le modèle FCAPS

Introduit pour la première fois au début des années 1980, **FCAPS *(Fault, Configuration, Accounting, Performance, Security)*** est le modèle de gestion des réseaux et systèmes proposé par l'organisme international de normalisation (ISO) [1][23][36]. Il présente les cinq domaines fonctionnels de la gestion des réseaux et systèmes, comme le montre la Figure 1-3.

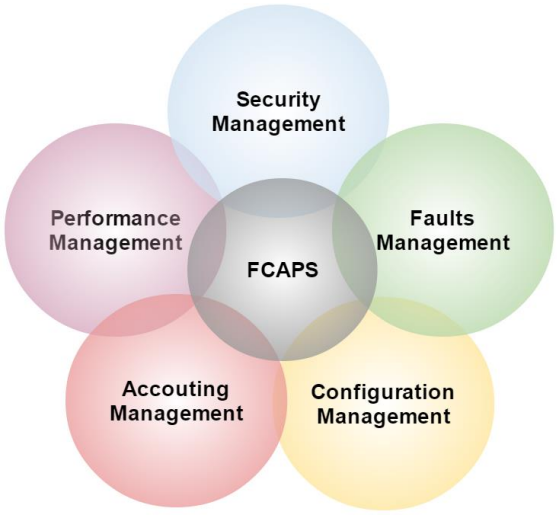

Figure 1-4. Les cinq domaines fonctionnels du modèle de gestion FCAPS.

1.4.1 La gestion des anomalies (*Faults Management*)

Appelée aussi gestion des fautes, son objectif est de garder le système dans un état opérationnel, en faisant appel aux fonctions suivantes :

- La surveillance : elle consiste à remarquer les pannes (fautes), consulter les journaux et collecter les alarmes.
- La localisation des pannes : elle se base sur la recherche dichotomique en utilisant des piles de tests.

1.4.2 La gestion des configurations et des noms (Configuration Management)

Elle permet de mettre en place le réseau et d'assurer son suivi. Elle est assurée par les fonctions suivantes :

- Installation: consiste à paramétrer un composant du réseau, de le mettre en service, le retirer du service et de mettre à jour les versions utilisées.
- Contrôle et surveillance: permet de contrôler et surveiller l'état des composants du réseau.
- Gestion des noms : permet de gérer les noms des composants pour pouvoir les identifier.

1.4.3 La gestion de la comptabilité (Accounting Management)

Elle est souvent utilisée à des fins de facturation et de gestion de quotas. Les accès des utilisateurs aux ressources sont comptabilisés par des statistiques sur des métriques comme: le taux d'utilisation du temps CPU, de la mémoire vive, des liaisons, du disque, etc. Cette gestion permet aussi d'empêcher la surcharge du réseau. Ce contrôle est généralement utilisé par les grandes organisations ainsi que les fournisseurs de services, mais rarement applicables dans les entreprises de petite à moyenne taille.

1.4.4 La gestion des performances (Performance Management)

Permet d'évaluer les performances du système afin de prévoir et quantifier la qualité de service, et d'identifier et paramétrer les outils nécessaires pour satisfaire cette dernière.

Les différentes fonctions de la gestion des performances sont :

- La surveillance : permet de collecter des informations (statistiques) sur les performances des différents composants du système.
- La gestion du trafic : permet contrôler le trafic du réseau en fonction des statistiques précédemment collectées, comme par exemple la mise à jour des tables de routage pour répartir le trafic de façon équilibrée.
- L'observation de la qualité de service : consiste à faire des tests pour mesurer certains paramètres de la qualité de service, comme le temps de repense d'un service.

1.4.5 La gestion de la sécurité (Security Management)

Son objectif est d'assurer la sécurité, physique et logique, du système et des informations qu'il héberge. Ses principales fonctions sont :

- La configuration et la gestion des outils de sécurité : consiste à configurer et auditer les composants qui assurent la sécurité, comme les pare-feux, les systèmes de détection d'intrusion, les listes de contrôle d'accès, etc.

- La gestion de la confidentialité : permet d'assurer la confidentialité des informations stockées ou transitant sur le réseau. Elle est assurée principalement par des mécanismes de chiffrement, comme les infrastructures à clé publique, les protocoles de transport sécurisé, etc.
- La surveillance des journaux : permet de prévoir et/ou détecter les attaques et d'émettre des alertes. Elle peut même inclure des mécanismes automatiques de prévention qui agissent en repense aux alertes.
- La gestion des abonnés aux différents services: elle est assurée par la détermination et le maintien des droits de connexion des utilisateurs aux différents services.

1.5 Le cadre architectural de la gestion ISO

L'architecture présentée dans cette section est décrite par la norme ISO10040. Cette norme traite les trois aspects suivants[1][23] : la structure de la gestion, les fonctionnalités de la gestion et la base des informations de gestion. Cette architecture repose sur un service appelé CMIS et un protocole d'échange d'information appelé CMIP.

1.5.1 La structure de la gestion

La norme décompose la structure de gestion en 3 niveaux : la gestion système, la gestion de couches et les opérations de couches.

1.5.1.1 La gestion système (System Management)

Elle concerne les mécanismes d'échange d'informations de gestion relatives aux objets du système. Il s'agit d'échanges de niveau 7 de l'architecture OSI entre des entités, appelées **SMAE (*System Management Application Entity*)**, qui assurent les fonctions de gestion du réseau (présentées dans la section FCAPS plus haut) via un protocole spécifique appelé **SMP (*System Management Protocol*)**.

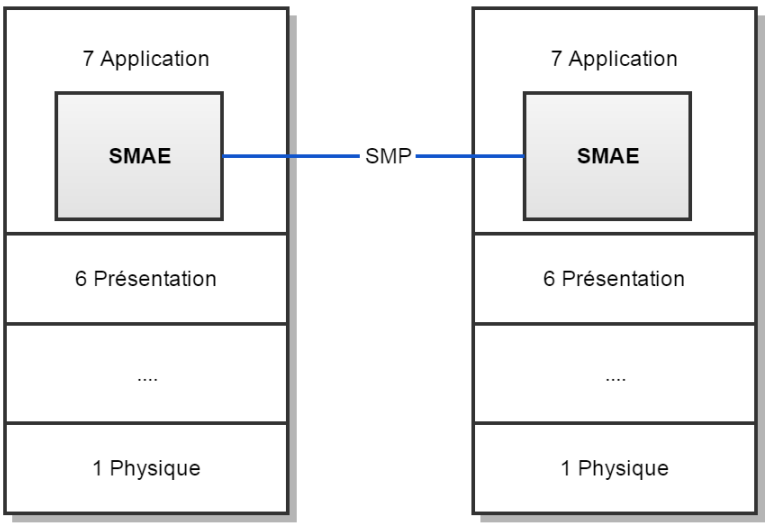

Figure 1-5. Échange d'informations de gestion système[1].

1.5.1.2 La gestion de couches (Layer Management)

Elle concerne l'ensemble des fonctionnalités de gestion spécifiques à des entités d'une couche de communication N, appelées **LME (Layer Management Entity),** à travers des protocoles spécifiques à cette couche notés **N-LMP (N Layer Management Protocol).**

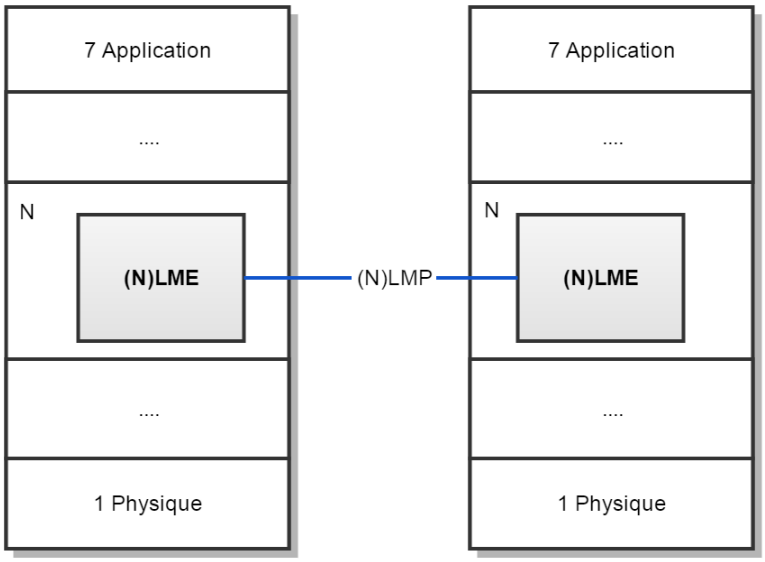

Figure 1-6. Échange d'informations de gestion de couches.

1.5.1.3 Les opérations de couche (Layer Operation)

Les opérations de couche N concernent l'ensemble des mécanismes gérant une couche de communication particulière, et utilisant le protocole de la couche N existant (N-LMP).

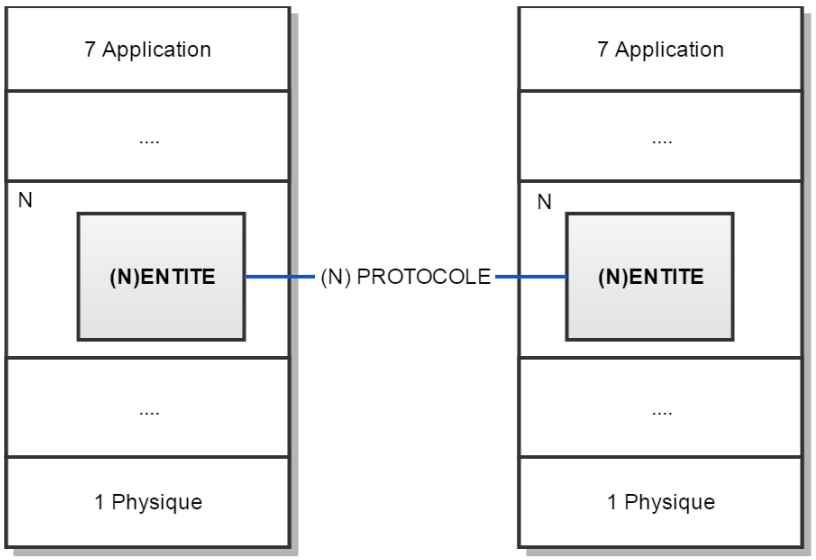

Figure 1-7.Echange d'informations de gestion système.

1.5.2 Le modèle conceptuel Agent-Manager

Ce modèle se base sur une architecture hiérarchique qui regroupe les entités de gestion ainsi qu'une répartition des rôles. Comme la plupart des modèles conceptuels de gestion, il utilise les deux entités de base qui sont : l'agent et le manager.

L'agent est le processus qui exécute les opérations de base sur les ressources de bas niveau qu'il gère, et fournit le service de gestion. Le manager est le processus qui fait appel aux agents pour réaliser des opérations sur les ressources pour des fins de gestion

La communication manager/agent se fait à travers des messages de commandes envoyés par les managers, et des messages de réponse envoyés par les agents. Il est possible que des messages de notification d'événements soient envoyés directement par un agent, pour signaler des événements intervenus à son niveau.

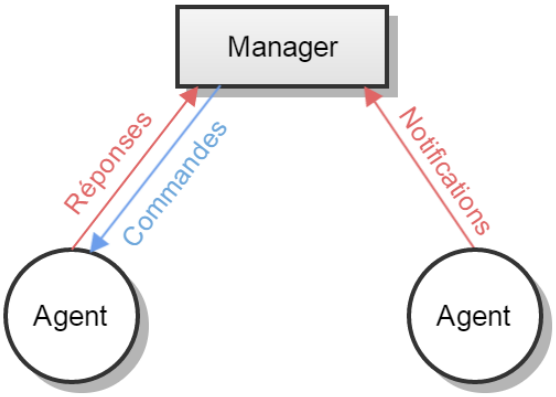

Figure 1-8. Le modèle Agent-Manager.

Il est possible qu'un agent n'utilise pas les mêmes normes de communication que le manager. Dans ce cas, une entité tierce appelée Proxy-Agent, située au niveau de l'agent ou du manager, est utilisée pour l'adaptation.

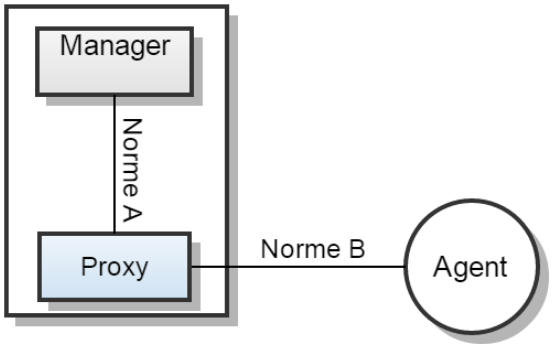

Figure 1-9. Les messages échangés entre un manager et un agent.

1.5.3 Le modèle d'information de gestion

Pour modéliser l'information de gestion, l'ISO propose un modèle d'information orienté objet. C'est à dire, l'ensemble des éléments (physiques et logiques) du réseau est vu de manière abstraire comme un ensemble d'objets gérés. Un objet est l'instanciation d'une classe et est caractérisé par un ensemble de propriétés et de méthodes.

L'ensemble des objets gérés sont stockés dans une base de données appelée **MIB *(Management Information Base)*** organisée sous forme d'arbre appelé **MIT *(Management Information Tree)***. Chaque agent maintient sa propre MIB qui contient ses objets gérés. Le manager maintient une MIB globale qui est le récapitulatif de toutes les MIB des agents.

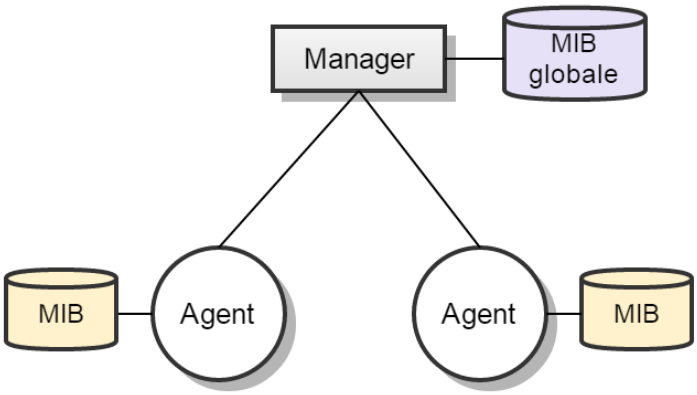

Figure 1-10. Répartition des MIB dans un système de gestion

1.5.4 Le service CMIS

Le service **CMIS *(Common Management Information Service)*** est un service de gestion implémenté dans une architecture de gestion ISO[23]. Il offre 3 types de services :

- Les services d'opérations : permettent à un manager de demander à un agent distant l'exécution d'opérations, selon des règles passées en paramètre.
- Les services de notification : permettent à un agent d'envoyer spontanément des messages à un manager
- Les services d'association : permettent à des agents et à des managers d'établir des connexions au niveau application.

L'entité qui propose les services CMIS est appelée **CMISE (*Common Management Information Service Element)***. Elle utilise le protocole **CMIP**, qui sera détaillé dans la section suivante, pour le transport d'informations de gestion. L'appel des services CMIS se fait par des primitives, comme la primitive **M-Event-Report** qui est invoquée par un agent rapportant un événement concernant un objet géré pour un manager.

1.5.5 Le protocole CMIP

CMIP *(Common Management Information Protocol)* est un ensemble de protocoles qui assure les échanges entre les services (éléments) CMISE. Comme tout protocole, il définit des règles de création et d'échange des PDU (Protocol Data Unit). CMIP fait appel à deux autres services appelés ROSE *(Remote Operation Service Element Protocol)* et ACSE (*Application Control Service Element)* qui ne sont pas détaillés dans ce document[23].

Une machine utilisant CMIP transmet au CMISE toutes les trames valides, et renverra toutes celles qui ne le sont pas en précisant la cause du rejet.

Le protocole CMIP a été adapté aux réseaux utilisant la suite de protocoles TCP/IP. Cette nouvelle variante du protocole a été nommée **CMOT *(CMIP Over TCP/IP)***.

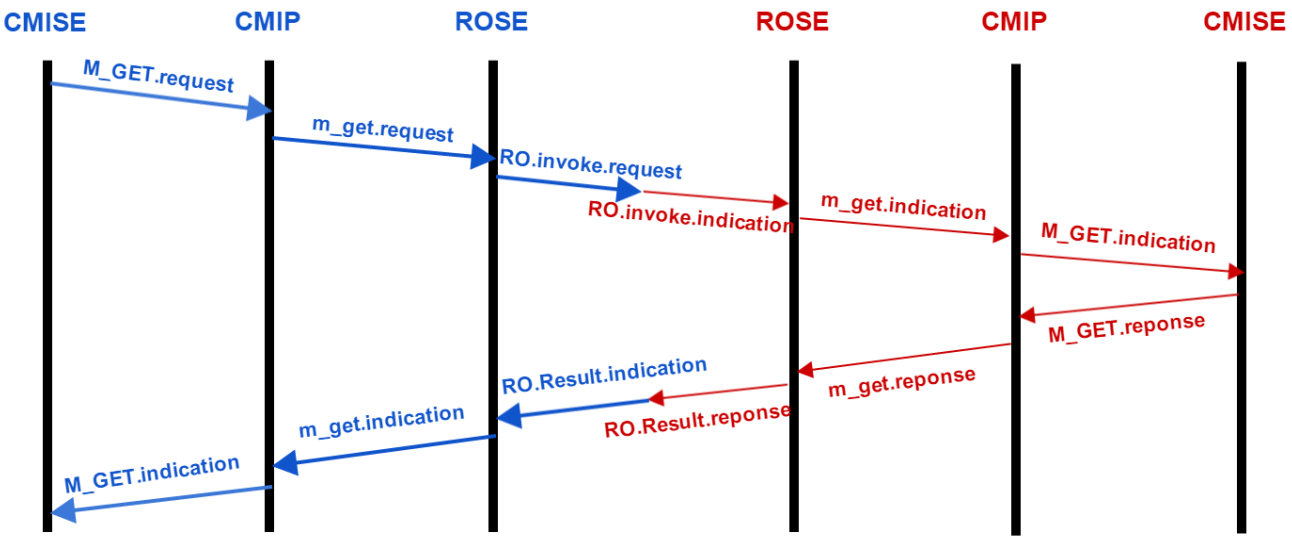

Figure 1-11. Utilisation du protocole CMIP dans l'invocation de la primitive M-Get.

1.6. La gestion SNMP

SNMP ***(Simple Network Management Protocol)*** représente l'approche de gestion des réseaux et systèmes proposée par l'**IETF** ***(Internet Engineering Task Force)*** au milieu des années 1990, pour satisfaire le grand besoin d'une gestion sophistiquée. À l'origine, SNMP vient d'un standard antérieur appelé **SGMP** ***(Simple Getway Management Protocol)*** destiné à la supervision des passerelles réseaux [2][34].

En 1988, l'**IAB** ***(Internet Architecture Board)*** approuva le SNMP comme étant une solution à court-terme, tandis que le CMOT comme étant une solution à long-terme.

1.6.1. L'architecture du protocole SNMP

L'architecture SNMP a été initialement développée pour les systèmes UNIX utilisant la pile de protocoles TCP/IP. Elle utilise les datagrammes UDP pour l'échange d'informations et se base sur quatre éléments :

- Un ensemble de stations gérées.
- Une ou plusieurs stations de gestion.
- Des base d'informations de gestion appelée MIB.
- Un protocole de gestion.

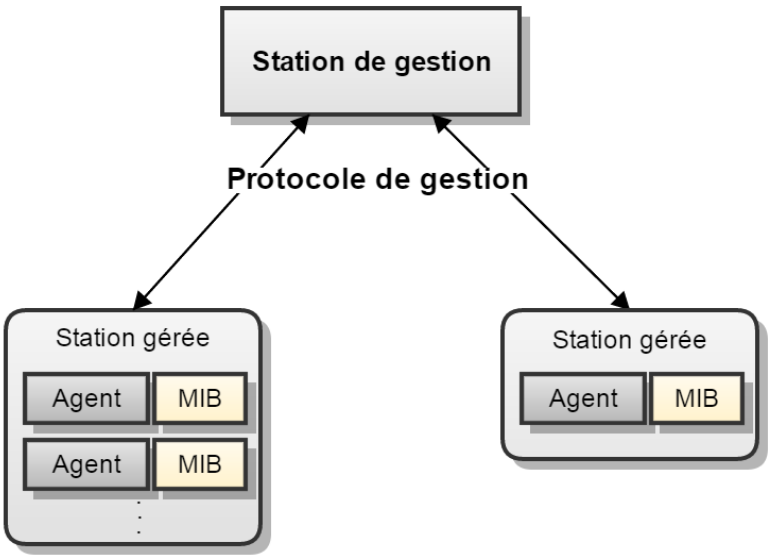

Figure 1-12. Les éléments de base d'une architecture SNMP.

1.6.1.1 Stations de gestion

Aussi appelées **NMS *(Network Management Station),*** une station de gestion est similaire à un manager dans l'architecture ISO. Elle gère le réseau en contrôlant les différentes stations par l'intermédiaire des agents de gestion qui sont en contact direct avec ces stations. Une station de gestion assure les opérations de gestion suivantes :

- La collecte d'information provenant des stations gérées.
- Le maintien de sa base d'information.

Une station de gestion doit avoir les éléments suivants :

- une interface pour l'administrateur : elle lui permet d'exécuter ses tâches.
- des mécanismes d'interaction avec les éléments gérés : pour pouvoir récupérer des informations (taux d'occupation d'une liaison, capacité du disque, etc.) sur les éléments gérés et changer leur configuration.
- une MIB globale : cette base d'informations de gestion est construite à partir des MIB des agents de gestion.

1.6.1.2 Station gérée

Une station gérée peut être n'importe quelle entité informatique connectée au réseau et exécutant un agent SNMP. Un agent SNMP est un programme qui tourne sur une station gérée et qui expose une interface pour répondre aux demandes d'informations et de commandes envoyées par une station de gestion SNMP. L'agent SNMP interagit directement avec la station gérée par l'exécution des commandes et la collecte d'informations demandées par le gestionnaire.

1.6.1.3 Base d'information de gestion (MIB)

Ayant le même principe qu'une MIB dans l'architecture ISO[36], une MIB SNMP contient un ensemble de variables qui représente les informations de gestion[2]. La MIB est mise à jour (en affectant de nouvelles valeurs aux variables) par un agent puisque c'est la seule entité qui a un contact direct avec

la station gérée. Une MIB appelée MIB globale est gérée par la station de gestion. Elle contient le récapitulatif des informations contenues dans les autres MIB des agents.

La MIB contient des définitions précises des éléments gérés (on parle aussi d'objets gérés) par l'agent. Ces informations sont représentées sous forme d'arbre de variables. La structure de ces informations de gestion est présentée dans la section 6.x de ce document.

1.6.1.4 Protocole de gestion

Le protocole de gestion SNMP est l'élément de l'architecture SNMP qui permet l'échange d'informations entre une station de gestion et un agent de gestion (ou entre deux stations de gestion dans le cas d'une architecture décentralisée). Il est connu par sa simplicité de compréhension, sa simplicité d'implémentation et sa stabilité.

1.6.2 L'architecture du protocole de gestion SNMP

Le protocole SNMP est un protocole applicatif sans connexion qui se base sur le protocole UDP. Il permet l'échange de requêtes, de réponses et de messages de notifications dans une architecture SNMP. Tous les échanges sont traduits en termes d'opérations de lecture/écriture de variables à l'aide de 05 opérations : **GetRequest, GetNext, SetRequest, GetRepense et Trap.**

1.6.2.1 Les cinq opérations SNMP

Les 5 opérations suivantes sont à la base de tous les échanges entre les entités d'une architecture SNMP [2][34][35]:

- **GetRequest :** c'est une requête envoyée par une station de gestion, pour **lire une variable** se trouvant dans la MIB d'un agent. Elle prend en argument l'identifiant de la variable et le nom de l'agent désiré. Une réponse à cette requête est une opération **GetResponse**. Si l'identificateur de la variable est invalide, la valeur NULL est retournée par l'agent.
- **GetNext :** c'est une requête qui permet à une station de gestion de **récupérer l'identificateur** de la variable qui se trouve après la variable passée en argument.
- **SetRequest :** permet **d'affecter une valeur à une variable** sur une station gérée. Elle retourne la valeur de la variable même si l'opération échoue. Cette requête permet aussi la création et la suppression de variables.
- **GetRepsponse :** permet à un agent de renvoyer la valeur d'une variable demandée par une opération GetRequest. Toutefois si la variable demandée n'est pas disponible, le GetResponse sera accompagné d'une erreur noSuchObject.
- **Trap:** c'est un message de notification (une alerte) qui permet à un agent de signaler un évènement.

Voici deux exemples d'opérations SNMP :

- **GetRequest(1.2.6.1.2.4)** provoque le retour de **GetResponse(tcpMaxConn = x)** où **x** est le résulta de la requête.
- **SetRequest(1.3.6.2.1.6.13.1.1 = 12)** provoque le retour de **GetRespense(tcpConnState = 12).**

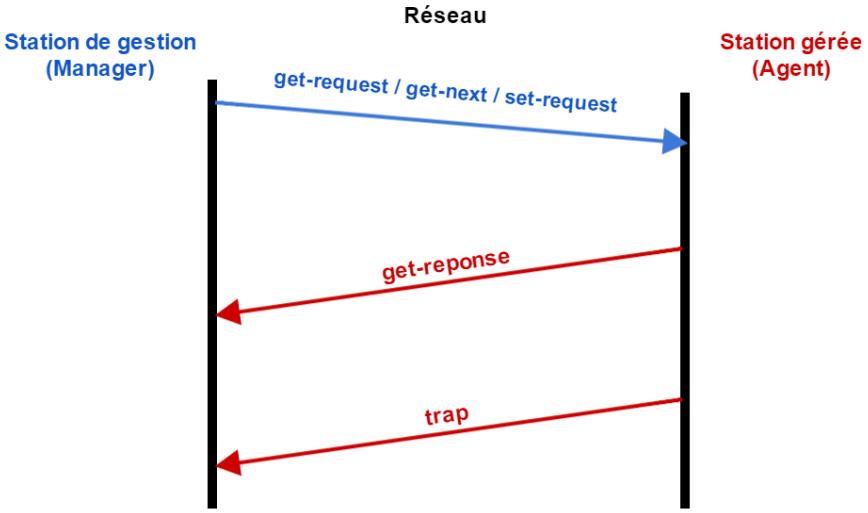

Figure 1-13. Échange de messages dans le protocole SNMP.

Le protocole SNMP utilise le port 161 pour les requêtes/réponses et le port 162 pour les notifications.

1.6.3 La structure de l'information de gestion dans SNMP

Avant d'aller plus loin, nous donnons ici la définition du concept d'**information de gestion,** cité plusieurs fois plus haut pour désigner les informations contenues dans les MIB.

Une information de gestion est un ensemble de variables modélisant une entité gérée ainsi que son état, comme par exemple l'état d'un port de commutateur (activé/désactivé), le nombre de paquets reçus par une interface de routeur, etc. Les administrateurs consultent et modifient ces variables pour accomplir leurs tâches de gestion, puisqu'elles permettent d'avoir une vision globale et claire des composants du réseau géré. Pour définir l'état d'un objet géré, trois attributs sont utilisés[35]: l'identifiant, la syntaxe et le codage.

1.6.3.1 Identifiant d'objet

Chaque agent localise les variables dans sa propre MIB et échange les données associées, avec la station de gestion qui devrait retrouver les Mêmes définitions de variables dans sa MIB. Les variables, constituants la MIB, doivent être parfaitement définies suivant des règles de nommage qui permettent leur désignation d'une manière universellement unique et sans ambigüité.

Suivant ses règles, les objets de la MIB sont hiérarchisés en fonction de leurs fonctions, sous forme d'un arbre appelé **Arbre d'Information de Gestion** ou **MIT *(Management Information Tree)***, similaire au MIT dans l'approche ISO[2][36]. Chaque nœud de l'arbre est représenté par un entier, ce qui nous permet d'identifier n'importe quel nœud par une suite d'entiers. Cette suite de numéros est appelée identifiant de l'objet ou **OID *(Object Identifier)***. Les nœuds de l'arbre sont étiquetés. L'étiquette d'un nœud représente l'association d'une courte description textuelle et du nombre entier relatif à ce nœud.

Pour des raisons d'organisation, les variables SNMP sont regroupés en listes (sous-arbres). Les objets gérés sont représentés par le sous-arbre **management** qui se situe, respectivement, après les nœuds **ISO**, **Identified-Organisation**, **DOD *(Departement of Defense)*** et **Internet**

Le nœud Internet, alloué à l'IAB, est le premier nœud de la DOD. Il est la racine de quatre autres sous-arbres : Directory, Management (celui qui contient les objets gérés), Experimental et Privete[35].

La figure 6.3.1 ci-dessous, présente un exemple d'OID dans un MIT SNMP. L'objet géré est un hôte du réseau. Il a pour OID **1.3.6.1.2.1.25** qui se traduit par :

iso.Identified-organisation.dod.internet.mgmt.mib.host.

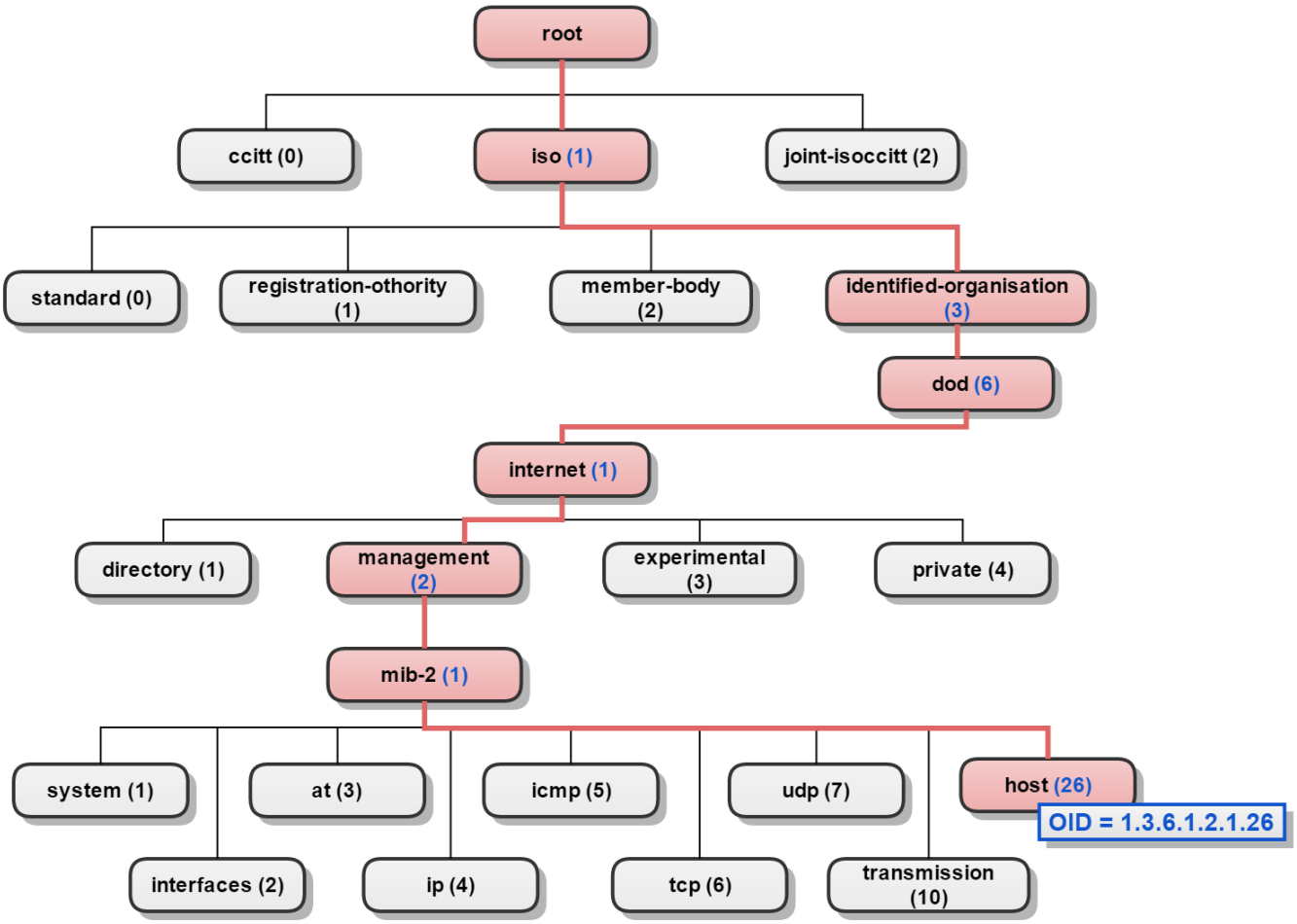

Figure 1-14. Exemple d'OID dans un arbre d'information de gestion SNMP[35].

1.6.3.2 La syntaxe et le codage

SNMP utilise la notation ASN.1 pour définir le nom et le type d'une variable de la MIB. ASN.1, qui est un standard international permettant de décrire le format des messages SNMP lors de la communication entre une station de gestion et une station gérée. Ce standard intervient dans la description syntaxique de ces messages, qui parle des types de données. Cette description ne sera pas détaillée dans les sections à venir.

ASN.1 inclut aussi un ensemble de règles de codage appelé **BER(*Basic Encoding Rules)***. SNMP respecte ces règles lors du codage des données avant de les transmettre sur le réseau sous forme de données utiles concaténées à un en-tête, comme le montre la figure 6.3.2.

Figure 1-15. En-tête du protocole SNMP[34].

1.6.4 Evolution du protocole SNMP

Le protocole SNMP a connu, du à son importance et à sa sensibilité, plusieurs évolutions, notamment en ce qui concerne l'amélioration de ses opérations et de sa sécurité, à chaque évolution correspond une version du protocole que sont [2][34]:

- **SNMPv1 :** C'est la première version du protocole. Elle est connue par sa sécurité triviale, car le seul mécanisme d'authentification se base sur une chaîne de caractères appelée **communauté** utilisée comme mot de passe. L'authentification précède chaque accès à une variable d'une MIB. Cette version est le premier standard du protocole SNMP.
- **SNMPsec :** C'est une version qui ajoute de la sécurité au SNMPv1 en se basant sur des groupes de sécurité. Maintenant, cette version est largement oubliée car aucun constructeur (ou très peu) ne l'a utilisée.
- **SNMPv2p :** Afin de mettre à jour du protocole SNMPv1, cette version propose une modification de ses opérations ainsi que d'autres nouvelles. Elle propose aussi des nouveaux types de variables. Concernant la sécurité, elle utilise le mécanisme de groupes de la version SNMPsec.
- **SNMPv2c :** Appelé **community stringbased SNMPv2**, cette version est une amélioration des opérations et des types de variables de SNMPv2p. Cependant, elle utilise l'authentification par la chaîne de caractères communauté qui est la lacune majeure de SNMPv1.
- **SNMPv2u :** Appelé **community stringbased SNMPv2**, cette version est une amélioration des opérations et des types de variables de SNMPv2c. Elle propose un mécanisme de sécurité basé sur les usagers.
- **SNMPv2* :** Cette version utilise les meilleurs parties de SNMPv2p et SNMPv2u. Elle est peu connue car, aucun document n'a été publié pour la décrire ??.
- **SNMPv3 :** Cette version représente le standard actuel du protocole SNMP. Elle combine la sécurité basée sur les usagers et les types et les opérations de SNMPv2p, avec plus de capacité pour les proxies.

Aujourd'hui, la version SNMPv1 a été abandonnée au profit de la version SNMPv2 qui reste la plus largement utilisée même si la version SNMPv3 est la plus recommandée. SNMPv2 a permis d'introduire le caractère décentralisée de la gestion, en permettant à un agent de jouer le rôle de gestionnaire vis-à-vis d'équipements non accessibles directement au gestionnaire centrale. Cette dernière caractéristique permet par exemple l'exploration et la construction automatique de la topologie d'un réseau complexe et à grande échelle. Notons enfin que La majorité des équipements et implémentations logiciels de SNMP maintiennent les deux versions 2 et 3 pour garder la compatibilité arrière avec les applications de gestion compatibles SNMPv2.

1.6.5 La sécurité dans SNMPv3

La sécurité apportée par SNMPv3 vise essentiellement les transactions entre les entités[34] [36]. Elle comprend l'identification et l'authentification des entités qui communiquent ainsi que la confidentialité des transactions qui peuvent être locales ou publiques.

Cette approche se base sur deux modèles[35] :

- **USM *(User-based Security Model)*** qui gère la sécurité des échanges agent/manager.
- **VACM *(View-based Access Control Model)*** qui contrôle l'accès aux informations du modèle de sécurité USM contenues dans une MIB spéciale appelée **SNMP User-Based-SM-MIB *(User-Based Security Model MIB)***. Ce modèle ne sera pas présenté dans la suite de ce document.

1.6.5.1 Le modèle USM

Dans ce modèle, trois mécanismes sont utilisés : l'authentification, le chiffrement et l'estampillage de temps.

- **L'authentification :** Elle permet de s'assurer que les entités qui communiquent sont vraiment celles qui prétendent l'être et que l'information qu'ils transmettent est intègre.

-**Le chiffrement :** Le chiffrement dans SNMPv3 est réalisé avec l'algorithme **DES *(Data Encryption Standard)***. Cet algorithme utilise une clé symétrique (la même clé est utilisée pour le chiffrement et le déchiffrement), gardée comme mot de passe secret, partagée entre le manager et l'agent.

L'estampillage de temps : Il permet d'arrêter les attaques par rejoue. Une attaque par rejoue, ou Replay Attack, consiste à capter un paquet pour l'envoyer de nouveau sans le modifier. Pour éviter cette attaque (ou la minimiser tant soit peu), le temps de l'envoi et estampillé sur le paquet. La date et heure de la réception du paquet SNMPv3 sont comparées à celles enregistrées dans le paquet. Si la différence est supérieure à 150 secondes, le paquet est ignoré.

1.7 Conclusion

La gestion des réseaux et des systèmes est une activité complexe qui demande beaucoup d'efforts, que ce soit pour les concepteurs de modèles de gestion ou les administrateurs. Cette complexité, ne nous empêche pas on peut dire que les anciens standards utilisés jusqu'à présent, à l'instar de SNMP, ont réussi leurs mission. Cependant, avec l'évolution exponentielle des nouvelles technologies ainsi que les nouveaux besoins en terme de gestion de systèmes complexes, le passage à des nouveaux concepts est devenu une nécessité.

Dans le cadre de ce projet, nous avons choisi le standard **WBEM *(Web-Based Enterprise Management)***, que nous détaillons dans le chapitre suivant, afin de mettre en place notre système de gestion de politiques de sécurité.

CHAPITRE 2

GESTION WBEM

2.1 Introduction

Depuis plusieurs années, beaucoup de standards de gestion de réseaux existent. SNMP est le plus connu d'entre eux, car, il est facile à implémenter et permet une gestion simple est efficace. Cependant, SNMP est devenu insuffisant puisqu'il ne permet de gérer que des équipements relativement simples (routeurs, commutateurs, etc.). Ainsi, un besoin de standards qui assurent la gestion de systèmes complexes est apparu.

L'initiative **WBEM *(Web-Based Entreprise Management)*** *est* apparue en 1996, comme étant le fruit de la collaboration de plusieurs organisations dont les plus connues sont Cisco, IBM, Intel et Microsoft entre autres. Leur objectif était de développer un ensemble de standards qui permettent de décrire les ressources matérielles et logicielles d'une entreprise, selon un schéma commun, et communiquer ces informations via un mécanisme standardisé (voir figure 1). Cependant, WBEM ne peut pas remplacer les standards de gestion existants telles que SNMP et CMIS car ils sont bien incrustés dans les environnements de gestion actuels, mais il permet par contre de les intégrer en les présentant avec son propre système d'information en assurant ainsi l'homogénéité des informations présentées aux applications de gestion. En juin 1998, la tâche de standardisation de l'initiative a été remise au **DMTF *(Distributed Management Task Force)***, consortium regroupant les entreprises concernées par le projet.

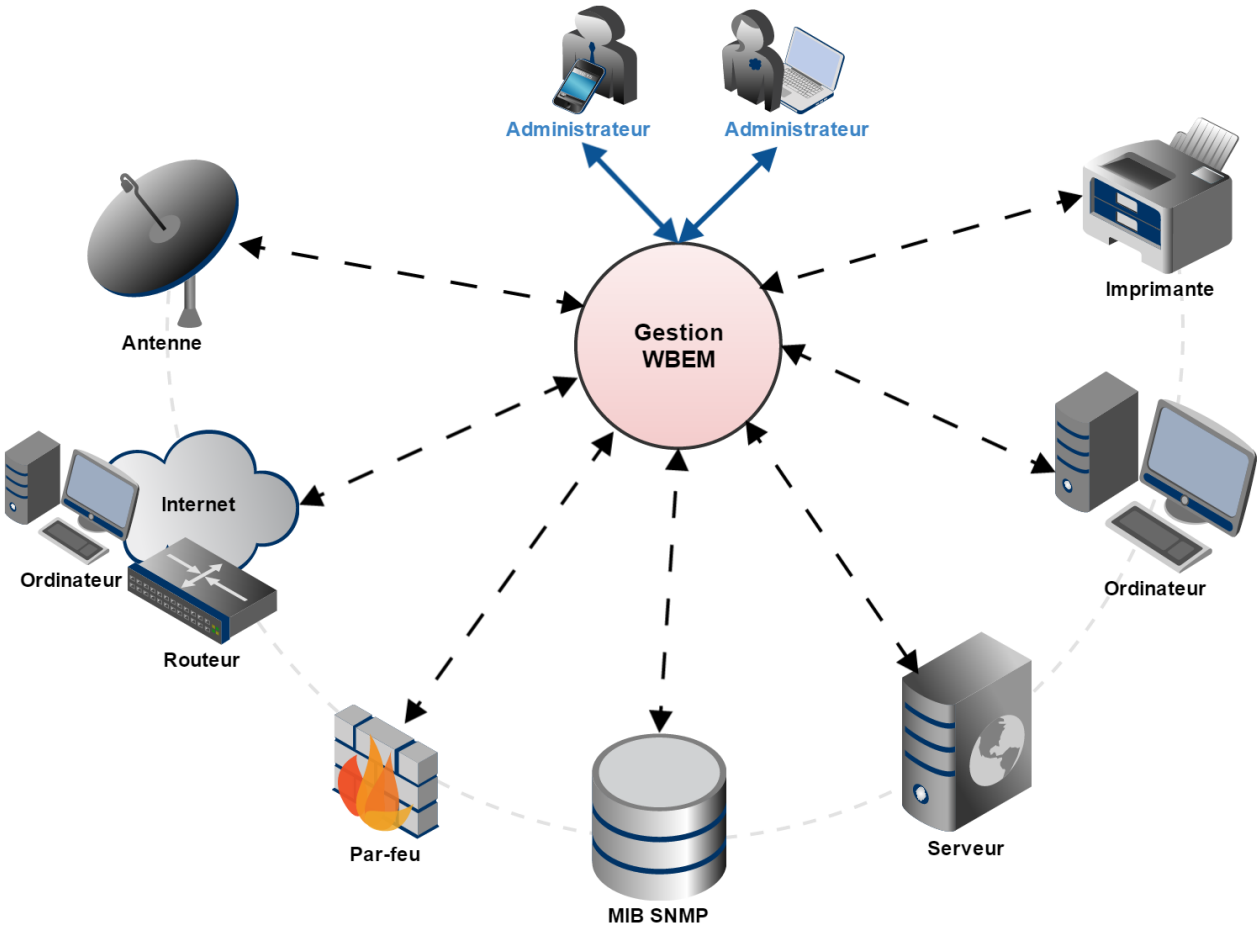

Figure 2-1. Gestion WBEM homogène d'un réseau hétérogène.

L'approche WBEM définit ses composants de base de la façon suivante [8][15][18]:

- **Une architecture fonctionnelle** qui définit les acteurs intervenants dans la gestion, leur répartition, leurs fonctions et leurs rôles.
- **Un modèle informationnel** qui représente une approche de modélisation et un formalisme pour la spécification de nouveaux modèles de gestion (syntaxe et sémantique). Il intègre aussi un modèle fonctionnel générique (se basant sur le modèle informationnel) comportant un ensemble d'objets gérés de base.
- **Un modèle de communication** qui comprend un service et un protocole assurant la communication entre les entités.

Le modèle d'information inclue dans l'approche WBEM est appelé **CIM *(Common Information Model)***. Il permet de représenter les ressources gérées d'une façon unifiée et extensible (à l'aide du concept d'héritage). De plus, l'architecture utilise un système d'encodage **CIM-XML** permettant de représenter l'information CIM en XML, pour pouvoir la transporter entre les différents acteurs à l'aide des opérations du protocole **HTTP/HTTPS**.

2.2 L'architecture fonctionnelle

Comme toute autre approche de gestion, WBEM propose une architecture se basant sur un ensemble d'entités [15]. Ces entités sont au nombre de quatre [15][18] (voir figure 2-2) :

- **L'application de gestion :** aussi appelée client CIM, elle constitue l'interface entre le domaine géré et le gestionnaire. Elle présente à ce dernier un accès aux informations qu'elle récupère auprès du gestionnaire d'objets CIMOM.

- **Le gestionnaire d'objets CIMOM *(CIM Object Manager)*:** aussi appelé serveur CIM, il représente l'entité principale dans l'architecture WBEM. Il assure le maintient de la base d'informations de gestion (MIB) qui est appelée **CIMOR *(CIM Object Repository)*** dans la terminologie WBEM. Le rôle du CIMOM consiste à donner aux applications de gestion l'accès aux informations contenues dans cette CIMOR, et qui donnent une vue globale et homogène des ressources gérées, puisqu'elles sont toutes modélisées avec CIM et suivant des modèles de références (voir section3.4).

- **Le provider :** Il assure la liaison entre le gestionnaire d'objets CIMOM et les ressources gérées. C'est un composant essentiel dans l'architecture WBEM car, il permet d'appliquer les opérations CIM, qu'un gestionnaire invoque sur les éléments gérés. Un provider peut être vu comme un traducteur, puisqu'il permet de traduire les informations CIM qu'il communique avec le CIMOM en actions ou en valeurs concrètes relatives aux ressources qu'il gère. Un provider possède donc deux interfaces :

    - **Une interface native** qui communique avec les ressources gérées pour lui transmettre les directives et les requêtes d'informations demandées par le serveur CIMOM.
    - **Une interface CIM** qui communique avec le gestionnaire d'objet CIMOM qui traduit les informations et réponses retournées par les ressources gérées en instances d'objets CIM.

Si on prend l'exemple d'un provider qui permet de gérer une interface réseau d'un routeur, il doit pouvoir récupérer l'état de cette interface (activée/désactivée) via son interface de commande native, et transmettre cette information au CIMOM sous format d'objet CIM.

- **L'agent :** Dans cette architecture, il représente la ressource gérée à laquelle est attachée un provider WBEM. Il permet d'exécuter les opérations de gestion requises par le gestionnaire.

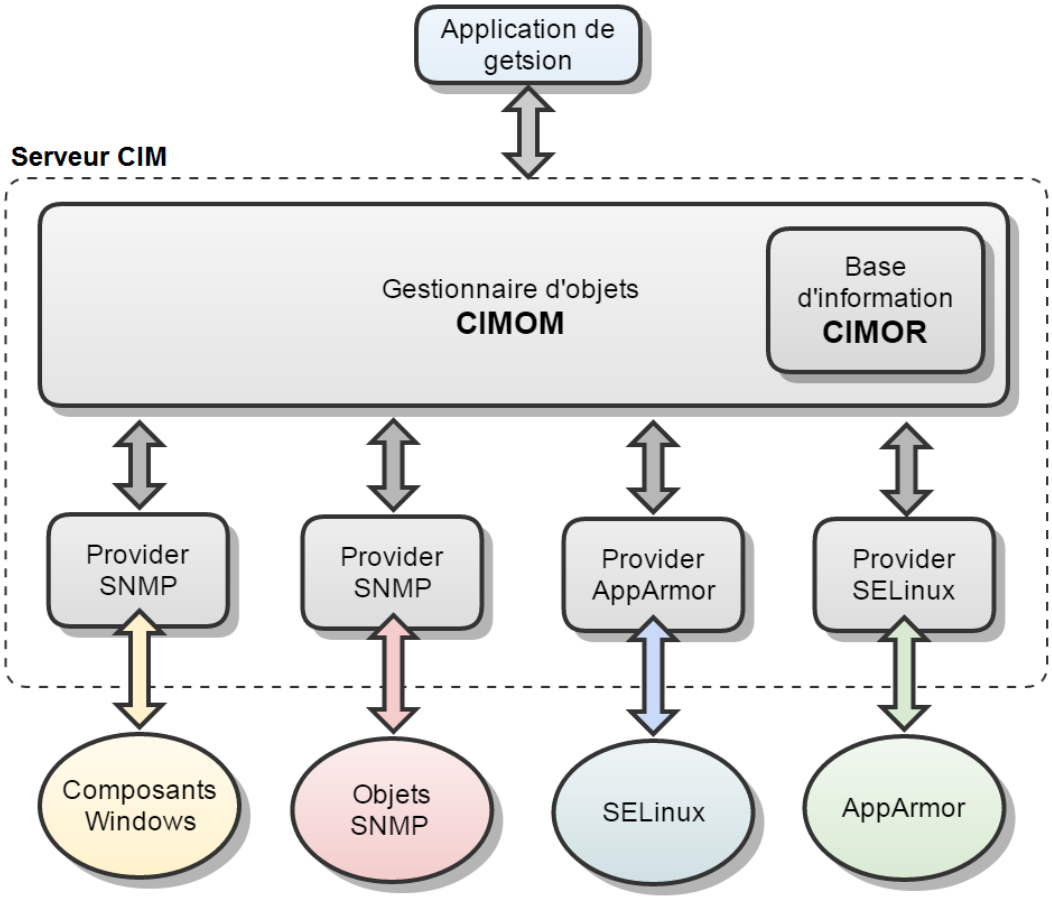

Figure 2-2.Composants d'une architecture WBEM

L'architecture définie offre une couche d'abstraction permettant d'assurer une gestion homogène centralisée d'un ensemble de ressources hétérogènes distribuées, tout en conservant les technologies existantes.

L'architecture WBEM présentée est purement théorique. Le choix de la distribution de cette architecture sur le réseau a été laissé à l'implémentation. Généralement, dans une implémentation réelle, les entités WBEM sont réparties de la façon suivante (figure 2-3) :

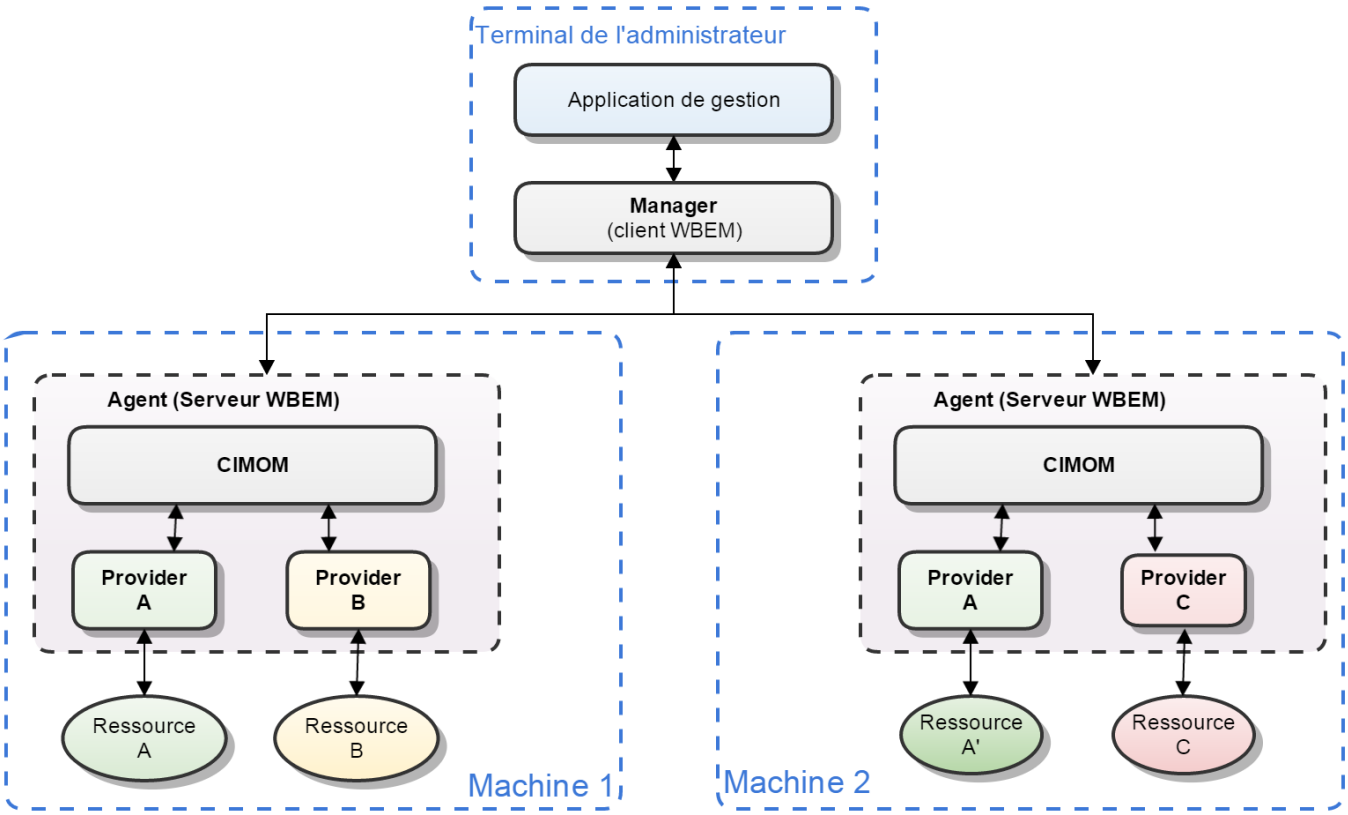

Figure 2-3. Distribution pratique de l'architecture CIM.

Le serveur WBEM est l'entité qui regroupe généralement le gestionnaire d'objets CIMOM, le référentiel CIM *(CIM Repository)*, les providers, ainsi que des modules qui assurent la communication (voir section 4.1).

2.3 Le modèle d'information commun CIM

Le modèle d'information commun (Common Information Model) CIM, définit comment les éléments d'un système sont modélisés, ainsi que les relations entre eux. C'est un méta-modèle extensible qui permet d'organiser logiquement les informations de gestion de manière unifiée et cohérente, même si l'environnement géré est hétérogène. CIM est un méta-modèle orienté objet qui tente d'unifier et d'étendre les standards de gestion existants, comme SNMP et CMPI, en les représentants avec le même méta-modèle. C'est à dire, qu'après la modélisation en CIM des systèmes (matériel et logiciels) issus de différents constructeurs, leur gestion ne nécessitera que des connaissances CIM au lieu d'apprendre les différentes API d'instrumentation offertes par les constructeurs et les éditeurs, ainsi tous les systèmes qui assurent les mêmes fonctions (OS, Routeurs, Switchs etc.) peuvent être gérés de la même manière.

La force de ce modèle est son niveau d'abstraction permettant de modéliser n'importe quelle entité gérée: réseaux d'ordinateurs, Smartphones, utilisateurs, politiques de sécurité, politique de qualité de service, etc. De plus, CIM propose des mécanismes pour gérer les

relations entre les objets gérés, ainsi qu'un mécanisme d'héritage pour étendre les modèles existants.

CIM présente principalement quatre éléments [6]:

1. **Un méta-modèle** décrivant les composants de base du modèle qui permettent la définition des entités gérées.
2. **Un schéma de nommage** des composants gérés.
3. **Un langage** pour la spécification des objets gérés.
4. **Un schéma de base et un schéma commun** qui sont des modèles génériques, devant être étendus pour la spécification de nouveaux systèmes.

2.3.1 Le méta-modèle

Etant un modèle orienté objet, le méta-modèle CIM présente l'ensemble des éléments de base permettant de construire des spécifications. Ces éléments représentent les briques de base du modèle, dont certains sont similaires à des concepts utilisés dans la programmation orientée objet [DMTF]. Ces éléments sont (voir figure 3.1):

- **Le schéma:** c'est un ensemble de spécifications constituées de **classes**, d'**instances** et d'**associations** modélisant un domaine de gestion. Un schéma peut être vu comme un module SNMP puisqu'il permet d'avoir une vue abstraite du domaine.
- **La classe:** permet de déclarer les propriétés communes d'un type d'objet. Elle est définie par des **attributs** dont les valeurs représentent l'état des objets de cette classe, ainsi que des **méthodes** qui représentent son interface d'interaction.
- **L'attribut:** représente une propriété de l'objet dans lequel il est défini. Chaque attribut est caractérisé par son type de données (entier, caractère, booléen, etc.) et un ensemble d'opérations (lire, affecter, etc.) que l'on peut exécuter sur lui.
- **La méthode:** définit une opération que l'on peut invoquer (exécuter) sur une instance d'objet. Une méthode comporte des paramètres et une valeur de retour.
- **L'association:** c'est une classe spéciale permettant de définir des dépendances plus ou moins forte entre les classes. Elle comprend au moins 2 pointeurs qui pointent vers 2 classes d'objets. Les deux classes sont donc associées par cette association.
- **L'instance:** c'est une occurrence particulière d'une classe ou d'une association. Plusieurs instances peuvent être issues de la même classe, on parle d'instanciation de classe.
- **Le qualificateur :** c'est une métadonnée qu'on associe à un composant (classe, attribut, méthode ou instance) afin de lui ajouter certaine spécificités. Parmi les qualificateurs les plus connus on retrouve :
  - **Description :** peut être associé à tous les composants. Il permet de leur ajouter une description textuelle (voir le code de la section 2.3.3 p x).
  - **Key :** associé aux attributs clé d'une classe, permettant de définir d'une manière unique les instances issues de cette classe.
  - **Valuemap/Values : deux liste contenant toutes les valeurs possibles d'un attribut.**

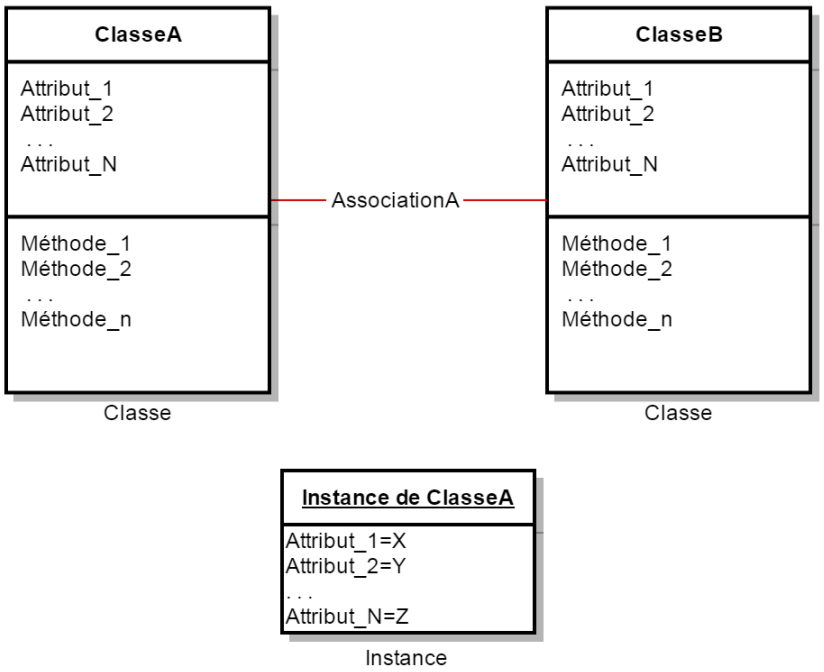

Figure 2-4. Les éléments de base du méta-modèle CIM.

Les éléments du méta-modèle sont spécifiés à l'aide d'un langage de spécification textuel appelé **MOF *(Managed Object Format)*** qui sera détaillé dans la section 3.3. Par ailleurs le langage visuel UML est adopté afin de représenter d'une manière synthétique (et tout aussi expressive que MOF) les détails d'un modèle en précisant les associations et les qualificateurs de chaque élément.

2.3.2 Le schéma de nommage et l'architecture de la MIB

Dans l'architecture CIM, les objets gérés, représentés à l'aide d'un ensemble de classes et de relations entre elles, sont maintenues par le gestionnaire d'objets CIMOM. Ces objets représentent la base d'information MIB appelée CIMOR dans le contexte WBEM.

L'architecture de cette base d'information repose sur le principe d'espace de nommage. Un espace de nommage est un conteneur d'objets (classes, instances et relations) dans lequel chaque composant est identifié d'une manière unique. Un espace de nommage peut contenir d'autres espaces de nommage, ce qui donne une arborescence de nommage similaire à l'arborescence des répertoires sous UNIX.

Pour que tout objet de la MIB soit identifiable de manière unique, CIM définit un mécanisme de nommage d'objet indiquant le nom de la classe caractérisé par son label (nom de cette classe dans la spécification) précédé par le chemin de la classe dans l'espace de nommage. Par exemple, une classe qui a comme label **ClasseEmployé** située dans l'espace de nommage **cimv2**, qui se trouve au dessous de l'espace **root**, a pour nom **/root/cimv2:ClasseEmployé**. Pour identifier les instances, le nom de la classe d'origine est des noms et valeurs des attributs formants la clé de l'objet (ceux qui sont précédés par le qualificateur KEY dans la définition de la classe). Par exemple, le nom d'une instance qui est issue de la classe **ClasseEmployé** présentée plus haut, et qui a comme clé les valeurs d'attributs suivantes :**ID = "1"** et **Nom = "bob"**, a pour nom **/root/cimv2:ClasseEmployé.ID="1".Nom="bob"**, comme le montre la figure 2-5.

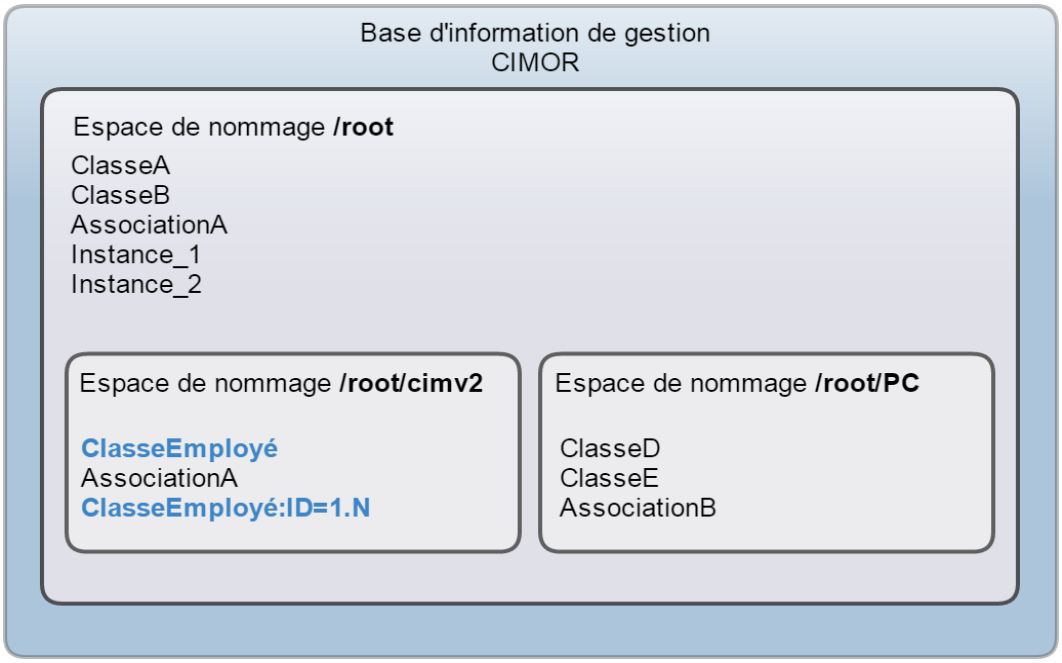

Figure 2-5. Exemple des composants d'un espace de nommage.

2.3.3 Le langage de spécification

Dans l'architecture WBEM, le langage utilisé pour la spécification est le **MOF *(Managed Object Format)***. Ce langage fournit un support textuel pour la formalisation des modèles de l'information et propose une syntaxe et une sémantique permettant de spécifier les composants d'un modèle CIM en se basant sur des éléments du méta-modèle (classes, associations, etc.).

MOF est un langage de spécification orienté objet dérivé du lange IDL (Interface Definition Language) défini par l'OMG (Object Management Group) principalement utilisé par les applications CORBA. La syntaxe et la sémantique ne sont pas détaillées dans ce document. Voici un exemple de code MOF représentant la classe **ClasseEmployé** déjà vu dans la section 2.3.2 :

```
[Description("classe représentant un employé")]          // qualificateur description
class ClasseEmployé{

[Key, Description("Identifiant de l'employé")]           // qualificateur clé et description
uint16 ID;                                               // entier non signé de taille 16 bits

[ Key, Description("Nom de l'employé")]
string Nom;                                              // chaîne de caractères

[ Description("'Méthode pour changer l'ID de l'employé")]

boolean SetNouveauID( uint16 NouveauID);                 // méthode qui prend un ID en
                                                         // argument et qui retourne un
                                                         // booléen

};
```

2.3.4 Le modèle commun

Afin d'unifier la gestion des ressources gérées et d'assurer leur interopérabilité avec l'architecture WBEM, le DMTF a définit deux schémas de référence sur lesquelles doit s'appuyer tout travail de modélisation en CIM. Ces deux schémas, dont chacun représente un certain niveau d'abstraction, sont : **le schéma de base (Core Schema)** et **le schéma commun (Common Schema).**

2.3.4.1 Le schéma de base

Il définit les classes et les relations de base qui représentent les concepts relatifs à la gestion de n'importe quel système. Ce modèle maintient un niveau d'abstraction très élevé, puisqu'il regroupe les éléments communs dans tous les systèmes gérables, indépendamment de toute implémentation. Toutes les classes de ce modèle héritent de la classe de base **CIM_ManagedElement**, dont les propriétés sont communes pour n'importe quel objet géré (InstanceID, ElementName, etc.).

La figure 3.4.1 comporte un diagramme de classes UML définissant le modèle de base simplifié (sans attributs ni méthodes) :

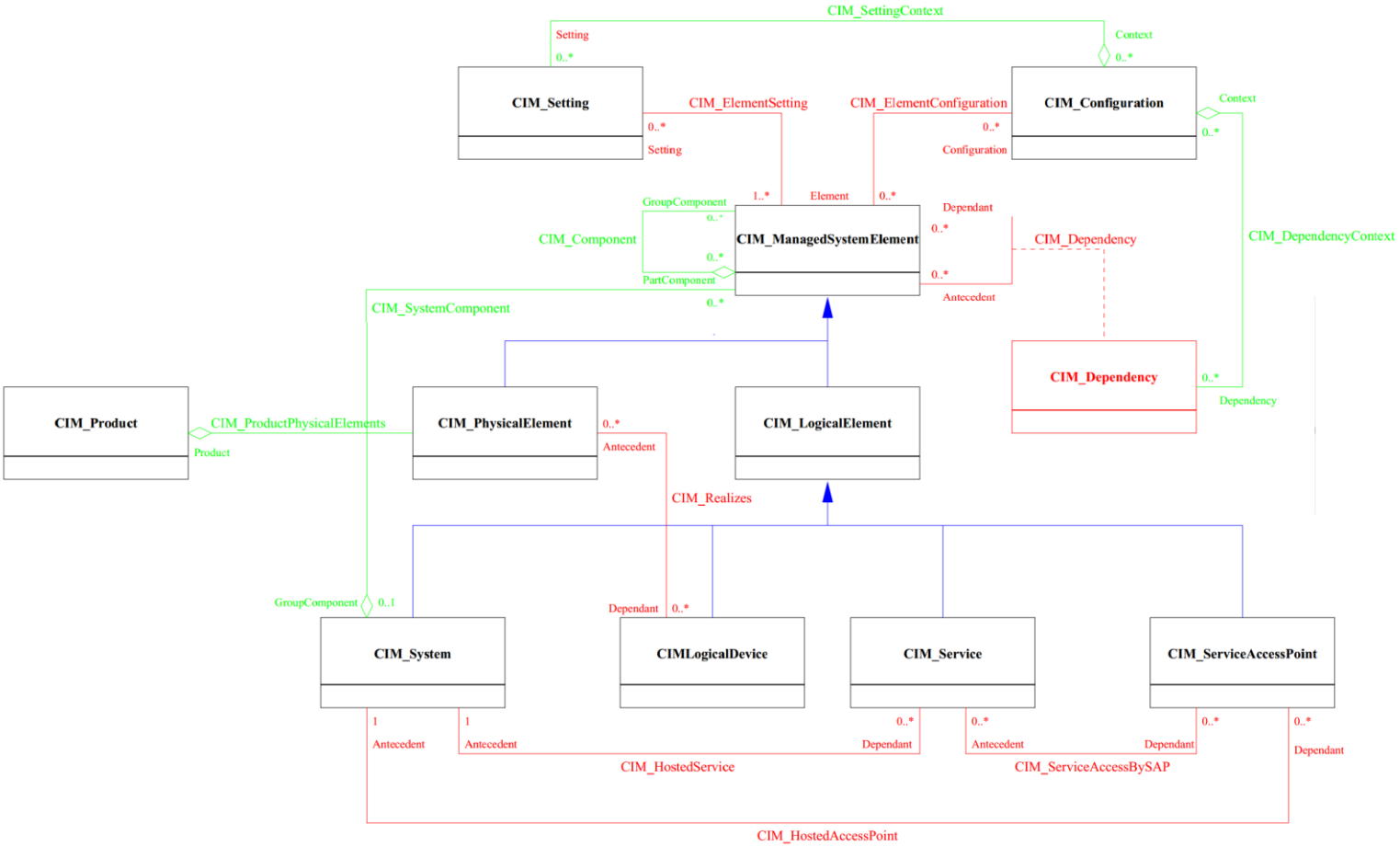

Figure 2-5. Diagramme de classes UML définissant le modèle de base simplifié [7].

Ce méta-modèle a été proposé par le DMTF pour que tous les méta-modèles dérivés soient unifiés en offrant un maximum d'homogénéité. Pour cette raison, toute modélisation d'une entité doit obligatoirement être dérivée de ce schéma ou d'un de ses dérivés.

2.3.4.2 Le schéma commun

Ce schéma est une extension du schéma de base. Il maintient un niveau d'abstraction inférieur à celui de ce dernier car, il présente des spécifications applicables à des domaines spécifiques. Ce schéma se décompose en 13 sous-schémas associés aux domaines de gestion suivants [8]:

- **Le domaine System :** il permet de modéliser, de manière générique, n'importe quel système distribué. Il présente un ensemble de classes qui permettent de représenter un ordinateur (processeur, mémoire, etc.), un groupe d'ordinateurs, les objets de virtualisation d'un système, et même les composants logiques d'un système (système d'exploitation, bios, processus, service de boot, thread, etc.). C'est l'un des schémas les plus stables et les plus complets.

- **Le domaine Network :** il permet de modéliser tout les composants d'un réseau. Il maintient la gestion des services réseaux, des protocoles, de la qualité de service, etc. Ce schéma est énorme, incomplet et instable. Il est très difficile de se retrouver dans ce schéma et de le comprendre, raison pour laquelle le DMTF a publié un livre blanc dédié uniquement à ce schéma.

- **Le domaine Physical :** il permet de modéliser les composants physiques d'une entité gérée. Il comprend des objets qui permettent de représenter n'importe quel composant physique d'un système géré (rack, puce, carte, transformateur, etc.).

- **Le domaine Application :** il permet de modéliser une application distribuée client/serveur. Il permet aussi la caractérisation de l'état d'une application distribuée (installable, exécutable, en cours d'exécution, etc.).

- **Le domaine Device :** il permet de modéliser les périphériques ainsi que les dépendances entre eux. Ce schéma a été raffiné plusieurs fois afin d'y inclure un schéma de sauvegarde contenants tous les objets gérés participants à cette tâche (interface série, contrôleur de bus USB, disque dur, DVD, etc.).

- **Le domaine Policy :** il permet de modéliser n'importe quelle politique pouvant être exprimée de la façon suivante :

IF (condition1 AND condition2) OR (condition3 AND condition4) THEN action

Une politique est composée d'un ensemble de règles. Une règle se compose à son tour de conditions (relatives à un état du système géré) et d'actions (applicables sur le système géré). À l'origine, ce schéma a été proposé pour la gestion de la qualité de service. Cependant, son niveau d'abstraction, relativement élevé, lui permet de

modéliser n'importe quel type de politique comme : **les politiques de sécurité**, **les politiques de contrôle d'accès**, **les politiques de routage**, etc.

- **Le domaine User :** il permet de modéliser les utilisateurs dans un système géré. Les informations modélisées correspondent aux informations d'identification, d'adresse, de comptes, de modèles d'authentification et de relation d'appartenance aux groupes. Tout comme le schéma Network, ce schéma est complexe et incomplet, malgré qu'il a été simplifié plusieurs fois.

- **Le domaine Event :** il permet de modéliser les événements pouvant se produire dans un système géré. l'importance de la notification des événements dans la gestion offre à ce schéma une grande importance, malgré sa simplicité.
- **Le domaine IPsecPolicy :** est une extension du schéma Policy qui permet de modéliser des politiques IPsec.

- **Le domaine Metrics :** permet modéliser des systèmes de collecte et de gestion dynamiques des informations de métrique. Ce schéma relativement simple est généralement couplé avec d'autres schémas.

- **Le domaine Support :** permet de modéliser les informations qui doivent être échangés à des fins de support, de dépannage et de débogage. Ce modèle est dédié à la formalisation des connaissances relatives aux solutions à des problèmes identifiés et résolus et pouvant concourir à résoudre des problèmes similaires. Il s'agit d'un support d'échange (et non pas de stockage) d'informations générées par des « producteurs de connaissances» pour être utilisées par des consommateurs.

- **Le domaine Data base :** il permet de modéliser les composants d'un environnement de gestion de base de données. Ses trois composants essentiels sont : le **database system software**, le **common database entity** et les **database services.**

- **Le domaine Interop :** il définit les composants de gestion qui décrivent l'infrastructure WBEM et comment d'autres composants WBEM comme les providers et les adaptateurs de protocoles, peuvent interagir avec elle. Ce schéma est à son tour subdivisé en 4 modèles : **CIM Object Manager Model**, **Namespace Model**, **Provider Model**, **Protocole Adapter Model**.

2.4. Le modèle de communication

Afin de véhiculer les informations de gestion CIM entre les différentes entités de l'architecture WBEM, le DMTF propose un protocole de communication fondé sur deux standards du web : le langage **XML *(eXtensible Markup Language)*** et le protocole **HTTP *(Hyper-Text Transport Protocol)***.

En plus de ces deux standards, le DMTF propose un ensemble standard d'opérations de gestion appelées **opérations CIM** permettant l'accès et la manipulation des données CIM [7][8][15][18].

2.4.1 Le protocole de communication XML/HTTP

Ce protocole est un couplage des deux standards **XML** et **HTTP**. Le premier est utilisé pour encoder l'information CIM alors que le second est utilisé pour le transport proprement dit. La figure 4.1 représente une communication client/serveur entre deux entités WBEM.

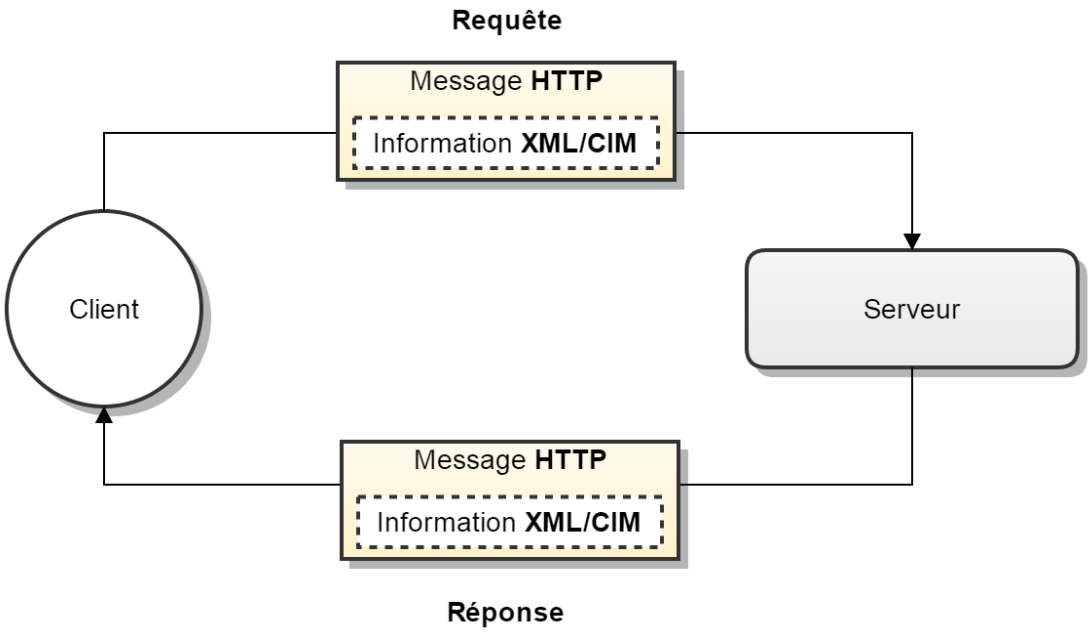

Figure 2-6. Communication client/serveur WBEM.

Au niveau de l'implémentation, cette communication est assurée par des modules dédiés qui sont :

- Le **serveur HTTP** et l'**encodeur/décodeur CIM** au niveau du serveur CIM.
- **Le client HTTP** et **l'encodeur/décodeur CIM** au niveau du client CIM.

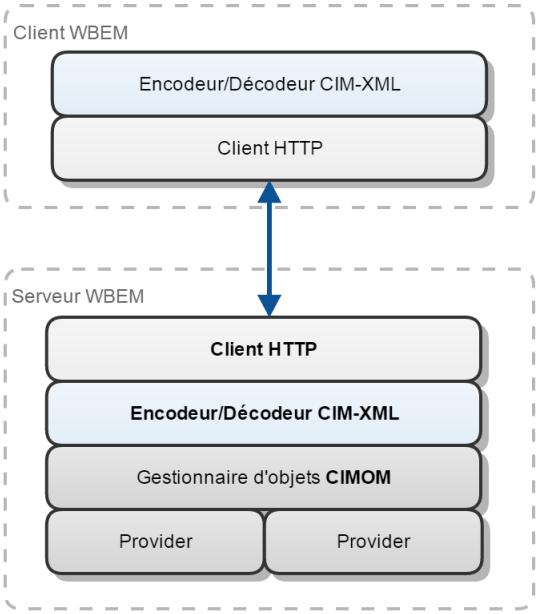

Figure 2-7. Modules de communication WBEM.

2.4.2 Les opérations CIM

Les opérations CIM sont définies comme des appels de méthodes sur des objets gérés. Bien entendu, ces appels seront traduits par des requêtes de bas niveau sur les véritables entités gérées. Deux types d'opérations CIM sont distingués : les **opérations intrinsèques** et les **opérations extrinsèques**.

- Les opérations Intrinsèques : Ces opérations permettent au client CIM de demander au serveur CIMOM, d'obtenir ou de modifier l'état des objets de l'espace de nommage cible (classes, instances, etc.). Conceptuellement, ces opérations correspondent aux opérations de gestion classique de type M-GET et M-SET dans le protocole CMIP proposé par l'ISO. Les opérations intrinsèques sont classifiées en 7 familles, leur implémentation dépend du modèle fonctionnel. C'est à dire, chaque modèle fonctionnel d'un système géré nécessite l'implémentation d'au moins une famille. Les familles d'opérations sont :

1. **Les opérations de lecture de base (Basic Read) :** elles regroupent les opérations de lecture simple des données maintenues par un gestionnaire d'objets CIMOM. Parmi les opérations les plus importantes on retrouve :
   - GetClass : retourne la définition d'une classe implantée dans l'espace de nommage cible. Elle prend en arguement obligatoire le nom de la classe souhaitée.
   - **EnumerateClasses :** retourne la liste de toutes les classes définies dans l'espace de nommage cible, ou seulement les classes dérivée d'une classe donnée comme argument.
   - **GetInstance :** retourne la définition de l'instance donnée comme argument.
   - **EnumerateInstances :** retourne l'énumération de toutes les instances d'une classe.

2. **Les opérations d'écriture de base (Basic Write) :** cette famille d'opérations contient une seule opération qui est la modification de la valeur d'une propriété d'une instance donnée. Elle doit être implémentée dans tous les systèmes utilisant les opérations de lecture de base. Cette opération est :

- **SetProperty :** permet de d'affecter une nouvelle valeur à une propriété d'une instance donnée, dans l'espace de nommage cible.

3. **Les opérations de manipulation de schémas (Schema Manipulation) :** elle regroupe les opérations qui permettent de créer, supprimer ou modifier une classe dans l'espace de nommage cible. Cette famille contient 3 opérations :
   - **CreateClass : permet de** créer une nouvelle classe dans l'espace de nommage cible. Elle requière comme argument la définition en langage MOF de la nouvelle classe.
   - **ModifyClass :** permet de modifier la définition d'une classe dans l'espace de nommage cible. Elle requière comme argument la définition en langage MOF de la classe modifiée.
   - **DeleteClass :** permet de supprimer la définition d'une classe dans l'espace de nommage cible. Elle requière donc comme argument le nom de cette classe.

4. **Les opérations de manipulation d'instances (Instance Manipulation) :** similaire à la famille précédente, cette famille contient les opérations : **CreateInstance, ModifyInstance** et **DeleteInstance**, qui sont invoquées respectivement dans le but de créer, modifier ou supprimer des instances d'objets dans l'espace de nommage cible.
5. **Les opérations de parcours d'associations (Association Traversal) :** elles permettent de lancer des requêtes sur des définitions et des instances d'associations (relations) entre les objets CIM. Il s'agit ici d'opérations très puissantes qui font toute la force de l'approche CIM/WBEM. En effet elles permettent à partir d'une instance définie et du nom d'une association, données en argument, d'identifier la classe et toutes instances dérivées associées à l'instance de départ. Comme exemple, il est possible d'identifier tous les composants physiques et logiques d'un ordinateur (instance de CIM_ComputerSystem) à travers les associations qu'il possède avec les différentes classes (CIM_Processor, CIM_Memory, CIM_NetworkPort etc.).
   L'opération la plus importante est **Associators :** elle permet de retourner l'ensemble des classes et des instances en relation avec l'objet (classe ou instance) donné en argument. Un autre paramètre peut être donné pour spécifier l'association qui assure les relations. C'est à dire, ce paramètre permet d'afficher les objets qui ont l'association donnée en paramètre, avec l'objet concerné.

6. **Les opérations d'exécution de requêtes (Query Execution) :** Cette famille, un peu spéciale, comporte une seule opération qui est une demande d'exécution d'une requête d'interrogation CIM sur la base CIMOR. Il s'agit là encore d'un aspect très puissant de l'approche CIM/WBEM qui permet de sélectionner des objets CIM (classes ou instances) de la même manière qu'une requête SQL sélectionnera des lignes de tables sur les bases de données relationnelles. Ce sont des requêtes multicritères sur tout type d'attributs (clé ou simple) qui vont permettre ensuite d'exécuter des opérations de base sur les résultats. Cette opération est **ExecQuery : elle permet à un client de lancer des requêt**es sur les objets maintenus dans l'espace de nommage cible. Cette opération prend comme arguments le nom du **langage de la requête** suivi de **la requête de sélection elle même.** WBEM définit pour le moment un seul langage de requêtes appelé le **CQL *(CIM Query Language)*.**

7. **Les opérations de déclaration de qualificateurs (Qualifiers Declaration) : elles per**mettent de lancer des requêtes sur des qualificateurs. Parmi ces opér**ation*s on retrouve :***

   - *G*etQualifier : permet de demander la définition d'un qualificateur dans l'espace de nommage ciblé.

   - **SetQualifier :** permet de demander la création ou la modification d'une définition de qualificateur dans l'espace de nommage ciblé.

   - **DeleteQualifier :** permet de demander la suppression de la définition d'un qualificateur dans l'espace de nommage ciblé.

Les opérations Intrinsèques : Elles représentent les méthodes pouvant être invoquées sur n'importe quel type d'objets. Ce sont les méthodes définies dans les classes CIM, telle que la méthode **SetNouveauID** donnée dans l'exemple de la classe **Employé** (section 3.3). Ce type d'opération est équivalent au service **M-Action** dans la norme ISO.

2.5 Sécurité de WBEM

Les mécanismes de sécurité implémentés dans l'approche WBEM peuvent être regroupés en fonction de leurs objectifs qui sont : **la sécurisation des communications** et **le contrôle des accès au CIMOM** (qui donne accès au CIMOR et aux *Providers*) [21].

2.5.1. Sécurisation des communications

Pour garantir la **confidentialité** et **l'intégrité** des données échangées entre un client et un serveur WBEM ainsi que leur **authentification**, WBEM fait appel au protocole **HTTPS *(HyperText Transfer Protocol Secure)***. On rappel que le HTTPS n'est qu'un couplage des protocoles **HTTP** et **SSL *(Secure Sockets Layers)*** qui se base sur le principe des certificats numériques, le chiffrement asymétrique et le hachage. SSL permet de contrer les attaques de l'**homme du milieu (Man in the middle)**. Toutes les implémentations de WBEM proposent un outil dédié à la configuration de SSL.. Pour le projet OpenPegasus (implémentation open source de CIM/WBEM) c'est le programme cimtrust qui assure cette fonction.

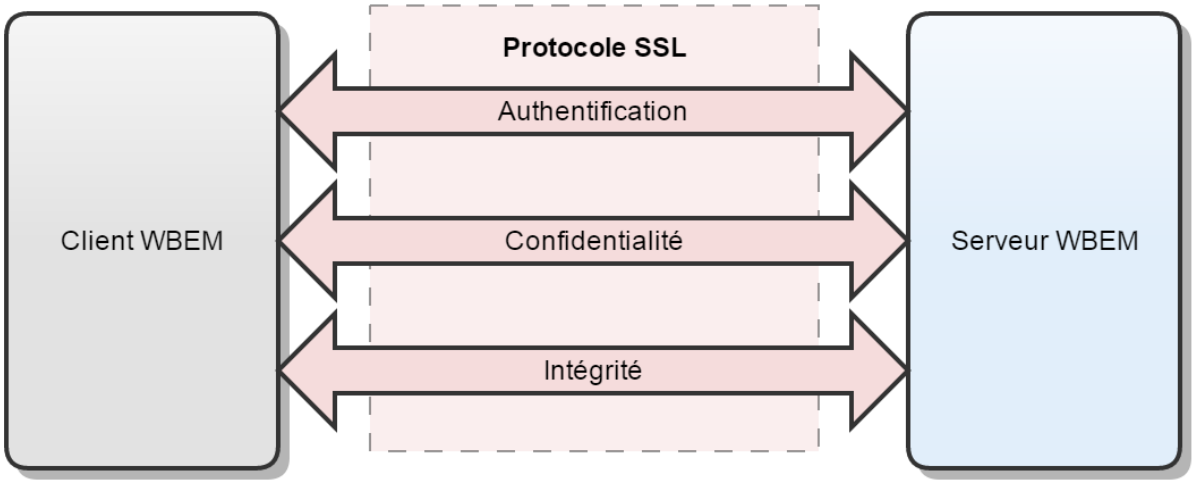

Figure 2-8. Sécurité des échanges entre client et serveur WBEM.

Les détails du protocole SSL, qui est très connu, ne sont pas présentés dans ce document.

5.2. Contrôle des accès au CIMOM

Lorsque la connexion entre le client et le serveur WBEM est établie correctement, le serveur WBEM utilise le nom de l'utilisateur (déjà authentifié) pour contrôler ses accès aux espaces de nommage. WBEM prend en charge deux types de contrôle d'accès qui sont :

- **Les listes de contrôle d'accès *(ACL)*** qui se basent sur les permissions des utilisateurs sur les espaces de nommage présents dans le CIMOR. Chaque tentative de lecture/écriture d'un client dans un espace de nommage (en invoquant une méthode donnée) est autorisée si et seulement si, ce client à le droit de lecture/écriture sur l'espace de nommage cible.
- **Contrôle d'accès basé sur les rôles *(RBAC)*** qui n'est pas utilisé dans les implémentations de WBEM. C'est un contrôle d'accès *RBAC* standard permettant de définir des rôles qui peuvent être attribués à des utilisateurs, pour ensuite, contrôler les accès sur la base de ces rôles.

Pour garantir la **non-répudiation** et permettre l'**audit** du système, WBEM inclue un mécanisme permettant au serveur de journaliser certains événements comment les tentatives d'authentification des utilisateurs (accomplies et échouées) ainsi que leurs accès non autorisés.

La configuration des mécanismes contrôle d'accès est assurée par l'outil **cimauth *(CIM Authentication)*** proposé par le projet OpenPegasus.

2.6 Gestion des politiques dans WBEM

Contrairement aux anciens standards de gestion, qui permettent généralement de gérer des entités relativement simples, WBEM apporte non seulement la possibilité de gérer des systèmes complexes mais aussi, la possibilité de gérer des entités abstraites. Parmi ces entités, la gestion des politiques pour laquelle le DMTF propose le modèle **CIM policy [9]** déjà vu dans la section 2.3.4.2. Les politiques modélisées peuvent être vues comme des configurations du système permettant de contrôler son comportement. Comme le cas des politiques de contrôle d'accès et les politiques de qualité de service.

Afin de baisser le niveau d'abstraction et d'offrir un modèle répondant aux besoins des organisations, le DMTF propose en plus de *CIM policy profile[12]* un autre modèle dédié aux politiques de contrôle d'accès. Ce modèle est appelé ***CIM Integrated Access Control Policy Management model*** **[13]** et son diagrammes de classes UML.

2.7 Conclusion

En synthèse, l'approche WBEM est un standard permettant la gestion homogène d'un ensemble de ressources hétérogènes complexes (composants physiques, services, utilisateurs, etc.). Contrairement à toutes les autres approches de gestion, WBEM prend en compte l'existence et le déploiement des autres approches de gestion. Cette prise en compte signifie que les objets gérés issus d'approches différentes sont vus comme des objets CIM ordinaires, accessibles par n'importe quel client WBEM.

Grâce à sa puissance d'expression, aux diverses implémentations propriétaires et open source et au soutient dont il bénéfice, WBEM est considéré comme l'un des standards les plus prometteurs du domaine. Aujourd'hui, plusieurs implémentations de WBEM sont proposées. Comme par exemple: **WMI *(Windows Management Instrumentation)*** de Microsoft, **Solaris WBEM Services software** de Oracle, ainsi que **OpenPegasus [27]** qui est l'implémentation open source fourni par l'OpenGroup[28] , le projet SBLIM soutenu par IBM [30]et OpenLMI de Fedora[27].

Nous avons choisi WBEM pour la réalisation de notre projet, à cause de sa capacité de gérer des entités complexes, vu que la gestion des politiques n'est pas un aspect supporté par n'importe quelle approche. De plus, l'interopérabilité offerte par WBEM a répondu à notre besoin de réalisation d'un un système de gestion standardisé pouvant être intégré dans d'autres systèmes.

CHAPITRE 3

POLITIQUES DE SECURITE SELINUX

3.1 Introduction

Présenté pour la première fois en mars 2001 par le bureau de la sécurité de l'information de la ***National Security Agency (NSA)***, ***Security Enhanced Linux (SELinux)*** représentait à l'époque, un patch pour le noyau **Linux** implémentant une architecture de contrôle d'accès poussée de type obligatoire ou ***MAC (Mandatory Access Control)***. Depuis son apparition, SELinux a été développé par la communauté open source en collaboration avec la NSA.

Depuis l'apparition de l'infrastructure ***LSM (Linux Security Modules,*** *voir la prochaine section)* , *SElinux* a été extrait du noyau sous la forme d'un module s'appuyant sur cette infrastructure qui permet d'intégrer n'importe quel module de contrôle d'accès, tout en modifiant le moins possible le noyau Linux. SELinux est actuellement intégré par défaut dans certaines distributions Linux comme **Fedora**, **Red Hat** et **CentOS.** Il existe aussi sous forme de packages pour toutes les autres distributions comme **Debian**, **Ubuntu**, **SuSe** et **Gentoo**.

Afin de combler les limitations du contrôle d'accès standard ou *DAC (Discretionary Access Control) de* Linux, et qui sont notamment **l'utilisation de seulement deux grandes catégories d'utilisateurs qui sont les administrateurs et les autres** et **le principe très dangereux du super utilisateur root qui peut tout faire,** SELinux offre une couche de sécurité rajoutée au-dessus de ce contrôle d'accès discrétionnaire standard (droits d'accès standards de Linux). Le contrôle d'accès obligatoire est un contrôle d'accès où les décisions sont prises en fonction de certaines règles fixes qui représentent une politique de sécurité. Contrairement au contrôle d'accès discrétionnaire, le MAC ne permet pas aux utilisateurs de changer les droits d'accès relatifs à leurs propres objets.

3.2 Linux Security Modules

LSM est une infrastructure permettant au noyau Linux de prendre en charge divers implémentations de contrôle d'accès, sans qu'elles y soient intégrées. Au lieu que toute architecture de contrôle d'accès soit intégrée dans le noyau, LSM ajoute des crochets (*hooks)* permettant d'intercepter les appels systèmes représentant des accès aux objets. Pour assurer la compatibilité avec les applications existantes, LSM positionne ses crochets de manière à ce que les vérifications DAC soient effectuées en premier lieu, ainsi, LSM n'intervient que si elles aboutissent.

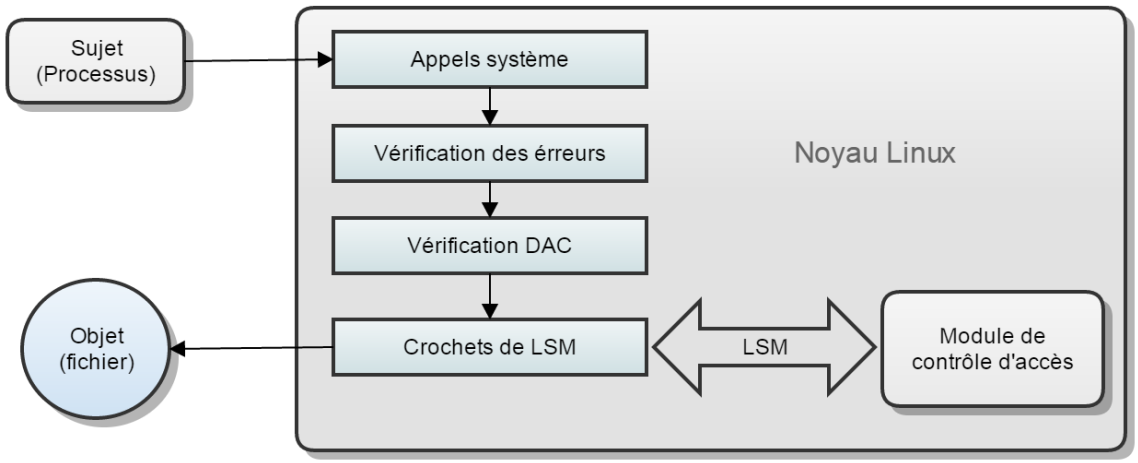

Figure 3-1. Architecture de LSM[15].

3.3 Architecture interne de SELinux

L'architecture interne de SELinux se base sur trois composants : ***SELinux Filesystem***, ***Security Server*** et ***Access Vector Cache (AVC)***.

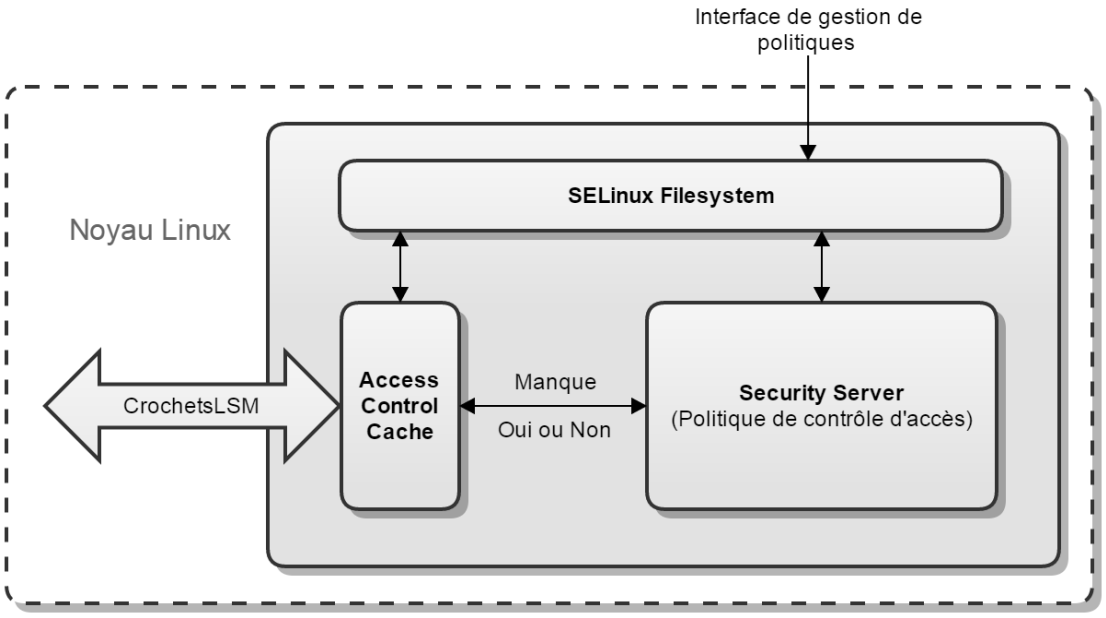

Figure 3-2. Architecture interne de SELinux [15]

Le ***SELinux Filesystem*** sert essentiellement à contenir les règles actives du système qui seront chargées dans le ***Security Server*** ainsi qu'à récolter des statistiques du ***AVC***. L'**AVC** contient seulement une partie des règles afin de réduire le temps du contrôle (au lieu de chercher dans toutes les règles) pour ne pas ralentir le système. Si l'AVC ne contient pas les règles nécessaires pour répondre à une demande d'accès, il envoi une requête au ***Security Server*** qui contient toutes les règles activées.

3.4 Contrôle d'accès obligatoire et le contrôle d'accès discrétionnaire

Afin de répondre aux besoins de sécurité, plusieurs modèles de contrôles d'accès existent. Ces modèles présentent un ensemble de règles qui permettent de contrôler l'accès aux ressources, dépendamment des besoins de l'organisation.

Le contrôle d'accès standard de Linux est de type « Discrétionnaire » ou DAC. Il permet au propriétaire d'un objet de définir pour un ensemble de permissions relatives à lui même, son groupe et les autres. Ce modèle doit son nom au fait que l'attribution des permissions d'accès sur un objet est faite à la « **discrétion »** de son propriétaire.

Contrairement au DAC, Le contrôle d'accès obligatoire *(MAC)* les décisions sont prises en fonction de certaines règles fixes représentant la politique de sécurité. Ces règles sont généralement définies par une entité tierce, le plus souvent externe à la politique de sécurité. Ainsi, même les utilisateurs ayant des privilèges élevés ne peuvent pas intervenir dans l'attribution des droits d'accès, chose qui peut s'avérer utile pour régler le problème des supers utilisateurs (le **root** dans Linux) pouvant tout faire.

3.5 Les concepts de base des mécanismes de SELinux

La compréhension des concepts de base de SELinux se fonde sur la compréhension du fonctionnement des différents mécanismes de contrôle d'accès qu'il utilise. Ces mécanismes qui sont utilisés de façon imbriquée, sont le **renforcement de type** ou **TE** ***(Type Enforcement)***, le **contrôle d'accès basé sur les rôles** ou **RBAC** ***(Role-Based Access Control)*** et la **sécurité multi-niveau MLS** ***(Multi-Level Security)*****[15][16][24].** Les deux premiers mécanismes sont détaillés respectivement dans la section 4.2 et la section 4.3. Alors que le dernier mécanisme (**MLS)** est abordé brièvement dans la section 3.4.4.

3.5.1 Le contexte de sécurité pour le renforcement de type

Tout mécanisme de contrôle d'accès se base sur un certain type d'attributs de contrôle associé à tous les couples sujets/objets du système (processus/fichiers dans Linux). Par exemple, dans le DAC de Linux l'attribut utilisé est **le nom d'utilisateur** (root, system, user, etc.). C'est à dire, le contrôle est appliqué en fonction de ce nom d'utilisateur et de ses privilèges.
Dans SELinux, l'attribut de contrôle d'accès est appelé **Contexte de sécurité.** Tout sujet/objet a un contexte de sécurité constitué de trois éléments : **identité**, **rôle** et **type**.
L'identité SELinux est similaire au **nom d'utilisateur** Linux. Lors de l'ouverture d'une session, l'utilisateur (identifié par son nom d'utilisateur) se voit attribué une identité SELinux qu'il ne changera jamais. Le rôle est le type sont aussi attribué aux utilisateurs en fonction de certaines règles. L'attribution des rôles et des types sera détaillée plus loin dans ce chapitre.

Chaque partie du contexte de sécurité est utilisé par un mécanisme de sécurité. L'identité et le rôle sont utilisé par le RBAC, tandis que le type est utilisé par le TE.

La forme la plus utilisée pour représenter textuellement le contexte de sécurité est la suivante :

Identité_u : Rôle_r : Type_t

Par convention, ces éléments sont suffixés de la manière suivante:

- le suffixe **_u** pour les identités.
- le suffixe **_r** pour les rôles.
- le suffixe **_t** pour les types (domaines et étiquettes).

Généralement, les **types** associés aux objets (fichiers) sont appelés **étiquettes** ou **Labels**, alors que les **types** associés aux sujets (processus) sont appelés **domaines**.

Le tableau 3.1 ci-dessous présente quelques exemples concrets de contextes SELinux attribués par défaut à quelques utilisateurs.

Tableau 3-1. Exemples de contextes de sécurité SELinux attribués à des utilisateurs (sujets)

Utilisateur	Identité	Rôle	Domaine	Contexte
System	system_u	system_r	system_t	System_u:system_r:system_t
Root	Root	sysadm_r	sysadm_t	root:sysadm_r:sysadm_t
User	unconfined_u	unconfined_r	unconfined_t	unconfined_u: unconfined_r: unconfined_t

Dans le deuxième exemple, l'utilisateur **root** a l'identité SELinux **root**, le rôle **sysadm_r** et le domaine **sysadm_t**. Le contexte SELinux de l'utilisateur **root** est donc **root:sysadm_r:sysadm_t.**

Le tableau 3-1. ci-dessous montre des exemples de contextes SELinux attribués à des fichiers.

Tableau 5.1- Exemples de contextes de sécurité SELinux attribués à des fichiers (objets).

Fichier	Identité	Rôle	Etiquette	Contexte
/etc/shadow	system_u	object_r	shadow_t	system_u:object_r:shadow_t
/var/www	system_u	object_r	httpd_sys_content	system_u:object_r:httpd_sys_content_t

Pour le fichier **/etc/shadow** par exemple, l'identité est **system_u**, le rôle est **object_r** et l'étiquette est **shadow_t**. Le contexte SELinux de ce fichier est donc **system_u:object_r:shadow_t**.

À un certain moment, chaque objet/sujet possède **un seul contexte** de sécurité. Dans le contexte d'un objet, **seul le type est utilisé**, c'est-à-dire on ne parle jamais du rôle d'un objet ni de son identité, mais seulement de son type.

L'identité SELinux d'un utilisateur est similaire au nom d'utilisateur standard de Linux. Elle représente un paramètre d'identification d'un utilisateur ainsi que le conteneur de rôles. Autrement dit, une identité SELinux permet d'identifier un utilisateur et de lui définir des rôles. La relation entre les rôles et les identités est détaillée dans la section 3.5.3.

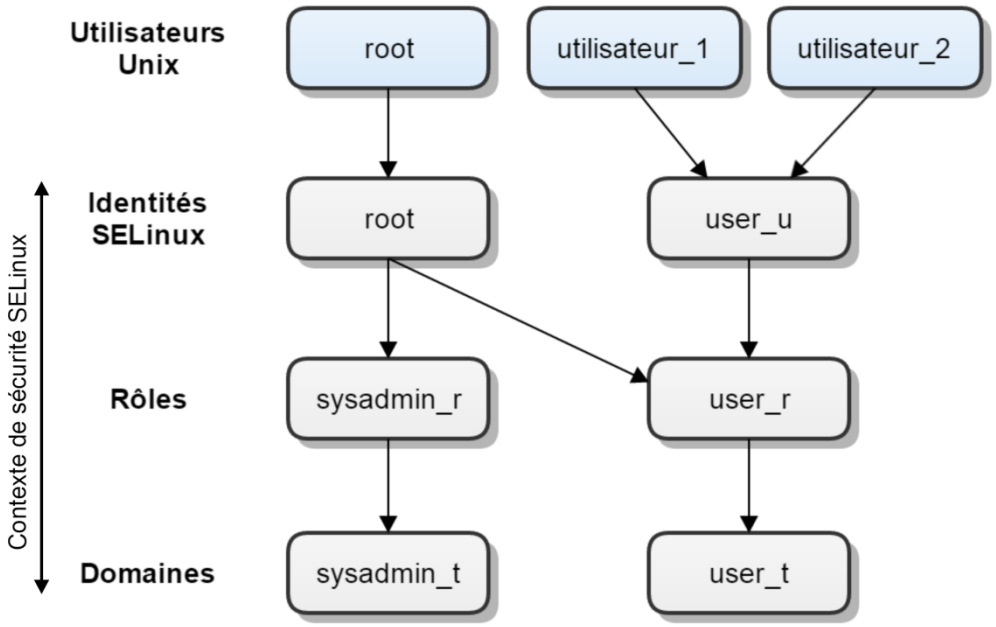

Figure 3-3. Contexte de sécurité dans SELinux [16]

Lors de l'installation de SELinux, une option, qui est **-Z**, est ajoutées à certaines commandes. On retrouve par exemple :

ls -Z : affiche le contexte de sécurité des fichiers ou des répertoires listés.

ps -Z : affiche le contexte de sécurité des processus listés.

id -Z : affiche le contexte de sécurité de l'utilisateur courant (son Shell).

3.5.2 Renforcement de type

Le contrôle d'accès par renforcement de type ***(Type Enforcement)***, ou encore appelé **imposition de contrôle d'accès par Type**, est le principal mécanisme de SELinux [15][16][24]. Il intervient au moment précis où un sujet veut accéder à un objet, pour l'autoriser ou l'interdire. La décision prise par SELinux dépend des règles du TE spécifiées dans la politique utilisée. Le TE, suit le principe général de SELinux, celui de **Whitelisting**, c'est-à-dire tout accès doit être explicitement permis. En d'autres termes, aucun accès n'est autorisé par défaut, contrairement au principe du **root** dans la sécurité standard de Linux, où tout est permis pour lui.

Dans le TE, l'accès d'un processus à une ressource doit être autorisé par des règles appelées **règles AV** (Access Vector Rules). Ainsi, une bonne politique donne aux processus le minimum de droits permettant leur bon fonctionnement. Une règle AV autorise l'accès d'un **domaine** (type de sujet) à une certaine **étiquette** (type d'objet). Ainsi, pour qu'un processus de domaine **D** accéder à un fichier d'étiquette **E**, une règle AV doit l'autoriser. La syntaxe d'une règle AV est :

allow domaine_t étiquette_t : class { permission_1 ... permission_n };

domaine_t : représente le domaine du processus auquel on permet l'accès.

étiquette_t : représente l'étiquette de l'objet auquel l'accès est permis.

class : représente la classe de l'objet auquel l'accès est permis (file, dir, socket, etc).

permissions : représente la permission accordée dépendant de la classe (read, write, exe pour un objet de classe file par exemple).

SELinux classe les objets en plusieurs **classes**. Ces classes permettent de déterminer les permissions qu'on peut avoir sur un objet. Par exemple, on peut avoir la permission **connect** sur un objet de classe **socket** et non pas sur un objet de classe **file**.

Le tableau 5.2-1 ci-dessous présente les classes d'objets supportées par SELinux alors que le tableau 5.2-2 présente les permissions relatives à la classe SELinux File

Object Class	Description
blk_file	Block files
chr_file	Character files
dir	Directories
fd	File descriptors
fifo_file	Named pipes
file	Ordinary files
filesystem	Filesystem (for example, an actual partition)
lnk_file	Symbolic links
sock_file	UNIX domain sockets

Figure 3-4. Classes SELinux relatives aux fichiers [15].

Permission	Description
append	Append to file contents (that is, opened with O_APPEND flag).
create	Create new file.
entrypoint*	File can be used as the entry point of the new domain via a domain transition.
execmod*	Make executable a file mapping that has been modified (implied by a copy-on-write).
execute	Execute; corresponds to x access in standard Linux.
getattr	Get attributes for file, such as access mode (for example, stat, some ioctls).
link	Create hard link to file.
unlink	Remove hard link (delete).
rename	Rename a hard link.
write	Write file contents; corresponds to w access in standard Linux.

Figure 3-5. Permissions relatives à la classe SELinux File[15]

Les accès interdis (seulement) par SELinux, sont enregistrés par défaut dans les journaux. Cependant d'autres règles AV existent et permettent de donner plus de flexibilité au renforcement de type. Ces types de règles sont :

- **dontaudit** : Annule l'écriture d'une action bloquée dans les journaux. Ces règles peuvent être utiles dans le cas d'un application qui subit beaucoup d'interdictions, alors qu'on connait la raison et on sait qu'elle ne présente aucune menace.
- **auditallow** : Écrit une action donnée dans les journaux même si elle est autorisée. Ces règles sont généralement utilisées dans les environnements de test (audit, expérimentation, etc).
- **neverallow** : Définit une action qui ne sera jamais autorisée même si une règle est ajoutée pour le faire.

3.5.2.1 Exemple de règle AV

Prenons l'exemple d'une règle qui autorise aux processus de domaine **user_t** l'accès en lecture et en exécution (read et execute qui sont similaires aux permissions Linux standards) aux objets étiquetés **etc_t** et dont la classe est **file**. Cette règle s'écrit comme suit :

allow user_t etc_t : file { read execute };

D'une façon plus simple, cette règle permet aux processus dont le domaine est **user_t** de lire et exécuter les fichiers étiquetés **etc_t** comme le montre la figure 3.6.

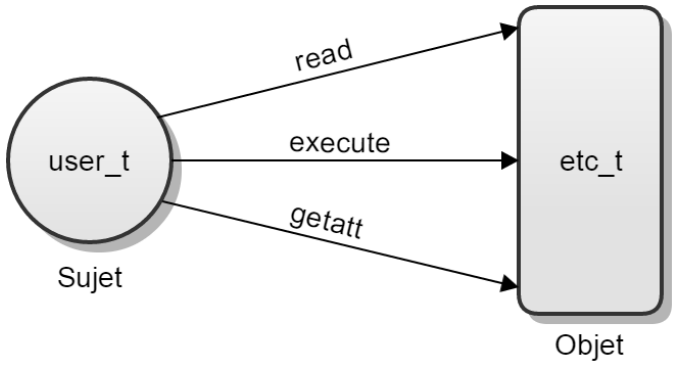

Figure 3-6. Exemple d'actions permises par une règle de renforcement de type[15]

Dans un cas pratique par exemple, pour restreindre l'accès au fichier critique **/etc/shadow**, on associe l'étiquette **shadow_t** à ce dernier. Ensuite, on permet au processus **passwd** (l'exécution du programme **/usr/bin/passwd**) de s'exécuter seulement dans le domaine **passwd_t** créé spécialement pour lui. Après avoir permis (avec une règle AV) seulement à **passwd_t** d'accéder à l'étiquette **shadow_t**, on est sûr que seul le processus **passwd** aura accès au fichier **/etc/shadow**, et c'est ce qui fait la force du TE de SELinux.

3.5.2.2 Transitions de domaines

La transition de domaine est un concept très important dans SELinux [15], car, elle permet d'avoir une certaine flexibilité pour des cas particuliers. Un utilisateur qui se connecte se voit attribué un contexte de sécurité, et donc, au moment de sa connexion il se trouve dans un domaine précis. Si cet utilisateur est obligé d'accéder à des objets non accessibles à partir de son domaine, SELinux lui offre la possibilité d'effectuer « une transition de domaine », à condition qu'elle soit permise.

Une transition de domaine est autorisée par une règle AV dont la syntaxe est la suivante :

allow domaine_src_t domaine_dst_t : process transition

domaine_src_t : domaine du processus qui veut faire la transition de domaine.
domaine_dst_t : domaine vers lequel le processus veut transiter.
process transition : l'action souhaitée qui est la transition du processus.

Dans l'exemple de l'accès au fichier **/etc/shadow**, on a autorisé seulement l'accès aux processus dont le domaine est **passwd_t**. Or, quand un utilisateur lance la commande **passwd** à partir du **shell** pour changer son propre mot de passe, le processus **passwd** se trouve dans le même domaine que le **shell** (vu qu'il est lancé par l'appel system **execve()** après un **fork()** à partir du **shell**). Par conséquent, le processus **passwd** doit transiter du domaine de son père, **user_t** par exemple, vers le domaine **passwd_t** pour pouvoir changer de mot de passe. Cette transition est permise par la règle suivante :

allow user_t passwd_t : process transition ;

La règle précédente autorise les transitions du domaine **user_t** vers le domaine **passwd_t**. Le processus **passwd** peut donc accéder au fichier **/etc/shadow** après une transition vers **passwd_t** comme l'illustre la figure 5.2.2.1.

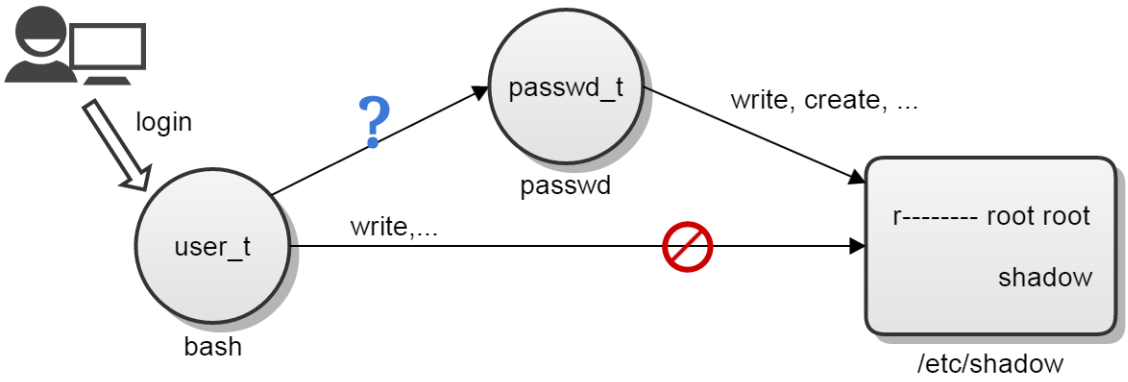

Figure 3-7. Transition de domaine avant l'accès au fichier /etc/shadow[15]

En réalité, pour qu'une transition de domaine soit autorisée, d'autres actions doivent être autorisées systématiquement. Tout d'abord, le lancement de l'exécutable **/usr/bin/passwd** (dont l'étiquette est **passwd_exec_t**) par le **bash** (dont domaine est **user_t**) doit être autorisé. Ceci est fait, dans ce cas, par la règle AV suivante :

allow user_t passwd_exec_t : file { getattr execute } ;

Ensuite, on doit permettre au programme **/usr/bin/passwd** d'être une entrée vers le domaine **passwd_t**, c'est à dire, l'exécutable **/usr/bin/passwd** est autorisé à rentrer dans le domaine **passwd_t** une fois lancé par un utilisateur (en devenant un processus). Le principe de point d'entrée est extrêmement important. Il nous force à déterminer, un par un, tous les exécutables qui permettent de rentrer dans un domaine donné. La figure 5.2.2.2 représente un exemple de transition à partir d'un point d'entrée. Dans ce cas, la règle qui permet de spécifier ce point d'entrée est :

allow passwd_t passwd_exec_t : file entrypoint ;

Enfin, il faut permettre la transition à partir du domaine **user_t** vers le domaine **passwd_t**, ce qui est la permission de transition proprement dite, par la règle déjà donnée et qui est :

allow user_t passwd_t : process transition ;

En résumé, pour qu'une transition de domaine soit permise, trois règles **allow** doivent être présentes pour vérifier que :

1. Le nouveau domaine du processus a un point d'entrée (**entrypoint**) à partir d'un fichier **exécutable**.
2. Le domaine actuel du processus a un accès **execute** sur le fichier exécutable qui est le point d'entrée (**entrypoint**).
3. Le domaine actuel du processus a un accès de transition sur le nouveau domaine.

La figure 3.8 illustre ce mécanisme de transition dans SELinux

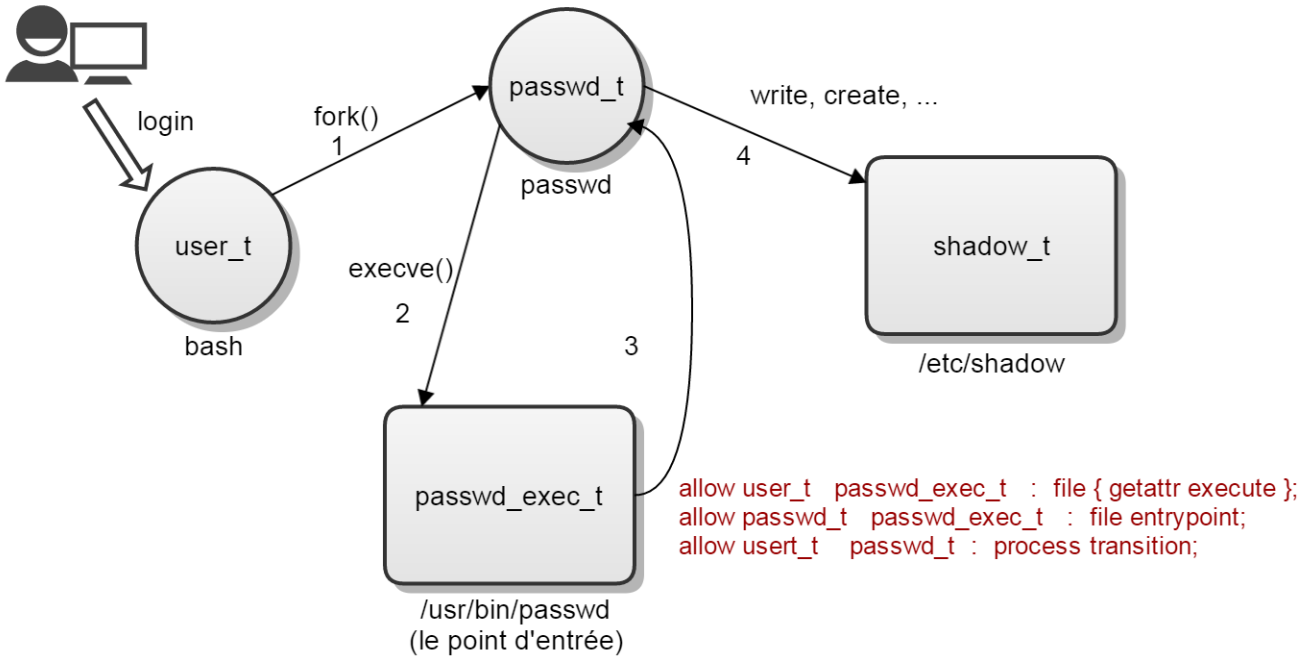

Figure 3-8. Transition de domaine avant l'accès au fichier /etc/shadow et le point d'entrée[15]

3.5.2.3 Transitions de domaines par défaut

Dans certains cas, et pour des raisons de simplicité, les transitions de domaine doivent être réalisées de façon transparente pour l'utilisateur. Comme dans le cas du fichier **/etc/shadow**, ou la transition du **user_t** vers le **passwd_t** doit être réalisée à chaque accès.

Pour gérer ces transitions par défaut, SELinux permet d'introduire un nouveau type de règles : **type_transition** dont la syntaxe est la suivante :

type_transition domaine_src_t étiquette_exec_t : process domaine_dst_t ;

Cette règle veut dire que par défaut, quand un processus de domaine **domaine_src_t** lance un exécutable d'étiquette **étiquette_exec_t**, une transition de domaine est tentée vers le domaine **domaine_dst_t**. On rappelle que le lancement d'un exécutable se fait par l'appel système **execve()** à partir du **shell.**

Il est très important de savoir qu'une règle **type_transition** tente une transition de domaine mais ne la permet pas. Pour qu'elle se produise, la transition doit être permise comme toute autre transition de domaine (par trois règles AV allow).

3.5.3 Contrôle basé sur les rôles

Le contrôle d'accès sous SELinux est principalement assuré par le renforcement de type. Cependant, le **contrôle basé sur les rôles (RBAC)** est ajouté à ce dernier, afin d'augmenter la flexibilité du système et de facilité sa gestion. Le principe du RBAC est la gestion des accès par rapport à des rôles qui servent à regrouper plusieurs privilèges. Un rôle peut être vu comme un profil d'utilisateur dans le système. Le RBAC implémenté par SELinux permet de créer un rôle, et de **lui définir un ensemble de domaines** vers lesquels il peut transiter. Ce rôle peut être affecté à une ou plusieurs identités SELinux, comme on peut affecter plusieurs rôles à la même identité.
La transition d'un utilisateur, défini par son identité SELinux, d'un domaine vers un autre est contrôlée non seulement par le renforcement de type (comme nous l'avons déjà vu dans la section 3.5.2.2), mais aussi par le RBAC. Ce dernier vérifie **si le domaine, vers lequel la transition est souhaitée, fait partie des domaines autorisés pour le rôle** de cet utilisateur.
Les rôles que peut activer un utilisateur à un instant donné et les domaines vers lesquels un rôle peut transiter doivent être préalablement définis par des déclarations. Le changement de rôle doit être autorisé par des règles *AV* qui ne sont pas présentées dans ce document.

3.5.4 La sécurité multi-niveaux

Pour certaines organisations, notamment celles qui ont une structure fortement hiérarchique, un autre mécanisme de contrôle d'accès obligatoire SELinux peut être utilisé, afin d'accroître la confidentialité des informations du système. Ce mécanisme est la **sécurité multi-niveau *(MLS)***. Le choix d'utilisation du MLS est arbitraire (on peut le désactiver sans désactiver SELinux), contrairement au renforcement de type, qui est de loin, le mécanisme le plus important dans SELinux.

Dans le MLS, les sujets/objets sont classés de deux manières : par **niveaux de sensibilité (s0,s1, etc.)** et par **catégories de sécurité (c0, c1, etc.)**. La première classification est hiérarchique, la deuxième est non-hiérarchique.
Entre les différentes sensibilités, une relation de dominance existe. C'est à dire, un niveau de sensibilité domine un autre (s0 < s1 < s3 < ...). Chaque sujet/objet se voit attribuer deux niveaux de sensibilités : un niveau **minimum** et un niveau de **maximum**. Évidement, le niveau maximum doit être le plus dominant des deux. Pratiquement, le niveau minimum est généralement le même (s0) pour tous les sujets/objets. Ce qui nous mène à ne considérer que le niveau de sensibilité maximum.
Pour les catégories de sécurité, qui ne sont pas hiérarchiques, aucune relation n'existe entre elles. C'est à dire, un sujet de catégorie **c1** n'a aucun lien avec un objet de catégorie **c2**. Un

objet peut avoir une ou plusieurs catégories. Ce qui ajoute au contexte SELinux d'un sujet/objet deux autres paramètres. La forme utilisée pour représenter l'intégralité du contexte SELinux est la suivante :

identité_t : rôle_r : type_t : sensibilité$_{min}$ - sensibilité$_{max}$: catégorie$_1$. catégorie$_n$

La gestion des accès suivant le mode MLS, après la classification des sujets/objets, se fait par le principe de **Bell et Lapadula.** Ce principe ne garantit que la confidentialité (il ne garantit pas l'intégrité). Son principe très simple est le suivant :

- un sujet ne peut lire un objet de sa catégorie que **si** son **niveau de sensibilité** est **supérieur** (domine) ou **égal au niveau de sensibilité** de cet objet.

- un sujet ne peut écrire dans un objet de sa catégorie que **si** son **niveau de sensibilité** est **inférieur** (dominé) ou **égal aux niveaux de sensibilité** de cet objet.

Ceci se résume en anglais à : **no read up no write down.** Ce principe permet de s'assurer qu'aucune information ne peut être transférée d'un niveau à un autre qui lui soit inférieur (pas digne de confiance) car, elle ne peut être lue depuis le bas ni écrite à partir du haut de la hiérarchie (comme le montre la figure 3-9 ci-après).

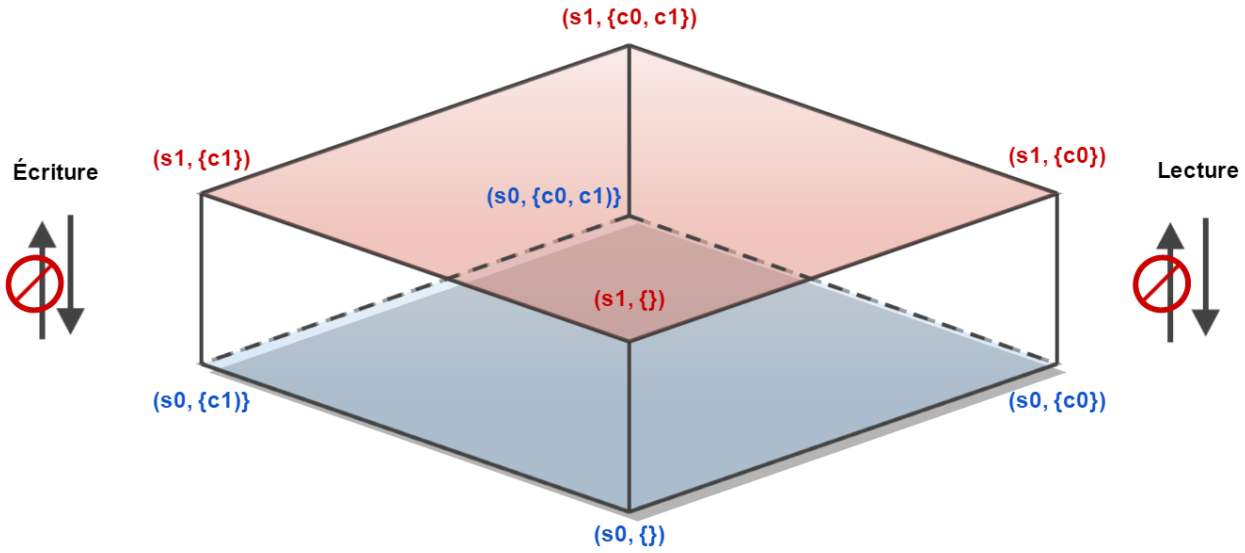

Figure 3-9. Hiérarchie composée de deux niveaux et deux catégories MLS SELinux[1]

À cause de sa complexité d'implémentation et son principe hiérarchique conçu pour les grandes organisations (gouvernements, grandes institutions sensibles, etc.), le mécanisme de sécurité multi-niveaux n'est pas pris en compte dans le cadre de ce projet.

3.6 SELinux dans la pratique

La compréhension des mécanismes de contrôles d'accès implémentés par SELinux n'est pas du tout évidente. Ainsi, la création de toutes les règles permettant le bon fonctionnement du système reste une chose impossible pour une simple organisation. Pour cette raison, la

[1] http://en.wikipedia.org/wiki/Multilevel_security

création des règles est le plus souvent laissée à des parties tierces qui proposent des packages, appelés modules. En général un module est créer par le développeur ou l'éditeur d'un package logiciel afin d'en permettre le bon fonctionnement (en plus de la sécurité) dans un environnement où SELinux est activé.

3.6.1 Modes de SELinux

Le LSM SELinux peut être configuré en trois différents modes :

- **Enforcing :** C'est le mode par défaut dans lequel la politique de sécurité est imposée. Les accès non autorisés sont bloqués et les actions sont écrites dans les journaux.
- **Permissive :** SELinux est activé mais n'impose pas la politique de sécurité. Les accès interdits sont seulement signalées et les actions sont écrites dans les journaux.
- **Disable :** SELinux est tout simplement désactivé.

Le définition du mode de SELinux peut se faire de façon temporaire (le changement n'est pas pris en compte lors du redémarrage du système) ou de façon permanante (le changement est pris en compte même après le redémarrage du système).

3.6.2 Politiques SELinux

Pour des raisons pratiques aussi, il est possible de définir des politiques SELinux regroupant des ensembles de règles qui répondent à un certain besoin de sécurité. Une politique SELinux peut être vu comme un ensemble de packages dont chacun contrôle l'accès à une application donnée. Lors de l'installation par défaut de SELinux, deux politiques sont présentes mais seulement une peut être appliquée à un moment donné. Il s'agit de**:**

- **strict :** Cette politique contrôle les accès à toutes les ressources du système en affectant des contextes de sécurité à les entités du système (utilisateurs et objets). Elle est généralement appliquée par les organisations qui requièrent une sécurité élevée.
- **targeted :** C'est la politique par défaut dans plusieurs distributions (comme CentOS). Dans cette politique, le contrôle d'accès est appliqué sur les applications critiques seulement (applications réseaux notamment httpd, dhcpd, mysqld, etc.). Les autres applications se voient attribuer le domaine **unconfined_t** sur lequel aucun contrôle d'accès n'est appliqué.

D'autres politiques SELinux sont proposées par certaines organisations, comme la politique **mls** appliquant le contrôle d'accès basé sur les nivaux. Cependant, la création de politiques SELinux reste une tâche difficile destinée aux très grandes organisations (gouvernements, organisations militaires, etc.), puisqu'une simple politique peut contenir des centaines de milliers de règle.

La figure 7.2 ici-bas montre le résultat de la commande **sestatus** permettant de récupérer le statut de SELinux (activé/désactivé), le mode par défaut, le mode permanant, la politique appliquée ainsi que d'autres informations.

```
[root@localhost ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          permissive
Policy version:                 24
Policy from config file:        targeted
[root@localhost ~]#
```

Figure 3-10. Résultat de la commande sestatus.

3.6.3 Modules SELinux

Les modules SELinux renferment un ensemble de règles permettant d'assurer le fonctionnement sécurisé d'une application donnée. Les règles constituant une politique SELinux sont regroupées en plusieurs modules en fonction des applications qu'elles contrôlent. De plus, les applications critiques sont livrées avec leurs propres modules, sous forme de package (*Policy Package* dont l'extension est .pp), permettant leur bon fonctionnement. Parmi les modules SELinux présents par défaut dans la politique **targeted** on retrouve: **apache 2.1.2**, **dhcp 1.8.1**, **ipsec 1.10.2**, etc.

La gestion des modules SELinux se fait à l'aide de l'outils **semodule** qui permet de : lister les modules installés (voir figure 7.3), installer de nouveaux modules et désinstaller des modules existants.

```
[root@localhost ~]# semodule -l
abrt    1.2.0
accountsd       1.0.0
ada     1.4.0
afs     1.5.3
aiccu   1.0.0
aide    1.5.0
amanda  1.12.0
amtu    1.2.0
antivirus       1.0.0
apache  2.1.2
apcupsd 1.6.1
arpwatch        1.8.1
asterisk        1.7.1
```

Figure 3-11. Liste de modules affichés par l'outil semodule.

3.6.4 Les booléens SELinux

Les booléens dans SELinux permettent de modifier une politique sans avoir la moindre connaissance sur la rédaction des règles. Techniquement, un booléen regroupe un ensemble de règles. La désactivation du booléen permet de désactiver les règles qu'il contient. Par exemple, dans la politique **targeted**, la permission de apache, d'agir comme un serveur FTP en écoutant le port FTP, est accorée par un ensemble de règles regroupées dans le booléen **httpd_enable_ftp_server**. Ainsi, l'activation/désactivation de ce booléen permet d'autoriser/interdire à apache d'agir ou non comme un serveur FTP.

La gestion des booléens SELinux est assurée par les outils: **getsebool** (voir la figure 3-11), **setsebool** et **semanage** (avec l'option boolean) qui permettent de lister l'état des booléens ainsi que leur activation/désactivation de façon temporaire ou permanente.

```
[root@localhost ~]# getsebool -a
abrt_anon_write --> off
abrt_handle_event --> off
allow_console_login --> on
allow_cvs_read_shadow --> off
allow_daemons_dump_core --> on
allow_daemons_use_tcp_wrapper --> off
allow_daemons_use_tty --> on
```

Figure 3-11. Liste de booléens affichés par l'outil getsebool.

3.7 Conclusion

SELinux est un LSM permettant d'ajouter au système Linux une autre couche de sécurité très robuste. Il place des crochets *(hooks)* au niveau des appels système afin de les intercepter pour vérifier leur légitimité en implémentant un contrôle d'accès obligatoire ***(MAC)***. SELinux utilise plusieurs mécanismes de contrôle d'accès dont le plus important est le **renforcement de type *(Type Enforcement)***, dans lequel les processus n'ont accès qu'aux ressources dont ils ont besoin. Le choix de SELinux pour l'application des politiques de sécurité dans le cadre de ce projet, est dû essentiellement au contrôle d'accès obligatoire très robuste qu'il implémente, ainsi qu'à ses mécanismes de journalisation très performants. De plus, les classes et les permissions dans SELinux lui donnent la possibilité d'appliquer un contrôle d'accès de très grande précision (comme donner la permission d'ajouter et non pas supprimer des données).

CHAPITRE 4

CONCEPTION DU SYSTÈME

4.1 Introduction

Nous rappelons que l'objectif de ce travail est la conception et réalisation d'un système de gestion de politiques de sécurité normalisé. Pour atteindre cet objectif, WBEM a été choisi pour satisfaire nos besoins en terme de standards de gestion (modélisation des règles, leur distribution dans le réseau, etc.). Pour l'application de ces politiques, le module de sécurité SELinux a été choisi à cause de ses mécanismes de sécurité très robustes (contrôle d'accès obligatoire basé sur le renforcement de type, nouveaux types permissions, etc.).

Bien que WBEM est aujourd'hui l'un des standards de gestion les plus prometteurs, l'implémentation de *Providers* permettant de gérer n'importe quel composant d'un système reste son plus grand défit. Nous avons donc décidé de participer à l'enrichissement de WBEM, en implémentant un *Provider* dédié à SELinux.

4.2 L'architecture de notre système de gestion

En s'inspirant des principes de l'architecture et du protocole de gestion de politiques appelé **COPS *(Common Open Policy Service)***, ainsi que le standard de gestion de politiques de sécurité de type RBAC, XACML nous avons défini deux entités principales assurant la définition et l'application de nos politiques. C'est entités sont :

- **Le Point d'Administration des politiques** : appelé **PAP (Policy Administration Point)** et représenté par un client CIM/WBEM, il fait appel aux services de PDP (ci-après) afin de définir, distribuer et appliquer des politiques
- **Le point de décision des politiques:** appelé aussi **PDP *(Policy Decision Point)***, représenté par le serveur CIMOM qui maintient la politique de contrôle d'accès appliquée dans le système géré. Concrètement, les règles sont définies par l'administrateur via un client CIM, et peuvent être activées et désactivées par la suite.
- **Le point de récupération des politiques** : appelé **PRP (Policy Recuperation Point)** représente l'entité qui stocke les politiques et permet de les lire. Dans notre cas c'est le serveur CIMOM qui constitue le point d'entrée au référentiel CIMOR contenant les politiques sous la forme d'instances CIM.
- **Le point de renforcement des politiques : appelé PEP (Policy Enforcement Point)** représente l'entité responsable de l'application des règles de contrôle d'accès définies par le PDP, dans notre cas c'est SELinux et le Provider WBEM associé.

Dans cette architecture, le client WBEM (ou le PAP) définit une politique (ensemble de règles) et la stocke sous format CIM dans la base CIMOR maintenue par le serveur WBEM (le PDP). Avant son application par SELinux (le PEP), cette politique est traduite en règles SELinux concrètes. La traduction CIM/SELinux est assurée par le serveur WBEM, plus exactement par le *Provider* SELinux (PEP aussi) que nous avons développé. Ce *provider* assure aussi la traduction inverse SELinux/CIM afin de pouvoir superviser l'état de SELinux (le status et le mode de SELinux, les règles appliquées, l'étiquetage des fichiers, etc.).

Comme tout autre *provider*, le notre possède deux interfaces :

- **Une interface CIM** qui permet d'interagir avec le CIMOM afin de lire les politiques définies est stockées dans la base CIMOR, et de présenter l'état du système sous format CIM.
- **Une interface SELinux** qui permet d'interagir avec SELinux afin de lui transmettre les règles concrètes à appliquer. Elle permet aussi de récupérer l'état de SELinux ainsi que celui des ressources qu'il contrôle.

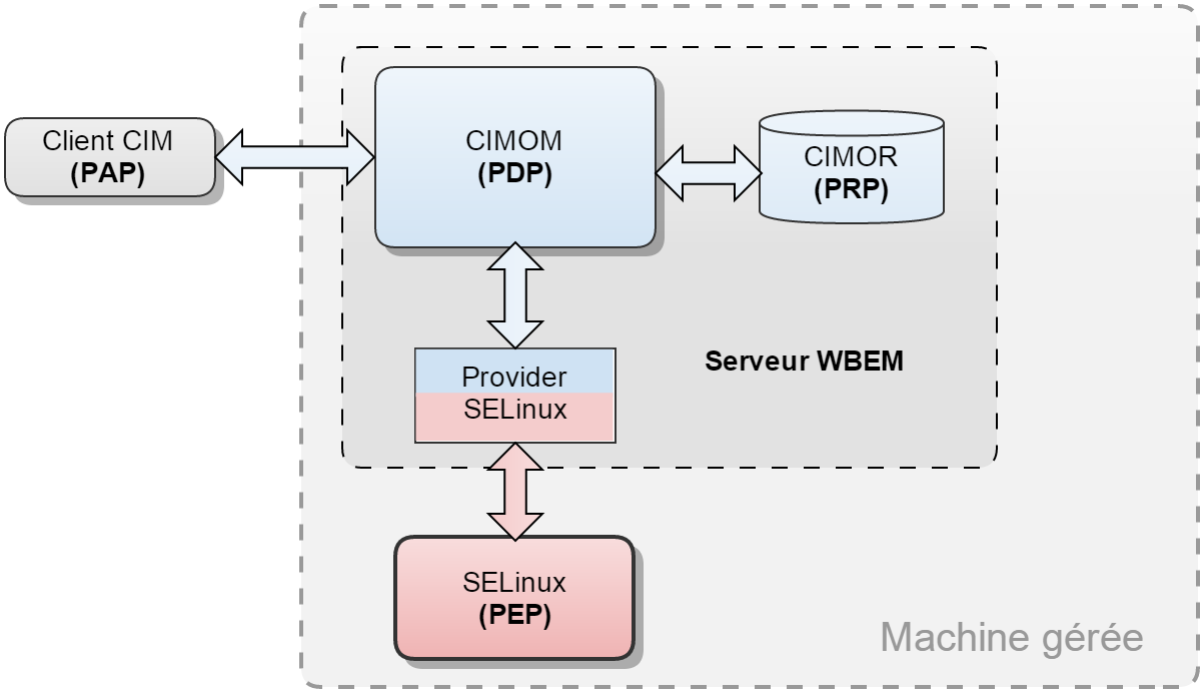

Figure 4-1. Architecture du système de gestion.

Le *provider* SELinux est en réalité un ensemble de *modules* dont chacun maintient une classe de notre modèle CIM. C'est à dire, chaque module assure l'exécution des méthodes d'une classe donnée ainsi que la gestion de ses instances (modifier et retourner les valeurs des attributs).

4.3 Modélisation CIM

Suivant les recommandations du DMTF exigeant que toute modélisation de système géré (machine linux utilisant SELinux) doit être conforme à l'un des profils proposés, nous avons proposé un modèle CIM répondant à nos besoins. Ce modèle est conforme à deux profils du DMTF: *policy profile* et *CIM Integrated Access Control Policy Management model*.

Notre modèle peut être divisé en deux :

- Une partie pour la définition des politiques.
- Une partie pour leur application dans le système.

Dans la terminologie CIM, on appel un schéma, tout modèle (ou un ensemble de modèle) proposé par une organisation pour satisfaire ses besoins en terme de gestion. Tout les noms des classes appartenant à un schéma donné, comportent un préfixe commun appelé schéma spécifique. Par exemple, les noms des classes appartenant aux modèles CIM proposés par le DMTF commencent tous par le schéma spécifique "**CIM_**". Ainsi, pour notre modèle, nous

avons défini "**SEL_"** comme schéma spécifique. Ceci implique que toutes nos classes commencent par le préfixe **:** SEL_.

4.3.1 Définition des politiques

La figure 3.1 représente la partie du modèle qui permet de représenter les politiques de contrôle d'accès gérées par la suite :

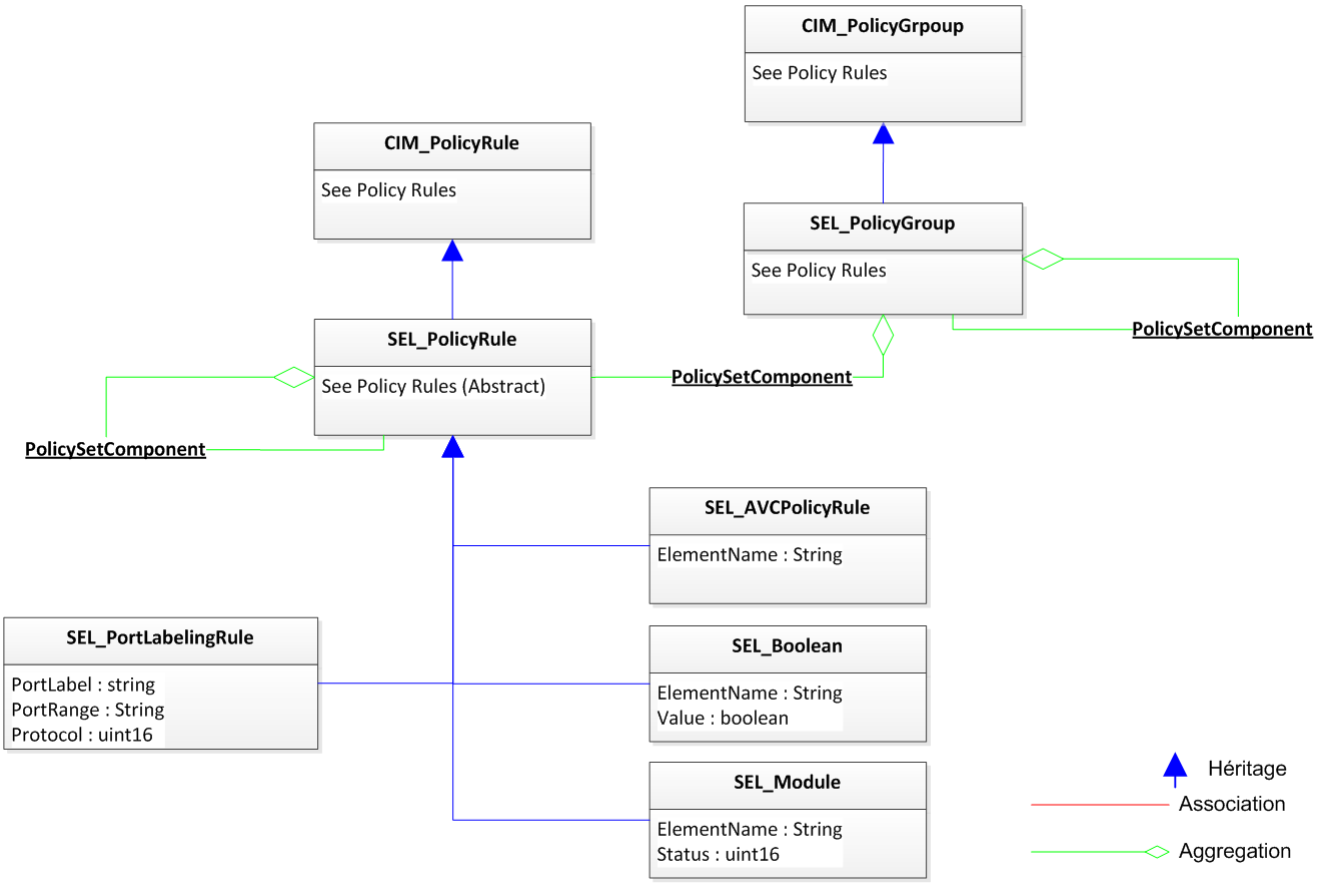

Figure 4-2. Diagramme de classes UML représentant la première partie de notre modèle.

Les classes présentes dans ce modèle permettent de définir les politiques définies dans le PDP/PRP. La classe principale de ce modèle est la classe abstraite **SEL_PolicyRule** qui représente une règle SELinux de manière générale. Elle hérite de la classe **CIM_PolicyRule** proposée par le DMTF, et ne défini pas de nouveaux attributs. Nous avons défini une autre classe appelée **SEL_PolicyGroup** qui permet de regrouper des instances de **SEL_PolicyRule** (ses classes dérivées) en fonction de certains paramètres. Une instance de **SEL_PolicyGroup** ne contient pas de règles mais juste un identifiant du groupe. Les instances de **SEL_PolicyRule** faisant partie du même **SEL_PolicyGroup**, sont seulement associées à ce dernier via la relation d'agrégation (classe-association spéciale) **PolicySetComponent** définie dans le *policy profile***.**

Quatre autres classes héritent de **SEL_PolicyRule** et représentant chacune d'entre elles un type de règles SELinux spécifique. Ces classes sont : **SEL_Module, SEL_Boolean, SEL_PortLabelingRule** et **SEL_AVCPolicyRule.**

4.3.1.1 SEL_Module

Cette classe représente un module SELinux. Un module peut être considéré comme une métarègle qui permet d'activer ou désactiver un ensemble de règles de base faisant partie du module SELinux. Un seul attribut hérité (de CIM_PolicyRule) a été utilisé lors de l'implémentation. Cet attribut, appelé **ElementName,** est une chaîne de caractères contenant le nom du module. Un autre attribut nouveau appelé **Status** est défini, c'est un entier représentant le statut du module.

Les méthodes implémentées pour cette classe sont seulement les deux méthodes intrinsèques **EnumerateInstances** et **Load.**

Load est une méthode qui se lance à chaque démarrage du serveur WBEM pour créer les instances dynamiquement. C'est à dire, à chaque démarrage du serveur, le *provider* crée les instances en fonction de l'état des modules SELinux, et les stocke dans la base CIMOR.

Nous rappelons qu'un module SELinux est un package contenant un ensemble de règles. Pratiquement, les modules sont conçus par les des tierces parties pour permettre le bon fonctionnement de leurs services. Par exemple, dans la configuration par défaut de SELinux, un module dédié au serveur apache est installé. Ce module contient toutes les règles permettant le fonctionnement sécurisé du serveur.

4.3.1.2 SEL_Boolean

Elle représente un booléen SELinux qui est considéré comme une métarègle regroupant un ensemble de règles de base (règles AVC, des déclarations de rôles, etc.) permettant d'autoriser une certaine action. Une règle SEL_Boolean peut être vu comme une boite noire, c'est à dire, un utilisateur peut activer/désactiver l'ensemble des règles qu'elle regroupe sans pouvoir les contrôler individuellement. Tout comme SEL_Module, pour l'implémentation de cette classe nous avons utilisé l'attribut hérité **ElementName** uniquement pour contenir le nom du booléen, ainsi qu'un nouvel attribut appelé **Value** qui est une variable booléenne contenant l'état du booléen SELinux (on/off).

Comme pour SEL_Module, les méthodes implémentées pour cette classe sont **Load** et **EnumerateInstances**.

Les classes SEL_Module et SEL_Boolean correspondent à des règles de type **blackbox**. Puisque dans le profil *Policy Profile*, une règle de type **blackbox** représente une métarègle contenant un ensemble de règles de base (illisibles pour l'administrateur) et qui peut être seulement activée ou désactivée.

4.3.1.3 SEL_portLabelingRule

Elle représente une règle qui autorise un domaine SElinux d'exploiter un port réseau spécifique (UDP ou TCP). Par exemple, si on veut définir une règle permettant à une application, ayant comme étiquette de domaine **home_t**, d'accéder à un ou plusieurs ports, il suffit juste d'ajouter une règle qui autorise home_t à exploiter le port souhaité. C'est pour ça qu'une instance de cette classe défini trois nouveaux attributs qui sont les seuls utilisés lors de l'implémentation. Ces attributs sont :

- **PortLabel** qui est une chaîne de caractères contenant l'étiquette associée au port.

- **PortRange** qui est une chaîne de caractères contenant la valeur numérique du port qui peut être un numéro (par exemple "80"), plusieurs numéros (par exemple "21,23,80") ou une plage (par exemple "1-1023").

- **Protocol** qui est un entier représentant le protocole autorisé pour la communication via ce port. Ses valeurs possibles sont **tcp** et **udp**.

Les seules méthodes qui sont implémentées sont **EnumerateInstances** et **Load.**

4.3.1.4 SEL_AVCPolicyRule

Elle permet de représenter des règles SELinux de base. L'attribut hérité **ElementName**, qui est une chaîne de caractères, est le seul attribut utilisé dans l'implémentation de cette classe. Il contient la définiion textuelle de la règle qui est de la forme suivante : **"action:domaine_t:étiquette_t:classe:permission"**. Nous avons choisit de représenter une règle par une simple chaîne de caractères pour faciliter leurs introduction par l'administrateur. C'est à dire, au lieu de donner 5 attributs différents pour définir une règle, une seule chaîne de caractères suffit. Par exemple, une instance de SEL_AVCPolicyRule dont l'attribut
ElementName= "auditallow:passwd_t:shadow_t:file:*" représente une règle SELinux qui permet d'écrire dans les journaux la trace de tout accès des processus de domaine passwd_t au fichier /etc/shadow.

Contrairement au classes présentées plus haut, les méthodes **EnumerateInstances** et **Load** ne sont pas implémentées pour cette classe à cause du nombre très important de règles AVC présentes dans le système (enivrions 500 mille règles sont chargées lors de l'activation de SELinux).

4.3.2. L'application des règles

La partie du modèle qui permet d'appliquer les politiques définies est représentée par la figure 3.2 suivante :

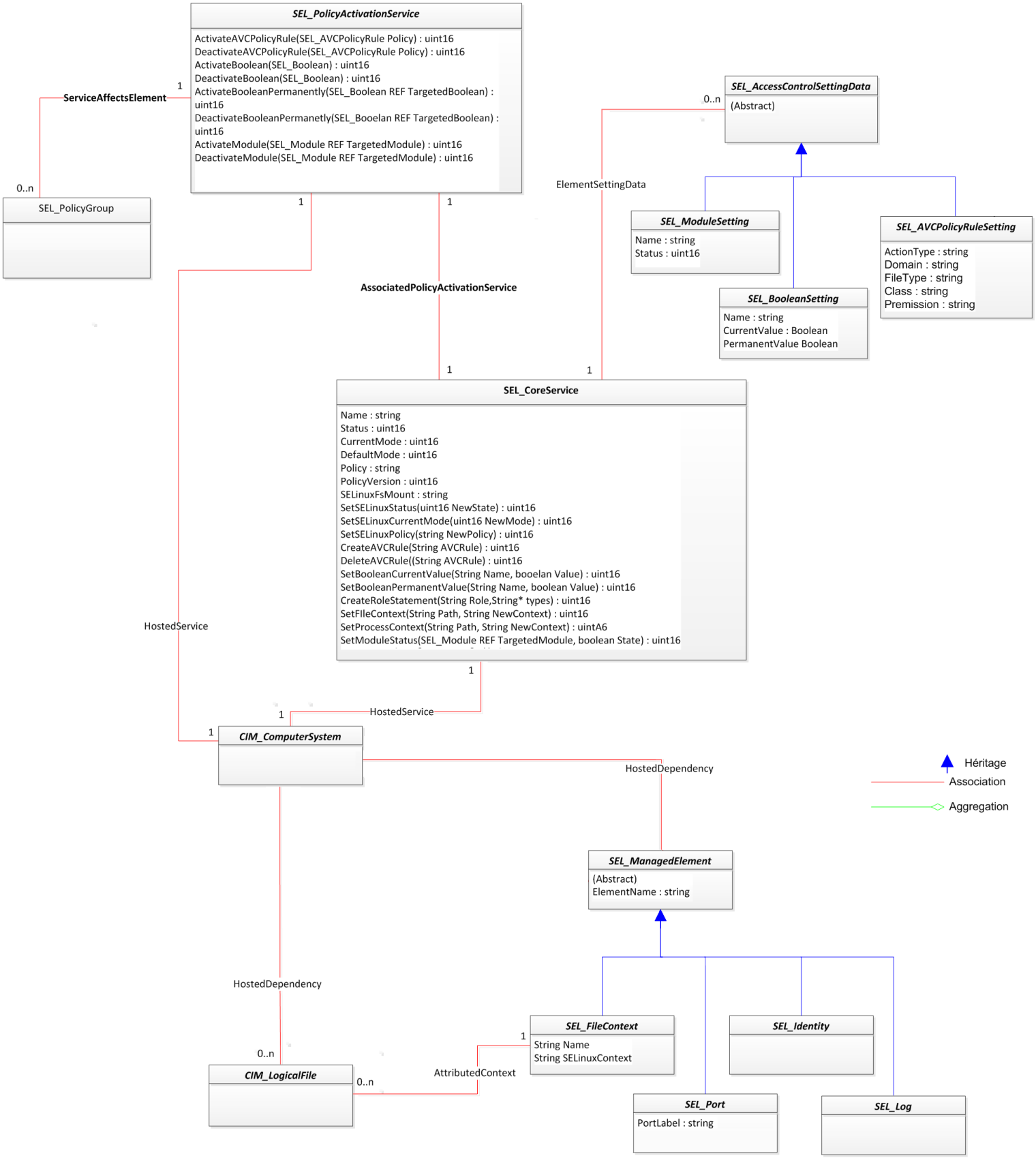

Figure 4-3. Diagramme de classes UML représentant la deuxième partie de notre modèle.

Le point de raccordement entre les deux parties du modèle est la classe SEL_PolicyGroup. Une fois les règles définies (création des instances de SEL_PolicyRule), elles peuvent être activées ou désactivées.

Les entités contrôlées par SELinux sont représentées par la classe abstraite **SEL_ManagedElement**. C'est la classe de base à partir de laquelle les éléments SELinux concrets sont dérivés. Tout ces éléments SELinux utilisent le paramètre **ElementName** comme unique identifiant. Les classes concrètes qui sont **SEL_File**, **SEL_Port**, **SEL_Type**, **SEL_Identity** et **SEL_Role**, sont présentées dans suite de ce document.

SEL_CoreService est la classe principale de cette partie. Elle permet de modifier l'état de SELinux dans le système. Elle permet aussi de présenter des informations de base sur SELinux et est associée au système (CIM_ComputerSystem) sur lequel notre *provider* réside, par l'association **CIM_HostedService**.

Les informations de configuration utilisée par SELinux sont représentées par la classe **SEL_AccessControlSettingData**. Une cohérence entre ces informations et les politiques définies doit être assurée par le système de gestion. C'est à dire, une règle définie par l'administrateur doit être présente sous forme d'instance de SEL_PolicyRule et présente sous forme d'instance de SEL_AccessControlSettingData.

Toutes les méthodes qui permettent de modifier l'état de SELinux (activer/désactiver SElinux, changer la valeur d'un booléen, activer/désactiver un module, etc.) sont définies dans la classe SEL_CoreService détaillée dans la section 3.2.2.

Le lien entre les règles définies par la première partie du modèle (instances de SEL_policyRule) et SELinux, est défini par la classe **SEL_PolicyActivationService**. Cette classe représente un service abstrait qui permet d'activer/désactiver les règles définies par l'administrateur, en exécutant des méthodes qui appellent des méthodes de la classe SEL_CoreService.

4.3.2.1. SEL_PolicyActivationService

On rappelle que cette classe représente une entité abstraite permettant d'activer et de désactiver les instances des classes issues de SEL_PolicyRule. Son implémentation est obligatoire pour être conforme au modèle *CIM Integrated Access Control Policy Management model*.

Aucun attribut (hérité) de cette classe n'est utilisé et aucune méthode intrinsèque n'est implémentée. Cependant elle contient des méthodes permettant d'activer et désactiver les règles. Ces méthodes sont :

- **ActivateAVCPolicyRule** qui permet d'ajouter une règle AVC dans SELinux en faisant appel à la méthode **CreateAVCRule** de la classe SEL_CoreService. Elle prend en argument une instance de SEL_AVCPolicyRule présente dans la base CIMOR.

- **DeactivateAVCPolicyRule** qui permet de supprimer une règle AVC présente dans SELinux en faisant appel à la méthode **DeleteAVCRule** de la classe SEL_CoreService.

Elle prend en argument une instance de SEL_AVCPolicyRule présente dans la base CIMOR. Cette instance sera supprimée si la méthode se termine avec succès.

- **ActivateBoolean** qui permet d'activer de façon temporaire un booléen SELinux. Elle fait appel à la méthode **SetBooleanCurrentValue** qui permet d'appliquer ce changement dans SELinux en mettant la valeur courante du booléen à "**on"**. Elle prend en argument une instance de SEL_Boolean dont l'attribut **Value** sera mis à **"on"** en cas de succès.

- **DeactivateBoolean** qui est similaire à ActivateBoolean mais qui permet de désactiver un booléen.

- **ActivateBooleanPermanetly** et **DeactivateBooleanPermanetly** qui sont similaires aux méthodes **ActivateBoolean** et **DeactivateBoolean** mais elles permettent un changement permanant de l'état des booléens (même après le redémarrage du système). Elles font appel à la méthode **SetBooleanPermanentValue** de la classe SEL_CoreService.

- **AactivateModule** qui permet d'activer un module SELinux en faisant appel à la méthode **SetModuleStatus** de la classe SEL_CoreService. Elle prend en argument une instance de SEL_Module.

- **DeactivateModule** qui est similaire à ActivateModule mais qui permet de désactiver le module donné en argument.

4.3.2.2. SEL_CoreService

Cette classe est la classe principale du modèle car elle permet de gérer l'état de SELinux. Une seule instance de cette classe peut être créée pour être associée au seul et unique service SELinux présent dans le système. Elle comprend les attributs suivants :

- **Name** qui est de type chaîne de caractères. C'est le seul attribut clé de cette classe et il représente le nom du service.

- **CurrentStatus** qui est de type **entier**. Il représente le statut de SELinux qui peut être **Disable** ou **Enable**.

- **CurrentMode** qui est de type **entier**. Il représente le mode courant de SELinux qui peut être **Enforcing** ou **Permessive.**

- **DefaultMode** qui est de type **entier**. Il représente le mode par défaut écrit dans le fichier de configuration de SELinux. Ses valeurs possibles sont : **Enforcing** et **Permissive.**

- **PolicyType** qui est de type **chaîne de caractères**. Il représente le nom de la politique appliquée par SELinux (le fichier *Policy*).

- **PolicyVersion** qui est de type **chaîne de caractères**. Il représente la version du fichier de la politique appliquée par SELinux.

- **SELinuxFsMount** qui est de type **chaîne de caractères**. Il représente le fichier racine de SELinux.

Cette classe comprend toutes les méthodes extrinsèques permettant d'interagir avec le système. La seule méthode intrinsèque implémentée est **EnumerateInstances** qui permet de retournée l'état de SELinux (les valeurs de tous les attributs).

Les méthodes extrinsèques sont :

- **SetSELinuxDefaultMode** qui permet de changer le mode par **défaut** de SELinux écrit dans le fichier de configuration. Cette méthode prend en argument le mode par défaut souhaité qui peut être : **Permissive**, **Enforcing**, ou **Disable**.

- **SetSELinuxCurrentMode** qui permet de changer le mode **courant** de SELinux. Elle prend en argument le mode courant souhaité qui peut être : **Permissive** ou **Enforcing.**

- **SetSELinuxPolicy** qui permet de changer la politique SELinux utilisée. Le nom du fichier de la politique (situé dans le répertoire /etc/selinux) est donnée en argument à cette méthode.
- **CreateAVCRule** qui permet de créer et d'ajouter une règle AVC à SELinux. La règle sous forme textuelle est donnée en argument.

- **DeleteAVCRule** qui est similaire à CreateAVCRule mais qui permet de supprimer une règle AVC.

- **SetBooleanCurrentValue** qui permet de changer la valeur courante d'un booléen SELinux. Cette classe prend en argument le nom du booléen et sa nouvelle valeur.

- **SetBooleanPermanentValue** qui est similaire à SEtBooleanCurrentValue mais qui permet de changer la valeur permanente du booléen.

- **SetModuleStatus** qui permet d'activer ou désactiver un module SELinux. Le nom du module ainsi que son statut (activé/désactivé) sont donnés en argument à cette méthode.

- **DeleteModule** qui permet de supprimer carrément un module SElinux. Le seul argument qu'elle prend est le nom du module concerné.
- **InstallModule** qui permet d'installer un nouveau module SELinux en donnant l'emplacement de ce dernier (le fichier .pp).

- **SetFileContext** qui permet de changer le contexte SELinux d'un fichier. Elle prend en argument l'emplacement du fichier concerné ainsi que le nouveau contexte.

- **SetProcessContext** qui permet de changer le contexte d'un exécutable et donc, des processus lancés à partir de ce dernier.

4.3.2.3. SEL_AccessControlSettingData

SEL_AccessControlSettingData est une classe abstraite qui hérite de la classe **CIM_AccessControlSettingData.** Elle permet de représenter les informations de configuration de SELinux qui doivent être cohérentes avec les règles définies. Trois classes sont dérivées de cette classe abstraite :

- **SEL_ModuleSetting** qui représente les informations de configuration des modules. Une instance de cette classe représente la configuration d'un module donné (activé ou désactivé). Deux attributs sont définis dans cette classe : **Name** et **Status**, qui représentent le nom et le statut du module.
- **SEL_BooleanSetting** qui représente les informations de configuration des booléens. Une instance de cette classe représente la configuration d'un booléen donné (activé/désactivé). Elle défini trois attributs qui sont : **Name, CurrentValue** et **PermanentValue**, qui représentent respectivement le nom, la valeur actuel et la valeur par défaut du booléen.
  /* ici, nous avons éclaté l'attribut RuleName (string) en cinq attributs */
- **SEL_AVCPolicyRuleSetting** qui représente une règle AVC définie. Chaque instance de cette classe représente une règle AVC déjà représentée par une instance de SEL_AVCPolicyRule. Elle définit cinq attributs qui sont tous de type **chaîne de caractères** :

ActionType qui représente le type de l'action appliquée par la règle qui peut être **allow, neverallow, auditallow, dontaudit**.

Domain qui représente le domaine (le sujet) auquel l'action est appliquée.

FileType qui représente l'étiquette (l'objet) sur laquelle l'action est appliquée.

Class qui représente la classe (file, dir, socket, etc.) de la ressource sur laquelle l'action est appliquée.

Permission qui représente la permission ou le type d'accès défini dans la règle.

4.3.2.4. SEL_ManagedElement

Le contrôle d'accès opéré par SELinux doit s'appliquer sur les ressources concrètes du système d'exploitation que sont : les fichiers au sens du système Linux (fichiers ordinaires, fichiers exécutables, répertoires, périphériques, etc.) ports réseaux (UDP ou TCP) et processus (domaines). /* mettre à jour ces classes */

Ces contrôles sont appliqués sur la base d'identités des utilisateurs, que des rôles qui leurs sont associés ainsi qu'aux types affectés aux fichiers (étiquettes) et aux processus (domaine)

- **SEL_FileContext** qui représente un contexte de fichier défini par la classe **CIM_LogicalFile**. Elle définit un seul attribut de type **chaîne de caractères** appelés **Path** qui contient le chemin du fichier. Cette classe n'est pas instanciée à cause du nombre de fichiers très important sur dans le système.
- **SEL_Port** qui représente un port géré par SELinux qui est défini par son numéro, son étiquette et son protocole.
- **SEL_Identity** qui représente une identité SELinux. Cette classe doit être associée d'une manière ou d'une autre à la classe CIM_Identity, le mieux par héritage. Elle permet d'afficher les rôles SELinux associés à l'identité qu'elle représente.
- **SEL_Log** qui ne peut être instanciée puisqu'elle ne permet de gérer aucun objet. Seule la méthode **EnumerateInstances** permettant de retourner les informations de journalisation de SELinux est implémentée.

4.4 Conclusion

Tout au long de ce chapitre, nous avons présenter et expliquer l'approche de conception de notre système standardisé (car il est conforme au *Policy Profile* proposé par le DMTF) de gestion de politiques de sécurité. Nous avons présenté l'architecture de gestion que nous avons implémenté et qui s'inspire du protocole de gestion de politiques **COPS**. Concernant, la modélisation des politiques ainsi que SELinux, nous avons proposé un modèle CIM permettant de définir des politiques SELinux pour les appliquer par la suite. Les classes principales de ce modèle sont : SEL_PolicyRule qui représente les modules et les booléens SELinux (règles de type **blackbox**), les règles AVC SELinux ainsi que les règles s'appliquant sur les ports réseaux, et SEL_CoreService qui représente SELinux.

La dernière étape de ce projet consiste à implémenter ce modèle sur un système Linux CentOS. Cette implémentation se traduit par l'installation d'une plateforme de gestion WBEM (client/serveur) ainsi que des outils nécessaires pour le développement des *providers* chargés de gérer les classes du modèle.

CHAPITRE 5

RÉALISATION DE NOTRE SYSÈME DE GESTION DE POLITIQUES DE SÉCURITÉ

5.1 Introduction

Ce chapitre présente les différentes étapes d'implémentation de notre système. Cette implémentation permet tout d'abord, de prouver la possibilité de gérer des politiques de sécurité abstraites en format CIM en les traduisant en politiques concrètes SELinux, mais aussi d'offrir une plateforme pouvant être exploitée par un administrateur afin de composer, diffuser et appliquer, via un client WBEM, des politiques de sécurité sur des systèmes en réseau.

Cette présentation retrace les phases de développement suivi que sont :

1. Les choix techniques adoptés permettant la programmation et la mis en place de l'ensemble du système.
2. L'implémentation de notre *Provider CIM/WBEM* appelé SELPovider, dédié à la gestion du module SELINUX et ses politiques.
3. Les détails de la de programmation d'un module particulier de notre Provider, relatif à la classe SEL_CoreService (voir la section 4.3.2.2 du chapitre conception) nommé SEL_CoreServiceProvider. Nous n'avons pas jugé utile de présenter la programmation des autres modules car nous avons suivi la même démarche.
4. Notre plan de tests.
5. Présentation du client de type GUI, que nous avons développé pour faciliter l'exploitation du système aux utilisateurs mais également pour démontrer toutes les possibilités offertes par notre implémentation.

5.2 Les choix techniques

Afin d'implémenter notre système, nous avons choisi comme plateforme un environnement **Linux CentOS 6.5**, puisque sur dans cette version, SELinux est installé par défaut. Toutefois notre implémentation est exploitable sur n'importe quel système de la famille Linux pourvu qu'il supporte le module SELinux. Nous avons opté pour le projet **OpenPegasus 12.2.0 [27]**, fourni par l'OpenGroup[28], qui implémente l'ensemble des composants de l'architecture WBEM ainsi que les différents points de gestion de politiques de notre architecture (PAP, PDP, PRP) à l'exception des providers CIM qui représentent les PEP. Nous avons choisi ce projet pour sa stabilité, sa maturité et pour l'exhaustivité de ses fonctionnalités **[23]**.

Pour le développement de notre *provider CIM*, nous avons choisi le *Framework de* développement **CIMPLE 2.0.24 [32]** dédié à la programmation de providers et de clients CIM. Cet outil facilite énormément l'implémentation par la génération de squelettes de programmes en C++ à partir de la définition en MOF des classes CIM relatives aux ressources cibles. La puissance de ce Framework réside en son approche qui consiste à générer des classes C++ qui correspondent (et qui sont identiques) aux classes MOF avec une puissante gestion des types de données et la génération des squelettes des méthodes intrinsèques et extrinsèques. Le travail du développeur consiste à coder les méthodes en fonction de la (des) ressource(s) visée(s) par le Provider ou le client **[7]**.

Pour l'instrumentation du module SELinux et ses politiques à travers les méthodes de notre Provider, nous avons utilisé les fonctions offertes par les bibliothèques de programmation natives à SELINUX que sont **libselinux** et **libsemanage [33].** Ces APIs étant écrites en langage

C, leur intégration dans les squelettes de programmes C++ générés par le Framework CIMPLE est tout simplement naturelle.

Pour tester notre Provider (gestion des politiques SELinux par l'invocation des méthodes des classes CIM correspondante) nous avons utilisé le client WBEM **wbemcli 1.6.1** comme point d'administration des politiques PAP. Cependant, pour faciliter l'exploitation de notre système de gestion de politiques (CIM/SELinux) et en tirer profit au maximum, nous avons développé une IHM intuitive, en utilisant le langage **Python**. Deux bibliothèques **Python** ont été utilisées : **PyWBEM [28]** pour la communication avec le serveur WBEM, et **PyQt[31]** pour la création de l'interface graphique[18].

5.3 Implémentation des providers

Il faut rappeler que notre *Provider* représente l'entité qui interagit directement avec SELinux. Ce *Provider* est constitué de plusieurs modules dont chacun permet de gérer une classe de notre modèle présenté dans la section 4.3 du chapitre conception. Notons ici que nous n'avons pas besoin de développer les Providers CIM pour lesquelles des implémentations existent, notamment à travers les projets SBLIM (maintenu par IBM) et OpenLMI (maintenu par le projet FEDORA)**[7]**. Comme exemple, la classe **CIM_LogicalFile** qui représente tous les fichiers ordinaires d'un système (ciblés par les politiques SELinux), est déjà prise en charge par des Providers tiers (les deux projets mentionnés plus haut) dans notre implémentation nous avons juste rajouté le support de la classe SEL_Contexte et l'association SEL_AttributedContext qui la relie à la classe CIM_LogicalFile (voir la section 4.3.2 du chapitre conception). Ceci démontre l'interopérabilité et l'intégration de Providers issus de différentes implémentations dans le même système WBEM, car en effet, le client CIM ne manipule que des instances de classes « abstraites » qui sont appariées par le serveur CIMOM avec l'implémentation correspondante.

Dans la suite de ce rapport, un module qui permet de maintenir une classe, dont le nom est **"SEL_ClassA"**, est nommé **"SEL_ClassA_Provider"**.

5.3.1 Le module SEL_Module_Provider (classe SEL_Module)

Ce composant du provider permet de récupérer l'état des politiques SELinux de type module présents dans le système et modélisées par la classe SEL_Module (voir la section 4.3.2.2 du chapitre conception). Il implémente les méthodes intrinsèques suivantes:

- **Load** qui s'exécute automatiquement lors du chargement du Provider (en général lors du (re)démarrage du serveur WBEM). Cette méthode permet de récupérer dynamiquement l'état des modules SELinux en utilisant les fonctions des bibliothèques de ce dernier, pour ensuite créer et enregistrer les instances correspondantes dans le CIMOR. Lors de la création d'une instance, les attributs **PolicyRuleName, Version** et **Status** sont fixés respectivement avec le **nom, la version** et le **statut** du module correspondant. La méthode Load permet en fait d'assurer la synchronisation et la cohérence de l'état actuel des modules SELinux avec les politiques CIM correspondantes abstraites présentes dans le CIMOR.

- **EnumerateInstances** Permet de retourner toutes les instances présentes dans le CIMOR (créées par la méthode Load). Elle est invoquée par un client CIM pour connaitre l'état des modules. Cette méthode ne fait appel à aucune fonction de la bibliothèque SELinux, car elle interagit seulement avec le CIMOM.
- **GetInstance** Permet de retourner une instance dans le CIMOR. Elle reçoit en paramètre les valeurs des attributs clés.
- **CreateInstance** Permet de créer une instance de SEL_Module dans le CIMOR en recevant en argument les valeurs de ses attributs. Cette méthode doit être invoquée seulement lors de l'installation du module par le SEL_CoreService_Provider si non, une invocation manuelle par l'administrateur risque de créer des problèmes de cohérence (instance crée sans module installé).
- **DeleteInstance** qui est similaire à CreateInstance mais qui permet de supprimer une instance du CIMOR lorsqu'un module est désinstallé.
- **ModifyInsctance** qui permet de modifier une instance en donnant en argument les nouvelles valeurs de PolicyRuleName et Status. Elle est invoquée lors de l'activation/désactivation d'un module.

5.3.2 Le module SEL_Boolean_Provider (classe SEL_Boolean)

Similaire à SEL_Module_Provider, il permet de récupérer l'état des booléens SELinux. Il implémente les méthodes suivantes :

- **Load** qui est similaire à celle du module précédent. Elle récupère l'état des booléens et les enregistre dans le CIMOR sous formes d'instances de SEL_Boolean.
- **EnumerateInstances** retourne les instances de type SEL_Boolean présentes dans le CIMOR. Invoquée par un client WBEM, elle récupère l'état des booléens (préalablement chargé par la méthode Load).
- **GetInstance** permet de récupérer l'état d'un booléen dont la clé est donnée en argument.

5.3.3 Le module SEL_PortLabelingRule_Provider (classe SEL_PortlabelingRule)

Il permet de récupérer l'état des ports réseaux maintenus par SELinux, présenté sous forme de règles de type PortLabelingRule. L'implémentation de ce module est similaire à celle de SEL_Module_Provider. La méthode Load permet de récupérer dynamiquement l'état des ports. **EnumerateInstances** et **GetInstance** retournent l'état des ports.

5.3.4 Le module SEL_AVCPolicyRule_Provider (classe SEL_AVCPolicyRule)

Ce module permet de récupérer les instances de SEL_AVCPolicyRule ajoutées manuellement par l'administrateur. La récupération des règles AVC présentes dans le système n'est pas implémentée, à cause de leur nombre trop important dans le système (500 mille règles AVC présentes lors de l'installation de SELinux). Un administrateur ne peut donc récupérer, supprimer ou modifier que des instances individuelles identifiées par leurs clés. Les méthodes implémentées dans ce module sont **CreateInstance** qui est invoquée lorsque l'administrateur veut crée une nouvelle règle AVC, **GetInstance**, **DeleteInstance** **EnumerateInstance** et **ModifyInsctance**, qui sont identiques à celle de

SEL_Module_Provider puisqu'elles ne manipulent que les instances présentes dans le CIMOR.

5.3.5 Le module SEL_PolicyActivationService_Provider

Ce module contient les méthodes qui permettent d'activer ou désactiver tous les types de politiques présents dans le système. Il ne fait qu'invoquer des méthodes de la classe SEL_CoreService afin d'appliquer les changements imposés par la politique donnée en argument.

Les règles activées/désactivées par ce module sont les booléens et les modules SELinux existants dans le système, ainsi que les règles de type AVC ajoutées par l'administrateur à travers notre systèmes de gestion.

5.3.6 Le module SEL_CoreService_Provider

Ce module regroupe toutes les méthodes qui permettent de modifier l'état de SELinux. Rappelons que l'état de SELinux est défini par les paramètres suivants : son mode, la politique qu'il applique, l'état actuel des modules, etc. Les méthodes implémentées par ce module sont :

- **SetSELinuxStatus** qui permet de changer le mode par défaut de SELinux ou de le désactiver. Elle reçoit en argument un entier dont les valeurs possibles sont : -1 pour *Disable*, 0 pour *Permissive* et 1 pour *Enforcing*. Cette méthode modifie le fichier /etc/selinux/config, les changements ne sont pris en compte qu'après le redémarrage du système.
- **SetSELinuxCurrentMode** qui permet de changer le mode courant de SELinux (*Enforcing* ou *Permissive*). Elle reçoit en argument un entier qui peut être : 0 pour *Permissive* ou 1 pour *Enforcing*.
- **SetSELinuxPolicy** qui permet de changer la politique utilisée par SELinux. Elle reçoit en argument une chaîne de caractères contenant le nom de la politique souhaitée.
- **CreateAVCRule** qui permet d'ajouter une règle AVC à la politique SELinux appliquée. Sous SELinux, une règle doit être contenue dans un module, c'est pour ça que cette méthode place la règle créée dans un module dédié. L'ajout d'une règle AVC se fait en trois étapes comme le montre la figure 5.1 suivante :

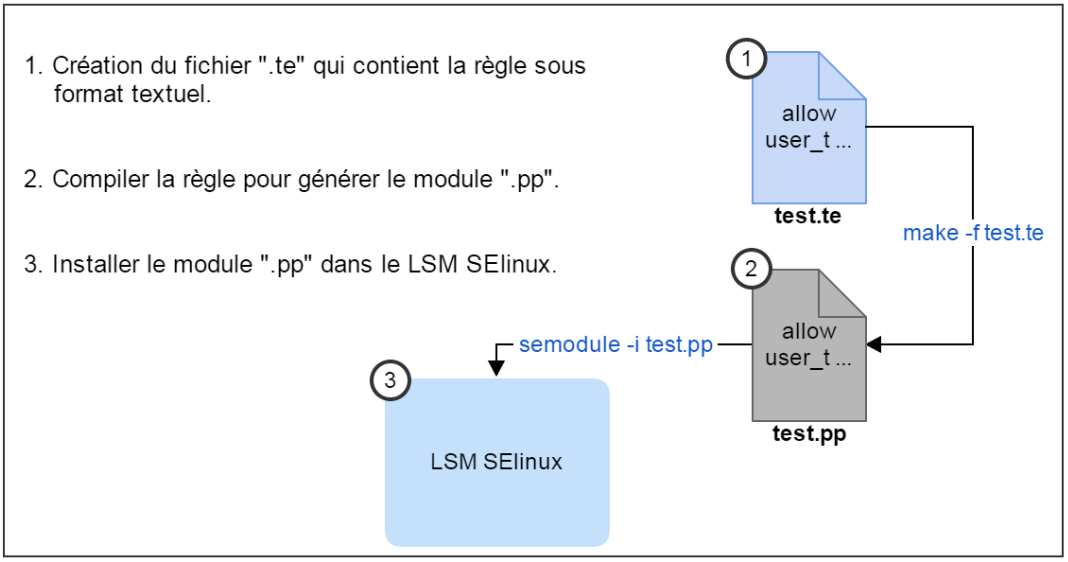

Figure 5-1. Étapes de l'ajout d'une règle AVC.

Si les tapes s'exécutent correctement, la méthode **CreateInstance** de la classe **SEL_AVCPolicyRuleSetting** est invoquée pour créer une instance représentant une information de configuration correspondante à la règle ajoutée et permettant à l'administrateur de vérifier le succès de son opération. Cette méthode ne doit pas être invoquée directement par le client, mais via le module **SEL_ PolicyActivationService_Provider.**

DeleteAVCRule qui permet de supprimer une règle AVC dont le nom est donnée en argument. Si la suppression réussit, elle supprime l'instance de **SEL_AVCPolicyRuleSetting** correspondante. Comme la méthode CreateAVCRule, elle ne doit pas être invoquée directement par un client.

SetBooleanCurrentValue et **SetBooleanPermanentValue** qui permettent de changer l'état d'un *boolean* SELinux, dont le nom est donné en argument. La première le fait de façon temporaire (jusqu'au prochain redémarrage) et la seconde de façon permanente. Le changement apporté par l'une de ces méthodes est appliqué sur l'instance de SEL_BooleanSetting Correspondante au booléen concerné.

SetFIleLabel et **SetProcessDomain** qui permettent de changer respectivement l'étiquette d'un fichier et le domaine d'un processus (lancés partir du nom de d'un fichier exécutable) . Chaque méthode reçoit en argument le chemin du fichier/exécutable et l'étiquette/domaine concerné.

SetModuleStatus qui reçoit le nom d'un module ainsi que son statut souhaité (activé/désactivé). Elle est invoquée par les méthodes **ActivateModule** et **DeactivateModule** de la classe **SEL_ PolicyActivationService** pour changer le statut du module concerné. Si ce changement réussit, l'attribut **Status** de l'instance de SEL_ModuleSetting est mis à jour.

InstallModule et **DeleteModule** qui permettent respectivement d'installer et désinstaller un module SELinux. La première reçoit en argument le chemin vers le fichier (.pp) représentant le module à installer, et la seconde reçoit le nom du module à supprimer.

5.3.7 SEL_AccessControlSettingData_Provider

On parle ici de trois modules qui maintiennent les classes : **SEL_ModuleSetting**, **SEL_BooleanSetting** et **SEL_AVCPolicyRuleSetting**. On rappelle que ces classes représentent la configuration actuelle des règles SELinux actuellement. Les méthodes implémentées pour chacune de ces classes sont :

- **Load** qui permet de récupérer l'état actuelle des règles concernées (règles AVC, booléens ou modules).
- **EnumerateInstances** et **GetInstance** qui retournent les instances à partir du CIMOR.
- **CreateInstance**, **DeleteInstance** et **ModifyInstance** qui permettent de gérer les instances de la classe concernée. Ces méthodes sont invoquées par les méthodes de **SEL_CoreService** pour gérer les instances en fonction du changement de l'état du système, afin d'en garantir la cohérence.

5.3.10 SEL_FileContext _Provider

C'est un module très léger qui implémente juste une méthode qui reçoit en argument le chemin de l'emplacement d'un fichier, et retourne la valeur de l'attribut SELinux Context qui représente son contexte SELinux (identité, rôle et étiquette) .

5.3.12 SEL_Identity _Provider

C'est un module léger aussi qui permet de lister toutes les identités SELinux présentes dans la politique SELinux actuelle, ainsi que leurs rôles associés.

5.3.13 SEL_Log_Provider

C'est un module léger qui permet de récupérer les informations sur les accès bloqués par SELinux à partir du fichier /var/log/audit/audit.log.

5.4 Phase de développement : détails de la programmation du module SEL_CoreService_Provider

On rappelle que la classe **SEL_CoreService** est l'une des deux classes principales de notre modèle. C'est pour cette raison que nous avons choisi de présenter la démarche de l'implémentation de ce module qui est la même démarche suivi pour les autres.

Nous avons exploité quelques attributs de la classe SEL_CoreService que sont : **Name**, **Status**, **CurrentMode**, **DefaultMode**, **Policy**, **PolicyVersion** et **SElinuxFSMount**, représentent l'état global de SELinux.

Le module permettant de maintenir cette classe et d'exécuter ses méthodes a été développé suivant les étapes suivantes :

a. Définition de la classe en format MOF :

Nous avons écrit la classe **SEL_CoreService** en langage MOF dans un fichier (.mof) de la manière suivante :

```
[Description ("This class represents the SELinux Service")]
class SEL_CoreService : CIM_AccessControlService
{
//attributs
[Key, Override ( "Name" ), Description ("The Service Name")]
 string Name;

[Description ("The SELinux current status"), ValueMap { "0", "1","-1"},
  Values { "Disable", "Enable","unknown"}]
uint16 CurrentStatus ;

[Description ("The SELinux current mode"), ValueMap { "0", "1","-1"},
Values {"Permissive","Enforcing","unknown"}]
uint16 CurrentMode ;
. . .

// methods
 ([Description ("SELinux new default mode") ,ValueMap { "0", "1", "-1"},
Values {"Permissive","Enforcing", "Disable"}]
uint16 NewMode                    ); // method end

}; // class end
```

b. Génération du projet C++ :

En utilisant le Framework **CIMPLE**, nous avons généré la classe **SEL_CoreService** en langage C++ en utilisant la commande suivante :

```
# genproj SELProvider SEL_CoreService
```

Cette commande génère tous les fichiers permettant la programmation du *Provider* (fichiers sources et en-têtes).

c. L'ajout des codes des méthodes :

Dans cette étape, nous avons intégré notre code permettant de faire le travail souhaité au niveau de chaque méthode. Le code C++ suivant représente la petite méthode **SetBooleanCurrentValue** écrite dans le fichier **SEL_CoreService_Provider.cpp** :

```
Invoke_Method_Status SEL_CoreService_Provider::SetBooleanCurrentValue(
   const SEL_CoreService* self,    const Property<String>& BooleanName,
   const Property<boolean>& NewValue,    Property<uint16>& return_value)
{
int i=security_set_boolean(BooleanName.value.c_str(),NewValue.value);
 if (i==0){if (security_commit_booleans()==0){
            Client c; c.connect();
            SEL_BooleanSetting_Hnd hnd;
            hnd.InstanceID_value((cimple::String)BooleanName.value.c_str());
            hnd.BooleanName_value((cimple::String)BooleanName.value.c_str());
            hnd.BooleanCurrentValue_value(NewValue.value);
            const String& __name_space= "root/cimv2";
            c.modify_instance( __name_space,    hnd);
            hnd.print();
            return_value.set(0);return INVOKE_METHOD_OK;
}}
}
```

Il est à noter que l'exécution de certaines méthodes du modèle (récupération de l'état des module par exemple) est relativement lente, chose qui est très logique puisque les fonctions proposées par les librairies de SELinux sont couteuses en temps.

d. Compilation du *Provider :*

La génération du provider ne se fait qu'après la programmation de tous les modules de ce dernier. Le *Provider* est généré sous forme de librairie (.o).

La figure 4 résume les étapes de la programmation de notre P*rovider :*

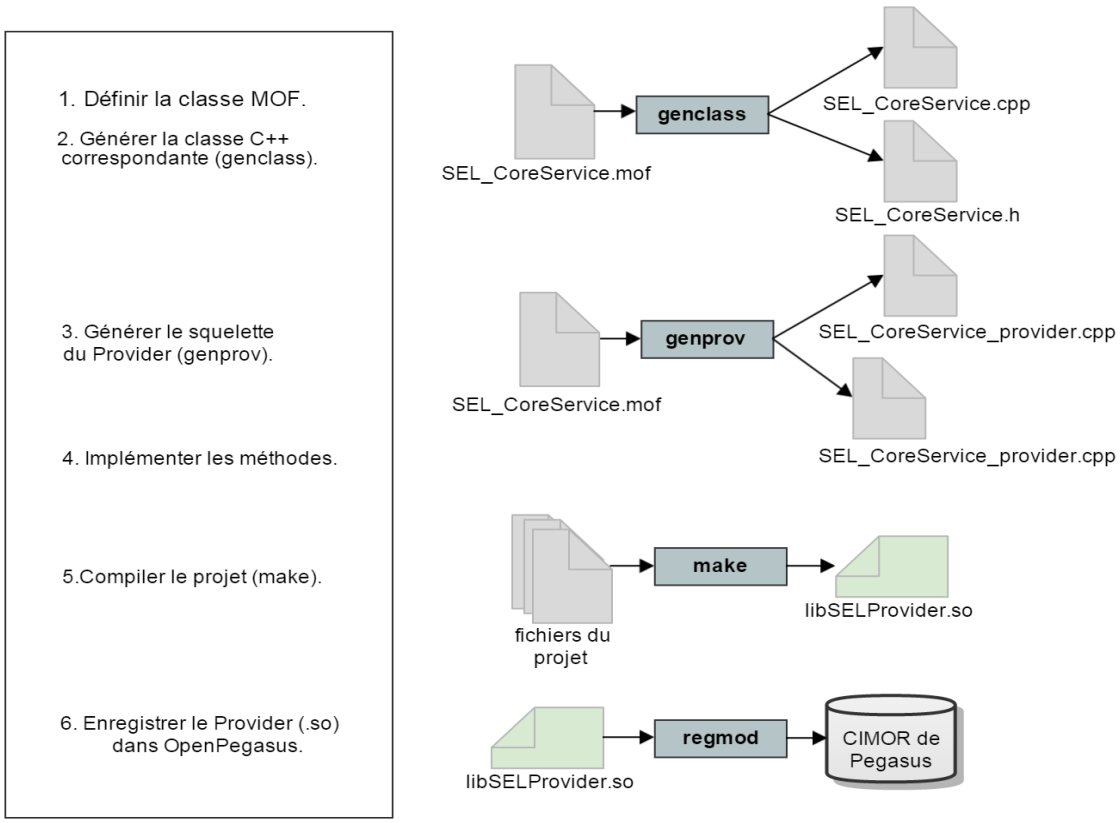

Figure 5-2. Étapes de programmation de notre *Provider*.

La compilation du *Provider* a été faite après la génération et l'adaptation de toutes les classes C++ de notre modèle, pour générer une bibliothèque nommée **libSELProvider.so**. Cette bibliothèque représente le cœur de notre travail car, après son enregistrement au niveau d'un serveur WBEM, n'importe quel client pourra invoquer les méthodes qu'elle contienne pour gérer les politiques SELinux mais en format CIM.

5.5 Plan de test

Nous présentons dans cette section notre plan de tests qui se focalise les fonctionnalités les plus importantes de notre système de gestion qui sont : la **modification** et **vérification de la configuration** de base de SELinux, l'**activation/désactivation** de politiques SELinux de type module, est l'**ajout** d'une politique SELinux de type **AVC**. Tous ces tests sont réalisés à l'aide d'un client **wbemcli** qui permet de communiquer localement avec le CIMOM qui gère SELinux à l'aide de notre *Provider*.

Afin de démontrer que notre Provider réalise effectivement ce qu'on attend de lui, nous avons préparé, pour chaque type de politique CIM, des scénarios pour vérifier que le contrôle d'accès attendu de cette politique est bel et bien traduit en terme de règle SELinux et inversement lorsque la politique CIM est désactivée.

Les tests présentés ici ont été réalisés à l'aide de l'interface ligne de commande du système Linux, nous avons donc veillé à décortiquer les sorties afin d'interpréter les résultats.

5.5.1 Désactivation d'un module

Afin de démontrer que le module SEL_Module_Provider réalise ce qu'on attend de lui, nous avons élaboré un petit programme appelé **test** qui ne fait qu'écrire un texte sur la sortie standard (l'écran). Nous avons également élaboré un module SELinux appelé **permit_exec** qui devrait autoriser notre programme à s'exécuter.

Le scénario consiste à :

- Compiler le module **permit_exec** dans le système SELinux, pour qu'il puisse être pris en charge par notre Provider
- Vérifier, via le client CIM que la politique CIM correspondante au module existe
- Activer la politique CIM correspondante au module
- Lancer le programme test et vérifier qu'il peut écrire sur l'écran
- Désactiver la politique CIM correspondante au module et vérifier que le programme ne peut plus écrire sur l'écran.
- Vérifier ce dernier rejet à travers l'énumération des instances CIM correspondantes aux Logs de SELinux

La figure 5-3 présente le scénario de ce test.

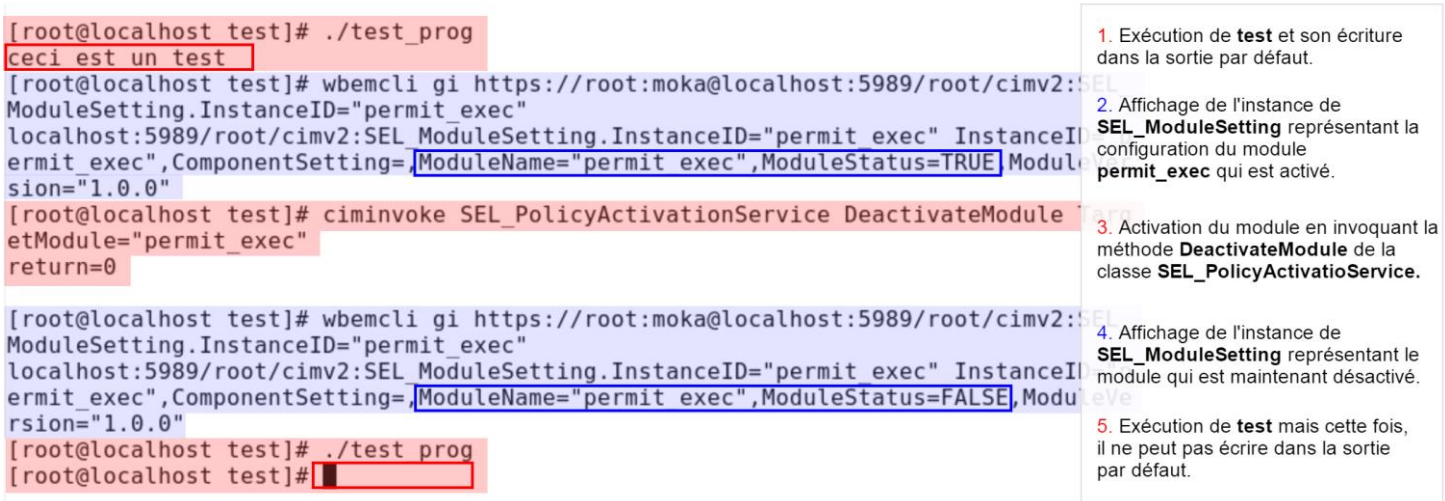

Figure 5-3. Scénario de test illustrant un exemple de désactivation d'une règle SEL_Module.

On rappel que la règle CIM abstraite correspondante au module **permit_exec** est représentée par une instance de **SEL_Module (2)** créée automatiquement par la méthode **Load** de cette classe. L'état de ce module (désactivé) est représenté par l'attribut **ModuleStatus** de l'instance de **SEL_ModuleSetting** correspondante, elle aussi, créée automatiquement par la méthode **Load** de **SEL_ModuleSetting (2)**. Après la désactivation du

module par la méthode **DeactivateModule** de **SEL_PolicyActivatioService (3)** (vérification de l'activation dans **(4))**, l'exécutable **test ne** peut plus écrire dans la sortie par défaut **(5)**.

La méthode **EnumerateInstances** de **SEL_Log** peut être utilisée pour vérifier que c'est bien la désactivation de notre règle CIM qui bloque l'accès en écriture au programme **test**, en utilisant SELinux. La figure 5.4 montre la trace de cet évènement dans le fichier **/var/log/audit/audit.log** récupérée par la méthode **EnumerateInstances** de **SEL_Log**.

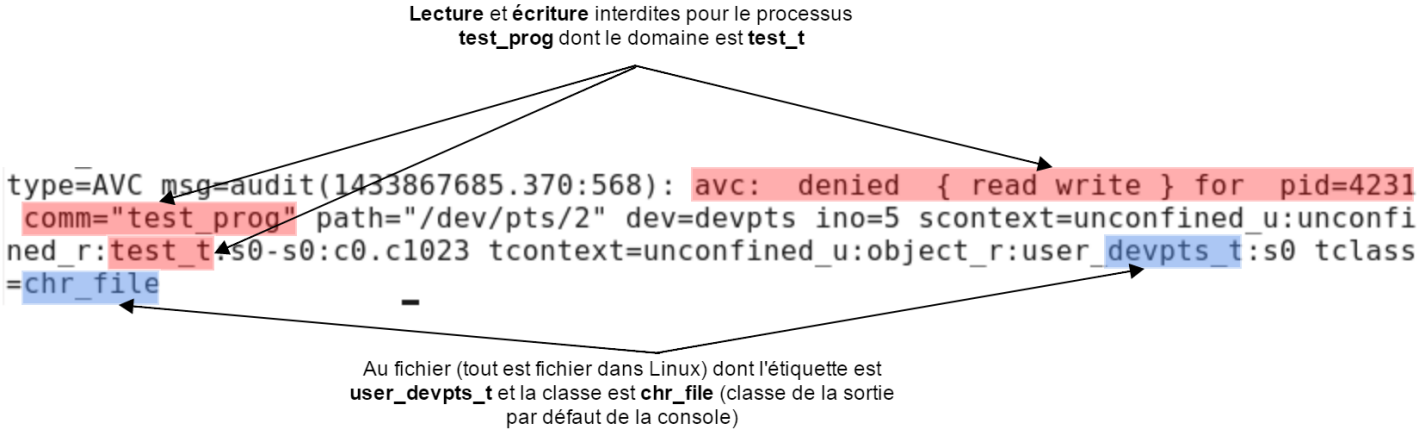

Figure 5-4. Trace du blocage de l'exécutable test dans les journaux.

5.5.2 Création et activation de règles SEL_AVCPolicyRule

Dans cette partie du test, nous utilisons l'éditeur de texte **gedit**, dont le domaine est **unconfined**_t, pour lui permettre l'accès en écriture à un fichier, étiqueté **fichier_t**, dont il a seulement la permission de lecture. Cette autorisation est accordée par la création et l'activation de deux politiques CIM **SEL_AVCPolicyRule**.

Au début, un accès en écriture est tenté à partir d'une session super utilisateur **root** via l'éditeur gedit. La figure 5-5 montre que cet accès est refusé.

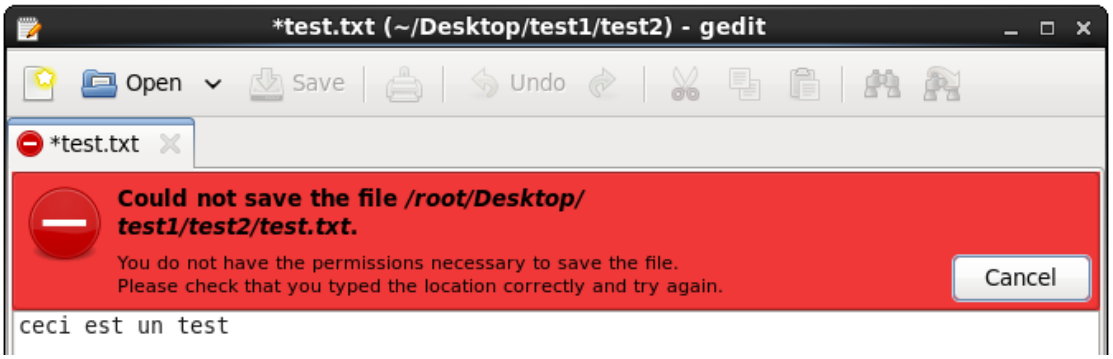

Figure 5-5. Modification du fichier **test.txt** refusée par SELinux.

La consultation du fichier de journalisation **/var/log/audit/audit.log** prouve que l'accès a été bloqué par SELinux, comme le montre la figure 5-6 suivante ;

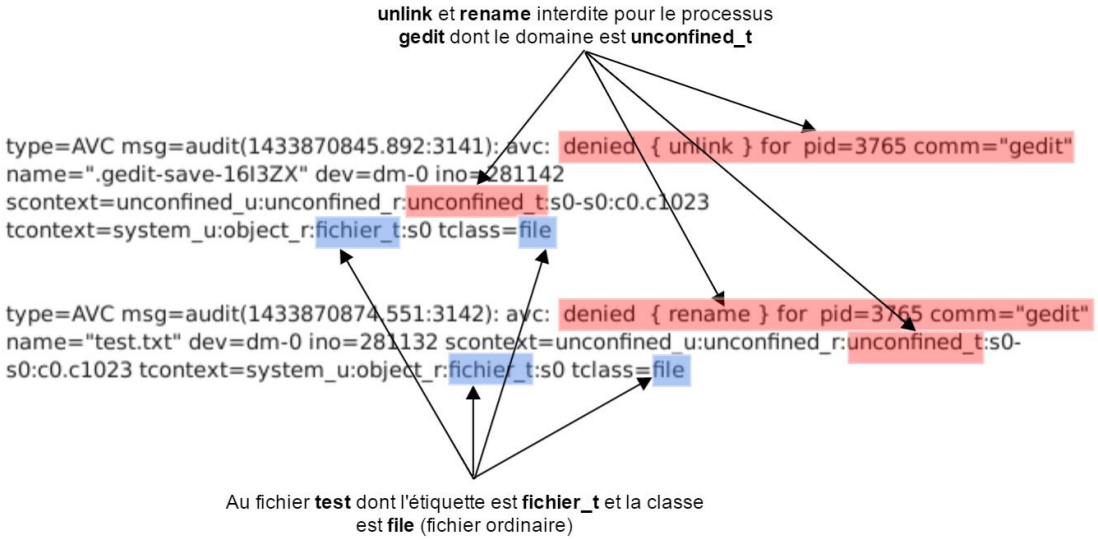

Figure 5-6. Trace du blocage de l'exécutable gedit dans les journaux.

On ajoute ensuite deux règles SEL_AVCPolicyRule permettant d'autoriser la modification du fichier **test** par **gedit**.

La figure 5.7 présente le scénario du test.

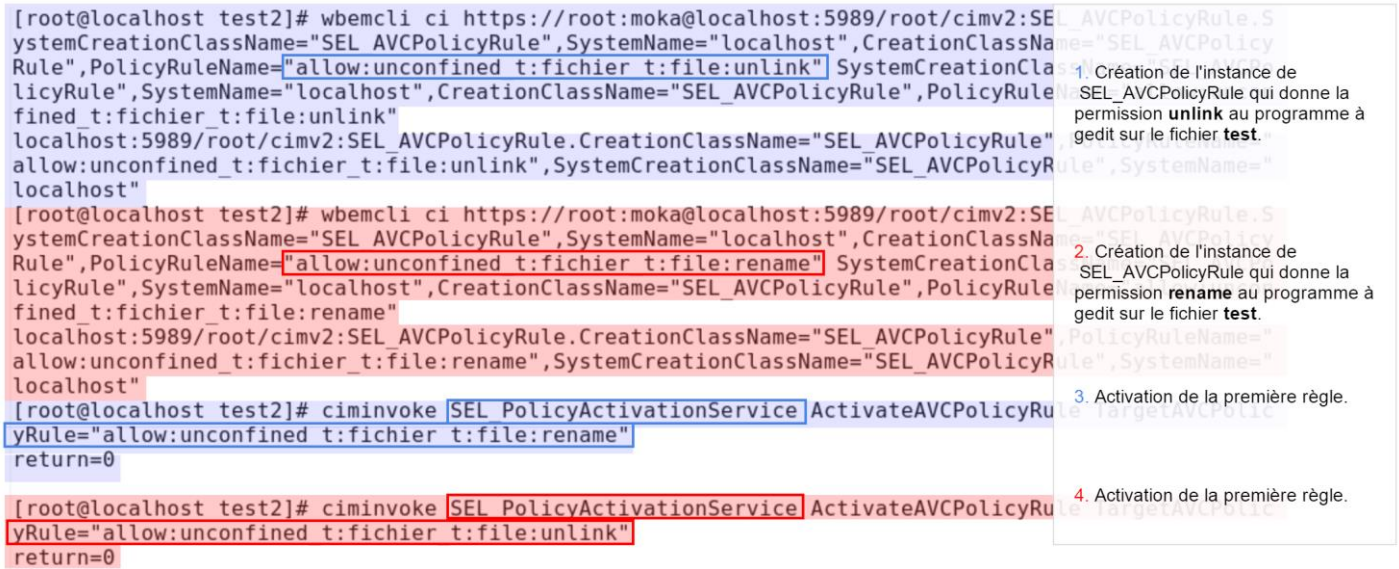

Figure 5-7. Scénario de test illustrant un exemple d'ajout de règles SEL_AVCPolicyRule.

Après l'activation des deux règles SEL_AVCPolicyRule ajoutés plus haut, **gedit** peut modifier le fichier **test** puisque on lui a attribué les permissions **rename** et **unlink**.

5.6 L'IHM de gestion des politiques SELinux

Nous rappelons que le noyau de notre travail est la réalisation du *Provider* CIM permettant d'appliquer des politiques SELinux. L'interopérabilité offerte par CIM permet à n'importe quel client WBEM autorisé, d'utiliser notre P*rovider* pour bénéficier de ses fonctionnalités.

Afin de faciliter l'exploitation de notre système et en tirer le maximum de profit,, nous avons programmé une IHM dédiée, qui offre une interface graphique conviviale permettant de se connecter au serveur afin de gérer les politiques SELinux en toute simplicité.

Pour sa programmation, nous avons opté pour le langage Python. Ce choix est justifié par la puissance de ce langage couplé au faible cout de son cycle de développement (apprentissage, programmation, maintenance, tests et mise au point etc.). En plus de ses avantages, il existe des bibliothèques spécialisées comme **PyWBEM** et **PyQt** qui permettent respectivement, d'offrir les fonctions d'un client CIM et de programmation d'interfaces graphiques.

Dans ce qui suit, nous présentons un scénario possible avec notre IHM illustré par des copies d'écrans :

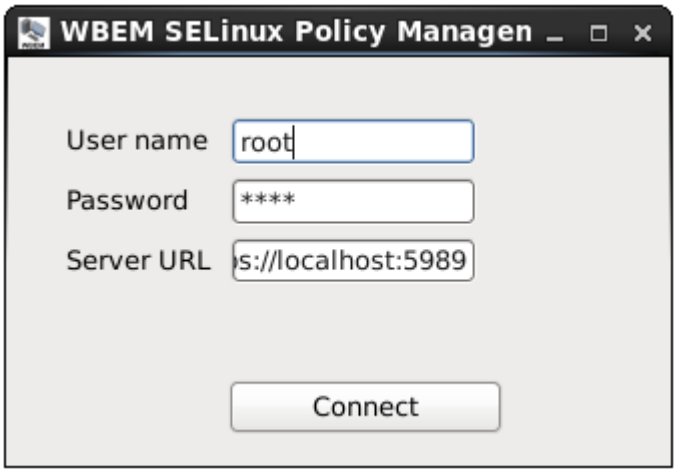

Figure 5-3. Fenêtre de connexion.

Après la connexion, Une autre fenêtre contenant plusieurs onglets s'affiche.

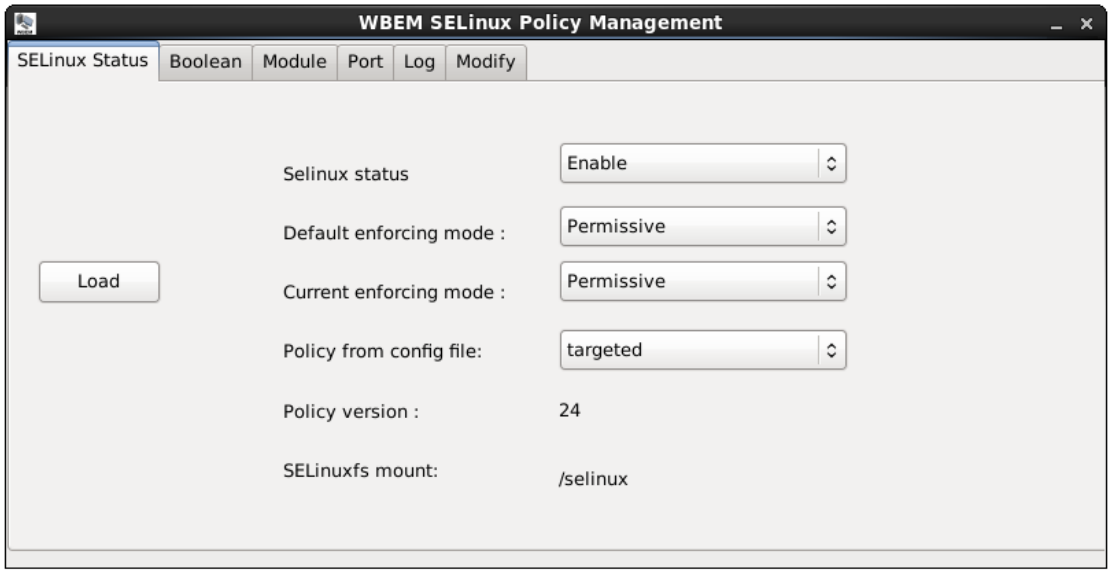

Figure 5-4. Onglet SELinux Status.

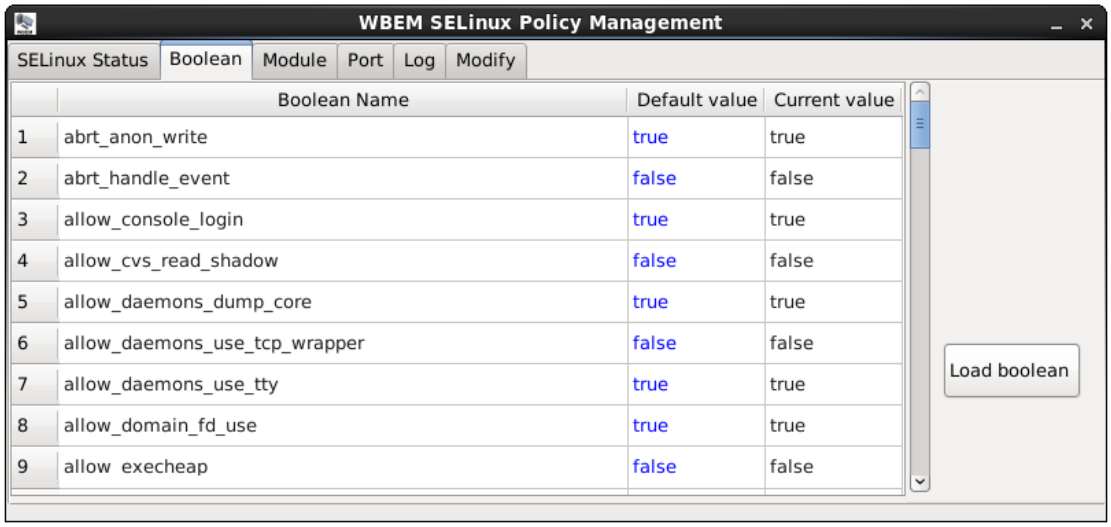

Figure 5-5.Onglet Boolean.

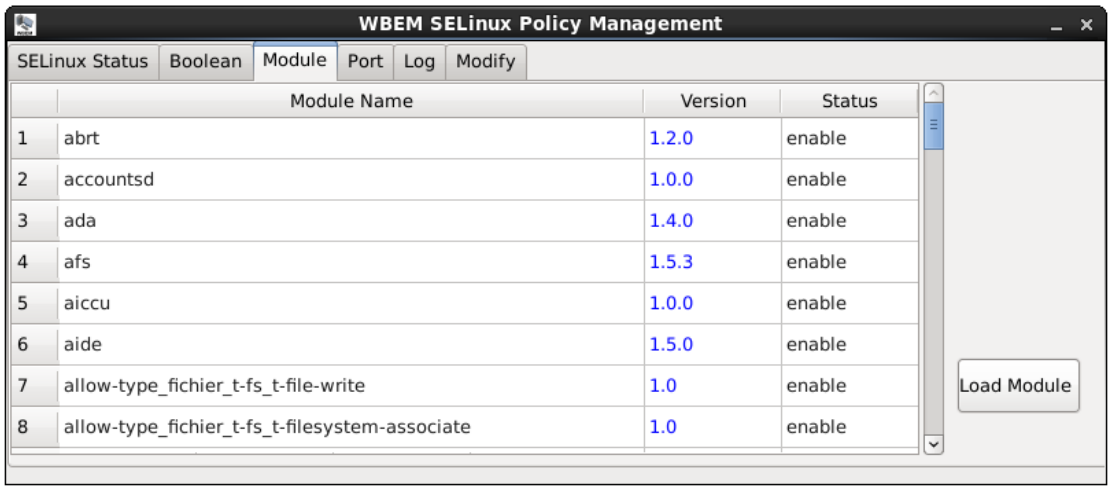

Figure 5-6.Onglet Module.

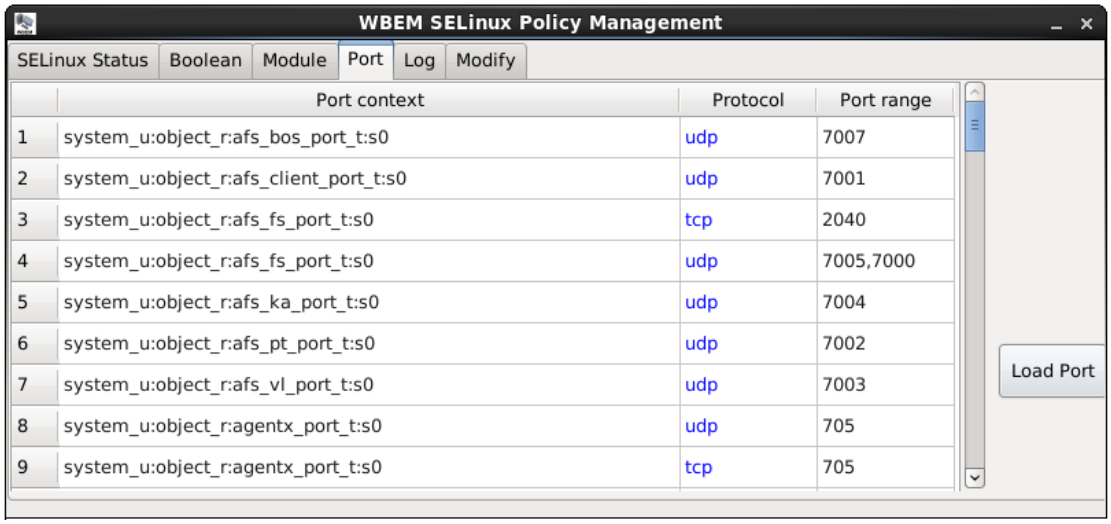

Figure 5-7. Onglet Port.

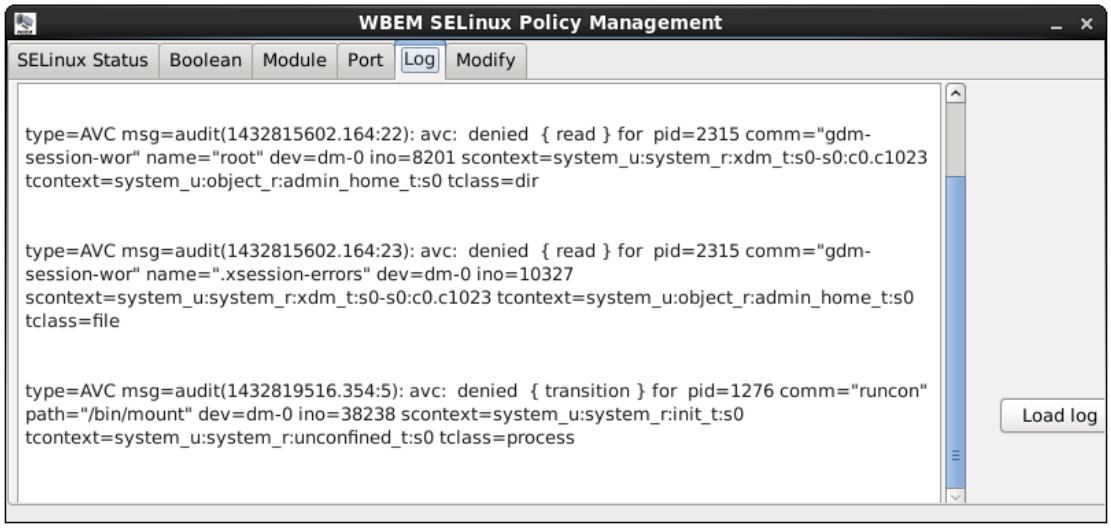

Figure 5-8. Onglet Log.

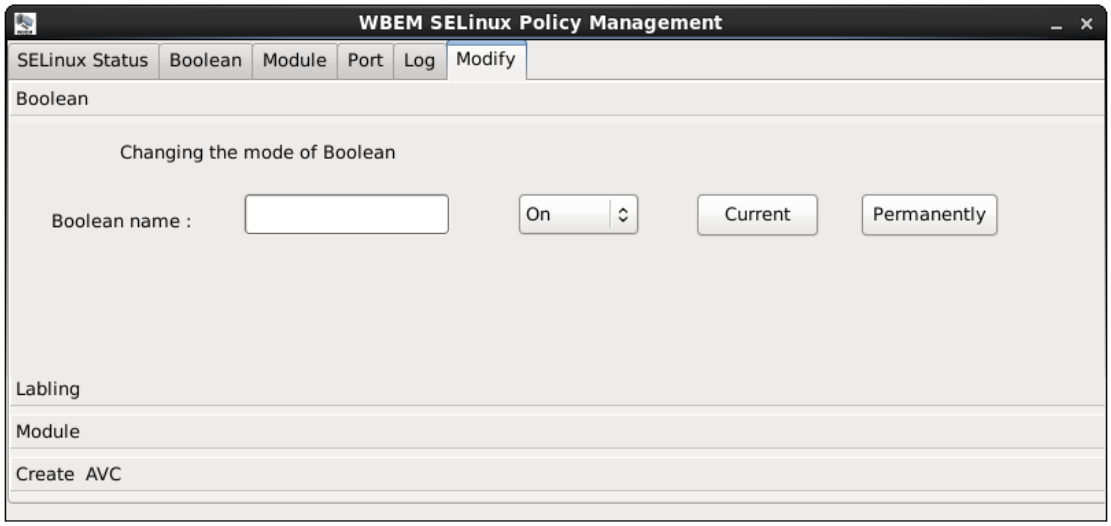

Figure 5-9. Onglet Modify partie Boolean.

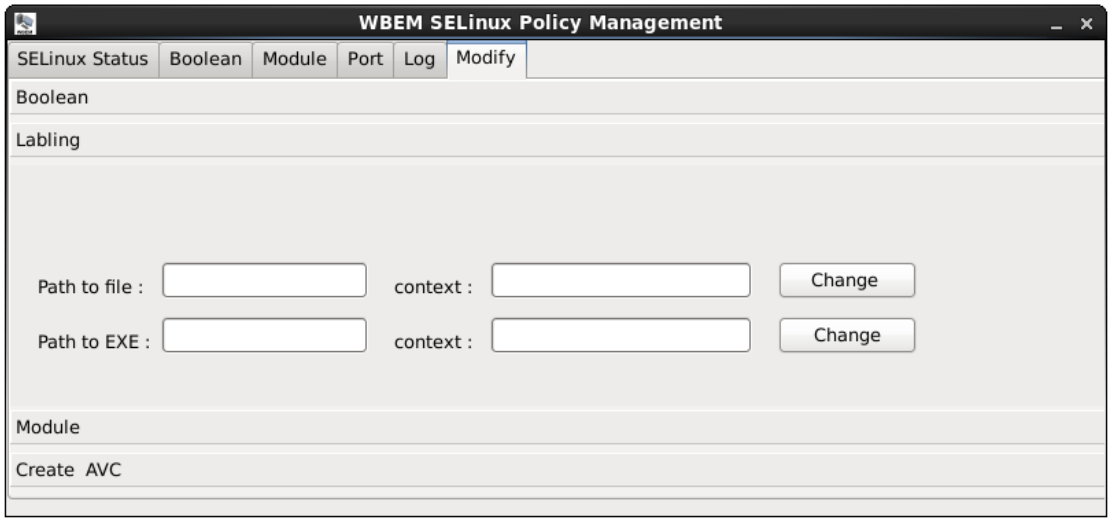

Figure 5-10. Onglet dédié à la manipulation du Labeling.

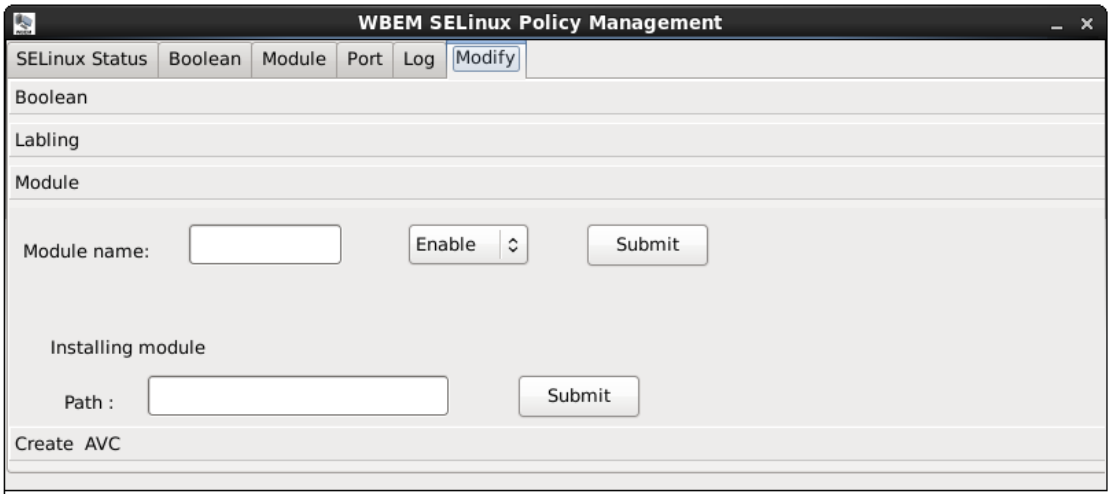

Figure 5-11. Onglet dédié à la manipulation de Modules

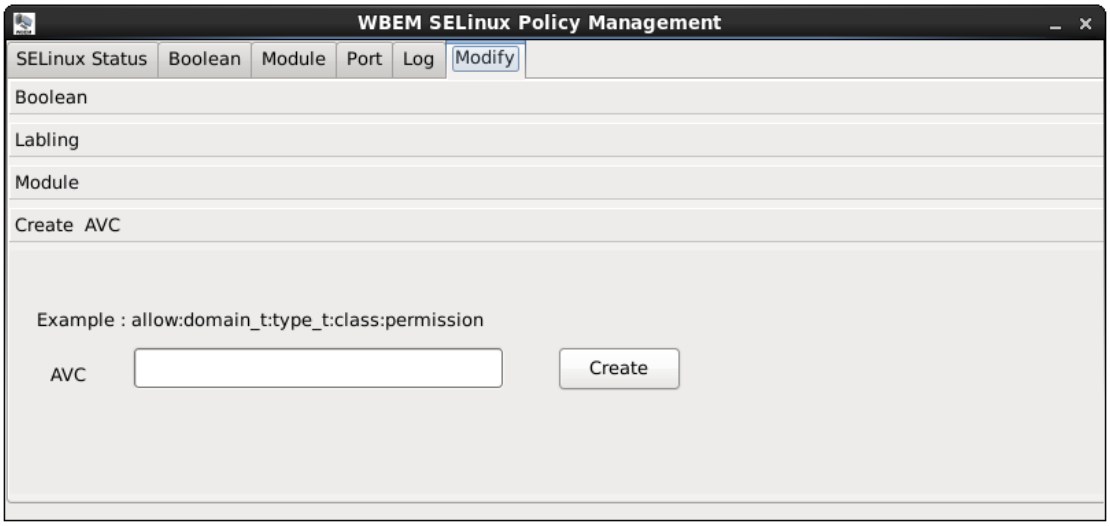

Figure 5-12. Onglet dédié à la création de règles AVC.

5.7 Conclusion

En synthèse, pour implémenter notre système standardisé de gestion de politiques de sécurité, nous avons installé le serveur OpenPegasus ainsi que le client wbemcli dans une plateforme Linux Centos. Ensuite, nous avons installé tous les outils nécessaires pour le développement du *Provider* qui représente le noyau de notre travail d'implémentation. Le fruit de ce travail est une bibliothèque (.so) qui représente le *Provider* permettant d'opérationnaliser notre modèle. Cette librairie peut être enregistrée dans n'importe quel serveur WBEM pour pouvoir gérer des politiques SELinux abstraitement dans suivant le format CIM. Pour faciliter la gestion et offrir un exemple d'exploitation pratique de notre système, nous avons développé une IHM dédié à la gestion des politiques SELinux se basant sur une API CIM client pour manipuler notre Provider.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Ce document présente la conception et réalisation d'un système standardisé dédié à la gestion des politiques de sécurité, qui représente une adaptation du standard CIM/WBEM au LSM SELinux. La conception de ce système montre la possibilité de modéliser, composer, distribuer, appliquer et retirer des politiques SELinux en format CIM et à travers un environnement de gestion CIM/WBEM. Le contrôle d'accès obligatoire robuste ainsi que les mécanismes de journalisation offerts par SELinux représentent les points forts sur lesquelles notre système repose en terme de sécurité.

Ce travail se positionne dans le domaine de la gestion des réseaux et systèmes dont les concepts et principes ont été présentés dans le premier chapitre. En effet la gestion des politiques de sécurité est un aspect de la gestion de la sécurité qui ne peut être isolée des autres composants participants à la bonne gestion IT (configuration, installation, performances, audit etc.)

Dans ce domaine (la gestion IT) qui est aussi complexe que la complexité et hétérogénéité des produits et environnements informatiques existants, diverses initiatives sont proposées par diverses entités chargées de formaliser les bonnes pratiques et standardiser les concepts, protocoles et services de gestion. Tout comme le modèle de référence OSI dédié à la communication en réseau, l'ISO a publié ses recommandations pour un protocole et un système d'information dédiés à la gestion des réseaux et systèmes, il s'agit du CMIP/CMIS. Ce modèle de référence a été adapté par le DMTF afin de produire le standard de gestion WBEM et le modèle d'information commun CIM qui représentent l'évolution la plus importante dans ce domaine. En effet, selon l'approche CIM, toutes les entités informatiques sont représentées sous la forme d'objets abstraits conçus selon les concepts de l'OMG (Object Managament Group) le DMTF qui est une association regroupant un grand nombre d'industriels et éditeurs informatiques, conçoit et publie également des profils CIM spécialisés pour des domaines de gestion précis, parmi lesquels on trouve : CIM Security Profile, CIM Policy Profile, CIM Integrated Access Control Policy Management Profile, CIM Simplified Policy language. Ce dernier a été adopté par l'IETF, l'organisation chargé de normaliser les protocoles Internet.

Dans le cadre de notre projet, nous avons adapté le système de renforcement de sécurité SELinux au standard CIM/WBEM en prenant en compte les spécificités de ce système et en se conformant aux exigences du DMTF en ce qui concerne la réutilisation de profiles existants, en particulier nous avons repris les recommandations et spécifications des profiles cités plus haut. Une bonne étude et maitrise de SELinux était nécessaire pour atteindre cet objectif et dont une synthèse est présentée au chapitre 3.

Nous avons ensuite repris les modèles CIM adéquats pour les étendre avec les éléments permettant d'exprimer les caractéristiques et règles SELinux. Une fois notre modèle CIM spécifique validé, nous avons procédé à son implémentation en commençant par l'installation de l'environnement de gestion WBEM (via le projet OpenPegasus) et puis nous avons apporté notre contribution à ce système par le développement de deux composants logiciels intégrées à l'environnement existant :

- Le provider WBEM qui implémente les classes et associations de notre modèle de gestion de politiques appliquées à SELinux. Il s'agit du cœur de notre système réalisé

conformément à la spécification CMPI (Common Manageability Programming Interface) pour lui permettre de s'intégrer naturellement et d'une manière transparent à n'importe quel serveur CIM/WBEM existant.

- L'IHM de gestion de politiques SELinux qui s'appui sur l'environnement de gestion CIM/WBEM étendu avec notre Provider. La valeur ajoutée de ce programme par rapport aux autres applications graphiques de gestion de SELinux, est qu'il utilise les objets CIM dérivés de notre modèle pour modéliser des politiques de sécurités au lieu d'utiliser les règles natives de l'outil. D'un autre coté, et vu le caractère distribué de l'environnement CIM/WBEM, notre application permet de gérer, naturellement, des systèmes distants dotés de SELinux pourvu qu'un serveur WBEM soit aussi installé sur ces systèmes au sein duquel nos modèles sont compilés et notre Provider installé.

L'aspect standardisé de ce travail lui permet de s'intégrer dans un écosystème concourant à l'unification de la gestion de l'intégralité d'une infrastructure IT, notamment via une politique de sécurité globale exprimée d'une manière homogène (en utilisant le méta-modèle CIM). Pour atteindre cette finalité et en perspectives à notre travail, des projets complémentaires doivent être réalisés visant l'adaptation des autres systèmes de sécurité comme XACML, AppArmor, GrSecurity et TOMOYO, au standard CIM/WBEM. Une fois que la gestion standardisée de ces systèmes est opérationnelle, la gestion des politiques de sécurité d'une infrastructure hétérogène peut être centralisée via une seule console d'administration qui s'appui sur l'API client CIM/WBEM. Un tel système d'administration permettra aux administrateurs de définir des politiques abstraites fondées sur le standard CIM-SPL, qui s'appliquerait à tout le parc informatique.

BIBLIOGRAPHIE

[1] Aiko Pras, Network Management Architectures, CTIT Ph. D-thesis series No. 95-02, 17 fevrier 1995, accessible à l'URL: http://www.utwente.nl/ewi/asna/research/Ph.D.%20Theses/pras-thesis.pdf

[2] Aurélien Méré, La gestion réseau et le protocole SNMP, FIFO04.

[3] BASBOUS Khaled , Security Enhanced for Linux, memoire de Bachelor, 22 juin 2012, disponible à l'URL: http://www.tdeig.ch/linux/Basbous_RTB.pdf

[4] Michael Brasher , Karl Schopmeyer, Using CIMPLE, A Practical Guide to Developing CIM Providers, disponible à l'URL: http://www.simplewbem.org/Using_CIMPLE.pdf, mai 04 2007, dernier accès 9 mars 2009

[5] Dennis John, « CIM Provider HOWTO » disponible à l'URL : https://fedorahosted.org/openlmi/wiki/CimProviderHowto , dernier accès mai 2015

[6] Distributed Management Task Force, DSP0004 Common Information Model (CIM) Metamodel, disponible à l'URL:

[7] Distributed Management Task Force, 2003. CIM Tutorial. disponible à l'URL http://www.wbemsolutions.com/tutorials/DMTF/dmtftutorial.pdf

[8] Distributed Management Task Force, 2003. CIM User and Security Model White Paper, disponible à l'URL: http://www.dmtf.org/standards/documents/CIM/DSP0139.pdf

[9] Distributed Management Task Force (2006). CIM Query Language Specification. disponible à l'URL: http://www.dmtf.org/standards/published_documents/DSP0202.pdf

[10] Distributed Management Task Force, DSP0108 .CIM Policy Model White Paper CIM Version 2.7.0, 18 juin 2003

[11] Distributed Management Task Force, DSP0200_1.3.1.pdf, Specification for CIM Operations over HTTP. 29/07/2009

[12] Distributed Management Task Force, DSP0231. CIM Simplified Policy Language (CIM-SPL) . 14 juillet 2009

[13] Distributed Management Task Force, DSP1003 .Policy Profile Version: 1.0.0a 12-02-2007

[14] Distributed Management Task Force, DSP1106 .Integrated Access Control Policy Management Profile Version: 1.0.0 16 septembre 2011

[15] Festor Olivier, Ben Youssef Nizar, WBEM, Rapport de recherche/INRIA , ISSN 0249-6399 ; 3927, 21 avril 2000

[16] Frank Mayer, Karl MacMillan, David Caplan, "SELinux by Example: Using Security Enhanced Linux" , Prentice Hall; Édition : 1 (27 juillet 2006), ISBN-10: 0131963694, ISBN-13: 978-0131963696

[17] Hertzog Raphaël, Mas Roland, « Le cahier de l'administrateur Debian », 2013, disponible à l'URL : https://debian-handbook.info/browse/fr-FR/stable/sect.selinux.html, dernier accès mai 2015

[18]Chris Hobbs, A Practical Approach to WBEM/CIM Management, CRC Press; 1 edition (February 11, 2004), ISBN-10: 0849323061, ISBN-13: 978-0849323065
[19]Mark Summerfield, Rapid GUI Programming with Python and Qt, ISBN: 9780132354189 Publisher: Prentice Hall, 19/10/2007, disponible à l'URL: http://www.cs.washington.edu/research/projects/urbansim/books/pyqt-book.pdf
[20]B. Moore IBM ,E. Ellesson, LongBoard, Inc.J. Strassner A. WesterinenCisco Systems, Policy Core Information Model -- Version 1 Specification, fevrier 2001, disponible à l'URL: https://tools.ietf.org/html/rfc3060
[21]The Open Group, OpenPegasus Administrator's Guide, ISBN: 1-931624-48-8, avril 2005
[22]Praveen Kumar Paladugu, Dell Linux Engineering Team, WBEM Based Management in Linux, fevrier 2011, A Dell Technical White Paper, disponible à l'URL: http://linux.dell.com/files/whitepapers/WBEM_based_management_in_Linux.pdf
[23]Ranc Daniel, Gestion de réseaux et services, une introduction, INT, 2004, accessible à l'URL : http://www-lor.int-evry.fr/~agirs/gestion-reseaux-v2.03.pdf, dernier accès mai 2015
[24]Yoo S-M., Hong. J. W-K., (2006). Performance Evaluation of WBEM Implementations, KNOM Review, 8, 7-13

Références WEB

[25] Gentoo Linux, SELinux/Quick introduction. https://wiki.gentoo.org/wiki/SELinux/Quick_introduction, dernier accès mai 2015
[26]Open Pegasus project http://www.openpegasus.org, dernier accès avril 2015.
[27]Open Group http://www.opengroup.org/, dernier accès avril 2015
[28]OpenLMI http://www.openlmi.org , dernier accès avril 2015
[29]PyWbem project http://pywbem.sourceforge.net/, dernier accès avril 2015
[30]PyQT http://www.riverbankcomputing.co.uk/software/pyqt/intro, dernier accès avril 2015
[31]SBLIM project http://sourceforge.net/projects/sblim, dernier accès avril 2015
[32]SELinux Project, libselinux 2.3 Library Functions. http://selinuxproject.org/page/LibselinuxAPISummary, dernier accès avril 2015
[33]SimpleWbem/CIMPLE project, http://www.simplewbem.org/
[34]SNMP Overview, https://www.webnms.com/snmputilities/help/quick_tour/snmp_and_mib/snmpmib_snmpoverview.html, denier accès avril 2015
[35]SNMP par_SebF & Sandra, http://www.frameip.com/snmp/, dernier accès avril 2015
[36]Gestion de réseaux, https://fr.wikipedia.org/wiki/Gestion_de_r%C3%A9seaux, dernier accès avril 2015.