

Sécurité du plan de gestion des réseaux IP

THÈSE

présentée et soutenue publiquement le 11 décembre 2006

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Vincent Cridlig

Composition du jury

<i>Rapporteurs :</i>	Omar CHERKAOUI	Professeur, UQAM, Canada
	Frédéric CUPPENS	Professeur, ENST-Bretagne
<i>Examineurs :</i>	Olivier FESTOR	Directeur de Recherche, INRIA Lorraine
	Stéphane FRENOT	Maître de conférences, INSA Lyon
	Claude GODART	Professeur, ESSTIN et UHP Nancy 1
	Radu STATE	Chargé de Recherche, INRIA Lorraine

Mis en page avec la classe thloria.

Remerciements

Je tiens tout d'abord à remercier Omar CHERKAOUI, Frédéric CUPPENS, Stéphane FRENOT et Claude GODART d'avoir accepté de prendre part à ce jury de thèse.

Je remercie mon directeur de thèse, Olivier Festor, qui m'a fait l'honneur d'accepter de diriger ma thèse, pour son soutien enthousiaste toujours accompagné d'une grande rigueur scientifique.

Je remercie chaleureusement mon encadrant, Radu State, qui m'a guidé tout au long de cette thèse et qui m'a apporté, jour après jour, son jugement scientifique pertinent et enrichissant mais aussi des nouvelles perspectives et orientations pour mon travail de recherche.

Je remercie l'ensemble des membres de l'équipe Madynes, permanents, doctorants, ingénieurs, stagiaires et assistante et en particulier mes collègues de bureau.

Je remercie également mes parents pour leur soutien et leur générosité sans faille tout au long de ces années de formation. Je tiens également à remercier mes frères et sœur, Joëlle, Raoul et Gautier, ainsi que ma grand-mère.

Je remercie l'INRIA et la région Lorraine pour m'avoir fait confiance en m'accordant une bourse de thèse. Je remercie les personnels de l'Université Henri-Poincaré et du LORIA qui ont contribué efficacement au bon déroulement de ma thèse.

Table des matières

Introduction

Chapitre 1 Introduction générale : un environnement hétérogène distribué	3
1.1 La supervision de réseau : un domaine critique pour la sécurité du réseau . . .	3
1.1.1 Des données sensibles	3
1.1.2 Des protocoles de gestion puissants mais faiblement sécurisés	4
1.1.3 Des spécificités propres à la gestion de réseau	4
1.2 Une hétérogénéité grandissante	5
1.2.1 Multiple équipements et plateformes de gestion de réseau	5
1.2.2 Multi-domaines de gestion	6
1.2.3 Multi-managers/providers	6
1.3 Le problème adressé	6
1.3.1 Organiser le contrôle d'accès	7
1.3.2 Définition de la cohérence locale	7
1.3.3 Définition de la cohérence globale	8
1.4 Contributions	8
1.4.1 Assurer la cohérence globale du contrôle d'accès	8
1.4.2 Détection de conflits et assurer la cohérence locale	9
1.4.3 Implantations et évaluation	9
1.5 Plan de la thèse	10
1.5.1 Partie I : La sécurité et les plateformes de gestion de réseau	10
1.5.2 Partie II : Vers une double convergence des modèles et politiques de sécurité	10
1.5.3 Partie III : Expérimentations et évaluations	10
 Partie I La sécurité et les plateformes de gestion de réseau	 11
 Chapitre 2 Les architectures, modèles et technologies de sécurité	 13

2.1	Modèle de menaces	13
2.2	Les services de sécurité	14
2.2.1	Authentification	14
2.2.2	Intégrité	14
2.2.3	Confidentialité	14
2.2.4	Contrôle d'accès	14
2.3	Les modèles de contrôle d'accès	14
2.3.1	Contrôle d'accès obligatoire (MAC)	14
2.3.2	Contrôle d'accès discrétionnaire (DAC)	15
2.3.3	Contrôle d'accès basé sur les rôles (RBAC)	15
2.4	Synthèse	21

Chapitre 3 La sécurité des plateformes de gestion : les approches et leurs limites 23

3.1	Couplage fort entre plateformes de supervision et mécanismes de sécurité	23
3.1.1	L'évolution SNMP : v1,v2,v3	23
3.2	Couplage faible	31
3.2.1	L'interface en ligne de commandes CLI	31
3.2.2	Le protocole de configuration Netconf	33
3.2.3	Un protocole destiné aux passerelles résidentielles : TR-069	35
3.2.4	Un exemple d'outil d'administration avec interface web : Webmin	37
3.3	Synthèse	38
3.3.1	Le groupe ISMS	40

Partie II Vers une double convergence des modèles et politiques de sécurité 41

Chapitre 4 Une approche pour la cohérence globale 43

4.1	Convergence du contrôle d'accès en amont	43
4.1.1	Unifier la gestion du contrôle d'accès	43
4.1.2	Automatiser le déploiement	44
4.2	Des modèles de convergence concrets	44
4.2.1	Passerelles multi-protocoles	44
4.2.2	Algorithme de conversion RBAC/CLI avec pertes	45
4.2.3	Une architecture de déploiement du modèle RBAC	50
4.3	Synthèse	58

Chapitre 5 Une approche pour assurer la cohérence locale	59
5.1 Introduction	59
5.1.1 Contexte	59
5.1.2 Définition formelle du modèle RBAC	60
5.2 Formalisation des modèles de contrôle d'accès hétérogènes	61
5.2.1 Définition pour la convergence du modèle VACM vers le modèle RBAC	61
5.2.2 Définition de la convergence entre le modèle de sécurité de CLI vers le	
modèle RBAC	63
5.2.3 Définition pour la convergence du modèle TR-069 vers le modèle RBAC	66
5.2.4 Implantation de l'algorithme	67
5.3 Détection de conflits par comparaison et approximations	67
5.3.1 Définition des opérations de comparaison	67
5.3.2 Algorithme pour la détection de conflits de politique	71
5.3.3 Conservation de l'équivalence de politique lors d'opérations de confi-	
guration	71
5.4 Synthèse	72
5.4.1 Un modèle formel pour vérifier la cohérence des politiques de contrôle	
d'accès	72
5.4.2 Limites et alternatives	73
Chapitre 6 Une proposition de contrôle d'accès dans Netconf	75
6.1 Un modèle RBAC dans Netconf	76
6.1.1 Une politique auto-protégée administrable à distance	76
6.1.2 XPath pour l'adressage des données du modèle	76
6.1.3 Spécification de la politique RBAC	76
6.1.4 Un gendarme pour le filtrage des données XML	78
6.2 Proposition d'extension pour Netconf : "#rbac capability"	78
6.2.1 Une nouvelle opération Netconf	78
6.2.2 Format des messages	79
6.3 Scénario d'utilisation	79
6.4 Synthèse et perspectives	82
Partie III Expérimentations et évaluations	87
Chapitre 7 Passerelle XML/SNMP sécurisée	89
7.1 Introduction	89
7.1.1 Les passerelles XML/SNMP	89

7.1.2	Motivation	90
7.1.3	Qualités requises pour la gestion de la politique de sécurité	90
7.2	Architecture fonctionnelle	90
7.3	Définition d'une politique d'autorisation	92
7.3.1	Lien entre RBAC et les données de gestion	92
7.3.2	Algorithme de translation unidirectionnel RBAC/VACM	92
7.4	Scénario d'utilisation	95
7.4.1	Processus de contrôle d'accès	95
7.4.2	Prise en comptes de nouveaux agents	96
7.4.3	Une interface pour l'administration de la passerelle	96
7.4.4	Un continuum d'authentification et de confidentialité	96
7.5	Synthèse	96
Chapitre 8 Implantation d'une suite logicielle pour Netconf		99
8.1	EnSuite : une plateforme Netconf de configuration de réseau	99
8.1.1	La plateforme EnSuite	99
8.1.2	Le côté agent de la plateforme EnSuite	100
8.1.3	Manager	104
8.2	Impact de la sécurité sur les performances	104
8.2.1	Performances générales	104
8.2.2	De l'utilisation de la compression	107
8.2.3	Intégrité et confidentialité	108
8.2.4	Combiner compression et chiffrement	109
8.2.5	Performance du filtrage XPath comparée au subtree filtering	109
8.2.6	Impact de la politique de contrôle d'accès	110
8.2.7	Sécurité XML intégrée vs SSH	110
8.3	Synthèse	111

Conclusion

Chapitre 9 Conclusions et perspectives des recherche		115
9.1	Conclusions	115
9.2	Perspectives	116
9.2.1	Performances des modèles de sécurité dans le cadre de la supervision de réseau	116
9.2.2	Contrôle d'intégrité des configurations	116
9.2.3	Une approche de fédération des identités	117

9.2.4	Couche logicielle de convergence bas niveau	117
9.2.5	Représentation des données	118
Publications		
Annexes		121
Annexe A Résultats de la conversion des modèles de contrôle d'accès		121
A.1	Exemple de conversion de VACM vers RBAC	121
A.1.1	Politique VACM en entrée	121
A.1.2	Résultat de la translation vers RBAC de la politique VACM	121
A.2	Exemple de conversion de CLI vers RBAC	121
A.2.1	Politique CLI en entrée	121
A.2.2	Résultat de la translation vers RBAC de la politique CLI	124
A.3	Modèle RBAC utilisé dans EnSuite	124
A.3.1	Politique RBAC en entrée	124
A.3.2	Résultat de la translation vers RBAC de la politique RBAC de EnSuite	124
Annexe B Les modèles et technologies XML émergents		129
B.1	Sécurité des web services	129
B.1.1	XML-Encryption	129
B.1.2	XML-DigitalSignature	130
B.2	SAML et XACML	131
B.2.1	Autorisation distribuée et Single-Sign On : SAML	131
B.2.2	Contrôle d'accès par politiques : XACML	132
B.3	Contrôle d'accès sur le contenu XML	133
Bibliographie		135

Table des figures

1.1	Coexistence de multiples protocoles de supervision de réseaux	5
2.1	Les variantes du modèle RBAC	16
2.2	RBAC 3 : hiérarchie et contraintes (DSD et SSD).	17
2.3	Les types de hiérarchies	20
3.1	Architecture de SNMPv3	25
3.2	Structure du message SNMPv3	26
3.3	Algorithme de calcul du MAC	26
3.4	Génération de la clé DES	27
3.5	Chiffrement avec CBC-DES	27
3.6	Génération des clés privKey et AuthKey	28
3.7	Distribution des clés localisées dans SNMPv3	28
3.8	Exemple de politique de contrôle d'accès avec VACM	30
3.9	Un extrait de commandes de configuration CLI	32
3.10	La sécurité de l'approche CLI	33
3.11	La pile protocolaire interne à Netconf	34
3.12	Les principaux composants d'une plateforme TR-069	36
3.13	Un exemple de configuration des ACL pour TR-069	38
4.1	Comparaison des modèles de contrôle d'accès	46
4.2	Problèmes de conversion des modèles de contrôles d'accès	46
4.3	Affectation des poids et lien avec les niveaux CLI	47
4.4	Application du K_means clustering à la conversion RBAC vers CLI	49
4.5	Algorithme K_means clustering	49
4.6	Les différentes composants du modèle	51
4.7	Représentation d'un arbre de clé	54
4.8	Encapsulation du message Netconf	55
4.9	Une ACL associée à un nœud particulier	57
5.1	Schéma de convergence des modèles de contrôle d'accès	60
5.2	Extrait de configuration CLI relatif à la sécurité	64
5.3	Traduction de la configuration de sécurité CLI	68
5.4	Exemples de comparaison de permissions	69
5.5	Les types de conflits identifiés	70
5.6	Algorithme pour la détection de conflit de politiques affichant le delta des permissions entre deux politiques pour un utilisateur donné.	72

6.1	Représentation XML de la politique RBAC dans l'agent Netconf	77
6.2	Implantation du système de contrôle d'accès	78
6.3	Filtrage par sous-arbre (subtree filtering) et par XPath	79
6.4	Nouveaux messages Netconf pour RBAC	80
6.5	Définition des rôles pour le routeur de bordure	82
6.6	Représentation XML de la politique RBAC dans l'agent Netconf (début)	83
6.7	Représentation XML de la politique RBAC dans l'agent Netconf (fin)	84
7.1	Architecture fonctionnelle de la passerelle sécurisée	91
7.2	Schéma général de la traduction RBAC/VACM	93
7.3	Algorithme de traduction de RBAC vers VACM	94
8.1	Vue globale de la plateforme EnSuite	100
8.2	Structure logicielle de EnSuite	101
8.3	Design pattern Command	102
8.4	Extrait de configuration relative au protocole RIP	103
8.5	Diagramme UML des commandes RIP	104
8.6	Temps de traitement CPU global d'une requête	106
8.7	Aperçu des tailles de messages Netconf	107
8.8	Résultat de compression et de chiffrement	108
8.9	Temps d'occupation CPU avec différentes options	109
8.10	Impact du nombre de permissions sur le temps d'évaluation	111
9.1	Deux approches pour le déploiement de politiques de contrôle d'accès	117
A.1	Configuration de sécurité originale de SNMP VACM	122
A.2	Configuration de sécurité SNMP VACM convertie	123
A.3	Configuration de sécurité originale de CLI	124
A.4	Configuration de sécurité CLI convertie	125
A.5	Configuration de sécurité originale de Netconf	126
A.6	Configuration de sécurité de Netconf (fonctionnalité RBAC) convertie	127
B.1	Exemple d'élément XML chiffré respectant la syntaxe XML-Encryption	129
B.2	Exemple de signature XML-DSIG	130
B.3	Exemple d'assertion SAML (Source : [57])	131
B.4	Exemple de politique XACML (Source : [3])	133

Introduction

Introduction générale : un environnement hétérogène distribué

Sommaire

1.1	La supervision de réseau : un domaine critique pour la sécurité du réseau	3
1.1.1	Des données sensibles	3
1.1.2	Des protocoles de gestion puissants mais faiblement sécurisés	4
1.1.3	Des spécificités propres à la gestion de réseau	4
1.2	Une hétérogénéité grandissante	5
1.2.1	Multiple équipements et plateformes de gestion de réseau	5
1.2.2	Multi-domaines de gestion	6
1.2.3	Multi-managers/providers	6
1.3	Le problème adressé	6
1.3.1	Organiser le contrôle d'accès	7
1.3.2	Définition de la cohérence locale	7
1.3.3	Définition de la cohérence globale	8
1.4	Contributions	8
1.4.1	Assurer la cohérence globale du contrôle d'accès	8
1.4.2	Détection de conflits et assurer la cohérence locale	9
1.4.3	Implantations et évaluation	9
1.5	Plan de la thèse	10
1.5.1	Partie I : La sécurité et les plateformes de gestion de réseau	10
1.5.2	Partie II : Vers une double convergence des modèles et politiques de sécurité	10
1.5.3	Partie III : Expérimentations et évaluations	10

1.1 La supervision de réseau : un domaine critique pour la sécurité du réseau

1.1.1 Des données sensibles

La supervision ou gestion de réseau rassemble toutes les activités qui permettent de s'assurer du bon fonctionnement du réseau. Elle consiste notamment à contrôler le trafic, contrôler l'état

de fonctionnement des équipements, gérer, maintenir, configurer les différents équipements d'un réseau, détecter les anomalies et les pannes, analyser des alarmes, se protéger des attaques. Cette tâche s'effectue le plus souvent à distance, depuis une console d'administration. Cette activité est critique dans la mesure où dorénavant, quasiment toutes les communications d'une entreprise transitent par le réseau informatique : email, voix sur IP, services web (serveurs de fichiers, application distribuées), etc... Il est donc important, pour une entreprise, un fournisseur d'accès ou un opérateur, de maîtriser cette architecture afin de garantir une grande disponibilité des services.

On distingue généralement deux grandes activités de supervision : le *monitorage* et la *configuration*. Le *monitorage* consiste à collecter et analyser l'information. C'est une activité d'observation principalement des flux (ou trafic) et de l'état des équipements. L'approche généralement adoptée consiste à interroger à intervalles réguliers les équipements via un protocole spécifique, de façon à récupérer l'information décrivant l'état du matériel : connexions réseaux, etc... L'étude de ces informations au cours du temps permet de suivre l'évolution de l'activité de l'équipement et éventuellement de détecter des anomalies (erreurs, suractivité). Il est possible de définir des seuils à partir desquels des alertes sont générées et envoyées aux administrateurs du réseau.

La deuxième grande activité est la *configuration*. Elle consiste à définir le comportement des équipements en positionnant les valeurs des variables présentes sur un équipement. Par exemple, il est possible de configurer la table de routage d'un routeur, d'activer ou de désactiver des interfaces. On peut aussi vouloir configurer à distance les règles de filtrage d'un pare-feu. Contrairement au monitoring qui est une activité d'observation, la configuration consiste à agir sur le comportement du réseau ou des équipements.

La supervision de réseau est une activité potentiellement à risque puisqu'elle autorise une grande possibilité de manœuvre. De ce fait, elle nécessite évidemment des mécanismes de sécurité pour restreindre les opérations de gestion aux seules personnes habilitées. Les données de gestion sont particulièrement sensibles dans la mesure où elles peuvent révéler l'architecture complète d'un réseau et les différentes politiques mises en œuvres : routage, localisation des serveurs, sécurité. La connaissance de telles informations peut permettre de planifier une attaque ultérieure.

1.1.2 Des protocoles de gestion puissants mais faiblement sécurisés

Les protocoles de gestion existants ont été conçus, pour la plupart, avant que l'Internet ne croisse dans les proportions que l'on connaît aujourd'hui et que les problèmes de sécurité liés n'apparaissent aussi fréquemment et violemment. Depuis, la sécurité de n'importe quel protocole intervient dès sa phase de conception. Encore aujourd'hui, de nombreux sites informatisés utilisent les protocoles de supervision de réseau de façon non sécurisée : mot de passe par défaut ("public") avec SNMPv1 transmis en clair sur le réseau, ainsi que toutes les données de gestion, login/mot de passe en clair également avec CLI (Command Line Interface) sur telnet. Un simple sniffer permet de récupérer ces tokens de sécurité et de commencer à gérer le réseau de façon illicite.

1.1.3 Des spécificités propres à la gestion de réseau

L'une des caractéristiques des protocoles de gestion de réseau est qu'ils sont supposés fonctionner en environnement dégradé, c'est-à-dire pendant les périodes instables où le réseau fonctionne mal. Cela peut se produire à l'installation du réseau ou en cas de pannes. Dans le contexte de sécurité, cela signifie qu'ils doivent éventuellement continuer à fonctionner sans avoir accès à un

serveur de distribution de clés, un serveur d'authentification ou autres.

Pour la plupart des plateformes de gestion de réseau et contrairement à une arborescence Unix, il n'y a pas de notion de propriété sur les données. Ceci a pour conséquence que les droits d'accès sont gérés de façon globale par un super utilisateur et non, par chaque utilisateur propriétaire (Discretionary Access Control DAC). Donc il y a un facteur d'échelle qui est introduit ici car la gestion des droits est moins distribuée que dans un système Unix.

1.2 Une hétérogénéité grandissante

1.2.1 Multiple équipements et plateformes de gestion de réseau

Les protocoles de gestion se sont adaptés aux nouvelles exigences de sécurité en proposant différents services de sécurité, principalement l'authentification, la confidentialité, l'intégrité, l'anti-rejeu. Les choix dans la mise en œuvre de ces services ont été différents en fonction des protocoles :

- refonte complète des message et de l'architecture logicielle comme c'est le cas avec SNMPv3,
- modification du protocole sous-jacent comme CLI qui a migré de telnet vers SSH.

Ce choix différent dans les mécanismes de sécurité introduit un nouveau facteur d'hétérogénéité entre les différents protocoles de gestion. En effet, il est courant qu'un administrateur utilise plusieurs de ces plateformes de supervision selon les besoin : CLI pour configurer les routeurs, SNMP ou syslog pour collecter les données statistiques et surveiller le bon fonctionnement des liens et des équipements (routeurs, serveurs, ...). On assiste également à l'éclosion des solutions basées sur le langage XML [65]. L'hétérogénéité des plateformes a été encore amplifiée par l'introduction des mécanismes de sécurité, ce qui pose des problèmes de gestion des politiques de sécurité, incompatibles d'une plateforme à l'autre. C'est d'autant plus flagrant pour les politiques de contrôle d'accès qui ont besoin de décrire les données protégées. Comme les données se recouvrent d'une plateforme à l'autre, il apparaît des problèmes de cohérence, c'est-à-dire de conflits entre politiques de contrôle d'accès.

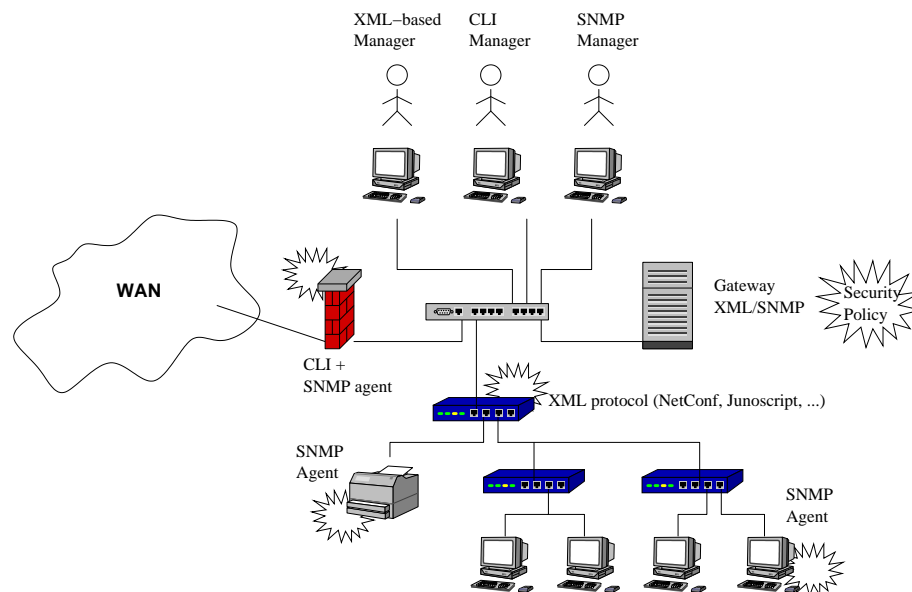


FIG. 1.1 – Coexistence de multiples protocoles de supervision de réseaux

La multiplicité de type d'équipements gérés est un problème dans le sens où chaque équipement affiche des caractéristiques différentes. Il met à disposition des MIBs standards et propriétaires qui nécessitent une adaptation des politiques de sécurité, surtout le contrôle d'accès. Chaque équipement propose une ou plusieurs interfaces de gestion, ce qui implique un coût non négligeable en terme de configuration des paramètres de sécurité.

1.2.2 Multi-domaines de gestion

De plus en plus, se pose le problème de recouvrement des domaines de gestion. En effet, il n'est pas rare qu'un équipement soit partagé et donc configuré par plusieurs entités appartenant à plusieurs domaines administratifs. De même, une entité peut vouloir accéder, même en lecture seule, à certaines parties de la configuration d'un équipement d'un domaine de gestion voisin. Cette situation peut se produire, par exemple, dans le cadre de BGP où un propriétaire de réseau peut vouloir consulter les *route-map* de ses voisins pour diagnostiquer d'éventuels dysfonctionnements ou des comportements inattendus. Des réflexions sont également en cours dans le cadre du protocole TR-069 pour fournir un cadre multi-provider où un utilisateur final a souscrit à différents services fournis par plusieurs fournisseurs. Dans ce contexte, il est nécessaire de définir une approche où les fournisseurs vont pouvoir accéder au même équipement sans introduire de conflits.

1.2.3 Multi-managers/providers

Un même domaine de gestion peut être administré par plusieurs gestionnaires qui sont chacun spécialistes dans un aspect particulier. Des managers doivent donc avoir des droits d'accès différents en fonction de leur fonction, pour définir les responsabilités et réduire les risques d'erreurs de configuration. Cela implique la définition de comptes utilisateurs, la distribution automatisée des tokens de sécurité dans le cas de réseaux à grande échelle, le déploiement de politiques de contrôle d'accès.

Sécuriser ne signifie pas forcément fermer tous les accès et permet au contraire d'offrir plus de pouvoir aux fournisseurs de configuration et de services autorisés de façon maîtrisée.

1.3 Le problème adressé

La supervision des réseaux n'est pas suffisamment sécurisée. Les ressources concernées par la gestion nécessitent des mécanismes de protection pour masquer leur contenu ou restreindre leur accès. Le domaine accuse un certain retard sur les autres domaines ou systèmes distribués. SNMP illustre parfaitement ce retard accumulé en matière de sécurité car il nécessite une gestion manuelle des clés cryptographiques ce qui pose un problème de passage à l'échelle et il utilise des algorithmes devenus faibles comme MD5 ou DES. De plus, le modèle de contrôle d'accès est difficile à maintenir et est trop dépendant des modèles de sécurité utilisés parallèlement. Certaines autres approches sont mieux pensées ou ont mieux évolué au cours des dernières années (par exemple CLI avec RADIUS) mais leur nombre et leurs différences rendent l'administration difficile.

Il y a un fort besoin d'homogénéisation des modèles qui s'il ne peut pas se traduire par une modification profonde des modèles existants, peut être approché par des sur-modèles ou des sous-modèles qui tendent vers une certaine convergence des politiques de sécurité. Un exemple de sur-modèle serait d'avoir une politique générique qui serait déployée automatiquement à grande

échelle et qui saurait s'adapter aux structures existantes. Un exemple de sous-modèle s'appliquerait entre le logiciel agent et le système pour fédérer les identités et le contrôle d'accès.

1.3.1 Organiser le contrôle d'accès

Lorsqu'on souhaite avoir un modèle globale avec un environnement aussi hétérogène, il se pose de nombreuses questions :

- Comment organiser les opérations de gestion et les privilèges afin de limiter le risque d'erreurs et limiter la complexité de configuration du contrôle d'accès ?
- Quelle approche ? centralisée ou distribuée ?
- Quel environnement ? nombre d'utilisateurs potentiels, de machines.
- Quel modèle de contrôle d'accès ?
- Quel modèle de données pour prendre en compte l'hétérogénéité ?
- Comment gérer les multiples gestionnaires sans créer de conflit ?
- Comment limiter les erreurs de configuration ?
- Faut-il conserver une trace des opérations de gestion pour conserver la responsabilité ?
- Comment éviter les erreurs de SNMP ?

Cet environnement a la particularité d'être hétérogène, multi-plateformes, distribué et multi-managers. Répondre aux questions précédentes implique forcément le concept de *fédération* que ce soit pour les identités, les ressources ou les privilèges. Dans un contexte différent qui est celui du monde HTTP et des sites d'e-business ou e-commerce, une initiative importante, SAML, a adressé un problème aux caractéristiques proches. En effet, SAML permet la gestion des identités entre plusieurs sites web en définissant un protocole d'échange d'assertions de sécurité, un mécanisme de fédération des identités et de Single Sign On. L'objectif est la coopération entre les différents sites pour simplifier la vie de l'utilisateur. Celui-ci a un compte sur chacun des sites mais ce sont les sites qui interopèrent pour gérer les sessions en se transmettant les informations d'authentification ou d'autorisation adéquates.

Cette approche manque à la gestion de réseau car elle améliorerait grandement l'interopérabilité entre les systèmes SNMP, CLI, Wbem, Netconf, etc... Cependant, il y a une différence de taille entre l'environnement cible de SAML et notre environnement. Dans le premier, les coopérations se font entre des dizaines de sites, peut-être des centaines, alors que dans le second, on parle actuellement de 30000 à 100000 équipements dans le cadre de Netconf. Le facteur d'échelle est également présent en terme de nombre d'utilisateurs, c'est-à-dire de managers.

De telles tailles nécessitent la plupart du temps, soit une approche centralisée couplée à des structures hiérarchiques, soit une approche distribuée pour résister au facteur d'échelle. Des approches existent pour répondre à ces problèmes à plusieurs niveaux d'abstraction. Par exemple, RBAC permet le passage à l'échelle en terme d'utilisateurs. Le multicast permet le passage à l'échelle en terme de distribution, par exemple, si l'on groupe les équipements par catégories. Des approches comme RADIUS permettent de gérer plus facilement les identités et de décharger les équipements de cette charge. La grande difficulté consiste à prendre tous ces éléments en compte et à établir un modèle adapté.

Dans la suite de ce chapitre, nous abordons les problèmes de cohérence entre politique de contrôle d'accès et nous définissons les concepts de cohérence globale et cohérence locale.

1.3.2 Définition de la cohérence locale

Pour des besoins de flexibilité, il est courant pour un équipement réseau d'avoir plusieurs interfaces de supervision. Chacune de ces interfaces dispose de ses propres modèles de sécurité.

Parallèlement, chaque interface permet de gérer un ensemble de données. Il est possible que de tels ensembles se recouvrent ce qui signifie clairement que deux interfaces de gestion permettent de gérer au moins un même paramètre de configuration. C'est le problème de *non-isolation* des données de configuration. C'est dans ce cas précis qu'il est possible de voir apparaître des conflits entre les politiques de sécurité, c'est-à-dire qu'un attaquant peut profiter de la situation en utilisant alternativement la plateforme la plus faible du point de vue sécurité pour aboutir à ses fins. La propriété de *cohérence locale* est vérifiée lorsque les privilèges d'un utilisateur sont les mêmes quelque soit la plateforme de supervision utilisée. Le terme *locale* est à prendre dans le sens *sur un même équipement*.

Comment garantir la cohérence locale des politiques de sécurité ? C'est le problème que nous aborderons dans les chapitres suivants. Il est nécessaire de prendre en compte l'hétérogénéité des plateformes qui se traduit en fait par plusieurs niveaux d'hétérogénéité :

- modèle de données,
- granularité des opérations de configurations,
- modèle de contrôle d'accès.

1.3.3 Définition de la cohérence globale

Lorsque la gestion de politiques de sécurité est faite à la main, sans réelle réflexion et planification, il est fort probable que les politiques de sécurité ne soient pas cohérentes et introduisent des failles importantes de sécurité. C'est un phénomène connu notamment dans le cadre des pare-feux. Si plusieurs pare-feux successifs sont mis en œuvre, il faut que leurs règles de filtrage soient interdépendantes pour améliorer leur efficacité. Dans notre environnement, il est souvent question de classes de machines, par exemple les routeurs, les stations de travail, les pools de serveurs ou les imprimantes. Il est évident qu'un administrateur réseau est susceptible d'avoir les mêmes privilèges pour un ensemble de machines ayant des fonctions similaires.

En pratique, avec SNMPv3, chaque politique est bien souvent définie machine par machine ce qui devient gênant lorsque leur nombre croît. Mais il est possible de remédier à ce problème simplement en distribuant les configurations à grande échelle. Les plateformes de supervision sont plus ou moins aptes à mettre en place une telle distribution et cette aptitude dépend parfois plus de l'implantation elle-même et non du protocole de supervision.

Le problème adressé est donc de garantir la cohérence dans un réseau à grande échelle multi-plateforme, et un niveau de sécurité constant sur les machines d'un même *groupe* par un manager donné

1.4 Contributions

1.4.1 Assurer la cohérence globale du contrôle d'accès

La première contribution [24] est une proposition de pondération du modèle RBAC (Role-Based Access Control) pour s'adapter au modèle de sécurité hiérarchique de CLI. Dans un premier temps, l'idée consiste à convertir un rôle (tel qu'il est défini dans le modèle de contrôle d'accès basé sur les rôles) vers un niveau de sécurité. Puis, par généralisation, l'algorithme utilise l'approche *K_means clustering* pour rassembler plusieurs rôles dans un ensemble de niveaux de sécurité de taille contraint, ce qui est le cas en pratique. La difficulté est que la hiérarchie des rôles est multiple alors que la hiérarchie des niveaux de sécurité est simple. La pondération proposée s'applique tout d'abord à chaque permission puis est ensuite propagée sur les rôles. Cette pondération sert de paramètre pour la clusterisation.

La deuxième approche [21] propose d'associer le concept de groupe multicast avec la notion de rôle du modèle RBAC. L'idée est que, à chaque rôle est associée une clé d'authentification et une clé de chiffrement. Cette clé (par exemple, pour l'authentification) est mise à jour en utilisant un groupe multicast et un algorithme multicast de gestion des clés qui assure de très bonnes performances en cas de message *join* pour rejoindre le groupe ou *leave* pour quitter le groupe. Seuls les managers assignés à un rôle donné ont la connaissance de la clé associée. Le simple fait de chiffrer ou d'authentifier un message avec une telle clé assure le destinataire de l'appartenance du manager au rôle.

Nous avons également proposer dans [25] une externalisation partielle de la politique de sécurité des agents SNMP de façon à homogénéiser la politique mise en œuvre au sein d'un réseau. Cette approche repose sur le protocole et l'architecture RADIUS qui est responsable de la prise de décision sur les accès aux ressources de gestion. L'avantage de cette approche est de mettre à profit des infrastructures déjà présentes sur le réseau et dédiées aux mécanismes de sécurité.

1.4.2 Détection de conflits et assurer la cohérence locale

La troisième contribution est une formalisation mathématique des modèles de contrôle d'accès mis en place dans SNMPv3 (VACM), CLI, TR-069 et Netconf. Ensuite, pour chaque formalisation, une définition formelle de conversion vers le modèle RBAC est proposée. Ces algorithmes conservent presque complètement les propriétés initiales de la politique de contrôle d'accès. Cette conversion a été implantée et permet d'obtenir des politiques RBAC très proches syntaxiquement. L'étape suivante a été de proposer des algorithmes de comparaison de politiques RBAC en simulant toutes les activations de rôles possibles, pour essayer de créer des situations où il ne serait pas possible d'acquiescer un ensemble de privilèges identiques. Nous nous sommes heurtés, dans cette dernière étape, à la difficulté de comparaison des modèles de données, nécessaire à la comparaison de permissions.

La quatrième contribution propose un modèle de contrôle d'accès RBAC dans Netconf. Cette nouvelle fonctionnalité (*capability*) étend le protocole Netconf avec de nouvelles opérations d'activation et de désactivation de rôles. Dans cette approche, l'agent stocke une politique RBAC qu'il utilise pour filtrer les résultats des opérations get-config et également donner le feu vert ou non pour les opérations d'écriture, de type edit-config ou copy-config. L'adressage des ressources repose sur XPath.

1.4.3 Implantations et évaluation

Une extension de passerelle XML/SNMP est proposée dans [20] pour garantir un continuum de sécurité depuis les managers XML [12] jusqu'aux agents SNMP. Une politique RBAC est stockée au niveau de la passerelle et est convertie et déployée dynamiquement vers les tables VACM des agents SNMP. Le premier avantage est qu'il n'est plus nécessaire de configurer les tables VACM des agents SNMP. Le second avantage de cette approche est qu'il reste possible d'accéder aux agents directement via SNMP.

Une implantation de Netconf [18, 19, 22] est réalisée pour évaluer les performances de notre modèle de contrôle d'accès et, d'une façon plus générale, fournir à la communauté de gestion de réseau un outil open source modulaire pour tester les fonctionnalités de Netconf et ses performances. Notre implantation fonctionne sous Linux et permet de gérer des routeurs RIP [42] et BGP [68], des serveurs Asterisk, les interfaces réseaux, et la politique de sécurité de l'agent

lui-même. D'autres projets d'extensions, internes à l'équipe ou externes, sont en cours de réalisation.

1.5 Plan de la thèse

1.5.1 Partie I : La sécurité et les plateformes de gestion de réseau

Dans la première partie, nous rappellerons d'une part les définitions des services de sécurité principaux et nous nous attarderons sur les modèles de contrôle d'accès avec en particulier celui basé sur les rôles. D'autre part, nous passerons en revue les principaux protocoles de supervision de réseau avec un regard focalisé sur leurs choix respectifs concernant les modèles de sécurité.

1.5.2 Partie II : Vers une double convergence des modèles et politiques de sécurité

La deuxième partie présente différentes propositions visant à améliorer les cohérences locales et globales des politiques de sécurité des protocoles de supervision de réseau. Nous avons développé deux directions : la première est une approche active qui, à partir d'une politique unique, déploie des politiques dérivées à travers le réseau. La deuxième approche consiste à vérifier les politiques déployées en les rapprochant vers un modèle commun puis en les comparant pour en extraire les conflits potentiels.

1.5.3 Partie III : Expérimentations et évaluations

La troisième partie présente nos expérimentations que ce soit pour les passerelles XML/SNMP sécurisées, l'extension de contrôle d'accès pour Netconf ou encore les tests de performance relatifs à la sécurité dans notre implantation du protocole de configuration Netconf.

tel-00134670, version 1 - 4 Mar 2007

Première partie

La sécurité et les plateformes de gestion
de réseau

Les architectures, modèles et technologies de sécurité

Sommaire

2.1	Modèle de menaces	13
2.2	Les services de sécurité	14
2.2.1	Authentification	14
2.2.2	Intégrité	14
2.2.3	Confidentialité	14
2.2.4	Contrôle d'accès	14
2.3	Les modèles de contrôle d'accès	14
2.3.1	Contrôle d'accès obligatoire (MAC)	14
2.3.2	Contrôle d'accès discrétionnaire (DAC)	15
2.3.3	Contrôle d'accès basé sur les rôles (RBAC)	15
2.4	Synthèse	21

2.1 Modèle de menaces

Avant d'étudier ou de proposer des modèles de sécurité pour la gestion de réseau, il est indispensable de définir un modèle de menaces (ou analyse des menaces).

L'interception de paquets permet à un attaquant d'écouter les échanges (eavesdropping), ou même de détourner les messages (hijacking) en se faisant passer pour le vrai destinataire. Dans notre cas, on peut imaginer qu'un pirate observe les messages de configuration pour obtenir des informations sensibles ou intercepte les messages de remontées d'alertes pour empêcher les administrateurs de détecter un problème. Ces problèmes peuvent être résolus en partie par des services d'authentification et de confidentialité. Dans le même ordre d'idée, un pirate peut intercepter un message, le modifier en créant de fausses valeurs et le réinjecter vers le véritable destinataire. Cette menace est liée à l'intégrité du message.

Un pirate peut également enregistrer des messages et les rejouer plus tard. Les conséquences varient en fonction du type du message. Par exemple, si il s'agit d'une simple requête pour récupérer une donnée statistique, les conséquences sont minimales. Si au contraire, il s'agit d'un message qui commande le redémarrage de l'équipement ou qui modifie la configuration de l'équipement, les conséquences sont beaucoup plus sérieuses. Il est donc nécessaire de mettre en place un mécanisme qui empêche de rejouer des messages et limiter ainsi les risques de déni de service.

Les menaces portant sur une plateforme de gestion sont communes à la plupart des systèmes distribués multi-clients et multi-serveurs. La complexité inhérente à la tâche de supervision, la dimension importante des réseaux actuels et le fait d'avoir plusieurs clients (managers) sous-entend que ceux-ci peuvent avoir différentes fonctions et activités et donc il est nécessaire de partitionner l'espace des privilèges en créant des règles d'accès aux ressources.

2.2 Les services de sécurité

2.2.1 Authentification

L'authentification est le mécanisme par lequel une entité (humain ou machine) peut s'assurer de l'identité d'une autre entité. En général, une entité s'authentifie en prouvant sa connaissance d'un secret. Le mécanisme d'authentification le plus connu est le mot de passe. Plus précisément, l'identification est assurée par le login et l'authentification par le mot de passe.

2.2.2 Intégrité

L'intégrité permet de s'assurer qu'une donnée n'a pas été modifiée pendant son transit. En particulier, cette donnée n'a pas été interceptée par un tiers, modifiée puis réémise vers le destinataire. L'intégrité est souvent mise en œuvre à l'aide d'algorithmes de hachage. Enfin, la signature électronique permet d'assurer à la fois l'intégrité et l'authentification du message en combinant un algorithme de hachage et de chiffrement. La signature électronique repose sur les architectures à clé publique.

2.2.3 Confidentialité

La confidentialité est le service qui assure que seules les personnes autorisées peuvent lire un document. Pour assurer la confidentialité, on chiffre la donnée avec une clé que seules les personnes autorisées connaissent. Seuls les possesseurs de la clé sont capables de lire la donnée. La confidentialité est nécessaire pour les communications personnelles (email, consultation de comptes bancaires), d'entreprises (secrets industriels), militaires (un groupe militaire ne souhaite pas que son ennemi connaisse sa stratégie ou ses positions). D'une manière générale, la confidentialité permet de protéger un secret.

2.2.4 Contrôle d'accès

Le contrôle d'accès définit l'ensemble des opérations qu'une entité du système peut réaliser ou non sur un ensemble de ressources. Les informations décrivant les droits d'accès peuvent être intégrées au système d'information (Unix) ou stockées à part. Pour faciliter l'administration des règles d'accès, il est commode de grouper les utilisateurs et/ou les ressources. Il existe plusieurs modèles de contrôle d'accès, à niveau de sécurité, hiérarchiques ou non, avec délégation ou non, pouvant contenir des contraintes spatio-temporelles et de séparation des pouvoirs.

2.3 Les modèles de contrôle d'accès

2.3.1 Contrôle d'accès obligatoire (MAC)

Le *Mandatory Access control (MAC)* [9] ou contrôle d'accès obligatoire est un modèle de contrôle d'accès dans lequel les données se voient affecter des niveaux hiérarchiques de sécurité.

Par exemple, *public*, *confidentiel*, *secret*, *top secret* est un ensemble de niveaux de sécurité ordonné et hiérarchique qui organise l'information en fonction de son caractère sensible. Parallèlement, chaque utilisateur, en fonction de son grade si l'on se place par exemple dans un contexte militaire, a accès au maximum à un certain niveau et à tous les niveaux inférieurs. Par exemple, un commandant a accès aux informations classées niveau *secret*, et par héritage, aux informations classées *confidentiel* et *public*. Ce modèle est très simple à implémenter et à maintenir car il suffit de comparer le niveau de privilèges de l'utilisateur à celui de l'information pour accorder l'accès ou non. Les règles d'accès MAC sont sous la responsabilité d'un administrateur.

2.3.2 Contrôle d'accès discrétionnaire (DAC)

Dans le modèle de contrôle d'accès discrétionnaire (ou *Discretionary Access Control (DAC)* [9]), chaque utilisateur peut créer une ressource et il devient alors le propriétaire de cette ressource. Le propriétaire a tous les droits sur cette ressource. En particulier, il est habilité à transmettre des droits d'accès à la ressource à d'autres utilisateurs via un mécanisme de délégation. C'est ce modèle qui est utilisé en environnement Linux, même si on voit apparaître les modèles MAC et RBAC avec SELinux. DAC est moins centralisé que MAC dans la mesure où chaque utilisateur est responsable des données qu'il crée.

2.3.3 Contrôle d'accès basé sur les rôles (RBAC)

Introduction

L'hétérogénéité des ressources distribuées sur le réseau rend particulièrement difficile la gestion des accès [75, 58, 59]. Au sein d'un système Unix ou Linux, on sait gérer les droits d'accès en définissant des groupes d'utilisateurs correspondant à une catégorie d'utilisateurs, des droits d'accès sur les fichiers et les répertoires en fonction du demandeur. Chaque ressource appartient à un groupe et à un utilisateur. Droits d'accès en lecture, en écriture et en exécution sont définis pour le propriétaire de la ressource, pour les membres du groupe et pour les autres utilisateurs.

On peut voir RBAC [33, 70] comme une généralisation de ce modèle destinée à s'appliquer à tout type de produits nécessitant un contrôle d'accès. La disparité dans les systèmes de contrôle d'accès ont conduit le NIST¹ à définir une architecture générale pour le contrôle d'accès et un ensemble de fonctionnalités permettant de faciliter la gestion de ce système.

Plus les ressources et la gestion des droits d'accès sont hétérogènes, plus le risque d'introduire des risques de sécurité est grand. RBAC permet de structurer de façon hiérarchique l'ensemble des utilisateurs agissant sur le système et de gérer les accès de façon centralisée. Il en résulte un passage à l'échelle facilité de son administration et donc une sécurité accrue.

Pour simplifier le travail de l'administrateur et des concepteurs de systèmes de contrôle d'accès, RBAC définit l'ensemble des fonctions standards que doit implanter tout système de gestion des droits d'accès. L'objectif est de rendre communes les fonctionnalités de base de tels systèmes.

L'architecture

RBAC définit et organise les différentes composantes nécessaires à l'implantation d'un système de contrôle d'accès. Comme illustré à la figure 2.1, RBAC se subdivise en plusieurs niveaux : Core RBAC, RBAC hiérarchique, contraintes statiques et dynamiques de séparation des pouvoirs [52]. Le premier niveau est le système de base. Il contient l'architecture principale. Les trois

¹National Institute of Standards and Technology

autres niveaux affinent le premier niveau en autorisant la hiérarchie des rôles, le contrôle d'accès statique et dynamique.

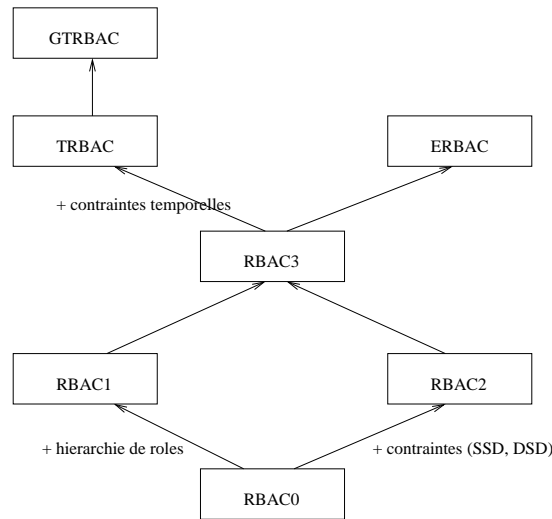


FIG. 2.1 – Les variantes du modèle RBAC

Core RBAC : ($RBAC_0$)

La première de ces composantes est l'utilisateur (USER) qui interagit avec le système. Ensuite, RBAC définit le concept de rôle qui décrit les différents rôles présents au sein d'une organisation : chercheur, maître de conférences, assistante de projet, chef de projet, dans le cas d'un laboratoire de recherche. RBAC définit aussi des opérations qui peuvent varier en fonction du système administré. Le plus souvent, ces opérations sont la lecture, l'écriture et l'exécution. Core RBAC décrit ensuite les objets (ressources) du système. Là encore, leur type et leur adressage peut varier en fonction des systèmes. Enfin, RBAC définit les sessions. C'est dans le cadre d'une session qu'un utilisateur peut acquérir des privilèges en activant ou désactivant des rôles. Une session se termine en général lorsque l'utilisateur demande la fermeture de la session.

Voyons maintenant comment RBAC organise ces composantes en définissant des associations entre elles. Un utilisateur est affecté à un ou plusieurs rôles. Par exemple, un utilisateur chef de projet peut être aussi un maître de conférence. Un utilisateur est associé à plusieurs sessions et une session correspond à un utilisateur unique. Une session peut réaliser plusieurs rôles qui correspondent généralement à un sous-ensemble des rôles de l'utilisateur de cette session. Les rôles sont associés à un certain nombre de permissions. RBAC définit une permission comme un couple (opération, objet). Donc un rôle peut réaliser des opérations sur des objets.

RBAC n'associe pas un utilisateur à des permissions, ceci pour ne pas définir les droits d'accès pour chaque utilisateur séparément. Gérer l'accès par utilisateur est envisageable pour une structure très petite mais à partir d'un certain nombre d'utilisateurs, c'est impossible. Ainsi, lorsqu'un nouvel utilisateur se présente, le travail de l'administrateur consiste simplement à affecter cet utilisateur à un rôle.

RBAC hiérarchique : ($RBAC_1$)

La variante hiérarchique de RBAC permet de classer les rôles suivant une hiérarchie. On peut voir RBAC hiérarchique comme un mécanisme d'héritage. Un rôle est toujours associé à un ensemble de permissions mais un rôle hérite en plus d'un ou plusieurs autres rôles, et donc de toutes leurs permissions associées. Par exemple, un professeur d'université est au-dessus d'un maître de conférences dans la hiérarchie des enseignants donc il hérite des privilèges de tout maître de conférences et il a des permissions supplémentaires.

On peut affiner la hiérarchie des rôles en la divisant en deux catégories :

- hiérarchie de rôles générale : l'héritage multiple est permis,
- hiérarchie de rôles limitée : l'héritage multiple n'est pas permis.

Une hiérarchie de rôles limitée contraint le modèle mais facilite l'administration de la politique de contrôle d'accès.

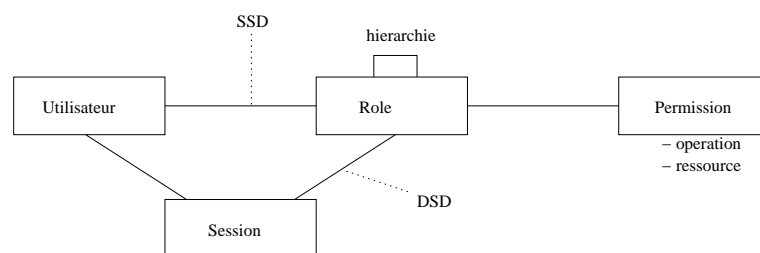


FIG. 2.2 – RBAC 3 : hiérarchie et contraintes (DSD et SSD).

RBAC contraint : ($RBAC_2$)

La séparation statique des pouvoirs (Static Separation of Duty, SSD) permet d'ajouter des contraintes sur l'association utilisateur-rôle. Il peut arriver qu'on souhaite restreindre le nombre de rôles d'un utilisateur. Par exemple, un utilisateur ne doit pas avoir plus de n rôles dans un sous-ensemble de rôles.

La séparation dynamique des pouvoirs (Dynamic Separation of Duty, DSD) a également pour objectif de placer des contraintes sur le nombre de rôles que peut endosser un utilisateur. La différence avec SSD est que la contrainte se situe ici sur le nombre de rôles endossables à l'exécution. Un utilisateur peut par exemple endosser deux rôles mais pas en même temps (ou dans une même session). Un chercheur ne peut pas être à la fois auteur d'un article et rapporteur d'un article même si il est possible qu'il soit auteur et rapporteur (statiquement) pour des articles différents.

Les fonctionnalités

Pour chacun des modules de RBAC, le NIST a défini un certain nombre d'opérations. Celles-ci peuvent être divisées en trois catégories (ici, les opérations pour Core RBAC) :

- les fonctions d'administration : créer, détruire, modifier des rôles, des utilisateurs, lier les utilisateurs aux rôles, lier les permissions aux rôles)
- les fonctions à l'exécution : ouvrir une session, activer un rôle, vérifier un accès
- les fonctions de monitoring : consulter les associations utilisateur-rôle, rôle-permission, rôle-session, ...

Pour les autres modules RBAC, on retrouve sensiblement les mêmes opérations mais avec des nuances spécifiques à la fonctionnalité du module.

Des extensions relatives au modèle RBAC

Le modèle RBAC a suscité un grand intérêt et a déjà été déployé dans nombre d'applications distribuées. Cependant, il subsiste quelques manques dans ce modèle, notamment au niveau des contraintes temporelles.

TRBAC Dans les systèmes opérationnels, il est très fréquent que les droits d'accès d'un utilisateur varient selon une contrainte de temps. Par exemple, un employé peut réaliser une opération seulement pendant ses heures de travail. A l'origine, le modèle RBAC ne permet pas ce type de contrainte.

Temporal-RBAC (TRBAC [5]) répond à ce manque du modèle RBAC en apportant une dimension temporelle. Cette contrainte supplémentaire porte sur la disponibilité d'un rôle (role enabling) pour un utilisateur, mais est différente de la contrainte de séparation statique des pouvoirs. La notion de *role enabling* est différente de l'assignation d'un rôle et d'un utilisateur et différente de l'activation d'un rôle. Pour rappel, la SSD limite les appartenances d'un utilisateur à des rôles les uns par rapport aux autres. C'est-à-dire que deux rôles ne peuvent pas être assignés à la même personne. Cependant, les deux types de contraintes portent sur la même relation, à savoir utilisateur-rôle. Ces contraintes temporelles s'expriment en terme d'intervalle de temps durant lequel la contrainte doit être considérée et en terme de période à l'intérieur de l'intervalle.

TRBAC introduit la notion de *triggers*, qui correspond à des actions exécutées automatiquement en fonction du temps. Ces actions peuvent être la mise à disposition d'un rôle, c'est-à-dire la possibilité d'activer un rôle (role enabling), ou, au contraire, l'interdiction d'activer un rôle (role disabling) lors d'un intervalle de temps. Lorsque de telles règles sont spécifiées, il est possible d'être confronté à des conflits portant sur l'application de règles contradictoires. C'est pourquoi un système de priorité a été proposé dans [5] pour résoudre ces conflits : à chaque contrainte est donnée une priorité.

GTRBAC Le modèle *Generalized Temporal RBAC* (GTRBAC [53]) généralise TRBAC en spécifiant plus largement les contraintes temporelles. Les points suivants donnent la spécification des contraintes et événements :

- Contraintes temporelles sur la disponibilité ou non d'un rôle. Une telle contrainte spécifie un intervalle global de validité et un ensemble de sous-périodes dans cette intervalle pour lesquelles le rôle est mis à disposition,
- Contraintes temporelles sur les assignations utilisateur/rôle et permission/rôle. C'est le même type de contrainte que le précédent, sauf que l'événement généré est une assignation ou désassignation de rôle,
- Contraintes d'activation : portent sur l'activation de rôle, par exemple la durée totale pendant laquelle un rôle peut être activé,
- Événements à l'exécution : permettent de prendre des mesures dynamiquement comme l'activation de rôles après un événement,
- Expression d'applicabilité de contrainte,
- Triggers : actions prises en réponse à un événement. Par exemple, *[enable R1 -> enable R2]* signifie qu'un événement, ici, une demande de mise à disposition d'un rôle, génère un autre événement qui demande la mise à disposition d'un second rôle automatiquement.

Cette approche a été instanciée dans [7] qui propose la spécification d'un langage XML exprimant le modèle GTRBAC et l'architecture nécessaire à l'évaluation dans un environnement concret de ce modèle de contrôle d'accès. Par ailleurs, un modèle d'administration décentralisé de cette plateforme a été proposée dans [8]. L'administration a été décentralisée pour répondre au problème de passage à l'échelle en terme d'administration de politiques, de la quantité de ressources à protéger et des domaines administratifs d'applicabilité de la politique.

Le modèle *Enterprise RBAC* Le modèle *Enterprise RBAC* (ERBAC [48, 49]) a été proposé pour répondre à la problématique du contrôle d'accès à l'échelle de l'entreprise et non plus d'un système. On peut voir ERBAC comme un sur-modèle qui se propage sur des systèmes cibles. Ces systèmes cibles ont leur propre mécanisme de contrôle d'accès mais ceux-ci peuvent varier d'un système à l'autre. ERBAC est une couche d'abstraction qui permet de masquer les systèmes concrets (ou systèmes cibles).

Lorsqu'un compte utilisateur est créé dans le modèle ERBAC, des comptes utilisateurs sont automatiquement créés sur un certain nombre de systèmes cibles concrets en fonction des spécifications. De plus lorsqu'un utilisateur est assigné à un rôle, l'utilisateur ne reçoit les permissions que pour ces systèmes cibles. Cette automatisation permet un gain de maintenance important des politiques de sécurité.

Ce travail est proche de celui présenté dans cette thèse mais a cependant des contraintes différentes. Les systèmes cibles ne partagent pas les mêmes ressources, contrairement à notre cas d'étude, où les ressources sont parfois partagées par différentes plateformes de gestion. De plus, les ressources ne sont pas décrites de façon générique, c'est-à-dire qu'une permission s'exprime en fonction du système cible et non d'un modèle générique de données. Cela a un avantage lors du déploiement mais coûte extrêmement cher en maintenance car il faut maintenir les permissions par système cible.

OrBAC OrBAC [47], un modèle de contrôle d'accès basé sur les organisations, part du principe qu'il manque certains concepts dans les modèles existants comme le contexte dans lequel une action est effectuée, ou encore la notion d'organisation dans laquelle plusieurs politiques peuvent coexister. OrBAC conserve la notion de rôle du modèle RBAC mais ajoute parallèlement aux permissions les notions d'obligation, d'interdiction et de recommandation. Ces concepts étendent le pouvoir d'expression des politiques de contrôle d'accès. OrBAC distingue les deux niveaux d'abstraction suivants :

Niveau concret Comme son nom l'indique, ce niveau décrit les composants concrets du modèle : Sujet, action, objet et permission,

Niveau abstrait Ce niveau définit les rôles comme un ensemble de sujets, les activités comme un ensemble d'actions, et les vues comme un ensemble d'objets.

Le grand avantage du niveau abstrait est qu'il donne une vision macroscopique de la politique de contrôle d'accès. En effet, on raisonne non plus sur des composants dont la granularité est trop fine pour pouvoir être gérée à grand échelle, mais plutôt en terme de concepts de haut niveau d'abstraction (rôle, activité). De ce point de vue, OrBAC est proche de ERBAC car il tente d'établir un sur-modèle qui abstrait le système concret. Par ailleurs, OrBAC permet l'expression d'interdictions ce qui n'est pas le cas de RBAC, car les interdictions sont susceptibles d'introduire des conflits. OrBAC cherche à résoudre les conflits d'abord automatiquement en priorisant les permissions et les interdictions, puis manuellement si le conflit persiste.

Comme RBAC, OrBAC autorise la hiérarchie des rôles mais OrBAC va plus loin en distinguant deux types de hiérarchies [26]. D'une part, la hiérarchie organisationnelle établit une

relation de supériorité hiérarchique entre deux rôles. Dans ce cas, le rôle du supérieur hiérarchique hérite des droits du second rôle. D'autre part, la hiérarchie induite par généralisation est similaire à celle utilisée par les langages orientés objets. Un rôle qui spécialise un autre rôle hérite de toutes ses permissions, comme une classe hériterait des attributs et des méthodes d'une autre classe. La figure 2.3 illustre un scénario définissant plusieurs rôles de supervision. Elle introduit tout d'abord un rôle général associé à l'activité de routage, *RoutingManager*, puis plusieurs autres rôles plus spécifiques à un protocole de routage donné : *RIPManager* pour le protocole RIP, *OSPFManager* pour le protocole OSPF, etc... C'est un exemple de hiérarchie par généralisation où le rôle *RIPManager* est une spécialisation du rôle *RoutingManager*. Le scénario propose également un rôle de gestionnaire du firmware pour le routeur, *FirmwareManager*. Dans ce cas précis, on a fait l'hypothèse suivante : le rôle *RoutingManager* est supérieur au rôle *FirmwareManager* hiérarchiquement. C'est un exemple de hiérarchie organisationnelle. Cette distinction faite par OrBAC entre types de hiérarchies est importante car elle s'exprime typiquement dans le sens inverse dans un schéma.

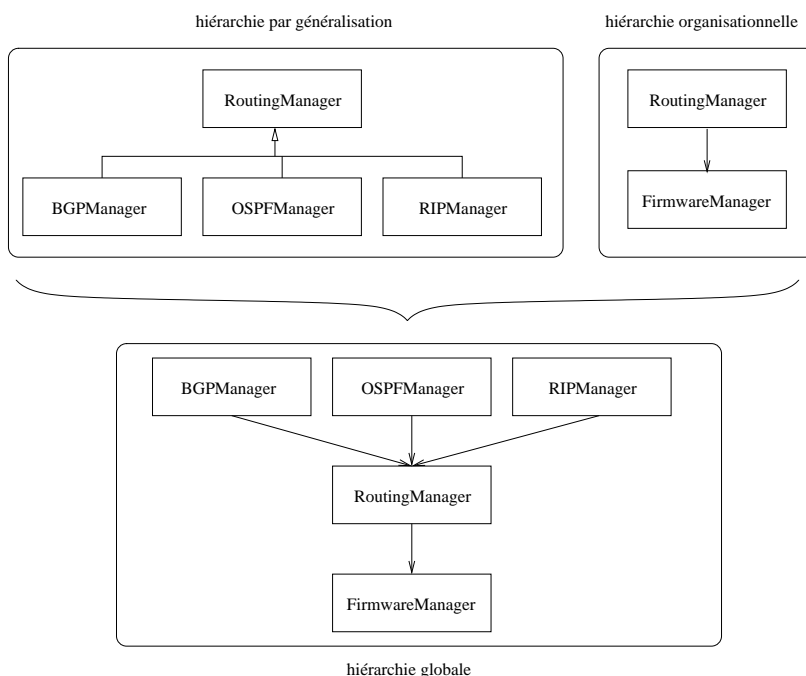


FIG. 2.3 – Les types de hiérarchies

OrBAC définit plusieurs types de contextes [27] qui font varier les permissions en fonction de la situation courante. Un contexte peut être temporel, spatial, déclaré par l'utilisateur, prérequis ou provisionnel. Par exemple, un contexte temporel définit à quel instant un privilège est valide ou non. Un contexte spatial définit la situation spatiale dans laquelle les privilèges sont valides (position géographique ou dans un réseau). D'autres travaux [63] ont développé des approches pour faciliter l'expression de contraintes de contexte dans un environnement RBAC.

La délégation dans OrBAC [28] peut également prendre différentes formes selon qu'elle est temporaire ou permanente, partielle ou totale (tout ou partie des droits sont délégués), monotone ou non-monotone indiquant si les droits sont conservés par le sujet qui délègue. D'autres formes de délégation sont définies par OrBAC et permettent de décrire finement le concept.

2.4 Synthèse

Dans ce chapitre, nous avons présenté le modèle de menaces dans le cadre de la gestion de réseau, puis nous avons rappelé brièvement les définitions des différents services de sécurité que sont l'intégrité, l'authentification, la confidentialité et le contrôle d'accès.

Modèle de contrôle d'accès	Concept central	Hiérarchique	Délégation	Contrainte Contexte	Contrainte SSD/DSD
MAC	niveaux	oui	non	non	non
DAC	propriété	non	oui	non	non
RBAC	rôle	RBAC1	non	non	RBAC2
TRBAC	temporel	RBAC1	non	temporelle	RBAC2
GTRBAC	temporel généralisé	RBAC1	non	temporelle	RBAC2
ERBAC	entreprise	RBAC1	non	non	RBAC2
OrBAC	organisation	oui	oui	temporel spatial ...	non

TAB. 2.1 – Synthèse des modèles de contrôle d'accès

Dans la suite, nous reviendrons sur chacun de ces services de sécurité mais nous nous intéresserons plus en détail aux problèmes liés au contrôle d'accès. Ce chapitre a également fait l'objet d'une présentation des principaux modèles de contrôle d'accès dont, plus particulièrement, le modèle basé sur les rôles, qui servira de fil conducteur dans la suite de cette thèse.

Le Tableau 2.1 résume les propriétés de plusieurs modèles de contrôle d'accès. Dans les grands systèmes de supervision, toutes les propriétés citées peuvent avoir un intérêt selon les scénarii envisagés. Les modèles les plus aptes à servir nos besoins sont les modèles ERBAC et OrBAC. Chacun vise à abstraire les politiques de contrôle d'accès vers des modèles macroscopiques. ERBAC est très intéressant dans notre cadre d'étude car son but est d'être déployé vers des systèmes cibles par forcément compatibles entre eux. L'environnement très distribué et hétérogène de la supervision des réseaux est un candidat parfait pour ce type d'approche. OrBAC apporte des concepts supplémentaires intéressants mais pas forcément transposables vers nos systèmes cibles que sont CLI, SNMPv3, TR-069 ou Netconf. Dans plusieurs des modèles proposés dans cette thèse, nous choisissons donc une approche de type ERBAC, c'est-à-dire se basant sur RBAC et avec comme systèmes cibles ceux précédemment cités.

Comme XML semble se profiler comme un candidat potentiel des futures approches de supervision de réseau, le lecteur est invité à se reporter aux annexes pour une présentation des modèles et technologies XML émergents relatifs à la sécurité. Nous y présentons notamment les travaux liés à la sécurité menés dans le cadre des web services, puis du langage d'expression de politique de contrôle d'accès XACML, et également du protocole SAML qui est un protocole de fédération des identités pour les services de l'Internet et qui sert à l'authentification unique (Single Sign On).

La sécurité des plateformes de gestion : les approches et leurs limites

Sommaire

3.1	Couplage fort entre plateformes de supervision et mécanismes de sécurité	23
3.1.1	L'évolution SNMP : v1,v2,v3	23
3.2	Couplage faible	31
3.2.1	L'interface en ligne de commandes CLI	31
3.2.2	Le protocole de configuration Netconf	33
3.2.3	Un protocole destiné aux passerelles résidentielles : TR-069	35
3.2.4	Un exemple d'outil d'administration avec interface web : Webmin	37
3.3	Synthèse	38
3.3.1	Le groupe ISMS	40

3.1 Couplage fort entre plateformes de supervision et mécanismes de sécurité

3.1.1 L'évolution SNMP : v1,v2,v3

SNMP (Simple Network Management Protocol) est l'un des protocoles les plus utilisés pour la gestion (management, monitoring) des réseaux. Il est basé sur le principe du manager et de l'agent. Un manager a sous sa responsabilité un certain nombre d'agents qu'il interroge via des requêtes SNMP. Un agent peut aussi communiquer de façon asynchrone avec son manager par un mécanisme de notifications. Un agent maintient une base de données appelée MIB (Management Information Base) qui stocke deux grandes classes de données : la première correspond aux données d'état de la machine (nombre de paquets reçus sur une interface, ...); la seconde correspond à la configuration de la machine (politique de sécurité ou de routage par exemple). Les données de la MIB sont modélisées de façon arborescente et peuvent être adressées par un OID (Object Identifier) qui correspond au chemin depuis la racine de l'arborescence des données de gestion jusqu'à l'objet, chaque nœud étant marqué par un identifiant (nombre et nom). 1.5.3.2.4 est un OID possible.

Les menaces et besoins de sécurité

La hausse continue des attaques par le réseau a fait apparaître le besoin de sécuriser le protocole SNMP. L'IETF² a donc proposé une nouvelle version (SNMPv3, voir [71, 41, 14, 56, 67, 66]) qui vise à apporter des outils pour sécuriser les échanges SNMP. Deux des objectifs principaux étaient de conserver un protocole simple d'utilisation et de rester compatible avec les anciennes versions de SNMP (v1 et v2). SNMPv3 apporte une nouvelle architecture qui répond à plusieurs besoins de sécurité correspondants à des menaces identifiées :

- Modification de l'information (intégrité),
- Usurpation d'identité (authentification),
- Modification de flux (rejeu),
- Affichage des données (confidentialité).

Dans le cadre de la gestion de réseau, la confidentialité est aussi importante que l'intégrité et l'authentification dans la mesure où les données échangées sont sensibles. Le besoin d'authentification est fort car on veut se protéger d'un attaquant qui se ferait passer pour un manager et qui reconfigurerait un équipement au détriment des utilisateurs. Le contrôle d'accès est également une des priorités. Certaines données ne doivent être accédées (que ce soit en lecture ou en écriture) que par des utilisateurs dûment autorisés.

Architecture

L'architecture fonctionnelle de SNMPv3 a été profondément remaniée par rapport à SNMPv2c. Deux modules de sécurité y ont été ajoutés : l'un pour l'authentification, l'intégrité et le chiffrement, l'autre pour le contrôle d'accès. La figure 3.1 illustre la nouvelle architecture fonctionnelle de SNMPv3. Le dispatcher y joue le rôle de chef d'orchestre pour les messages entrant et sortant. Il se divise en trois blocs principaux :

- *Transport Mapping* : responsable des interactions avec la couche inférieure (UDP),
- *PDU Dispatch* : interface fonctionnelle avec la couche supérieure (SNMP applications),
- *Message Dispatch* : en charge de la distribution des messages vers le sous-système de traitement des messages V1, V2 ou V3 selon la version annoncée par le message.

Plusieurs sous-systèmes de sécurité peuvent être implantés au sein de chacun des modules de sécurité. L'USM [10] est une implantation par défaut fournie par l'IETF. Elle permet de chiffrer, d'authentifier et d'éviter le rejeu pour chaque message traité. De même, VACM [79] est une implantation par défaut fournie par l'IETF. VACM permet de contrôler l'accès aux données stockées dans les MIB(s).

Chaque message SNMP inclut un champ `MsgSecurityModel` qui précise le modèle de sécurité utilisé : USM ou autre. Ceci permet au dispatcher de diriger les messages vers le module adéquat.

Format des messages

La figure 3.2 présente la structure d'un message SNMPv3. La principale caractéristique est que les mécanismes de sécurité sont profondément ancrés dans la structure du message. En particulier, les champs relatifs au contrôle d'intégrité, d'authentification et anti-rejeu sont inclus dans le message. C'est un choix qui est assez surprenant par rapport à l'objectif annoncé de modularité et qui est matérialisé par la possibilité de définir plusieurs modèles de sécurité. En réalité, ces champs sont dépendants du modèle de sécurité par défaut USM présenté par la suite, ce qui explique probablement qu'il n'y a jamais eu d'autres modèles proposés que celui-ci. Si,

²Internet Engineering Task Force, <http://www.ietf.org>

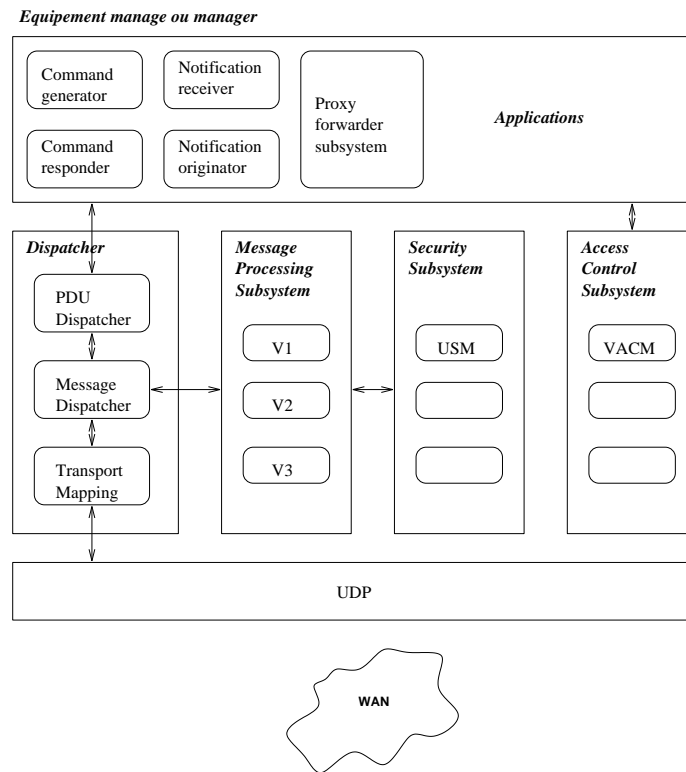


FIG. 3.1 – Architecture de SNMPv3

par hasard, un nouveau modèle de sécurité avait besoin de transmettre plus d'informations de sécurité, il ne le pourrait pas. Les différents champs du message sont détaillés par la suite.

Modèle de sécurité basé sur les utilisateurs (USM)

L'USM [10] (User-based Security Model) implante trois outils de sécurité. Le premier, *Privacy module*, permet de chiffrer les messages SNMP. Le second, *Authentication module* permet d'authentifier les messages SNMP. Enfin, le troisième, *Timeliness module*, est chargé de vérifier la fraîcheur d'un message (anti-rejeu, ...).

L'utilisation des deux premiers outils est optionnelle. Il existe un paramètre, *MsgFlags*, permettant de savoir quelles options de sécurité ont été choisies par l'émetteur. *MsgFlags* contient deux flags : *PrivacyFlag* et *Authflag*. Le premier indique si le message utilise le chiffrement, le second indique si le message utilise l'authentification. Ces champs sont indépendants du modèle de sécurité. Dans le cas le plus courant, ces informations permettent à l'USM de savoir s'il faut utiliser le Privacy Module et/ou l'Authentication Module.

Authentification et intégrité Pour s'authentifier, il faut générer et joindre un MAC au message à partir de la clé *authkey* localisée. Un destinataire peut retrouver quelle est la clé localisée à utiliser grâce à *msgAuthoritativeEngineID* et à *msgUserName*. Ces deux données identifient de façon unique une clé. Le champ *MsgAuthenticationParameter* contient le MAC du message.

Pour authentifier un message, un *non-authoritative engine* utilise la clé localisée correspondant à l'*authoritative engine* destinataire. Par convention, lors d'un message get-request, get-

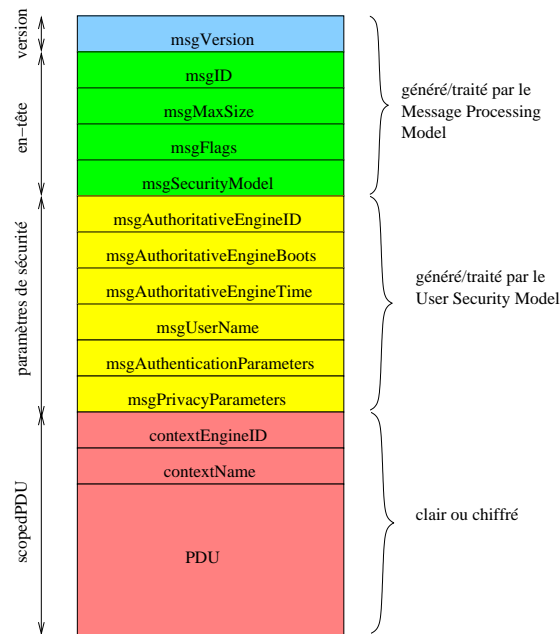


FIG. 3.2 – Structure du message SNMPv3

next-request, get-bulk-request, set-request ou inform, le récepteur est désigné *authoritative*. Pour tous les autres messages, le récepteur est désigné *non-authoritative*. Plus simplement, l'entité qui joue le rôle de manager est *non-authoritative*. Les entrées nécessaires à l'algorithme de génération de MAC sont données par la Figure 3.3. L'algorithme calcule K1 et K2, puis HMAC et finalement MAC, qui est ajouté au champ *msgAuthenticationParameters* du message SNMPv3 :

```

1 /* Paramètres en entrée */
2 ExtendedAuthKey = authkey + 44 octets de zéros
3 Ipad = 0x36 0x36 0x36 ... (64 fois)
4 /* Algorithme de calcul du MAC*/
5 Opad = 0x5c 0x5c 0x5c ... (64 fois)
6 K1 = extendedAuthKey XOR ipad
7 K2 = extendedAuthKey XOR opad
8 HMAC = H(K2, H(K1, wholeMessage))
9 MAC = 12 premiers octets de HMAC
    
```

FIG. 3.3 – Algorithme de calcul du MAC

Chiffrement Pour chiffrer le message, l'algorithme choisi est CBC-DES. Il faut tout d'abord générer la clé de chiffrement (*privkey*) avec MD5. Un destinataire retrouve la clé utilisée grâce aux champs *msgUserName* et à *msgAuthoritativeEngineID* du message. La figure 3.4 montre comment est obtenue la clé DES.

Pour chiffrer le message avec CBC (Cipher Block Chaining), on a besoin d'un vecteur d'initialisation (IV) de 64 bits. On calcule $IV = \text{preIV} \text{ XOR salt}$ avec :

- $\text{preIV} = 8$ derniers octets de *privkey*,

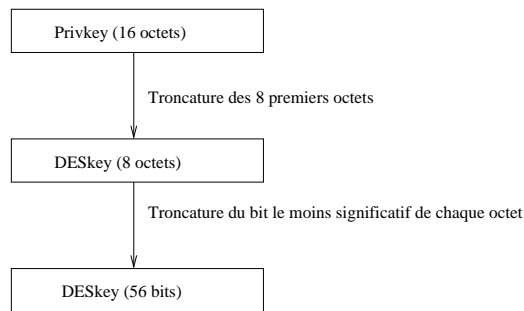


FIG. 3.4 – Génération de la clé DES

- salt = 4 premiers octets de snmpEngineBoots (Nombre de boots de l'équipement) + entier aléatoire (4 octets).

La figure 3.5 illustre la division de ScopedPDU en blocs B_i de 64 bits et le déroulement de l'algorithme CBS-DES :

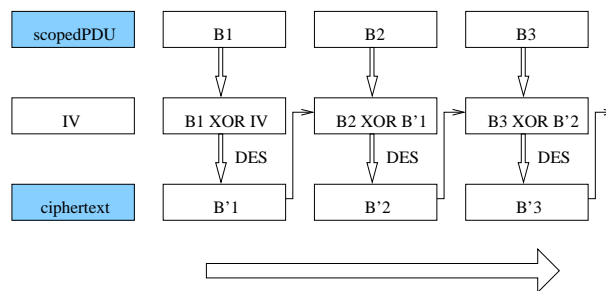


FIG. 3.5 – Chiffrement avec CBC-DES

Le paramètre *Salt* est ajouté au champ *msgPrivacyParameters* du message SNMPv3 pour que le récepteur puisse déchiffrer le message. Le déchiffrement consiste à déchiffrer B'_n avec l'algorithme DES qui est symétrique. La clé est la même pour le chiffrement et le déchiffrement. On obtient $B_n \text{ XOR } B'_{n-1} = \text{suite de bits correspondant à } B'_n \text{ déchiffré}$. Connaissant B'_{n-1} , on en déduit B_n . Pour le dernier (B_1), on utilise IV qu'on est capable de retrouver avec Salt.

Génération et distribution des clés Le manager génère deux clés : *privkey* et *authkey*. SNMPv3 propose l'utilisation de deux algorithmes de hachage (H) : MD5 et SHA [31]. Le processus de génération des clés est illustré par la figure 3.6. L'entrée nécessaire est le mot de passe du manager. La première étape consiste à répéter *passwd* jusqu'à l'obtention d'un mot de 1048 octets qu'on appelle *Digest0*. On applique ensuite une fonction de hachage sur *Digest0* (MD5 ou SHA) pour obtenir *Digest1* qui fait 16 ou 20 octets suivant l'algorithme choisi. L'étape suivante consiste à concaténer *EngineID* (celui de la machine courante) et *Digest1*. On réapplique le même algorithme de hachage que précédemment pour obtenir *Digest2*. *Digest2* est la clé finale.

Deux clés (*privKey* et *authKey*) sont requises pour assurer l'authentification et la confidentialité ; un manager doit retenir deux mots de passe qui seront utilisés par l'algorithme de génération de clés. Le manager dérive ensuite ses deux clés en clés localisées : une paire par agent géré. Une clé différente est utilisée pour chaque agent pour limiter les risques. La corruption d'un agent n'a pas de conséquences sur la sécurité des autres agents. Cela permet également à l'adminis-

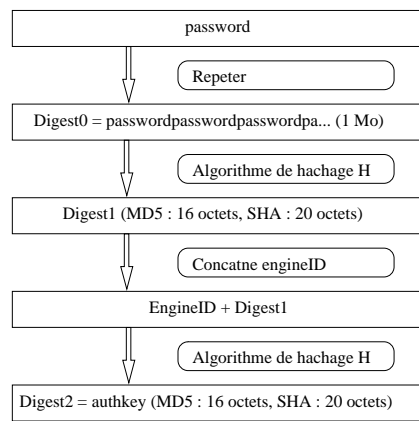


FIG. 3.6 – Génération des clés privKey et AuthKey

trateur de ne retenir que deux mots de passe pour l'ensemble des agents. Pour générer une clé localisée à destination de l'agent *authoritative* dont l'ID est *authoritativeSnmpEngineID*, on calcule $H(\text{digest1}, \text{authoritativeSnmpEngineID}, \text{digest1})$ où H est l'algorithme de hachage choisi et *digest1* est le même paramètre utilisé précédemment lors de la génération de la clé (*authkey* ou *privkey*). Ce calcul est fait une fois pour toute pour chaque agent géré, ce qui peut prendre un certain temps en fonction du nombre d'agents gérés.

La distribution des clés se fait "de façon sûre" à l'ensemble des agents gérés, c'est-à-dire par un mécanisme externe à SNMP. En pratique, la distribution est souvent faite à la main. Le mécanisme de distribution de clés localisées est illustré par la Figure 3.7.

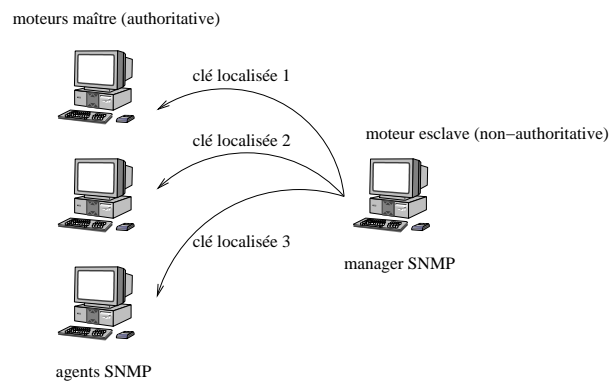


FIG. 3.7 – Distribution des clés localisées dans SNMPv3

Mécanisme anti-rejeu SNMPv3 rend possible la vérification de la cohérence dans l'arrivée des messages. Ce mécanisme permet d'éviter les problèmes de rejeu. Pour cela, chaque moteur *non-authoritative* maintient plusieurs variables concernant les moteur *authoritative* avec lesquels il communique. Ces variables définissent une fenêtre de temps qui délimite la période durant laquelle un message est considéré comme valable. Les variables nécessaires au mécanisme anti-rejeu sont :

- *AuthoritativeEngineBoots* : nombre de boots depuis l'installation de la machine. Cette valeur est maintenue par l'agent. Le manager garde une vue locale de cette valeur,
- *AuthoritativeEngineTime* : nombre de secondes depuis le dernier boot, incrémenté une fois par seconde par le moteur *non-authoritative*,
- *LatestReceivedEngineTime* : valeur du dernier *msgAuthoritativeEngineTime* reçu,
- *SnmpEngineID* : identifiant du moteur SNMP (authoritative ou non).

Prenons le cas d'un message émis par un agent et arrivant côté manager. Ce message est considéré comme invalide par rapport à la fenêtre temporelle l'une des deux conditions suivantes est vérifiée :

- $snmpEngineBoots = 2^{31} - 1$, valeur utilisée pour les cas d'erreurs,
- $connaissancelocaledesnmpEngineBoots > msgAuthoritativeEngineBoots$,
- $(connaissancelocaledesnmpEngineBoots = msgAuthoritativeEngineBoots)$ et $(msgAuthoritativeEngineTime < connaissancelocaledesnmpEngineBoots - 150)$.

Un manager a également une procédure de mise à jour des paramètres décrits précédemment, ce qui lui permet de rester synchronisé avec la fenêtre temporelle de l'agent.

Modèle de contrôle d'accès basé sur les vues (VACM)

VACM [79] (View-based Access Control Model) est le sous-système de contrôle d'accès proposé par défaut. Ce modèle a la charge de définir quels sont les droits des managers sur les MIBs par type d'opération (lecture, écriture, notification).

Les concepts du modèle Pour contrôler l'accès aux données, VACM définit un certain nombre de notions. Celles-ci permettent de faire le lien entre les ressources, des groupes d'utilisateurs, des droits d'accès et des niveaux de sécurité requis. Le premier concept introduit est le groupe : un groupe est un ensemble de couples (security model, security name). Un couple en entrée identifie un groupe de façon unique. La notion n'est pas très éloignée de la notion de rôles dans le modèle RBAC au sens où on tente d'agrèger des droits d'accès. Cependant, les groupes ne sont pas organisés de façon hiérarchique, ce qui rend difficile la gestion des groupes dans le cas où le nombre d'utilisateurs est grand.

Le deuxième concept est celui de Security level : il décrit les méthodes de sécurité déployés par l'utilisateur. Par exemple, *authentication and no privacy* indique que l'utilisateur souhaite utiliser l'authentification mais pas le chiffrement.

Le troisième concept est celui de contexte (voir [40, 72]) : un agent SNMP peut diviser l'information de gestion en plusieurs collections d'information appelées contextes qui permettent en particulier d'adresser plusieurs occurrences des mêmes types de données, c'est-à-dire ayant le même OID. La notion de contexte est très large : elle peut correspondre à plusieurs équipements physiques, à un seul équipement ou à une sous-partie d'un équipement. Mais un contexte définit un ensemble de données contenant au plus une occurrence d'une donnée.

Le quatrième concept est celui de famille d'arbres et de vues : une famille d'arbres correspond à un ensemble de sous-arbres (view subtrees), où chaque sous-arbre est un ensemble de données identifié par un OID couvrant cet ensemble. Un mécanisme de masque d'OID a été conçu pour adresser rapidement plusieurs branches. Par exemple, le couple (oid=1.3.6.1.2.1, masque=1.1.0.1.0.1) est équivalent à une notation XPath [16] (/1/6/*/1*/1) qui identifie un ensemble de nœuds. Chacun des nœuds sélectionnés est la racine d'une branche, l'ensemble de ces branches constituant finalement une famille d'arbres. Une vue est alors définie comme un ensemble de familles d'arbres, chaque famille étant incluse ou exclue de la vue.

<i>Security Model</i>	<i>Security Name</i>	<i>Group Name</i>
USM	Bob	SysAdmin
USM	Alice	NetAdmin

<i>View Name</i>	<i>Subtree</i>	<i>Mask</i>	<i>Type</i>
Mib II	1.3.6.1.2.1	11111111	Included
Network	1.3.6.1.2.1.6	11111111	Included
Network	1.3.6.1.2.1.2	11111111	Included
System	1.3.6.1.2.1.1	11111111	Included

<i>Group Name</i>	<i>Context Prefix</i>	<i>Security Model</i>	<i>Security Level</i>	<i>Context Match</i>	<i>ReadView Name</i>	<i>WriteView Name</i>	<i>NotifyView Name</i>
SysAdmin	""	USM	authNoPriv	exact	Mib II	System	None
NetAdmin	""	USM	authPriv	exact	Mib II	Network	None

FIG. 3.8 – Exemple de politique de contrôle d'accès avec VACM

Le dernier concept est celui de politique d'accès. Une politique d'accès lie un groupe, un contexte, un modèle de sécurité et un niveau de sécurité à trois vues : une correspondant à la vue accessible en lecture, une pour l'écriture et une pour le notify. Il se peut aussi qu'aucune vue ne corresponde à ces quatre entrées.

La figure 3.8 est une instantiation très simple d'une telle politique. La table *SecurityToGroupTable* définit deux groupes *SysAdmin* et *NetAdmin*, contenant chacun une seule entité, Bob et Alice. Ces deux entités appartiennent au groupe à condition qu'elles utilisent le module *USM*. La table *AccessTable* donne les ressources accessibles en fonction des différents paramètres. Par exemple, Le groupe *SysAdmin* a le privilège de modifier les ressources de la vue *System* à condition d'utiliser le bon modèle de sécurité (ici, *USM*) avec le niveau de sécurité adéquat (ici, authentication sans confidentialité). La table *ViewTreeFamilyTable* définit entre autres la vue *System* composée du sous-arbre ayant pour racine le nœud *1.3.6.1.2.1.1*.

Le processus de contrôle d'accès Les étapes de contrôles d'accès sont les suivantes. Tout d'abord, il s'agit de déterminer dans quel groupe se placer en fonction des paramètres *msgSecurityModel* et *msgUserName*. Puis, il faut vérifier l'existence du contexte (*contextName*), déterminer les vues accessibles pour ce groupe, ce contexte (*contextName*), ce modèle de sécurité (*msgSecurityModel*) et ce niveau de sécurité (*msgFlags*). Après ce traitement, trois noms de vues sont obtenues : une première vue en lecture, une deuxième en écriture et enfin une vue pour les notifications. L'étape suivante consiste à déterminer quel est l'accès demandé (*get-Request*, ... : *read*, *write*, *notify*) et à sélectionner l'une des trois vues accessibles. Lors de la dernière étape, l'agent vérifie que l'objet sur lequel porte la requête appartient bien à la vue sélectionnée.

Les limites de SNMPv3

SNMPv3 souffre de plusieurs défauts de conception qui justifient son faible déploiement par les opérateurs. Tout d'abord, VACM est très peu utilisé en pratique probablement du fait de son niveau de granularité très fin et la difficulté de maintenance. Il est également très difficile d'avoir une vue synthétique des droits d'accès car la notion de vues utilisant des masques est complexe, surtout lorsque l'on combine des familles de sous-arbres.

Bien que la notion de groupe soit intéressante, elle est trop fortement liée au modèle de sécurité. En effet, c'est le nom de l'utilisateur et le modèle de sécurité utilisé qui déterminent le groupe. De plus, la notion de hiérarchie de groupe n'existe pas. Pour un modèle donné, un

utilisateur ne peut appartenir qu'à un groupe. Par conséquent, il n'est pas possible de réutiliser un ensemble de permissions existant. Comme pour les groupes, le couplage entre le modèle de sécurité utilisé et les vues est trop important. En effet, le modèle de sécurité intervient une nouvelle fois dans la table liant les vues aux groupes.

Un modèle de sécurité, baptisé RSM et fondé sur RBAC, a été proposé pour SNMPv3 dans [55]. L'architecture est composée d'un système de gestion de sécurité gérant les politiques de façon centralisée. Ces politiques sont distribuées aux agents et managers SNMP. De nouvelles tables sont créées pour instancier un modèle RBAC :

- l'association Utilisateur - Rôle en créant une table `rsmUserToRoleTable`,
- l'association de hiérarchie Rôle - Rôle avec la table `rsmRoleHierarchyTable`,

Cependant, RSM ne construit pas de table de permissions (opérations sur des objets) pour faire ensuite une association Rôle-Permission. Une table regroupant un rôle, ses différentes vues et le modèle de sécurité associé a été préféré. Ceci implique qu'une permission n'est pas réutilisable dans RSM. L'implantation de ce système de contrôle d'accès de l'agent nécessite une modification de l'agent pour fonctionner avec ces tables. L'approche que nous présentons plus loin dédiée à l'utilisation globale de RBAC avec SNMPv3 ne nécessite pas de modification des agents, et donc reste compatible avec l'existant.

Une autre limitation pour le passage à l'échelle est qu'il est nécessaire de partager un secret "à la main" pour chaque agent et chaque manager. En effet, aucun mécanisme de distribution de clés n'a été proposé car ce problème était considéré hors des limites de la spécification de SNMPv3.

Enfin, les approches de sécurité se sont multipliées indépendamment de SNMPv3 et sont devenues très populaires : PGP, RADIUS/DIAMETER, SSH, etc... Cela a été un frein au déploiement de SNMPv3 car ces approches ont été déployées rapidement et largement et le besoin d'une approche supplémentaire spécifique à SNMPv3 n'était plus compris et était devenu trop coûteux. De plus, l'argument justifiant la non-utilisation d'une approche trois-tiers pour garantir un fonctionnement en environnement dégradé propre à la gestion de réseau ne tenait plus face aux multiples avantages des approches concurrentes : authentification forte (PKI), externalisation pour fédérer la sécurité de plusieurs plateformes ou protocoles, passage à l'échelle des approches P2P, redondance des serveurs d'authentification.

3.2 Couplage faible

3.2.1 L'interface en ligne de commandes CLI

Une syntaxe simple mais efficace

La *CLI*, qui est propriétaire contrairement aux autres interfaces de supervision présentées dans ce document, est de loin l'interface de configuration la plus utilisée pour la configuration des routeurs. CLI est un protocole de configuration de réseau centré sur le concept de commandes qui permet de configurer les paramètres des protocoles de routage déployés comme RIP, BGP ou OSPF, ou tout autre élément des équipements réseau. L'administrateur accède à un prompt d'où il entre les commandes en format textuel. Les commandes CLI sont relatives à leur contexte d'exécution, donc une même commande peut produire des résultats différents. Pour configurer un routeur, il faut tout d'abord passer en mode privilégié et activer le mode configuration. Ensuite, un manager peut agir sur la configuration courante (running), c'est-à-dire celle en cours d'exécution sur le routeur. Par exemple, il peut désactiver la diffusion des annonces de routes sur une interface (eth0) ou encore autoriser la redistribution des routes apprises d'autres protocoles

de routage (OSPF) dans les tables RIP.

```
1 > enable
2 # configure terminal
3 # router rip
4 # passive-interface eth0
5 # redistribute ospf
```

FIG. 3.9 – Un extrait de commandes de configuration CLI

Authentification, intégrité et confidentialité

Une session CLI s'appuie en général sur les protocoles telnet ou SSH. Si telnet est utilisé, toutes les commandes ainsi que les mots de passe (telnet et enable) sont transmis en clair sur le réseau. Cela semble évidemment archaïque mais telnet est encore utilisé par de nombreux administrateurs réseaux. Dans des réseaux plus évolués, une politique de sécurité est généralement mise en place. L'authentification du serveur est prise en charge par le protocole SSH et l'utilisation de clés publiques. SSH assure également les services d'intégrité et de confidentialité. Pour l'authentification du manager, l'approche *Authentication, Authorization, Accounting* (AAA) est la plus souvent utilisée. Cette architecture permet de centraliser et uniformiser les traitements et décisions relatifs à la sécurité en centralisant les bases de données utilisateurs, leurs tokens ainsi que les services associés.

AAA s'applique dans deux domaines distincts mais non exhaustifs : le service d'accès au réseau (e.g. via un modem) et la configuration des routeurs. Le premier domaine s'adresse plutôt aux utilisateurs finaux en leur donnant des droits d'accès. Ces utilisateurs disposent de credentials, en général un login et un mot de passe, pour ouvrir une connexion PPP avec le serveur d'accès au réseau. Ce dernier peut stocker localement une base de données contenant les informations relatives aux utilisateurs ou bien utiliser un serveur AAA dédié comme TACACS [34], TACACS+ ou RADIUS [69], lui-même pouvant utiliser une base de données (type Oracle ou autre).

Le second domaine s'adresse, quant à lui, aux administrateurs réseau qui ont besoin d'accéder aux routeurs. Les routeurs peuvent utiliser les mêmes serveurs AAA pour définir des comptes utilisateurs et les services qui leur sont autorisés. Les serveurs AAA jouent donc un rôle important dans l'uniformisation du plan de sécurité dans les réseaux d'accès et les réseaux de cœur.

Un modèle de contrôle d'accès à niveaux hiérarchiques

La plupart du temps, une politique du tout ou rien est mise en place pour le contrôle d'accès. Un administrateur passe du mode utilisateur au mode privilégié par la commande *enable*. Dans le mode utilisateur, les droits d'un administrateur se limitent à l'accès en lecture aux informations de configuration du routeur. Dans le mode privilégié, qui correspond au niveau de sécurité maximum (level 15), un administrateur a tous les droits. Cette configuration des niveaux de privilèges peut être modifiée en redéfinissant le niveau de privilège de certaines commandes.

Les niveaux de sécurité s'étalent dans une plage 1 à 15 et sont organisés de façon hiérarchique. Un administrateur se plaçant au niveau i à un instant t dispose des privilèges de niveau i , ainsi que de tous les privilèges des niveaux hérités (ou inférieurs dans la hiérarchie), c'est-à-dire les niveaux 1 à $i-1$. Il est intéressant de faire le parallèle avec les rôles du modèle RBAC et de noter

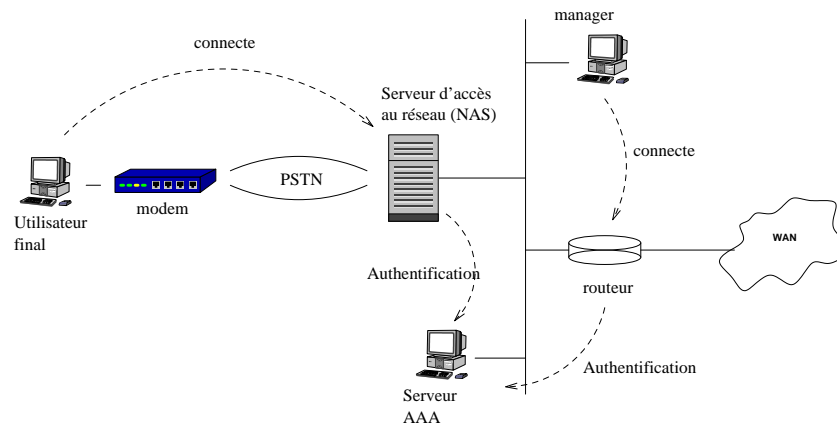


FIG. 3.10 – La sécurité de l'approche CLI

que, contrairement à celui-ci, la hiérarchie des niveaux dans CLI est linéaire. Un niveau donné n'a qu'un seul niveau senior, c'est-à-dire de niveau supérieur et un seul niveau junior, de niveau inférieur. C'est une limitation par rapport au modèle RBAC.

Dans CLI, toute commande peut être repositionnée à un autre niveau (e.g. au niveau 6), de façon à fournir un niveau beaucoup plus fin de configuration du contrôle d'accès. Mais, en pratique, la réaffectation des commandes dans les différents niveaux de privilèges est peu utilisée.

3.2.2 Le protocole de configuration Netconf

Netconf est un protocole de configuration de réseau très prometteur qui reprend le paradigme gestionnaire/agent. Il est soutenu par les plus grands fabricants du monde réseau. Se basant sur le langage XML pour la partie protocolaire mais aussi pour structurer les données, il profite des années d'expérience acquise lors de la croissance de l'Internet. Il va très probablement remplacer l'interface de commandes en ligne (CLI) et peut-être aussi SNMP. Par rapport à ces "protocoles" existants, il apporte principalement une gestion plus transactionnelle des configurations, grâce notamment au concept de configuration *candidate* et aux opérations qui lui sont propres : *validate* et *commit*. La première opération, *validate*, permet de valider la syntaxe et la sémantique de la configuration *candidate*, tandis que la seconde, *commit*, permet de remplacer la configuration *running* par la configuration *candidate*. Le fait qu'il repose sur le protocole sécurisé SSH est un deuxième point important. L'utilisation du langage XML rend également son implantation plus aisée grâce aux nombreuses bibliothèques existantes, qui disposent de toute la panoplie de fonctionnalités liées à XML : parsing de document, transformation, langage de requête, langage de spécification de structure.

Netconf se décompose en quatre couches d'abstraction illustrées par la Figure 3.11 :

protocole applicatif C'est le protocole de niveau inférieur, dans la structure en couche traditionnelle, sur lequel Netconf repose. Plusieurs protocoles sont possibles (SSH [78], SOAP [36], BEEP [54]) mais c'est SSH qui s'impose comme le protocole obligatoire, les autres devenant optionnels. SSH apporte un niveau très élevé de sécurité et fournit les services d'authentification, anti-rejeu, confidentialité et non répudiation,

RPC C'est la couche responsable de différencier les messages Netconf entre eux au sein d'une session. Elle définit un identifiant, à la manière d'un numéro de séquence, qui permet à un gestionnaire de savoir à quelle requête correspond une réponse,

opération Cette couche d'abstraction définit un ensemble d'opérations de base pour récupérer des données (get, get-config), modifier des données (copy-config, delete-config, edit-config), poser et relâcher des verrous (lock, unlock) pour acquérir un accès privilégié aux données, ou encore forcer la fermeture d'une session (close-session, kill-session),

contenu La couche d'abstraction *contenu* correspond aux données de gestion. Par exemple, il peut s'agir de données relatives à une politique de firewall, une configuration de routage BGP, RIP ou OSPF, ou toute autre configuration.

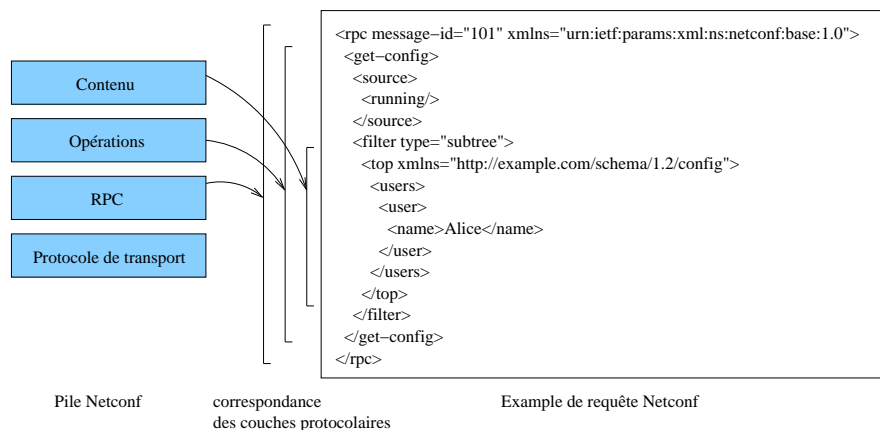


FIG. 3.11 – La pile protocolaire interne à Netconf

Des points clés du protocole

Le subtree filtering Le subtree filtering est une technique propre à Netconf qui permet de réaliser des requêtes de lecture sur une configuration XML. C'est l'équivalent du SELECT en langage SQL. Cette méthode de sélection définit un ensemble de règles de filtrage qui régissent le processus de construction du document résultat. Le principal intérêt de cette technique est qu'elle permet à un administrateur de construire des sortes de templates (document XML à trous) qui correspondent à une vue partielle de la configuration d'un équipement.

La deuxième méthode de sélection possible est basée sur XPath. Celle-ci a l'avantage de s'exprimer de façon beaucoup plus concise et de plus, est une technologie standard et répandue. Cependant, les constructeurs semblent plus favorables au subtree filtering pour des raisons de difficultés d'implantation.

Les trois types de configuration Netconf distingue trois types de configuration : running, startup et candidate. Ce modèle est en partie calqué sur le modèle CLI.

- #running correspond à la configuration qui est en cours d'exploitation par l'équipement réseau. Elle est modifiable à chaud à condition que l'équipement supporte l'extension #writable-running,
- #startup se rapporte à une configuration qui est chargée en mémoire lors du démarrage de l'équipement réseau. En cas de mise hors-tension puis de redémarrage de l'équipement, la configuration running est perdue et c'est la configuration startup qui redevient la configuration courante. Il est évidemment possible de sauvegarder la configuration running dans la configuration startup, grâce à l'opération copy-config,

- `#candidate` est une nouveauté par rapport à CLI. La configuration `#candidate` est une configuration temporaire de travail qui peut être utilisée au lieu de travailler directement sur la configuration en cours d'exploitation (running). Cela permet de faire des changements comme si on travaillait sur la configuration running puis de valider ces changements (validate) et enfin de les appliquer (commit). On peut également annuler les changements (discard-changes). L'utilisation de `#candidate` est un facteur de réduction des risques d'erreur de manipulation, car les changements ne prennent effet que lorsque l'administrateur pense que la configuration est valable. L'aspect transactionnel de cette fonctionnalité est l'un des plus importants introduits par Netconf par rapport à CLI.

Les extensions Netconf est un protocole extensible par deux de ses aspects : d'une part il est possible de définir de nouvelles opérations, d'autre part chacun peut étendre le jeu de données de configuration. Pour communiquer les extensions supportées, le manager et l'agent s'envoient mutuellement et de façon asynchrone un message hello contenant leurs extensions respectives lors de l'ouverture de la session Netconf. On utilise des URNs pour identifier de façon unique les extensions. Par exemple, si un agent annonce l'extension `urn : ietf : params : netconf : capability : xpath : 1.0`, cela signifie qu'il supporte la technologie XPath avec l'opération *get-config*.

Il existe deux types d'extensions : celles qui sont définies dans la spécification comme `#candidate`, `#xpath`, `#startup`, `#writable-running`, présentées précédemment et celles que les vendeurs vont développer pour se différencier des autres implantations.

La sécurité

L'utilisation sous-jacente du protocole SSH fournit un gage de sécurité en fournissant l'authentification de l'agent et du manager, l'intégrité et la confidentialité. L'externalisation des principaux mécanismes de sécurité vers une couche dédiée est un progrès important par rapport à SNMPv3. D'une part, cela permet de profiter des évolutions du protocole SSH de façon transparente. D'autre part, le protocole Netconf s'en trouve grandement simplifié car libéré des questions de cryptographie. C'est pour cette raison aussi que des travaux sont en cours (cf. le working group ISMS) pour définir un binding de SNMP sur SSL. Ces remarques s'appliquent également dans le cas de l'utilisation de netconf sur SOAP/SSL. SSH et SSL s'appuient tous les deux sur une infrastructure à clés publiques et utilisent les mêmes algorithmes de cryptage et d'authentification.

La problématique du contrôle d'accès pour Netconf est toujours en cours de discussion à l'IETF. Un draft sera probablement écrit pour décrire les modèles et mécanismes de contrôle d'accès pour Netconf. De notre côté, nous avons proposé un modèle et des extensions protocolaires décrits dans le chapitre 6.

3.2.3 Un protocole destiné aux passerelles résidentielles : TR-069

Le protocole TR-069, issu du DSL Forum, est destiné à permettre la gestion des passerelles résidentielles. Il s'adresse plus particulièrement aux terminaux des réseaux DSL, comme les *box* multi-services qui fleurissent dans toutes les offres des fournisseurs d'accès à Internet et autres opérateurs. Deux facteurs nouveaux sont apparus au cours de ces dernières années. Les services fournis sont passés d'un simple accès Internet à une multitude de services comme la voix sur IP (VoIP), la télévision sur ADSL, la vidéo à la demande (VoD), la connexion Internet et bientôt la convergence entre les téléphones mobiles type (GSM, GPRS) et les téléphones Wifi avec

les problèmes de handover pour le passage transparent d'une plateforme à l'autre. Le deuxième facteur est qu'on est passé d'un modèle mono-proviseur à un modèle multi-providers. Aujourd'hui, il n'est plus rare que des clients aient un opérateur pour la téléphonie classique, un opérateur pour l'accès à internet, et pourquoi pas un opérateur de VoIP (comme wengo).

Il arrive de plus en plus souvent que chaque fournisseur de services propose une *box*, appelé CPE (Customer Premise Equipment) dans le vocabulaire TR-069. Les services fournis par cet équipement évoluent au fur et à mesure des offres des opérateurs et nécessitent des mises à jour logicielles ou de leur firmware. De plus, il est nécessaire de pouvoir configurer les paramètres de ces équipements, comme le protocole de routage, le type de ligne ADSL (ADSL, ADSL2, ADSL2+, Re-ADSL, etc...) ou le numéro de téléphone IP. Les opérateurs souhaitent aussi monitorer ces équipements pour vérifier leur bon fonctionnement : nombre de paquets perdus, états des interfaces réseaux. TR-069 définit un modèle de données ordonné de façon hiérarchique à la manière de SNMP. L'adressage des données se fait à l'aide d'une notation pointée également comme SNMP.

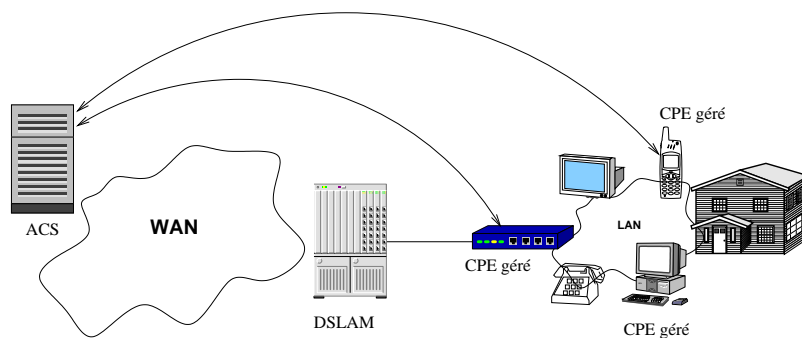


FIG. 3.12 – Les principaux composants d'une plateforme TR-069

Architecture protocolaire

Un Auto-Configuration Server (ACS) est une entité qui communique avec un ensemble de CPE. C'est l'ACS qui va collecter les données et configurer les CPE grâce au protocole TR-069. La figure 3.12 donne un aperçu très général de l'architecture. Pour cela, TR-069 définit un ensemble d'opérations de type RPC : SetParameterValues, GetParameterValues, SetParameterAttributes, Reboot, Download, Upload, etc... Pour faire la comparaison avec Netconf, le modèle de données de TR-069 n'est pas instancié dans un document XML contrairement à Netconf, même si les opérations de TR-069 sont codées en langage XML. De plus, TR-069 modifie les valeurs de façon atomique, c'est-à-dire en précisant à chaque fois le paramètre et la nouvelle valeur, alors que Netconf a plus une approche de type document où un manager transmet une sous-configuration qui est fondue dans la configuration ciblée. Netconf laisse à l'agent la responsabilité de savoir quelles sont les modifications à effectuer. Parallèlement, il est aussi possible de spécifier explicitement quel élément XML doit être créé, effacé, remplacé ou modifié dans Netconf. De ce point de vue, TR-069 est plus proche des opérations SNMP *get* et *set*.

En revanche, pour la pile protocolaire, TR-069 est quasi identique à Netconf sur SOAP. En effet, TR-069 fonctionne sur SOAP/HTTP/SSL(TLS)/TCP exactement comme Netconf si l'on considère SOAP comme le protocole applicatif.

Services de sécurité supportés

Authentification, confidentialité et intégrité En terme de sécurité, les conclusions sont les mêmes que pour Netconf : SSL est une approche qui fournit une authentification forte et également les services d'intégrité et de confidentialité. Concernant le modèle de distribution des clés, deux scénarii peuvent se produire qui vont conditionner le déroulement de la distribution des tokens de sécurité.

Dans le premier cas, l'association entre le CPE et l'ACS est connue au moment de l'achat, c'est-à-dire que le CPE est distribué par le fournisseur de service (souvent le cas des offres triple play) ou bien l'achat du CPE se fait au détail mais le fournisseur est connu. Ce cas de figure permet de fournir l'url de l'ACS et un secret partagé au client lors de l'achat. De son côté, le fournisseur de service configure l'ACS en lui donnant ce même secret. Une deuxième solution se base sur l'utilisation de certificats. L'ACS envoie son certificat et celui-ci est authentifié. Le CPE obtient l'identité de son ACS par un mécanisme de découverte (auto-configuration, DHCP, ACS par défaut dans la configuration du CPE). Cette identité est comparée à celle contenue dans le certificat. L'authentification du CPE est faite également par certificat. Le CPE génère un certificat grâce à une autorité de certification (CA) en ligne dont l'adresse est préconfigurée.

Dans le second cas, le CPE a été acheté sans la connaissance a priori d'un fournisseur de service. Le mécanisme d'authentification de l'ACS utilisé à l'initialisation est alors la même que précédemment utilisant les certificats : découverte de l'identité de l'ACS, vérification du certificat. Le CPE génère un certificat en ligne à l'aide d'une CA connue.

Modèle de contrôle d'accès Pour la partie contrôle d'accès, chaque paramètre (ou feuille) de l'arbre de données de gestion est associé à une liste d'attributs. Parmi ces attributs, on trouve une *AccessList* qui indique quelles entités sont autorisées à modifier le paramètre associé. Si la liste est vide, alors la seule entité ayant les droits en écriture est l'ACS. Sinon, les entités de la liste ont aussi ces droits. Pour l'instant, seule une entité a été définie : *Subscriber*. Ainsi, le client propriétaire du CPE peut être autorisé à configurer lui-même son équipement. Par défaut, la liste des entités contient l'ensemble des entités existantes.

Cette approche du contrôle d'accès par paramètre est très fine mais problématique du point de vue maintenabilité. En effet, maintenir une ACL par paramètre par équipement peut vite devenir un casse tête et poser un problème de passage à l'échelle. Cependant, TR-069 fournit un moyen d'appliquer une ACL à un ensemble de paramètres en une seule opération en désignant un sous-arbre au lieu d'un simple paramètre. La nouvelle liste d'accès est alors appliquée à l'ensemble des paramètres désignés. La Figure 3.13 illustre un exemple de configuration des règles de contrôle d'accès dans TR-069. Le paramètre de l'opération est le sous-arbre ayant pour racine le nœud *ManagementServer* car il y a un point à la fin de l'expression *InternetGatewayDevice.ManagementServer*. qui indique que l'opération s'applique à tous les paramètres sous ce nœud. Les trois nœuds enfants sont donc affectés par cette opération et la liste *AccessList* contient l'unique élément *Subscriber*.

3.2.4 Un exemple d'outil d'administration avec interface web : Webmin

Architecture protocolaire

Webmin est un outil d'administration basé sur un interface web et des scripts perl. Il adresse en particulier la configuration de serveur HTTP mais aussi DHCP ou DNS. C'est un outil qui est orienté gestion système mais aussi gestion de réseau. Webmin repose sur un système de module et, à ce titre, est considéré comme extensible. Plus loin dans ce document, nous verrons un

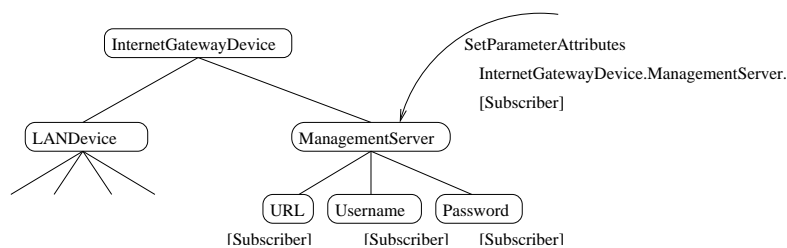


FIG. 3.13 – Un exemple de configuration des ACL pour TR-069

développement de module réalisé par des étudiants dans le cadre de cette thèse qui permet de gérer les politiques de contrôle d'accès d'agents SNMP.

Un point important est que Webmin n'est pas un protocole réseau dans le sens où chaque équipement a son propre serveur HTTP dédié à Webmin. Il n'y a donc pas de protocole spécifique de transport de données ou d'opération de gestion, mis à part les POST et GET du protocole HTTP. Comme un serveur web est nécessaire par équipement, Webmin ne passe pas à l'échelle. On n'imagine pas se connecter à 10000 routeurs, un par un, via une interface web. C'est un problème qui pourrait aussi se produire avec CLI, sauf que CLI permet de charger une configuration très rapidement via TFTP par exemple, ce qui accélère grandement la configuration d'un grand nombre d'équipements.

La sécurité dans Webmin

La sécurité de Webmin repose sur HTTPS, donc sur SSL. Comme pour Netconf ou CLI (telnet sur SSH), les mécanismes sont externalisés vers les couches inférieures : authentification forte, intégrité, confidentialité.

La gestion du contrôle d'accès est plutôt bien pensée dans Webmin. En effet, un bon compromis en terme de granularité du contrôle d'accès a été trouvé et les futures propositions de Netconf dans ce domaine auraient grand intérêt à s'en inspirer pour éviter un système similaire à VACM. Dans un premier temps avec Webmin, les droits d'accès sont attribués par module (moyen grain), ce qui évite de décrire trop finement les données réellement concernées. Par exemple, un utilisateur pourra configurer uniquement le serveur HTTP et le serveur DNS. Cependant, chaque module Webmin a la possibilité de dire quelles données spécifiques du module peuvent être lues ou modifiées. Pour résumer, Webmin considère un module comme unité principale de description des ressources au niveau des permissions, puis chaque module peut rédéfinir des permissions plus fines en décrivant ses propres ACL. Ce double niveau de granularité est très confortable à maintenir.

3.3 Synthèse

Par le passé, de nombreux modèles de supervision de réseau ont été proposés et d'autres, aujourd'hui, sont en cours de conception, chacun proposant son propre modèle de sécurité. Il n'y a pas eu de véritable réflexion sur les interactions entre ces différentes plateformes et en particulier les conflits entre leurs politiques de sécurité respectives.

Le Tableau 3.1 synthétise les services de sécurité offerts par les différentes plateformes de gestion de réseau en fonction du protocole sous-jacent. Notons que l'authentification du client avec SSH a lieu après la négociation SSH, à un niveau protocolaire supérieur, contrairement à

SSL. Nous n'avons pas présenté dans ce chapitre la plateforme WBEM (Web Based Enterprise Management) proposée par le DMTF³ et qui repose sur le modèle d'information CIM (Common Information Model). Pour la partie sécurité, cette plateforme repose sur le protocole sécurisé SSL. Son mécanisme de contrôle d'accès peut fonctionner par espace de nom. Par exemple, un utilisateur peut avoir le privilège d'écrire (*écrire* regroupant une dizaine d'opérations de plus haut niveau comme SetProperty ou CreateInstance) dans un espace de noms donné.

Plateforme de supervision	Authentification du serveur	Confidentialité	Intégrité des données	Authentification du client	Contrôle d'accès
SNMPv1, v2c				x mot de passe en clair	
SNMPv3		x USM [10] cbc-des	x USM [10] hmac-md5-96 hmac-sha-96	x mot de passe md5, sha	x VACM [79]
CLI/telnet				x mot de passe	x niveaux
CLI/SSH	x ssh-rsa ssh-dss	x 3des-cbc aes, ...	x HMAC [50] hmac-sha1 hmac-md5	x SSH Auth. Prot. [82] clé publique mot de passe	x niveaux
Netconf/SSH	x ssh-rsa ssh-dss	x 3des-cbc aes, ...	x HMAC [50] hmac-sha1 hmac-md5	x SSH Auth. Prot. [82] clé publique mot de passe	x Pas encore spécifié
Netconf/SOAP /HTTP/SSL [30]	x rsa dss	x des 3des, rc4	x HMAC [50] hmac-sha1 hmac-md5	x (optionnel) Certificat client mot de passe	x Pas encore spécifié
TR-069/SOAP /HTTP/SSL(TLS)	x rsa dss	x des 3des, rc4	x HMAC [50] hmac-sha1 hmac-md5	x (optionnel) userid/mot de passe Distribution non spécifiée	x une ACL par nœud
Webmin /HTTP/SSL	x rsa dss	x des 3des, rc4	x HMAC [50] hmac-sha1 hmac-md5	x (optionnel) Certificat client mot de passe	x Permissions par module

TAB. 3.1 – Synthèse des services de sécurité par plateforme de gestion

Certains modèles de sécurité sont complètement intégrés au protocole, comme avec SNMPv3. D'autres tentent d'externaliser les mécanismes de sécurité en réutilisant des protocoles existants comme SSH, SSL ou RADIUS. Concernant le contrôle d'accès, la tendance est la politique du *chacun pour soi*, probablement du au fait que ce type de politique est très fortement lié au modèle de données de la plateforme. Ces problèmes d'hétérogénéité ont tendance à s'amplifier au cours du temps, car il y a de plus en plus de protocoles qui coexistent, souvent incompatibles, et ouvrent d'autant plus de portes pour des personnes mal intentionnées. Il est parfois même envisagé de transporter certains protocoles de supervision dans d'autres, ce qui rend le problème encore plus difficile. Des réflexions sont menées pour, par exemple, transporter des commandes CLI ou des messages syslog dans le protocole Netconf.

Dans la suite de cette thèse, nous nous intéressons à ce besoin fort d'homogénéisation en abordant le problème par une approche globale visant à harmoniser le niveau de sécurité à travers les plateformes de gestion de réseau déployées, en particulier dans le domaine du contrôle d'accès.

³Distributed Management Task Force, <http://www.dmtf.org>

3.3.1 Le groupe ISMS

Parallèlement aux travaux que nous avons mené sur la problématique de la sécurité dans l'univers de la supervision des réseaux, des travaux ont été entrepris par le groupe de travail ISMS (Integrated Security Model for SNMP) au sein du protocole SNMP pour permettre l'intégration des mécanismes de sécurité de ce protocole avec les architectures de sécurité déployées sur le réseau comme un serveur AAA ou encore une infrastructure à clés publiques.

A l'origine des travaux été apparus avec [39] visant à introduire le concept de session sécurisée dans SNMPv3. Cette approche repose sur une PKI et se différencie de USM par le fait que les clés de session ont une durée de vie limitée. Concrètement, dans le groupe ISMS, il est question d'utiliser par exemple le protocole SSH pour l'établissement de communications chiffrées et authentifiées, et RADIUS pour l'authentification et l'autorisation du client (le manager). Rappelons que SSH ne gère que l'authentification du serveur et pas celle du client d'où l'utilisation de RADIUS après l'établissement de la session SSH.

Comme le groupe de travail souhaite que d'autres protocoles de transport puissent être utilisés (BEEP, TLS, ...), un mécanisme de conversion est défini pour associer une entité SSH (ou BEEP ou TLS, ...) à une entité SNMP, de façon assez similaire à ce qui avait été fait lors de la définition de l'USM et l'alias *securityName*. C'est un facteur de complexité à prendre en compte car dorénavant on n'a plus simplement plusieurs protocoles (SNMP, CLI, Netconf) avec chacun leur modèle de sécurité mais on a également certains protocoles qui dans leur propre structure sont susceptibles de mettre en oeuvre plusieurs modèles de sécurité. C'est le cas de SNMP avec SSH, BEEP ou TLS mais aussi de Netconf (SSH, BEEP, SOAP) et aussi de CLI (telnet, SSH). D'où une gestion potentiellement encore plus complexe des politiques de sécurité. Ne serait-il pas plus sage d'avoir une vision plus large et se limiter à un protocole au lieu de vouloir rendre les choses très génériques au prix d'une complexité difficilement gérable ? Il semble cependant qu'il y ait un dénominateur commun qui se dégage et qui est le protocole SSH (Netconf, SNMP via ISMS, CLI) ce qui est en soi un progrès. S'il est combiné avec RADIUS, ce sera une autre avancée dans l'uniformisation de l'authentification et de l'autorisation du manager.

Deuxième partie

Vers une double convergence des modèles et politiques de sécurité

Une approche pour la cohérence globale

Sommaire

4.1	Convergence du contrôle d'accès en amont	43
4.1.1	Unifier la gestion du contrôle d'accès	43
4.1.2	Automatiser le déploiement	44
4.2	Des modèles de convergence concrets	44
4.2.1	Passerelles multi-protocoles	44
4.2.2	Algorithme de conversion RBAC/CLI avec pertes	45
4.2.3	Une architecture de déploiement du modèle RBAC	50
4.3	Synthèse	58

4.1 Convergence du contrôle d'accès en amont

4.1.1 Unifier la gestion du contrôle d'accès

Unifier la gestion du contrôle d'accès passe par la définition d'une politique ayant un niveau d'abstraction des données plus haut que celui des plateformes spécifiques. Ensuite, il y a forcément une phase de conversion qui doit être capable de particulariser la politique globale sur des ressources plus concrètes. Cette phase est appelée phase de déploiement. Elle contribue à uniformiser les politiques appliquées au sein des équipements. Si l'on configure des équipements similaires manuellement, non seulement il se pose un problème de coût de configuration mais, en plus, il apparaît forcément des incohérences.

Il faut également définir des fonctions (métiers, rôles) de gestion de réseau et éventuellement les hiérarchiser pour construire par la suite de nouvelles fonctions à partir de briques existantes et ainsi simplifier la maintenance. L'agrégation des permissions dans ces fonctions est un avantage que l'on retrouve dans VACM et CLI. Mais VACM n'utilise pas de hiérarchie et CLI n'a qu'une hiérarchie directe. RBAC répond à tous ces problèmes grâce au concept de rôles hiérarchiques. L'utilisation de RBAC, de son mécanisme d'activation des rôles, et de séparation des pouvoirs (statique et dynamique), limite considérablement les risques de fautes, en limitant le champ d'action pendant une session.

Le défi principal est d'offrir une interface unique, de haut niveau d'abstraction, de gestion des droits et capable de s'adapter à l'hétérogénéité sous-jacente. C'est ce défi que nous relevons ici.

4.1.2 Automatiser le déploiement

Automatiser le déploiement à grande échelle de la sécurité et des politique de contrôle d'accès est faisable à condition de combiner plusieurs approches :

- Administrer la politique de sécurité de la plateforme par la plateforme elle-même (au lieu de gérer à la main),
- Mettre en place une PKI [1] pour l'authentification forte et la confidentialité,
- Utiliser un ou plusieurs repository,
- Manager des groupes d'équipements de façon transparente au lieu de gérer "par équipement".

Mais, selon les cas, ces approches ne sont pas toujours adaptées. Par exemple, dans le cas d'un protocole qui n'est pas orienté session comme SNMP, il est extrêmement coûteux d'utiliser une approche PKI, puisque qu'on n'amortit pas du tout le coût d'établissement de la session. L'envoi d'un seul message régulier dans le cadre d'une activité de monitoring, ne justifie pas un handshake SSL, avec vérification de certificats, l'échange de clés symétriques, puis l'échange d'un message SNMP.

De même, il n'est pas toujours possible de réaliser les mêmes opérations sur un ensemble d'équipements à grande échelle. C'est possible pour certaines opérations mais impossible pour d'autres, notamment les pare-feux ou les interfaces d'un routeur. Il y a un compromis à trouver entre généralisation du déploiement et spécialisation des requêtes selon la nature des équipements.

4.2 Des modèles de convergence concrets

4.2.1 Passerelles multi-protocoles

Des propositions de passerelles multi-protocoles ont été faites au cours des dernières années, la plupart étant orientées vers le langage XML. Certaines ([64, 83, 73]) se sont intéressées à la conversion du protocole SNMP vers le langage XML et d'autres [62] à l'étude des performances de telles approches. Une conversion automatique de la description du modèle de données a été proposée et implantée. Cette conversion est capable de générer des schémas XML à partir des MIB(s). L'intérêt est de faciliter le traitement des données du côté du gestionnaire car il existe de nombreuses bibliothèques pour ce langage.

Un deuxième type de passerelle a été proposé dans [62] pour la conversion des messages SNMP vers des messages de type SOAP et inversement, dans le cadre des propositions relatives aux *web services* (WS). Les web services ayant leurs propres mécanismes de sécurité, il est nécessaire de définir un modèle permettant de conserver les propriétés de sécurité des deux côtés de la passerelle (SNMP et WS).

On retrouve ces problèmes de sécurité quelque soit la passerelle utilisée, car les protocoles diffèrent dans la mise en œuvre des mécanismes de sécurité. Une architecture incluant une passerelle crée un modèle à trois entités là où il n'y avait que deux entités à l'origine. C'est une modification profonde du modèle et une nouvelle dimension pour la sécurité et en particulier pour la gestion des clés. Cela nécessite une réflexion sur la distribution des responsabilités de chacun. Par exemple, comment un agent SNMP peut identifier un manager et donc ses permissions, si c'est la passerelle et non plus le manager qui génère le message SNMP ? Comment fédérer les identités des managers dans une passerelle multi-protocoles ? L'introduction des passerelles est révélateur du problème de complexité qui se pose de plus en plus au sein de la gestion de réseau et qui est dû à la multiplicité grandissante des protocoles. Elle illustre notamment l'hétérogénéité des modèles de données, de modèles de sécurité et des format de messages.

Dans le chapitre 7, nous proposons une architecture de sécurité pour les passerelles XML/SNMP et son implantation. Cette architecture assure une sécurité de bout en bout et optimise le coût de maintenance, tout en garantissant un haut niveau de cohérence.

4.2.2 Algorithme de conversion RBAC/CLI avec pertes

Motivation

Pour distribuer une politique de contrôle d'accès centralisée et basée sur les rôles à des équipements réseaux compatibles avec CLI, un schéma de conversion du modèle RBAC vers les niveaux de sécurité CLI doit être défini. La politique ainsi convertie doit refléter le plus fidèlement possible la politique de sécurité originale, de façon à préserver les cohérences locales et globales.

Le contrôle d'accès dans CLI se fonde sur le principe des niveaux de sécurité. Ces niveaux peuvent être activés par un mot de passe ou via un serveur TACACS [34]. Comme une commande n'est accessible qu'à partir d'un certain niveau de privilège, un utilisateur peut exécuter cette commande que si il a lui-même accès au niveau de privilège requis. Toutes les commandes sont ainsi distribuées à un niveau de privilège de façon à organiser de façon hiérarchique les différentes opérations de configuration, et parallèlement les administrateurs en fonction de leur rôle, leur niveau de connaissance ou leur expérience. En effet, au cours d'une session CLI, un administrateur peut activer des niveaux de sécurité à la demande, à condition que les règles de sécurité l'y autorisent. Prenons l'exemple de la commande `show` que nous avons volontairement placée au niveau de sécurité 12. Seuls les administrateurs autorisés à activer via `enable` le niveau 12 ou supérieur, peuvent alors exécuter cette commande.

Même si c'est peu utilisé en pratique, il est possible de positionner arbitrairement le niveau de sécurité de chaque commande grâce à la commande `privilege`. Notons que le niveau de sécurité 15 joue un rôle particulier puisque il a le privilège de modifier l'attribution des privilèges.

Les niveaux de sécurité CLI sont organisés dans une hiérarchie linéaire. Le nombre usuel de niveaux est limité à 15. On estime que 15 niveaux sont suffisants pour la grande majorité des cas rencontrés. Ce modèle hiérarchique établit qu'un niveau i dispose, par héritage, des permissions du niveau inférieur. Par transitivité, un niveau i a tous les privilèges des niveaux inférieurs, c'est-à-dire du niveau i au niveau 0. Le niveau 15 a tous les privilèges de tous les niveaux. En pratique, lorsqu'un administrateur souhaite activer le niveau de privilège 12, il utilise la commande suivante : `enable 12`. Après cette commande, toutes les permissions du niveau 12 au niveau 0 lui sont données. Pour revenir au niveau 9, un administrateur doit entrer la commande : `disable 9`.

La Figure 4.1 illustre les deux modèles de contrôle d'accès, à savoir les niveaux de privilèges de CLI et le modèle RBAC. Dans le modèle RBAC, les rôles sont organisés de façon hiérarchique. Typiquement, un rôle *senior* hérite de toutes les permissions de ses rôles *juniors*. Récursivement, ces rôles *juniors* héritent des permissions de leurs propres rôles *juniors* et ainsi de suite. La Figure montre l'affectation d'un utilisateur à un rôle F et la propagation des rôles hérités (ici, A, B et D). Le modèle à niveaux de CLI est aussi hiérarchique. Un utilisateur a toujours un niveau courant de sécurité et toutes les commandes autorisées par les sous niveaux sont également permises. Par exemple, un utilisateur ayant le niveau 2 peut réaliser toutes les opérations permises par les niveaux 2, 1 et 0.

La principale différence qui existe entre ces deux modèles est la hiérarchie : simple pour le modèle à niveaux et multiple dans RBAC. Par conséquent, la principale difficulté à résoudre est la conversion des rôles vers les niveaux de sécurité. Le problème, illustré par la Figure 4.2, est qu'il n'est pas possible, dans le cas général, d'ordonner deux rôles, E et F, et de les convertir

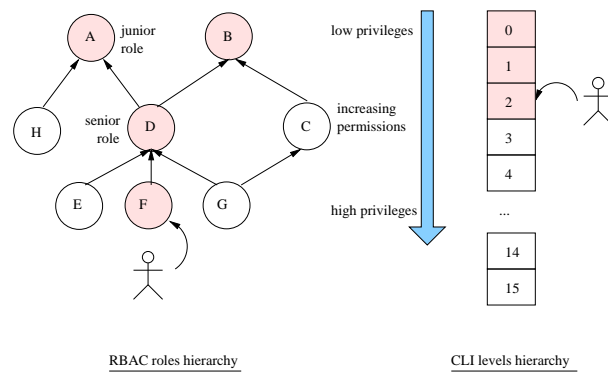


FIG. 4.1 – Comparaison des modèles de contrôle d'accès

dans deux niveaux 6 et 7. La raison est qu'une telle conversion donnerait au rôle F toutes les permissions de E, ce qui n'est pas acceptable. Le second problème est qu'il n'y a pas de critère trivial qui permettrait d'affirmer qu'un rôle est plus critique qu'un autre d'un point de vue sécurité, et qui permettrait de choisir le niveau relatif de sécurité. Nous pensons donc qu'une conversion parfaite du modèle hiérarchique RBAC vers le modèle de sécurité CLI basé sur les niveaux n'est pas possible sans introduire une perte d'information.

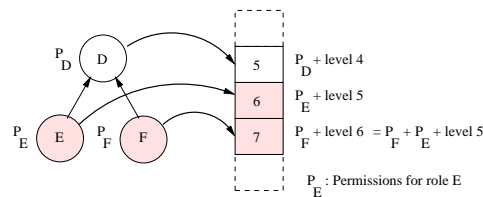


FIG. 4.2 – Problèmes de conversion des modèles de contrôles d'accès

Algorithme de conversion avec perte

Nous optons pour une solution compromis qui vise à minimiser les différences entre le comportement attendu relativement à la politique RBAC originale et le comportement de la politique de sécurité CLI résultante. Une assertion qui doit être vérifiée pour toute conversion est que le niveau d'un rôle senior doit être plus élevé que le niveau de tous ses rôles juniors. Dans la solution proposée à la section suivante, il se peut qu'un rôle gagne des privilèges. L'objectif est de minimiser le nombre de permissions qui seront ajoutées à un rôle. L'idée sous-jacente est qu'il est préférable d'allouer des permissions supplémentaires à un rôle disposant déjà de nombreux privilèges que d'ajouter des privilèges importants à un rôle à faibles responsabilités.

Pondération du modèle RBAC

L'algorithme proposé repose sur les poids affectés aux rôles. L'utilisation de ces poids permet la définition d'un ordre total entre les rôles. Un rôle ayant un poids supérieur à un second rôle aura un niveau de sécurité plus élevé que le second. De façon à allouer un poids aux rôles, nous proposons de pondérer arbitrairement les permissions. Il est plus facile d'attribuer un poids à une permission qu'à un rôle qui est, par définition, associé à un ensemble, potentiellement grand, de

permissions. Les poids des rôles découlent ensuite de cette attribution. Par exemple, la permission (write, firewall rules) aura un poids supérieur à la permission (read, network interfaces). De même, la permission (write, firewall rules) devrait avoir un poids supérieur à celui de la permission (read, firewall rules) pour des raisons évidentes. L'allocation des poids aux permissions est laissée à la charge de l'administrateur local. Cette allocation des poids est importante car elle influence sensiblement la façon dont les rôles sont converties sur les niveaux de sécurité. Une bonne pratique pour réaliser cette allocation est de trier les permissions par ordre d'importance puis, seulement ensuite, de leur donner un poids.

Avec cette approche, le poids affecté à un rôle est la somme des poids de tous ses rôles juniors, plus la somme des poids de ses permissions propres. Cette méthode satisfait, par construction, la contrainte énoncée précédemment et qui veut que le poids d'un rôle junior est plus faible que ceux de ses rôles seniors. Une fois que le poids des rôles a été calculé, l'ordre des rôles dans le modèle de sécurité CLI est donné par l'ordre de leurs poids respectifs. Si, par hasard, deux rôles ont le même poids, ils peuvent être rangés de manière aléatoire.

La Figure 4.3 illustre notre approche basée sur la pondération des rôles et la conversion obtenu dans la hiérarchie de niveaux CLI. Chaque rôle est lié à deux nombres : celui qui est dans le cercle indique le poids du rôle alors que celui en dehors du cercle correspond au poids des permissions locales. Par exemple, le poids du rôle H est de 12, ce qui correspond à la somme des poids de ses permissions locales (7) et de ses rôles juniors (rôle A de poids 5). Comme le rôle B a le plus petit poids, il sera affecté au niveau de privilège 0. Le rôle A est affecté au niveau 1. Les rôles H et D ont le même poids et sont donc ordonnés de façon aléatoire. Il est facile d'observer que certains rôles auront plus de privilèges qu'auparavant. Par exemple, le rôle D aura les permissions locales du rôle C parce que son poids est supérieur à celui de C. Cependant, les rôles C et D étaient des voisins et non des parents dans la hiérarchie initiale. Il se peut également que le rôle D obtienne les permissions du rôle H.

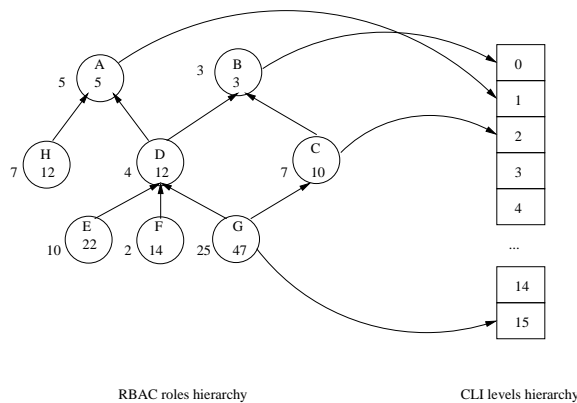


FIG. 4.3 – Affectation des poids et lien avec les niveaux CLI

L'algorithme proposé précédemment fonctionne à la condition que le cardinal de l'ensemble des rôles est inférieur au nombre de niveaux de sécurité CLI. Si cette condition n'est pas vérifiée, un algorithme plus complexe doit être établi. Le paragraphe suivant présente une solution basée sur l'algorithme de K-means clustering [45] pour résoudre ce problème. Ce nouveau scénario sous-entend la nécessité d'affecter plusieurs rôles par niveau de sécurité car il n'y a plus assez de niveaux pour placer les rôles niveau par niveau. La question qui se pose alors est de définir un critère permettant de rassembler des rôles dans des sous-ensembles correspondant à un niveau

CLI.

Une façon de répondre à ce problème est de considérer l'ensemble des poids de rôles, qui sont des entiers, et d'essayer de construire des sous-ensembles afin de minimiser l'hétérogénéité. La méthode de K_means clustering apporte une solution possible à ce problème. L'idée est que les rôles qui ont des poids proches doivent être associés à des responsabilités similaires. C'est pourquoi ces rôles peuvent être rassemblés dans un même sous-ensemble. Certains utilisateurs auront des nouveaux privilèges mais dans des proportions limitées. De cette façon, un rôle junior avec des faibles permissions ne gagnera jamais des privilèges correspondant à des responsabilités sensiblement plus importantes.

Le K_means clustering fournit la méthode théorique pour construire nos sous-ensembles de rôles. Il permet d'organiser un ensemble d'éléments dans des sous-ensembles en fonction d'un critère donné. Un tel critère peut être la minimisation de la distance de chaque élément d'un cluster avec la valeur moyenne de ce cluster. Un exemple typique de cette méthode est la clusterisation d'un ensemble de points dans un plan à deux dimensions. Notre proposition consiste à considérer les poids des rôles comme l'ensemble des éléments à clusteriser. Dans ce contexte, K est le nombre de niveaux de sécurité. Chaque rôle sera associé à un cluster (ou un niveau de privilèges). La valeur qui doit être minimisée est la distance entre le poids du rôle et la valeur moyenne du cluster.

Considérons le cas d'utilisation illustré par la Figure 4.4. Soit un ensemble des rôles avec leurs poids respectifs. Le nombre de niveaux de privilèges à atteindre est 3 : $K = 3$. L'algorithme K_means clustering consiste, tout d'abord, à affecter aléatoirement les rôles dans le K clusters destination. Ensuite, avec ce contexte courant, la moyenne des poids est calculée pour chaque cluster. Puis, les rôles sont redistribués dans le cluster ayant la moyenne la plus proche de leur poids. Cette opération est répétée jusqu'à ce que aucun rôle ne se déplace d'un cluster à un autre. Dans l'exemple de la Figure, le rôle f de poids 14 est plus proche du niveau 1 (11.33) que du niveau 2 (27.66). Donc le rôle f passe du niveau 2 au niveau 1 pendant la première étape. Aucun rôle ne change de cluster durant la deuxième étape donc l'algorithme se termine et l'affectation finale des rôles dans les niveaux est trouvée. Les rôles ont ainsi été distribués dans les niveaux de privilèges CLI.

Dans l'exemple, les rôles a et b seront associés au niveau de sécurité 0. Cela signifie que les utilisateurs qui activent un de ces rôles partageront le mot de passe pour ce niveau. Tous les utilisateurs qui ouvrent une session avec l'un de ces rôles activé auront les mêmes privilèges. Il y a donc perte partielle d'information.

L'algorithme K_means clustering appliqué au contexte courant est illustré par la Figure 4.5. Tant que au moins un rôle passe d'un niveau (cluster) à un autre, la boucle continue. Pour initialiser l'algorithme, la méthode *RandomRoleToClusterDistribution* distribue les rôles vers les clusters aléatoirement. Au lieu de cette distribution aléatoire des rôles, un raffinement consiste à trier dans un premier temps les rôles selon leur poids et de les affecter de façon homogène aux différents clusters. Ensuite, la méthode *computeClusterAverageWeight()* calcule la nouvelle moyenne de chaque cluster à partir des rôles qui composent ce cluster. La méthode *findNewCluster(role r)* cherche le nouveau cluster de chaque rôle, selon les résultats obtenus avec la méthode précédente. La méthode *currentCluster(role r)* retourne le cluster trouvé durant l'étape précédente.

Conversion des permissions

Après la première étape qui a consisté à convertir les rôles vers les niveaux de privilèges, la deuxième étape consiste à traduire les permissions vers la politique de sécurité CLI. Cette étape

Roles	b	a	c	d	h	f	e	g
Weights	3	5	10	12	12	14	22	47
CLI Levels	0	0	1	1	1	2	2	2
Weight average	4		11.33			27.66		

↓

Roles	b	a	c	d	h	f	e	g
Weights	3	5	10	12	12	14	22	47
CLI Levels	0	0	1	1	1	1	2	2
Weight average	4		12			34.5		

↓

Roles	b	a	c	d	h	f	e	g
Weights	3	5	10	12	12	14	22	47
CLI Levels	0	0	1	1	1	1	2	2
Weight average	4		12			34.5		

FIG. 4.4 – Application du K_means clustering à la conversion RBAC vers CLI

```

R={set of roles}
RandomRoleToClusterDistribution()
continue = true
while continue{
  continue = false
  computeClusterAverageWeight()
  for each role r of R
    c = findNewCluster(r)
    if c != currentCluster(r)
      continue = true
}

```

FIG. 4.5 – Algorithme K_means clustering

nécessite une conversion du modèle de données et une réallocation des permissions.

Une politique RBAC basée sur XML décrit la politique de contrôle d'accès au modèle de l'information XML par l'utilisation intensive d'expressions XPath. Ces expressions XPath décrivent les ressources protégées. Pour propager cette politique de sécurité vers les équipements réseau utilisant la CLI, il est nécessaire de savoir convertir les permissions, ce qui se concrétise par un changement de niveau de privilège des commandes CLI. Ainsi, toutes les permissions doivent être prises en considération et leurs commandes CLI relatives doivent être repositionnées en fonction du rôle auquel elles sont affectées. Ce mécanisme suppose qu'une expression XPath peut être convertie vers une commande.

Cette conversion est la principale difficulté à résoudre. La question est comment une permission (`read, /*[ifname='eth0']`) peut se traduire en une commande générique telle que `show interface`. Dans le modèle de sécurité de CLI, qui suit une approche orientée commande, il n'est pas permis de renseigner des paramètres dans la définition des privilèges. Dans notre politique XML, il est facile d'allouer des privilèges qui autorise, par exemple, à lire les informations relatives à une interface ou à une configuration de routage. Dans CLI, on ne sait pas faire la différence entre `show interface` et `show route`.

Le chapitre 5 apporte un début de réponse à ce problème, notamment en se basant sur la représentation arborescente Meta-CLI définie dans [29].

4.2.3 Une architecture de déploiement du modèle RBAC

Considérations sur l'architecture et son design

La supervision de réseau devient une tâche complexe dans les réseaux de grande taille, où des milliers d'équipements sont gérés par des groupes d'administrateurs de réseaux et de services. Parallèlement, le nombre de personnes en charge de maintenir le réseau augmente aussi de façon significative. L'organisation des administrateurs est importante dans un tel contexte et peut se faire par fonctions. Le modèle de contrôle d'accès basé sur les rôles (RBAC [52]) est particulièrement adapté pour ce type d'architecture. Il permet le passage à l'échelle en terme de nombre d'utilisateurs, puisque ceux-ci sont regroupés par rôles, eux-mêmes organisés de façon hiérarchique. Les utilisateurs peuvent être associés à des rôles dynamiquement et de façon distribuée. La difficulté qui consiste à faire coopérer tous les équipements et à fournir une qualité de service acceptable est fortement liée à la capacité à définir des tâches de gestion de réseau, tout en assurant les services de sécurité pour permettre aux personnes autorisées à réaliser les opérations de gestion.

Exécuter une requête sous la coupe d'un rôle nécessite au préalable un mécanisme d'authentification de l'administrateur. Un utilisateur doit s'authentifier auprès de tous les équipements réseau, ce qui implique souvent une infrastructure à clé publique (PKI [1]). Une alternative pour un utilisateur est de s'authentifier auprès d'un serveur pour obtenir ainsi des tokens de sécurité qui lui permettent par la suite de s'authentifier auprès des agents de gestion. De plus, au lieu d'utiliser des tokens spécifiques à un utilisateur, ces tokens peuvent être spécifiques à un rôle de façon à ce qu'un utilisateur s'authentifie indirectement à travers un rôle auprès d'un agent. Ainsi, les agents ne sont capables d'authentifier que des rôles ce qui réduit fortement le nombre de tokens à distribuer puisque le nombre de rôles est typiquement beaucoup plus faible que les nombre d'administrateurs dans un réseau à grande échelle. Cela réduit également la complexité du processus de contrôle d'accès au niveau de l'agent. Un tel mécanisme utilisant des clés pour l'authentification et le chiffrement nécessite une organisation efficace de distribution et de rafraîchissement de clés, en particulier dans un contexte où les utilisateurs peuvent avoir des fonctions

multiples au cours du temps. Des algorithmes performants existent dans le contexte de la sécurité multicast et peuvent être réutilisés dans notre contexte.

Partant des prometteuses approches de gestion de réseau basées sur XML, nous proposons un framework sécurisé pour la gestion de réseau avec des contraintes de performances et de passage à l'échelle, utilisant les mécanismes décrits précédemment. Nousinstancions cette approche dans le cadre de Netconf [32].

Un tel protocole introduit des contraintes de sécurité, comme par exemple le fait que seuls les entités autorisées doivent être capable de configurer les équipements réseau et ceci avec un certain nombre de services de sécurité mis en œuvre. Selon la spécification du protocole de configuration Netconf, le protocole de transport sur lequel Netconf repose est responsable de l'authentification, de l'intégrité et de la confidentialité des données [32]. Nous nous attachons dans cette proposition à lier le concept de groupe multicast avec le concept de rôle du modèle RBAC, pour fournir un framework sécurisé et passant à l'échelle.

Une plateforme de sécurité intégrée

La Figure 4.6 illustre les composants du modèle de sécurité proposé. Le cœur du modèle est divisé en quatre domaines de sécurité : authentification, intégrité, confidentialité et contrôle d'accès. Chaque domaine se rapporte à une ou plusieurs techniques ou concepts sous-jacents. Par exemple, XML-Encryption est dédié à la confidentialité. Le contrôle d'accès est un mécanisme flexible de gestion des accès aux ressources. L'aspect flexible est évalué en fonction de la dynamique et de la taille de la plateforme. Alors que la dynamique implique la capacité à modifier fréquemment les droits d'accès, la taille relève plutôt du nombre d'équipements et de managers.

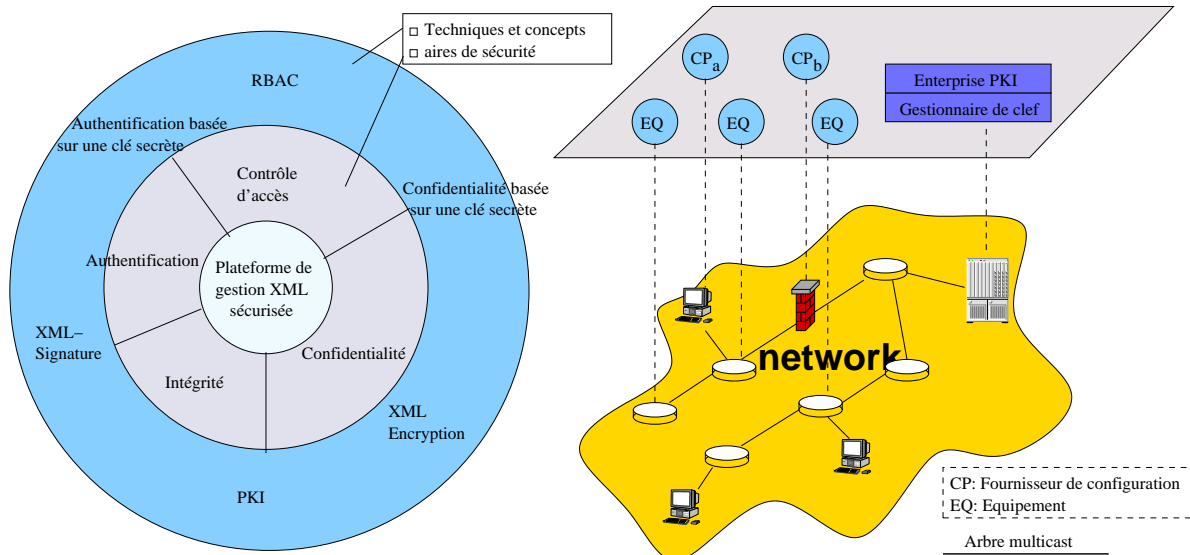


FIG. 4.6 – Les différentes composants du modèle

Nous avons proposé une solution pour le contrôle d'accès basée sur un modèle de type mécanisme de contrôle d'accès distribué basé sur les rôles (RBAC [52]). Dans cette approche, les rôles sont distribués à la volée. L'architecture délivre les tokens (clé d'authentification et de confidentialité) aux différentes entités. Pour prouver qu'elle est bien en possession de tokens donnés, une entité signe et chiffre les messages avec les clés correspondant aux rôles demandés. Pour gérer

cette dynamique, nous avons besoin d'un mécanisme d'échange de clés efficace, en particulier si l'on considère des événements comme la révocation de rôles.

Nous considérons ici un environnement de gestion basé sur XML et où les agents utilisent le protocole de configuration Netconf. C'est la raison pour laquelle nous réutilisons les paradigmes existants de sécurité XML (XML-Signature [4], XML-Encryption [44]) pour réaliser le contrôle d'accès.

Le modèle repose sur RBAC et plus particulièrement sur le concept de rôle. Chaque opération est effectuée sous la coupe d'un rôle : authentification avec la clé k_{role}^{auth} , confidentialité avec la clé k_{role}^{priv} et contrôle d'accès basé sur les rôles. La principale difficulté est de gérer les clés et de déployer les rôles de façon incrémentale. Les langages XML XML-Signature et XML-Encryption sont utilisés pour les services d'authentification et de confidentialité au niveau du message XML. Une infrastructure à clés publiques sera utilisée pour partager un secret initial et agira ainsi comme une étape de *bootstrapping*. Le passage à l'échelle implique des contraintes additionnelles que nous résolvons à l'aide des rôles.

Architecture Notre modèle de sécurité pour la configuration de réseau est construit autour de plusieurs entités principales illustrées sur la Figure 4.6. Alors que les fournisseurs de configuration (CP) agissent comme des sources de données, les équipements gérés (EQ) exécutent un agent Netconf qui reçoit des requêtes RPC de différents CP. Par exemple, CP_a et CP_b peuvent charger différentes configurations partielles dans un agent, créant ainsi une configuration multi-source. Ces configurations sont chargées en fonction de différents droits d'accès ; un agent doit être capable de décrypter des configurations entrantes et de leur attacher des droits d'accès. Nous décrivons dans cette section les mécanismes requis pour répondre à ces besoins.

Un manager RBAC agit comme un tiers d'un point de vue sécurité entre les fournisseurs de configuration et les équipements gérés. Cette entité fournit les différents services de sécurité suivants : authentification, intégrité des données, confidentialité et contrôle d'accès. Le manager RBAC stocke des politiques RBAC qui sont déployées dynamiquement sur les entités du réseau.

Pour authentifier les entités auprès du manager RBAC, ce dernier communique avec une infrastructure à clé publique (PKI) [1] qui délivre des certificats à la fois aux fournisseurs de configuration et aux équipements gérés. Ainsi, nous supposons que tous possèdent une paire de clé publique/privée pour fournir une authentification forte. Une entité s'authentifie auprès du manager RBAC en envoyant un message signé avec sa clé privée. Une fois que l'entité a été authentifiée, le manager RBAC peut distribuer les clés correspondant à un rôle particulier demandé par l'entité. Ces clés seront utilisées pour faire des opérations de gestion directement sur les agents. Ceci est équivalent au processus de Single-Sign On où les tokens distribués pour accéder au service (l'agent) sont les clés de rôle.

Nous proposons d'attacher chaque rôle conceptuel à une clé spécifique. Cela signifie que chaque entité qui veut réaliser une opération dans le contexte d'un rôle doit récupérer cette clé.

Le manager RBAC est responsable de la distribution des clés aux différentes entités. Ainsi, seules les entités qui possèdent la clé de rôle sont capables de déchiffrer et de chiffrer les messages relatifs à un rôle donné. En effet, lorsqu'une entité n'est plus attachée à un rôle donné, elle ne devrait plus avoir la possibilité de faire des opérations autorisées par un rôle. C'est pourquoi une distribution efficace des clés est nécessaire, pour redistribuer dynamiquement une nouvelle clé aux entités qui utilisent actuellement cette clé. Nous appelons cet ensemble d'entités un $KeyGroup_{role_i}$.

Nous donnons maintenant des définitions pour décrire formellement les tokens et ensembles utilisés dans cette architecture :

- E : l'ensemble des entités de l'architecture,
- $k_{public}^e/k_{private}^e$ où $e \in E$: les clés publiques et privées en possession de l'équipement e ,
- $roles(e)$ où $e \in E$: l'ensemble des rôles qui apparaissent au moins une fois dans les règles ACL de l'entité e . Cet ensemble représente toutes les clés dont un agent a besoin pour (dé)crypter les données relatives à un rôle. Une ACL est attachée à chaque nœud de la configuration XML de façon à permettre un contrôle d'accès dynamique. Une ACL est un nœud XML contenant un ensemble de règles décrivant un ensemble d'opérations disponibles pour un rôle donné. $roles(e)$ est construit en analysant toutes les ACLs de la configuration XML d'un agent et en ajoutant chaque rôle appartenant à cet ensemble. Nous définissons la syntaxe XML d'une ACL dans la section 4.2.3. L'ensemble $roles(e)$ est utilisé pour définir l'ensemble des clés nécessaires à une entité, chaque rôle étant liée à une clé,
- $k_{role_i} = \{k_{role_i}^{priv}, k_{role_i}^{auth}\}$: les clés cryptographiques symétriques correspondant au rôle $role_i$ et partagées par toutes les entités qui ont le rôle $role_i$. Chaque rôle utilise deux clés : $k_{role_i}^{priv}$ est utilisée pour crypter les messages XML envoyés sous le rôle $role_i$ alors que $k_{role_i}^{auth}$ est utilisé pour les authentifier. Chaque entité peut potentiellement envoyer des messages utilisant ces clés, à condition qu'elle soit autorisée par le manager RBAC à activer ce rôle. L'utilisation de ces clés est similaire à celle faite dans le modèle *User Security Model* [10] de SNMPv3. Une clé est utilisée pour l'authentification, alors que la seconde clé est utilisée pour le chiffrement. Cependant, une clé est attachée à un utilisateur dans SNMP, alors qu'une clé est attachée à un rôle dans notre approche,
- $KeyPath_{role_i}^e$: un chemin dans un arbre de clés. Il contient l'ensemble des clés cryptographiques dont e a besoin pour le rafraîchissement de clés pour le rôle $role_i$. Le $KeyPath$ de l'entité d est illustré en gris dans le second arbre de la Figure 4.7. d a activé le rôle $role_A$. La clé racine est utilisée pour chiffrer les données lorsqu'un message est envoyé sous le rôle correspondant. Les clés feuilles sont connues seulement par le membre sous-jacent et sont transmises en utilisant la PKI. Les clés intermédiaires sont utilisées pour envoyer les nouvelles clés d'une façon sûre et efficace, en minimisant le nombre de messages de renouvellement de clés. Nous décrivons le mécanisme de maintien des clés dans la section 4.2.3. L'ensemble $KeyPath_{role_A}^d$ est égal à l'ensemble $\{kad, kcd, kd\}$. Alors que kd est la clé initiale connue uniquement par d et obtenue via la PKI, kad est égal à k_{role_A} et kcd est une clé intermédiaire connue par c et d ,
- $KeyGroup_{role_i} = \{e \in E | role_i \in roles(e)\}$: l'ensemble des équipements qui appartiennent à $role_i$.

Le manager RBAC interagit aussi avec un gestionnaire de clé. Ce dernier gère différents arbres de clés (cf. Figure 4.7). Celles-ci sont distribuées aux fournisseurs de configuration et aux équipement gérés en utilisant le multicast IP pour assurer une distribution efficace. Tous les agents et les managers sont des membres d'un même groupe multicast de façon à ce qu'ils reçoivent les messages de rafraîchissement de clé.

Un arbre de clés est lié à un groupe multicast et correspond à un rôle. Les clés racine sont utilisées pour authentifier et chiffrer les messages Netconf entre les différentes entités. Les clés intermédiaires sont utilisées pour rafraîchir les clés efficacement. Les feuilles d'un arbre correspondent à une entité particulière. Une entité doit avoir la connaissance de toutes les clés de la feuille à la racine pour que le mécanisme de rafraîchissement fonctionne. La PKI est utilisée pour échanger la clé feuille initiale de l'entité. Une fois que cette clé a été reçue par un entité, la paire de clés publique/privée n'est plus utilisée pour optimiser l'efficacité de l'architecture. Nous discutons cela à la fin de la section.

Finalement, un administrateur RBAC est responsable d'éditer la politique de contrôle d'accès de l'entité *manager RBAC*. Ce dernier maintient les associations liant les utilisateurs aux rôles.

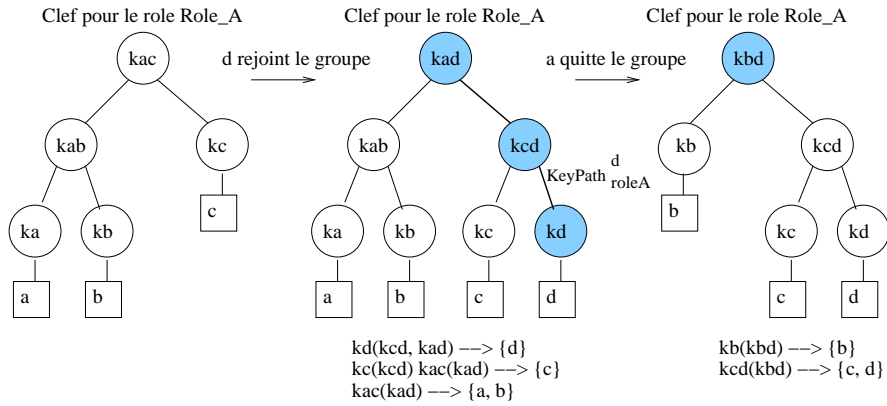


FIG. 4.7 – Représentation d’un arbre de clé

Lorsqu’une entité demande l’activation d’un rôle, le manager RBAC cherche dans la base de données des relations utilisateur-rôle pour décider si l’utilisateur peut activer le rôle.

Concrètement, un arbre de clé est utilisé pour chaque rôle pour fournir un rafraîchissement efficace des clés. Cela permet de rafraîchir la clé k_{role_i} des entités du groupe $KeyGroup_{role_i}$ sans utiliser leurs paires de clés publiques/privées. Pour rafraîchir les clés de groupe efficacement, nous utilisons l’algorithme *logical key hierarchy* (LKH) [76, 80, 77], qui minimise le nombre de clés à rafraîchir en utilisant le multicast.

Nous décrivons maintenant les interactions entre les fournisseurs de configuration, les équipements gérés et le manager RBAC :

- Quand un fournisseur de configuration souhaite réaliser des opérations sur un équipement géré, il doit activer un rôle. Pour cela, il envoie une requête au manager RBAC. Cette requête contient le `roleID` du rôle souhaité ainsi que l’identité de l’émetteur pour que le manager RBAC puisse retrouver sa clé publique. Ce message est chiffré avec la clé publique du manager RBAC de telle sorte que personne d’autre que le manager RBAC puisse lire le message. L’émetteur est également authentifié en joignant une signature électronique réalisée avec la clé privée de l’émetteur e . Voici la description formelle du message envoyé au manager RBAC pour souscrire à un rôle $roleID$:

$$k_{public}^{RBACmanager}(roleID, identity, k_{private}^e(hash(roleID, identity)))$$

Tout cela est rendu possible par une PKI qui délivre des certificats à la fois aux fournisseurs de configuration et aux équipements gérés,

- Si le message est valide, le manager RBAC cherche le rôle dans l’ensemble des rôles assignés à l’utilisateur. Si le rôle (`roleID`) demandé peut être activé par l’utilisateur, le manager RBAC interroge le gestionnaire de clés pour récupérer les clés relatives à ce groupe. Le manager RBAC doit envoyer $KeyPath_{role_i}^e$ de façon sécurisée au fournisseur de configuration correspondant. C’est pourquoi ke , la clé feuille de l’entité e , est hachée et signée avec la clé privée du manager RBAC, ce qui assure l’authenticité du message. De plus, le message est chiffré avec la clé publique du récepteur, ainsi personne ne peut déchiffrer le message :

$$k_{public}^e(ke, k_{private}^{RBACmanager}(hash(ke)))$$

Ensuite, $KeyPath_{role_i}^e$ est envoyé en utilisant le secret partagé ke . Ainsi, les services d’authentification et de confidentialité sont assurés lors de la transmission du `KeyPath` :

$$ke(KeyPath_{role_i}^e)$$

- Lorsqu'un rôle a été activé, un manager Netconf est capable d'envoyer des messages sécurisés à tous les équipements. On entend par sécurisé : confidentialité, authentification et intégrité en utilisant la clé de rôle k_{role_i} . Les agents ont un système de contrôle d'accès qui limite les opérations qu'un rôle peut effectuer. Formellement, une requête envoyée d'une entité manager et une entité agent sous un rôle $role_i$ est la suivante :

$$k_{role_i}(\langle rpc \rangle \dots \langle /rpc \rangle)$$

Ces services sont décrits dans la prochaine section,

- Quand une entité désactive un rôle, le manager RBAC en informe le gestionnaire de clés pour qu'il retire l'entité de l'arbre de clés. Toutes les clés de l'entité à la racine de l'arbre doivent être mises à jour pour que l'entité ne puisse plus réaliser de opérations sur les équipements gérés.

Extension de sécurité pour le protocole Netconf

Authentification, intégrité et confidentialité Pour sécuriser les échanges dans Netconf, il faut partager un secret entre les managers et les agents. En réalité, deux clés vont être utilisées, l'une pour le service d'authentification et l'autre pour la confidentialité. Une fois le manager Netconf authentifié auprès du manager RBAC, il communique avec l'agent en disposant des droits correspondant au rôle acquis précédemment. Le manager possède les deux clés k_{role}^{priv} et k_{role}^{auth} pour ce rôle. Chaque message Netconf est envoyé dans le contexte d'un rôle et non d'un utilisateur. Ceci facilite le contrôle d'accès pour l'agent et la distribution et le rafraîchissement des clés pour le gestionnaire de clés.

Chaque message Netconf est chiffré avec la clé k_{role}^{priv} et stockée dans un élément `<EncryptedData>` conformément à la recommandation XML-Encryption [44] du W3C⁴.

Concernant le service d'authentification, un élément XML est alors signé en utilisant la clé k_{role}^{auth} . Le résultat est un élément `<Signature>` conforme à la recommandation XML Digital Signature (XML-DS [4]) du W3C. Il contient la description des algorithmes utilisés pour construire la signature ainsi que les informations permettant de retrouver les tokens de sécurité à utiliser pour vérifier la signature (certificats, clés, ...). La Figure 4.8 illustre l'encapsulation XML correspondant aux processus d'authentification et de chiffrement.

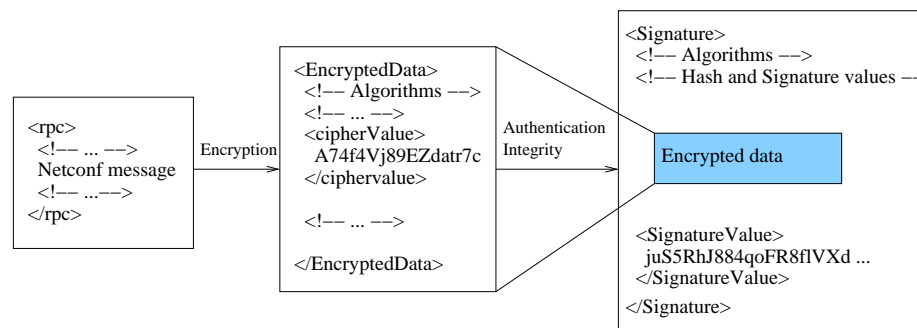


FIG. 4.8 – Encapsulation du message Netconf

⁴World Wide Web Consortium, <http://www.w3.org>

Contrôle d'accès De nombreux modèles de contrôle d'accès ont été adaptés pour la protection des données au format XML. La plupart reposent sur les rôles [52] ou proposent des adaptations vers RBAC [52, 61, 3]. Ces modèles utilisent souvent des permissions dont les ressources sont décrites à l'aide d'expressions XPath. Il existe deux approches pour stocker la politique de contrôle d'accès : *inline* et *out-of-line*. Dans la première approche, les permissions sont attachées à chaque donnée ou chaque nœud d'un arbre logique. C'est, par exemple, l'approche des systèmes Unix. Avec l'approche *out-of-line*, la politique de contrôle d'accès est centralisée. Elle décrit les droits d'accès pour l'ensemble des données. Par exemple, XACML, CLI, VACM suivent cette approche. Ici, nous choisissons la première approche en définissant des ACLs pour chaque nœud XML. Il n'y a pas une grande différence en terme de coûts de maintenance ou de complexité. Il suffit d'utiliser des mécanismes de propagation pour simplifier la gestion des droits d'accès, par exemple, lors de la création d'une ressource. Avec une politique centrale, il peut être plus difficile de visualiser quels sont les utilisateurs autorisés à accéder à telle ou telle ressource précise. Mais des outils facilitent la définition des politiques et cachent la complexité et la syntaxe sous-jacente. Les ACL définies dans la suite se basent sur les rôles et non sur les utilisateurs.

Opérations Netconf	Primitive de base de l'ACL
<get-config>	<get>
<edit-config> (operation=merge)	<merge>
<edit-config> (operation=replace)	<replace>
<edit-config> (operation=create)	<create>
<edit-config> (operation=delete)	<delete>
<copy-config>	<delete> et <replace> et <create>

TAB. 4.1 – Conversion des opérations Netconf sur des primitives bas-niveau

Les opérations Netconf sont organisées en terme de primitives de bas niveau, <get>, <merge>, <replace>, <create> et <delete>, comme illustré par la Table 4.1. Sur la gauche du tableau sont définies les opérations Netconf, et sur la droite sont données les opérations de bas niveau correspondantes. Quand une requête Netconf se rapporte à un nœud particulier, le processus de contrôle d'accès consiste à vérifier que les opérations correspondantes sont bien dans l'ACL. Le fait qu'un rôle soit autorisé à exécuter une opération particulière sur un nœud ne signifie pas forcément qu'il peut faire cette opération sur les nœuds enfants ou parents. Cela simplifie grandement le processus de contrôle d'accès et donne un niveau de granularité particulièrement fin. Chaque est ainsi responsable de son propre contrôle d'accès.

La Figure 4.9 montre comment une liste de contrôle d'accès est ajoutée à un nœud XML. Elle définit que le rôle *myFirstRole* peut faire les opérations <get/>, <replace/> et <merge/> sur ce nœud mais ne fait aucune supposition sur les parents et les enfants de ce nœud. Le second rôle établit que le rôle *mySecondRole* ne peut faire que l'opération <get/> sur ce nœud XML. Cet exemple illustre le fait que l'utilisation des rôles limite considérablement le nombre d'ACLs. Les ACLs sont créées en même temps que le nœud lui-même et avec des règles par défaut relatives au rôle couramment actif.

Par l'utilisation de ce modèle, il est possible de construire une réponse Netconf en fonction des droits d'accès du rôle utilisé. Tout d'abord, dans le cas de l'opération *Read*, le document XML résultat est construit sans se préoccuper des règles de contrôle d'accès. On obtient ainsi une sous-partie de la configuration. Ensuite, les branches pour lesquelles le rôle ne peut pas réaliser

```

1 <interface>
2   <ACL>
3     <rule roleRef="myFirstRole">
4       <operations><get/><replace/><merge/></operations>
5     </rule>
6     <rule roleRef="mySecondRole">
7       <operations><get/></operations>
8     </rule>
9   </ACL>
10  <name>Ethernet0/0</name>
11  <mtu>1500</mtu>
12 </interface>

```

FIG. 4.9 – Une ACL associée à un nœud particulier

l'opération sont coupées. L'étape suivante consiste à chiffrer l'arbre avec la clé correspondant au rôle en cours d'utilisation : k_{role}^{priv} . Puis, une signature (XML-Signature) est construite avec la clé k_{role}^{auth} . Dans le cas d'une opération d'écriture, des vérifications permettent de s'assurer que ce rôle est autorisé à écrire dans les nœuds concernés. L'élément **ACL** est un fils direct du nœud qu'il protège.

Gestion des clés Le gestionnaire de clés est l'entité responsable de la gestion des arbres de clés pour chaque groupe. Un arbre de clé est maintenu pour chaque rôle, et la racine de l'arbre est la clé du rôle. Puisque plusieurs entités peuvent activer le même rôle simultanément, chacune doit recevoir la clé de ce rôle. Comme la désactivation d'un rôle par un membre peut se produire à tout moment, il est nécessaire de mettre en place un mécanisme de rafraîchissement de clé pour empêcher, d'une part, un membre nouvellement arrivé de lire les messages passés et, d'autre part, un membre ancien de continuer à lire les messages après la désactivation du rôle.

Plusieurs algorithmes de renouvellement de clés ont déjà été proposés parmi lesquels *Marks* [13], *LKH* [76, 80, 77] et *One-way Function Tree* (OFT). Nous utilisons LKH qui est le plus adapté pour le passage à l'échelle, les performances et la dynamique. Marks suppose que la durée d'appartenance au groupe est connue au début ce qui n'est pas envisageable dans notre cas. LKH+ est un algorithme de renouvellement de clé ayant une complexité en $\log(n)$ que ce soit pour les messages *join* ou *leave*. La clé du groupe multicast (dans notre cas identique à la clé du rôle) est la racine de l'arbre de clés. Chaque membre est une feuille et a la connaissance de toutes les clés depuis la feuille jusqu'à la racine. Cet ensemble de clés est noté *KeyPath*. Sa propre clé (la feuille) est distribuée par le gestionnaire de clé à l'aide d'une paire de clé publique/privée.

Pour illustrer le fonctionnement de l'approche LKH, considérons les cas où un membre rejoint ou quitte le groupe. La Figure 4.7 présente trois arbres correspondant au même rôle : le premier est l'arbre de clé initial et contient ici trois membres $\{a, b, c\}$. Si un nouveau membre rejoint le groupe, l'arbre de clés doit être mis à jour aussi efficacement que possible. d doit recevoir la clé racine pour pouvoir manipuler les prochains messages diffusés par le groupe multicast. Cette clé racine est modifiée par le gestionnaire de clé pour que d n'ait pas accès aux anciens messages. La PKI est utilisée ici pour transmettre la clé symétrique kd au membre d chiffrée avec la clé publique de d . d est alors le seul à pouvoir décrypter ce message d'initialisation. Ensuite, les autres clés de l'ensemble $KeyPath_{role_A}^d$ sont envoyées à d et aux autres entités comme illustré sur la Figure. La transition du deuxième au troisième arbre illustre le cas inverse d'un membre a quittant le groupe. L'entité a a la connaissance de ka , kab et kad . Puisque les clés ka et kab

disparaissent de l'arbre lorsque *a* quitte le groupe, seul *kad* doit être mise à jour de sorte que *a* n'ai plus accès au trafic futur. C'est pourquoi le gestionnaire des clés génère une nouvelle clé *kbd* et l'envoie à toutes les entités sous-jacentes, c'est-à-dire *b*, *c* et *d*, de telle sorte à minimiser le nombre de messages. Par exemple, un seul message est nécessaire pour envoyer *kbd* aux entités *c* et *d* puisque elles partagent une clé commune *kcd*. Les deux messages sont donnés sous le troisième arbre.

L'utilisation de clé symétrique est évidemment beaucoup plus performante que les clés publiques/privées.

4.3 Synthèse

Dans ce chapitre, nous avons proposé un algorithme de conversion et de distribution *top-down*. L'algorithme est capable d'adapter un politique de contrôle d'accès basée sur les rôles à un modèle basé sur des niveaux hiérarchiques de taille limitée. Une fois adaptée au modèle, cette politique est déployée sur les équipements supportant l'interface en ligne de commande et assure ainsi la cohérence globale leurs politiques de sécurité.

Nous avons également proposé une architecture pour Netconf dans laquelle l'appartenance à un rôle se concrétise par la connaissance des tokens de sécurité liés au rôle. L'activation d'un rôle par un utilisateur se traduit par la distribution de la *clé de rôle*. Les échanges sont cryptés et authentifiés avec la clé de rôle et le contrôle d'accès est réalisé par rapport au rôle utilisé, lui-même identifié par la clé. Des algorithmes conçus à l'origine pour le multicast sont réutilisés de façon à optimiser la distribution et le rafraîchissement des clés. Le rapprochement est possible grâce à la similitude entre les groupes multicast et les rôles du modèle RBAC. Dans notre proposition, la politique de sécurité devient cohérente globalement, car elle s'applique quelque soit l'équipement géré. La principale limite de cette approche est que l'on perd la responsabilité des utilisateurs, car les opérations sont faites dans le cadre d'un rôle et non plus d'un utilisateur. Cependant, on pourrait en plus joindre à chaque message une signature générée avec la clé de l'utilisateur.

Une approche pour assurer la cohérence locale

Sommaire

5.1	Introduction	59
5.1.1	Contexte	59
5.1.2	Définition formelle du modèle RBAC	60
5.2	Formalisation des modèles de contrôle d'accès hétérogènes . . .	61
5.2.1	Définition pour la convergence du modèle VACM vers le modèle RBAC	61
5.2.2	Définition de la convergence entre le modèle de sécurité de CLI vers le modèle RBAC	63
5.2.3	Définition pour la convergence du modèle TR-069 vers le modèle RBAC	66
5.2.4	Implantation de l'algorithme	67
5.3	Détection de conflits par comparaison et approximations	67
5.3.1	Définition des opérations de comparaison	67
5.3.2	Algorithme pour la détection de conflits de politique	71
5.3.3	Conservation de l'équivalence de politique lors d'opérations de configuration	71
5.4	Synthèse	72
5.4.1	Un modèle formel pour vérifier la cohérence des politiques de contrôle d'accès	72
5.4.2	Limites et alternatives	73

5.1 Introduction

5.1.1 Contexte

Dans un monde parfait, deux interfaces de gestion, ouvertes sur un même équipement, devraient allouer les mêmes privilèges à une entité donnée. Si l'entité *A* peut gérer la configuration RIP d'un routeur via CLI, elle devrait aussi être autorisée à la gérer via Netconf. Inversement, si CLI n'autorise pas l'entité *A* à consulter les règles de pare-feu, Netconf devrait l'interdire aussi. Ce chapitre traite principalement de ce problème de cohérence du contrôle d'accès et, en particulier, cherche à démontrer formellement que ces politiques hétérogènes sont cohérentes ou non.

Le paragraphe suivant propose l'utilisation du modèle RBAC (cf. [52, 59]) comme modèle de convergence. La section 5.2 propose, quant à elle, une définition formelle des modèles de contrôle d'accès des principales plateformes de supervision ainsi que leur conversion vers le modèle commun. Ce schéma de convergence est illustré par la Figure 5.1. La section 5.3 définit une approche de détection de conflit dans les politiques RBAC obtenues après conversion. Elle donne aussi un éclairage sur les problèmes rencontrés, issus principalement de l'aspect dynamique de RBAC, c'est-à-dire l'activation et la désactivation des rôles. Les conflits étudiés dans cette section sont de type inter-politiques conformément à la taxonomie présentée dans [2, 38] qui se focalise sur les pare-feux mais reste applicable dans notre cas.

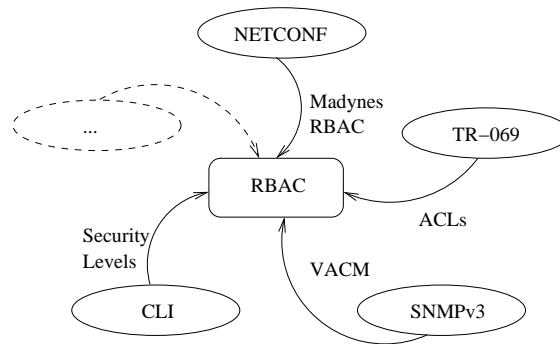


FIG. 5.1 – Schéma de convergence des modèles de contrôle d'accès

5.1.2 Définition formelle du modèle RBAC

L'objectif de ce chapitre est de prouver l'équivalence des politiques de contrôle d'accès venant d'un ensemble de plateformes de gestion. L'équivalence de deux politiques de contrôle d'accès assure qu'un utilisateur a exactement les mêmes privilèges quelle que soit l'interface de gestion utilisée. Pour atteindre cet objectif, la première étape consiste à être capable d'exprimer ces politiques dans un référentiel commun, avec un modèle de contrôle d'accès, un langage et une représentation des ressources communs. Cette section propose une approche pour réaliser cette première étape en s'appuyant fortement sur le travail fait dans [29].

Nous avons choisi le modèle de contrôle d'accès basé sur les rôles comme modèle de convergence, ceci pour plusieurs raisons. La première raison est que RBAC est un standard ITU⁵. Ensuite, le modèle RBAC a de nombreux avantages : passage à l'échelle en terme de nombre d'utilisateurs, organisation claire des rôles (ou fonction) de chaque utilisateur, facilité de maintenance grâce au concept de rôle et à leur organisation hiérarchique.

Les définitions suivantes sont extraites du standard décrivant RBAC (Core RBAC [52]). Nous rappelons ici les notations qui seront réutilisées plus tard dans ce chapitre. Ces définitions décrivent, en premier lieu, l'ensemble des composants du modèle RBAC puis les relations fonctionnelles qui les lient : assignation des utilisateurs à des rôles, puis des permissions à des rôles. La partie dynamique du modèle n'est pas décrite ici car elle ne correspond pas à des composants persistants.

- USERS, ROLES, OPS, et OBS (les utilisateurs, rôles, opérations et objets respectivement),

⁵Union internationale des télécommunications

- $UA \subseteq USERS \times ROLES$, une relation n-n entre l'ensemble des utilisateurs et des rôles,
- $assigned_users : (r : ROLES) \rightarrow 2^{USERS}$, la conversion du rôle r vers un ensemble d'utilisateurs. Formellement : $assigned_users = \{u \in USERS | (u, r) \in UA\}$,
- $PRMS = 2^{(OPS \times OBS)}$, l'ensemble des permissions,
- $PA \subseteq PRMS \times ROLES$, la relation n-n entre les permissions et les rôles,
- $assigned_permissions(r : ROLES) \rightarrow 2^{PRMS}$, la conversion du rôle r vers un ensemble de permissions. Formellement : $assigned_permissions = \{p \in PRMS | (p, r) \in PA\}$,
- $Op(p : PRMS) \rightarrow op \subseteq OPS$, la conversion d'une permission vers une opération, qui donne l'ensemble des opérations associées à la permission p,
- $Ob(p : PRMS) \rightarrow ob \subseteq OBS$, la conversion d'une permission vers un objet, qui donne l'ensemble des objets associées à la permission p.

Dans notre cas, les objets (ou ressources) sont représentés selon la représentation compacte utilisée dans [29]. Par exemple, $(host : /a/b)$. Nous étendons cette notation avec le caractère spécial * qui signifie *tout élément*. Par exemple, l'expression suivante devient permise : $(host : /a/* /c)$. Soit la fonction *compact()* qui convertit une commande en sa représentation compacte. De plus, nous faisons l'hypothèse que l'ensemble des opérations possibles se limite à "r" et "w" pour les opérations de lecture et d'écriture. Dans ce modèle, une permission (op, ob) où ob=(host : /a/b) donne accès tout le sous-arbre identifiée par la racine /a/b. Le comportement est calqué sur le processus de décision du modèle *View-Based Access Control Model* (VACM [79]) : avoir une permission en lecture sur le nœud *system* avec un masque ("FF") signifie que tous les objets sous ce nœud sont accessible en lecture. Cette approche évite d'avoir à définir spécifiquement tous les nœuds du sous-arbre. On ne donne que la (ou les) racine du sous-arbre.

5.2 Formalisation des modèles de contrôle d'accès hétérogènes

5.2.1 Définition pour la convergence du modèle VACM vers le modèle RBAC

Définition formelle du modèle VACM

Le modèle VACM est défini comme suit : il consiste en un ensemble d'utilisateurs, de groupes, de modèles de sécurité, de niveaux de sécurité, de vues et de permissions. La façon dont ces composants s'articulent entre eux est définie par les tables VACM suivantes : *VacmSecurityToGroupTable*, *VacmViewTreeFamilyTable*, *VacmAccessTable*.

- $U = \{USMusers\} \approx \{SecurityName\}$ où *SecurityName* est le nom de l'utilisateur dans le modèle USM,
- $G = \{vacmGroupName\}$,
- $O = \{OID(s)\}$,
- $M = \{Mask\}$,
- $B = \{"included", "excluded"\}$,
- $V = \{View\} \in 2^{(O \times M \times B)}$. $2^{(O \times M \times B)}$ est l'ensemble des combinaisons possibles de taille quelconque de l'ensemble $O \times M \times B$
- $SM = \{SecurityModel\} = \{"v1", "v2c", "usm", \dots\}$,

- $SL = \{SecurityLevel\} = \{ "noAuthNoPriv", "authNoPriv", "authPriv" \}$,
- $C = \{Context\}$,
- $UA_{VACM} \in 2^{(U \times SM \times G)}$. $2^{(U \times SM \times G)}$ est l'ensemble des combinaisons possibles de taille quelconque de l'ensemble $U \times SM \times G$, qui est un ensemble de couples de type (user1, "usm", group1),
- $PA_{VACM} \in 2^{(G \times SM \times SL \times V \times V \times V)}$. $2^{(G \times SM \times SL \times V \times V \times V)}$ est l'ensemble des combinaisons possibles de taille quelconque de l'ensemble $G \times SM \times SL \times V \times V \times V$, qui est un ensemble de couples de type (group1, "usm", "authPriv", mib2view, mib2view, mib2view). Pour simplifier, PA_{VACM} est un ensemble de relations entre un groupe et des permissions.

Conversion formelle du modèle VACM vers le modèle RBAC

Cette section décrit la conversion du modèle VACM vers le modèle RBAC. Les composants du modèle RBAC et leurs associations sont obtenues à partir des tables de VACM à l'aide d'un algorithme de conversion. Pour faciliter la compréhension des notations, l'annotation *construct* indique explicitement que l'ensemble correspondant est le résultat d'une conversion :

- Chaque utilisateur VACM est directement converti vers un utilisateur RBAC :

$$USERS_{VACM}^{Construct} = U$$

- Chaque groupe VACM est directement converti vers un rôle RBAC :

$$ROLES_{VACM}^{Construct} = G$$

- L'assignation des utilisateurs à des rôles est construite à partir de l'ensemble UA_{VACM} . Définissons la fonction de translation f_1 telle que :

$$\forall ua = (u, s, g) \in UA_{VACM}, f_1(ua) = f_1((u, s, g)) = (u, g)$$

La définition précédente établit que le modèle de sécurité est ignoré, d'une part pour simplifier la transformation et d'autre part parce qu'on ne veut pas rendre le contrôle d'accès dépendant du modèle de sécurité utilisé par ailleurs. Formellement, f_1 est une projection mathématique d'un ensemble à trois dimensions vers un ensemble à deux dimensions. Généralisons cette définition à l'ensemble complet des assignations UA_{VACM} :

$$UA_{VACM}^{Construct} = f_1(UA_{VACM})$$

- Chaque OID est traduit, dans un premier temps, vers une notation avec les noms de nœuds complets. Par exemple, *1.3.6.1.2.1.2.1* se traduit en */iso/org/dod/internet/mgmt/mib-2/interfaces/ifNumber*. Cette conversion est grandement facilitée par l'utilitaire *snmptranslate -Of*. En fonction du masque de la vue considérée, des jokers (étoiles ou wildcards) peuvent apparaître dans la représentation de la ressource. Par exemple, la vue

$$(name, oid, mask) = (myview, 1.3.6.1.2.1.2.1, 11011110)$$

est convertie vers la représentation suivante : */iso/org/*/internet/mgmt/mib-2/interfaces/**. La deuxième étape est la projection de cette expression vers le modèle de données générique. Cette fonction de conversion est notée :

$$dmapper : O \rightarrow OBS$$

Nous supposons ici que toute représentation générée par la première étape peut être traduite vers une représentation dans notre modèle de données de référence. C'est le rôle de la fonction *dmapper*. Une autre hypothèse importante est que l'algorithme de conversion présenté ignore les parties *excluded* de la vue considérée et issue de la table *VacmView-TreeFamilyTable*. Nous faisons ce choix non seulement pour simplifier le processus mais aussi parce que c'est une recommandation du standard RBAC d'autoriser uniquement les permissions positives. Par défaut, un rôle n'a aucune permission. Les seuls privilèges sont ceux explicitement spécifiés dans la politique de sécurité. Cependant, combiner les parties incluses et exclues pour construire des permissions uniquement positives reste possible.

- Les permissions sont construites à partir de l'ensemble PA_{VACM} . Chaque tuple de PA_{VACM} est converti en un ensemble de trois permissions. Pour illustrer cela, considérons l'exemple de la case appartenant à la colonne *readView* avec comme valeur *systemView*. Cette case est convertie en une permission (*systemView*, "r") où la vue *systemView* est elle-même convertie selon l'item précédent. Définissons la fonction de conversion $f_2 : PA_{VACM} \rightarrow PRMS$ telle que :

$$\forall pa = (g, m, l, v_1, v_2, v_3) \in PA_{VACM}, f_2(pa) = f_2((g, m, l, v_1, v_2, v_3)) = \{(v_1, "r"), (v_2, "w"), (v_3, "n")\}$$

Construisons maintenant l'ensemble des permissions :

$$PRMS_{VACM}^{Construct} = f_2(PA_{VACM})$$

- Les assignations des permissions et des rôles sont construites à partir de l'ensemble PA_{VACM} . Définissons la fonction de conversion

$$f_3 : PA_{VACM} \rightarrow PA_{VACM}^{Construct} \in PA$$

en utilisant le produit scalaire noté ".", tel que :

$$\forall pa = (g, m, l, v_1, v_2, v_3) \in PA_{VACM}, f_3(pa) = f_3((g, m, l, v_1, v_2, v_3)) = (g.f_2(pa))\}$$

Construisons l'ensemble des assignations :

$$PA_{VACM}^{Construct} = f_3(PA_{VACM})$$

Par ailleurs, ajoutons la contrainte de séparation dynamique des devoirs (DSD) : un seul rôle peut être activé à la fois dans cette politique. La raison est la suivante : pour simuler le comportement de SNMP, un utilisateur peut appartenir à un seul groupe VACM pour une requête donnée car le modèle de sécurité est donné. C'est pourquoi la session RBAC ne peut avoir qu'un seul rôle actif à tout instant. Cependant, il n'y a pas de contraintes statiques additionnelles car VACM ne le permet pas. Donc l'algorithme de conversion ne génère aucune contrainte SSD.

5.2.2 Définition de la convergence entre le modèle de sécurité de CLI vers le modèle RBAC

Définition formelle du modèle de sécurité de CLI

Le modèle de sécurité de CLI est basé sur une hiérarchie de privilèges dans la plage de niveau de sécurité allant de 0 à 15. Les niveaux les plus importants sont les niveaux 1, aussi appelé niveau utilisateur et le niveau 15, appelé niveau privilégié. Un utilisateur se déplace d'un niveau de sécurité à un autre grâce aux commandes **enable** et **disable**. Puisque les niveaux de

sécurité sont hiérarchiques, accéder à un niveau donne accès non seulement aux permissions de ce niveau mais aussi à toutes les permissions des niveaux inférieurs. Chaque niveau est associé à un ensemble de commandes autorisées. Définissons formellement les composants de ce modèle :

- $U = \{CLIusers\}$
- $LEVELS = \{ \text{niveaux de sécurité CLI} \}$
- $UA = 2^{U \times LEVELS}$
- $CLASS = \{Exec, configuration\}$
- $COMMANDS = \{ \text{commandes CLI} \}$
- $PA = 2^{CLASS \times COMMANDS \times LEVELS}$

Le code illustré Figure 5.2 donne un exemple de configuration de privilèges dans un équipement réseau CLI. Pour chaque utilisateur, la politique définit optionnellement son niveau de privilège maximum et son mot de passe. Par exemple, *john* qui a le mot de passe *doe* est autorisé à atteindre le niveau 9. Ensuite, cette configuration définit des relations entre les commandes et les niveaux, de façon similaire à l'assignation des permissions aux rôles dans le modèle RBAC. Par exemple, configurer *snmp-server community* nécessite d'être en mode configuration et d'être au moins au niveau 8.

```

1 username john privilege 9 password 0 doe
2 username six privilege 6 password 0 six
3 username poweruser privilege 15 password poweruser
4 username inout password inout
5 privilege configure level 8 snmp-server community
6 privilege configure level 3 command ping
7 privilege configure level 5 mode enable command configure
8 privilege configure level 15 command aaa
9 privilege configure level 15 command aaa-server
10 privilege configure level 15 command access-group
11 privilege configure level 15 command activation-key

```

FIG. 5.2 – Extrait de configuration CLI relatif à la sécurité

Conversion formelle du modèle de sécurité CLI vers le modèle RBAC

Cette section décrit la conversion des composants du modèle de sécurité CLI vers les composants du modèle RBAC. Comme souligné précédemment, les niveaux de sécurité sont très similaires à une hiérarchie de rôles linéaire. Ainsi, une solution directe consiste à convertir chaque niveau i vers un rôle $role_i$. Ensuite, chaque rôle $role_i$ a un rôle junior $role_{i-1}$ et un seul. La conversion des commandes vers une représentation abstraite est plus subtile. Tout d'abord, pour se conformer à la représentation des ressources que nous avons choisie précédemment, l'ensemble des commandes et des valeurs doit être représenté sous forme de structure de données arborescente. [29] décrit un modèle de configuration Meta-CLI pour la gestion des équipements réseaux. Dans un premier temps, notre approche repose sur ce travail de façon à construire un arbre des ressources gérées et définir des politiques de contrôle d'accès générique basées sur cet arbre. Définissons formellement la conversion du modèle de sécurité de CLI vers le modèle commun :

- Chaque utilisateur CLI est converti directement vers un utilisateur RBAC :

$$USERS_{VACM}^{Construct} = U$$

- Chaque niveau de privilège CLI est converti directement vers un rôle RBAC. Le nombre de rôles créés est donc égal au nombre de niveaux de sécurité initial :

$$ROLES_{VACM}^{Construct} = LEVELS$$

La hiérarchie des rôles est définie comme suit :

$\forall l_i, l_{i-1} \in LEVELS$ tels que $l_i > l_{i-1}$, $f(l_i) = role_i$ et $juniorRoles(role_i) = \{f(l_{i-1})\}$, où $juniorRoles$ a la même signification que dans [52] et désigne l'ensemble des rôles juniors du rôle fourni en paramètre.

- L'assignation des utilisateurs aux rôles est construite à partir des commandes de configuration CLI *username*. L'algorithme de conversion établit que chaque utilisateur appartient au rôle correspondant à son niveau de privilège :

$$\forall ua = (user, l_i) \in UA, f_1(ua) = f_1((user, l_i)) = (user, f(l_i))$$

- Les permissions sont construites à partir des représentations obtenues. Chaque commande est traduite en une représentation compacte selon le modèle présenté dans [29]. Par exemple, **snmp-server community** se traduit par deux permissions (read et write), parce que le fait d'avoir le privilège de configuration via une commande donne l'accès à la fois en lecture et en écriture :

$$\begin{aligned} (A : /configuration/snmp-server/community, "r") \\ (A : /configuration/snmp-server/community, "w"). \end{aligned}$$

$$\forall pa = (configuration, l_i, command) \in PA, f_5(pa) = f_5((configuration, l_i, command)) = compact(command)$$

Généralisons cette définition à l'ensemble des permissions converties :

$$PRMS_{CLI}^{Construct} = f_5(PA_{CLI})$$

- La relation entre les rôles et les permissions est construite à partir de la commande CLI *privilege* qui lie une commande et un niveau de sécurité. Chaque privilège donne lieu à deux permissions dans l'ensemble des permissions $PA_{CLI}^{Construct}$: la première correspond à la permission en lecture et la seconde à l'écriture. Chaque commande est ainsi transcrite selon la méthode *compact()*. Formellement :

$$\forall pa = (configuration, l_i, command) \in PA, f_2(pa) = f_2((configuration, l_i, command)) = \{(compact(command), "r"), (compact(command), "w")\}$$

On peut maintenant généraliser cette définition à l'ensemble des permissions :

$$PA_{CLI}^{Construct} = f_2(PA_{CLI})$$

Comme pour SNMP, la contrainte de séparation dynamique des pouvoirs est la suivante : seul un rôle peut être actif à un instant t . En effet, activer deux rôles n'a pas de sens puisque l'un des deux sera toujours un rôle junior ou senior de l'autre. Il est donc inutile d'empêcher l'activation simultanée des rôles 3 et 5 car le rôle 5 hérite de toute façon du rôle 3. En ce qui concerne la séparation statique des pouvoirs, la contrainte générale est la suivante : un utilisateur ne peut être assigné qu'à un rôle par construction puisqu'un utilisateur appartient à un niveau de privilège et un seul dans la configuration CLI.

5.2.3 Définition pour la convergence du modèle TR-069 vers le modèle RBAC

TR-069 [37] est un protocole de gestion dédié aux technologies ADSL et aux équipements domestiques grand public. Ce protocole repose sur le paradigme manager/agent dans lequel le manager est appelé un serveur d'auto-configuration (ACS) et l'agent est dénommé équipement propriété du client (CPE). TR-069 est un protocole de type RPC et fonctionne sur le protocole SOAP initialement développé dans le cadre des services web. TR-069 définit un modèle de données arborescent et un schéma d'adressage sous forme de notation pointée, très fortement inspiré de SNMP. Un ensemble d'opérations simples a été proposé pour récupérer (get) ou modifier (set) les valeurs des différents paramètres de configuration ainsi que les meta-informations qui leur sont associées. Ces dernières contiennent notamment l'information de contrôle d'accès utile à notre modélisation.

Définition formelle du modèle de sécurité de TR-069

Dans TR-069, la structure des paramètres est constituée de différents attributs parmi lesquels une ACL (Access Control List) est définie. Cette ACL est une liste d'entités ayant l'accès en écriture sur le paramètre considéré. Si cette liste est vide, cela signifie que seul l'ACS a le droit d'écriture sur le paramètre. Pour l'instant, dans la spécification de TR-069, la seule entité définie est *Subscriber*, qui en un sens peut être considérée comme un rôle.

- $U = \{ACS, Subscriber\}$
- $O = OBJECTS$
- $ACL \subseteq O \times U$, une relation entre un paramètre et une entité.

Conversion formelle du modèle TR-069 vers le modèle RBAC

L'algorithme de conversion est simple. Les deux seuls rôles définis sont *ACSRole* et *SubscriberRole*. Le rôle *ACSRole* a toutes les permissions possibles alors que le rôle *SubscriberRole* dispose des permissions uniquement sur les données pour lesquelles l'ACL contient le mot clé *Subscriber*. Définissons formellement la construction pour TR-069 :

- Chaque utilisateur TR-069 est directement associé à un utilisateur RBAC :

$$USERS_{TR-069}^{Construct} = U$$

- Chaque entité TR-069 est directement convertie vers un rôle RBAC :

$$ROLES_{TR-069}^{Construct} = \{ACSRole, SubscriberRole\}$$

- Les assignations entre utilisateurs et rôles sont réalisées de façon triviale :

$$UA_{TR-069}^{Construct} = \{(ACS, ACSRole), (Subscriber, SubscriberRole)\}$$

- Permissions :

$$\forall o \in O, f(o) = \{(o, "w"), \text{ où } Subscriber \in acl(o)\} \cup \{(/, "w")\}, PRMS_{TR-069}^{Construct} = f(O)$$

- Assignations des permissions aux rôles :

$$PA_{TR-069}^{Construct} = (PRMS_{TR-069}^{Construct}, SubscriberRole) \setminus ((/, "w"), Subscriber) \cup \{((/, "w"), ACSRole)\}$$

5.2.4 Implantation de l'algorithme

L'approche, proposée dans la section précédente et qui consiste à convertir le modèle de contrôle d'accès de plusieurs plateformes de gestion vers un modèle commun, a été implantée en langage Python et repose sur une analyse syntaxique. Le prototype utilise la librairie Python Yapps2 pour générer automatiquement un analyseur syntaxique en code Python à partir d'une grammaire abstraite. Ce prototype prend en entrée une politique de sécurité VACM (respectivement CLI, Netconf avec notre proposition d'extension RBAC) et génère une politique RBAC exprimée dans la syntaxe proposée dans [29]. Par ailleurs, la représentation des ressources protégées respecte la notation compacte également proposée dans [29]. L'annexe A détaille les formats de configuration en entrée et les résultats obtenus. La figure 5.3 donne un très rapide aperçu de la conversion.

5.3 Détection de conflits par comparaison et approximations

La section précédente a visé la convergence de modèles de contrôle d'accès de plusieurs plateformes de gestion de réseau. L'objectif est maintenant d'utiliser ces représentations communes pour en extraire les différences de privilèges d'un utilisateur donné ou de démontrer leur cohérence (ou équivalence) respective. Nous nous plaçons maintenant dans un contexte où les politiques de contrôle d'accès sont homogènes d'un point de vue syntaxique et modélisation. Dès lors, le principal challenge est d'établir des correspondances entre les rôles de différentes politiques, tout en prenant en compte le caractère dynamique du modèle RBAC et ses contraintes portant sur l'activation (respectivement désactivation) des rôles, pour parvenir à des conclusions.

5.3.1 Définition des opérations de comparaison

Cette section définit les opérateurs de comparaison des éléments de base de RBAC nécessaires à l'exploitation des politiques et leur comparaison. Tout d'abord, une définition des opérateurs est faite pour chaque entité du modèle (utilisateurs, ressources, permissions), puis des opérateurs de plus haut niveau sont construits, pour comparer, par exemple, des rôles et deux politiques.

Comparaison des utilisateurs

- equality (=) : deux utilisateurs sont égaux s'ils possèdent le même nom.

Comparaison des ressources

- égalité (=) : deux représentations de ressources sont égales si elles adressent le même ensemble d'objets,
- inclusion (\subseteq) : une représentation de ressource a est incluse dans une représentation de ressource b si l'ensemble des objets adressés par a est inclus dans l'ensemble des objets adressés par b . Plus précisément :
 - $a \subseteq b$ si b est le début de a : $/a_1/a_2/a_3 \subseteq /a_1/a_2$
 - $a \subseteq b$ si b est le début de a mis à part les jokers : $/a_1/a_2/a_3/a_4 \subseteq /a_1/* /a_3$

Comparaison des permissions

Les points suivants énumèrent les règles suivies pour la comparaison des permissions. Ils définissent deux opérateurs : l'égalité et l'inclusion. La Figure 5.4 illustre des exemples de com-

```
1 username john privilege 9 password 0 doe
2 ...
3 username inout password inout
4 privilege configure level 8 snmp-server community
5 privilege configure level 3 command ping
6 ...
7 privilege configure level 15 command access-group
8 privilege configure level 15 command activation-key
```

```
1 - rbac:
2   -roles:
3     -role: '1'
4       -name: 'role9'
5     -role: '2'
6       -name: 'role6'
7   -users:
8     -user: '1'
9       -name: 'john'
10    -user: '2'
11      -name: 'six'
12  -permissions:
13    -permission: '1'
14      -scope: '/configure/snmp-server/community'
15      -operation: 'w'
16    -permission: '2'
17      -scope: '/configure/command/ping'
18      -operation: 'w'
19    -permission: '3'
20      -scope: '/configure/mode/enable/command/configure'
21      -operation: 'w'
22  -user-role-assignments:
23    -user-role-assignment: '1'
24      -user: '1'
25      -role: '1'
26    -user-role-assignment: '2'
27      -user: '2'
28      -role: '2'
29  -permission-role-assignments:
30    -permission-role-assignment: '1'
31      -permission: '4'
32      -role: '3'
33    -permission-role-assignment: '2'
34      -permission: '5'
35      -role: '3'
```

FIG. 5.3 – Traduction de la configuration de sécurité CLI

paraisons de permissions.

- égalité ($=$) : deux permissions sont égales si la représentation de leurs ressources et leurs opérations sont égales,
- inclusion (\subseteq) : une permission a est incluse dans une permission b si les conditions suivantes sont satisfaites :
 - la représentation de ressource a est incluse dans la représentation de ressource b ,
 - l'opération de a est égale à l'opération de b .

```

1 (read, /a/b) est incluse dans (read, /a)
2 (read, /a/b) n'est pas incluse dans (write, /a)
3 (read, /a/b/c) est incluse dans (read, /a/*)
4 (write, /a/b/c) est incluse dans (write, /a/*/c)
5 (read, /a/*) n'est pas incluse dans (read, /a/b)

```

FIG. 5.4 – Exemples de comparaison de permissions

Comparaison des rôles

- égalité ($=$) : deux rôles sont égaux si leurs ensembles de permissions sont égaux ainsi que leurs ensembles d'utilisateurs. Formellement,

$$\forall r_1, r_2 \in ROLES, (assigned_permissions(r_1) = assigned_permissions(r_2)) \wedge (assigned_users(r_1) = assigned_users(r_2)) \Rightarrow r_1 = r_2$$

- inclusion (\subseteq) : un rôle a est inclus dans un rôle b si l'ensemble des permissions du rôle a est inclus dans l'ensemble des permissions du rôle b et l'ensemble des utilisateurs assignés au rôle a est inclus dans l'ensemble des utilisateurs du rôle b ,
- disjonction : deux rôles a et b sont disjoints si aucune permission de a (respectivement b) n'est incluse dans b (respectivement a).

Comparaison des politiques

Détecter des conflits entre deux politiques RBAC n'est pas trivial pour plusieurs raisons. Tout d'abord, il est important de noter que deux politiques peuvent être égales même si leurs ensembles de rôles respectifs ne sont pas égaux. Paradoxalement, les définitions de rôles peuvent être totalement différentes tout en allouant les mêmes permissions aux utilisateurs. C'est la distribution des permissions dans les rôles qui peut varier. La concordance partielle permet de vérifier la partie dynamique du modèle RBAC par l'activation des rôles. Un ensemble composé de deux rôles dans la politique P^A peut être égal à un ensemble de deux rôles dans la politique P^B , même si aucun des rôles de P^A n'est égal à un rôle de P^B . C'est ce que nous entendons par concordance partielle. A l'issue de la conversion des politiques de sécurité vers le modèle commun, la probabilité d'avoir généré des rôles égaux est très faible. Cependant, l'assemblage des rôles et la comparaison de ces ensembles est très utile pour la détection de conflit de politique. La Figure 5.5 illustre différents types de conflits qui sont susceptibles de se produire.

- Figure 5.5.a : il est extrêmement rare que deux rôles provenant de deux politiques différentes correspondent exactement, c'est-à-dire aient les mêmes permissions et les mêmes utilisateurs,

- Figure 5.5.b : Un ensemble de permissions correspondant à un rôle issu de la politique P^A peut être distribué dans plusieurs rôles dans la politique P^B ;
- Figure 5.5.c : Un ensemble de n permissions de la politique P^A peut être égal à un ensemble de m ($m \neq n$) permissions dans la politique P^B . Un cas particulier est qu'une permission issue de la politique P^A soit égale à n permissions dans la politique P^B .
- il se peut qu'il y ait des contraintes dynamiques de séparation des pouvoirs (DSD) qui rendent la comparaison encore plus complexe.

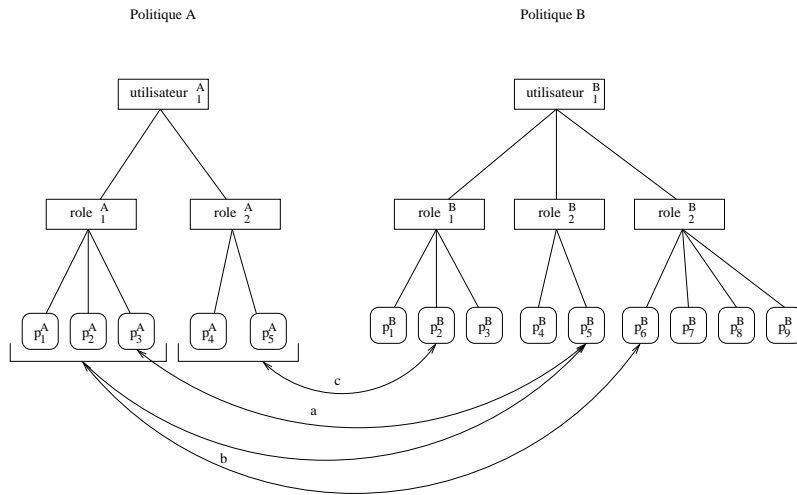


FIG. 5.5 – Les types de conflits identifiés

Évaluer les politique RBAC nécessite de prendre en considération, non seulement les différentes conditions d'activation de rôles, mais aussi les contraintes de séparation des pouvoirs.

- égalité ($=$) : deux politiques sont égales si elles donnent le même ensemble de permissions aux mêmes utilisateurs,
- inclusion (\subseteq) : une politique P^A est incluse dans une politique P^B si, pour chaque utilisateur u , l'ensemble des permissions de u dans la politique P^A est inclus dans l'ensemble des permissions de u dans la politique P^B . Plus formellement :

$$P^A \subset P^B \Leftrightarrow \begin{aligned} &\forall R^A \in \wp(\text{assigned_roles}(\text{user})), R^A \text{ satisfait les contraintes DSD,} \\ &\exists R^B \in \wp(\text{assigned_roles}(\text{user})), R^B \text{ satisfait les contraintes DSD,} \\ &R^A \subset R^B \end{aligned}$$

où :

- $\text{assigned_roles}(\text{user})$ est l'ensemble des rôles statiquement associés à l'utilisateur user ,
- $\wp(\text{assigned_roles}(\text{user}))$ est l'ensemble des partitions possibles, c'est-à-dire les différents ensembles de rôles possibles qui peuvent être activés.

La proposition précédente établit que tout ensemble de rôles actifs dans la politique P^A peut être simulé dans la politique cible P^B . Rappelons qu'une collection non ordonnée de p rôles distincts appartenant à l'ensemble des rôles $ROLES$ est appelée une combinaison et est notée C_n^p . Le nombre des combinaisons possibles de l'ensemble des $ROLES$ est donc :

$$\text{Card}\left(\bigcup_{k=1}^n \wp_k(ROLES)\right) = \sum_{p=1}^n C_n^p = \sum_{p=1}^n \frac{n!}{p!(n-p)!}$$

Par exemple, si $ROLES = \{r^1, r^2, r^3\}$, alors :

$$Card(\bigcup_{k=1}^3 \wp_k(ROLES)) = \sum_{p=1}^3 C_3^p = \sum_{p=1}^3 \frac{3!}{p!(3-p)!} = 7$$

Cette assertion est vérifiée car :

$$Card(C_{r^1, r^2, r^3}) = Card(\{r^1, r^2, r^3, (r^1, r^2), (r^1, r^3), (r^2, r^3), (r^1, r^2, r^3)\}) = 7$$

Cet ensemble de combinaisons est réduit par les contraintes DSD qui peuvent s'exprimer par exemple de la façon suivante : r^1 et r^2 ne peuvent pas être activés simultanément. Dans ce cas, le nombre de combinaisons devient :

$$Card(C_{r^1, r^2, r^3}) = Card(\{r^1, r^2, r^3, (r^1, r^3), (r^2, r^3)\}) = 5$$

- Correspondance partielle : les politiques P^A et P^B correspondent partiellement si, quel que soit l'ensemble de rôles R^A de P^A activés par l'utilisateur u , un ensemble de rôles R^B de P^B peut être activé par le même utilisateur u , tel que $permissions(R^A) \subseteq permissions(R^B)$ et respectivement.

5.3.2 Algorithme pour la détection de conflits de politique

L'algorithme, donné à la Figure 5.6 en pseudo-langage, affiche le delta, les différences, qui existe entre deux politiques P^A et P^B . Pour un utilisateur donné, cet algorithme considère toutes les conditions d'activation de rôles possibles, et prend en compte les contraintes de séparation dynamique des pouvoirs. A tout instant, un utilisateur peut avoir activé un nombre quelconque de rôles. Pour reproduire ces conditions, nous considérons toutes les situations qu'un utilisateur peut créer lors d'une session. Une partition de rôles est un sous-ensemble de l'ensemble des partitions de $ROLES$ défini dans la section précédente. L'algorithme est capable de trouver les conditions de privilèges possibles par la politique P^A mais pas par la politique P^B et respectivement. C'est pourquoi un *delta*, ou conflits entre les politiques P^A et P^B peut être détecté.

5.3.3 Conservation de l'équivalence de politique lors d'opérations de configuration

D'un point de vue contrôle d'accès, les opérations de gestion de réseau peuvent être partitionnées en deux ensembles : celles qui conservent le jeu de données autorisées et celles qui ne le conservent pas. L'ensemble des opérations susceptibles de modifier les données autorisées peuvent être de deux types : celles qui modifient la politique de contrôle d'accès elle-même et celles qui modifient des valeurs de telle sorte qu'un privilège a une plus grande portée que précédemment. Ce dernier type d'opérations existe car les politiques de contrôle d'accès reposent souvent sur des expressions décrivant les ressources qui dépendent des valeurs du modèle de données. Par exemple, l'accès aux interfaces réseau qui portent le nom *eth0* peut être donné à un utilisateur. Potentiellement toutes les opérations d'écriture peuvent modifier l'ensemble des données auxquelles un utilisateur a accès.

A partir d'une situation stable, où les politiques de contrôle d'accès sont égales parmi un ensemble de plateformes de gestion de réseau, il est intéressant de qualifier les opérations qui conservent cette égalité. Définissons l'ensemble des opérations qui conservent l'égalité des politiques de contrôle d'accès au sein d'une plateforme donnée NMF_1 :

```

1 function showDelta(pa, pb, user){
2   // get the roles statically assigned to user in RBAC generated policy pa:
3   roles_a = pa.assigned_roles(user)
4   // get the roles statically assigned to user in RBAC generated policy pb:
5   roles_b = pa.assigned_roles(user)
6   // build the partition set of user from policy pa, taking DSD into account:
7   partition_set_a = pa .partitions(roles_a)
8   // build the partition set of user from policy pb, taking DSD into account:
9   partition_set_b = pa .partitions(roles_b)
10  // try to simulate all partitions from partition_set_a in partition_set_b
11  for partition_a in partition_set_a:
12    included = False
13    for partition_b in partition_set_b:
14      permission_set_a = assigned_permissions(partition_a)
15      permission_set_b = assigned_permissions(partition_b)
16      if permission_set_a.includedIn(permission_set_b):
17        included = True
18    if not included:
19      print "partition_a can not be simulated in policy pb."
20      print partition_a.toString()
21  }

```

FIG. 5.6 – Algorithme pour la détection de conflit de politiques affichant le delta des permissions entre deux politiques pour un utilisateur donné.

$$F_{conserve} = \{ \text{opérations } f \text{ telles que } ac(f(NMF_1)) = ac(NMF_1) \}$$

Caractériser de telles opérations assurerait la préservation de la cohérence au cours du temps. Une conséquence directe est que cela évite d'avoir à vérifier la cohérence après chaque opération. Cependant, l'identification et la classification de ce type d'opération est un problème complexe. Le processus de détection des opérations qui ne préservent pas la cohérence suppose l'évaluation de la politique de contrôle d'accès par rapport au modèle de données. En effet, les deux composants que sont la politique et le modèle de données ont pu être modifiés par l'opération. Ce problème est également très dépendant de la façon dont les opérations sont définies : spécialisées (setInterface) ou non (set(interface)). Le premier cas semble plus simple à traiter car il demande une analyse des paramètres moins poussée. Le deuxième cas peut être rendu complexe par la nature des paramètres comme par exemple une opération edit-config dans Netconf. La caractérisation de la conservation de la cohérence par les opérations de gestion est une de nos perspectives de recherche.

5.4 Synthèse

5.4.1 Un modèle formel pour vérifier la cohérence des politiques de contrôle d'accès

Dans ce chapitre, nous avons présenté un mécanisme de convergence adapté au modèle de contrôle d'accès de différentes plateformes de supervision de réseau. L'objectif était de pouvoir comparer deux à deux des politiques qui, à l'origine, sont hétérogènes mais protègent des ressources communes. Nous nous sommes servi du modèle RBAC comme référentiel commun pour cette étude. Nous avons formalisé le modèle de contrôle d'accès de plusieurs plateformes, puis

nous avons défini le mécanisme de conversion qui permet d'exprimer ce modèle vers le modèle RBAC, et enfin un algorithme de comparaison de politiques RBAC a été proposé.

L'avantage de cette approche est qu'elle est utilisable dans le cadre d'un réseau déjà configuré. Dans un premier temps, il suffit simplement de récupérer les politiques existantes, de les rendre comparable, et de les analyser pour en extraire les conflits. Dans un second temps, il est possible de prendre des mesures correctives de façon à réduire au maximum les conflits locaux.

5.4.2 Limites et alternatives

Adaptabilité du modèle RBAC

Si, dans le futur, un autre modèle de contrôle d'accès est utilisé dans une plateforme de gestion, il y a trois alternatives pour notre approche :

- Le nouveau modèle est adaptable, c'est-à-dire peut être simulé, par un modèle RBAC. Dans ce cas, l'approche qui consiste à utiliser RBAC comme modèle de convergence fonctionne toujours,
- Le modèle RBAC peut se traduire vers le nouveau. Par transitivité, les modèles de contrôle d'accès considérés peuvent aussi être convertis vers le nouveau modèle via RBAC,
- Le nouveau modèle et RBAC sont totalement incompatibles : la probabilité de ce cas est très faible. Cependant, une approximation ou une extension du modèle RBAC peut être envisagée.

Représentation du modèle de données

Pour pouvoir comparer les politiques de contrôle d'accès, il est nécessaire d'avoir défini au préalable un modèle et une représentation des données génériques. Chaque plateforme de gestion de réseau a son propre modèle de l'information et son propre schéma d'adressage. Par exemple, si l'on considère SNMP, le modèle de données est décrit par des fichiers définissant la structure des MIBs (Management Information Base). Ces MIBs décrivent une structure arborescente et définissent un schéma d'adressage basé sur une notation pointée et dans laquelle les identifiants peuvent être soit des noms, soit des nombres. Dans CLI, une configuration s'exprime sous forme de commandes, de paramètres et de valeurs selon une syntaxe bien définie. [29] a montré que, même cette forme de représentation peut être modélisée sous la forme d'un arbre. Dans Netconf, le modèle de l'information n'a pas encore été standardisé mais il le sera probablement dans un langage dédié à la définition de structure de document XML, c'est-à-dire probablement des schémas XML [74], des DTD, ou RelaxNG. Les données de configuration sont alors représentées sous la forme d'un document XML qui est ni plus ni moins qu'un arbre. C'est pourquoi la construction d'un modèle de données générique est une tâche réalisable.

Le challenge est d'établir un lien entre ces différentes représentations arborescentes de la configuration. Ce processus peut être fait en deux étapes. Tout d'abord, un algorithme a été développé pour construire un arbre à partir de plusieurs plateformes (SNMP, CLI, Netconf) : ces arbres peuvent être adressés conformément à la représentation étendue proposée dans [29]. Cependant, à ce stade, les noms des nœuds et l'organisation des données sont toujours différents.

La seconde étape, qui consiste à réorganiser l'arbre ou traduire les adressages des permissions d'un modèle à l'autre, est beaucoup plus difficile. Les adressages (scopes) sont les représentations des ressources sous la forme d'une chaîne de caractères. La seule façon de pouvoir comparer ces scopes est de les convertir par rapport à un modèle de données générique. Ce problème n'est pas trivial. Une solution possible serait d'implanter un système d'apprentissage automatique de

façon à construire progressivement la traduction entre ces modèles hétérogènes. Cette approche n'a pas encore été investiguée.

Nous avons fait l'hypothèse forte qu'il existe une fonction qui prend en paramètre un scope relatif à un modèle et retourne le scope dans le modèle commun.

6

Une proposition de contrôle d'accès dans Netconf

Sommaire

6.1	Un modèle RBAC dans Netconf	76
6.1.1	Une politique auto-protégée administrable à distance	76
6.1.2	XPath pour l'adressage des données du modèle	76
6.1.3	Spécification de la politique RBAC	76
6.1.4	Un gendarme pour le filtrage des données XML	78
6.2	Proposition d'extension pour Netconf : "#rbac capability" . . .	78
6.2.1	Une nouvelle opération Netconf	78
6.2.2	Format des messages	79
6.3	Scenario d'utilisation	79
6.4	Synthèse et perspectives	82

Le protocole de configuration Netconf est en cours de standardisation à l'IETF. Il s'inspire des protocoles existants que sont SNMP ou CLI pour définir un nouveau protocole cette fois basé sur le langage XML. De SNMP, il conserve le paradigme gestionnaire/agent, un modèle de données hiérarchique, un protocole applicatif dédié et un besoin fort de passage à l'échelle. De CLI, il conserve le côté orienté configuration, les multiples configurations (running, startup) et le protocole de transport SSH.

Comme Netconf n'a pas encore défini son modèle de contrôle d'accès, nous avons proposé un modèle qui tire parti des meilleurs modèles de contrôle d'accès, des technologies XML, des fonctionnalités propres à Netconf et de son contexte. Ce modèle s'intègre parfaitement avec nos autres propositions, notamment celle étendant une passerelle XML/SNMP pour ainsi fournir une portée plus grande à l'homogénéité globale de la sécurité de la gestion de réseau. Le second objectif est de séparer les tâches de gestion pour accroître la sûreté en limitant les erreurs humaines, et répartir le nombre toujours croissant des gestionnaires. Rappelons ici que Netconf doit pouvoir gérer jusqu'à 100000 équipements. Des industriels citent des cas réels de 30000 équipements. Il va de soi qu'un tel réseau ne peut être géré par une seule personne. L'approche suivie définit des classes de gestionnaires, chacune ayant des privilèges particuliers, pour distribuer les responsabilités et réduire les risques.

6.1 Un modèle RBAC dans Netconf

6.1.1 Une politique auto-protégée administrable à distance

Dans cette proposition de contrôle d'accès pour Netconf, chaque agent a sa propre politique. Nous n'avons pas choisi de centraliser le contrôle d'accès dans un serveur dédié pour ne pas augmenter la quantité de messages nécessaires au bon fonctionnement de l'agent. De plus, en environnement dégradé, il se peut qu'un tel serveur ne soit pas joignable. Un tel contexte, où le réseau fonctionne de façon limitée, est très courant puisque ce sont en général ces opérations de configuration qui permettent de faire fonctionner ou de rétablir la connectivité du réseau.

Cependant, rien n'empêche un administrateur de mettre en place un serveur de stockage contenant des politiques de sécurité prédéfinies et de les déployer automatiquement via Netconf. En effet, l'approche présentée dans cette section permet à tout administrateur de gérer la politique de sécurité à distance via Netconf lui-même. On pourra qualifier cette approche, similaire au choix fait dans SNMPv3, d'autogestion. De plus, le protocole Netconf permet l'utilisation d'url pour réaliser des opérations de type `edit-config`; donc un serveur de configuration est tout à fait envisageable pour déployer les droits d'accès, tout en facilitant l'administration de la politique de sécurité. C'est un principe qui a souvent été utilisé dans le passé, notamment dans le cadre de CLI avec la mise en place de serveurs TFTP.

6.1.2 XPath pour l'adressage des données du modèle

[43, 51, 84, 15] ont proposé des approches de contrôle d'accès sur les documents XML.

Un peu à la manière de VACM, nous définissons des vues sur lesquelles des opérations pourront être ajoutées pour définir des permissions. La différence est que les vues, ici, utilisent une syntaxe standard (XPath) qui permet à toute personne connaissant ce schéma d'adressage de comprendre le contenu de la permission. Dans VACM, les tables de vues reposent sur un système de familles d'arbres utilisant des masques (comme les adresses IP), pas plus complexe que XPath mais moins puissant et demandant un certain temps d'adaptation.

XPath permet d'exprimer très naturellement une requête sur un arbre XML et aura pour résultat un sous-ensemble des nœuds de l'arbre. XPath permet une sélection :

- par type : on peut sélectionner toutes les interfaces : `//interface`
- par instance : on peut sélectionner l'interface eth0 : `//interface[@name='eth0']`

Nous ne posons aucune restriction sur l'utilisation de XPath pour l'adressage des ressources. XPath permet un système de contrôle d'accès très fin reposant sur une technologie connue.

6.1.3 Spécification de la politique RBAC

La Figure 6.1 montre un échantillon de politique RBAC au format XML. Elle définit un ensemble d'utilisateurs, de rôles, de permissions et leurs affectations respectives. Dans cet exemple, un seul utilisateur est défini (ligne 10). Il est autorisé à endosser le rôle *rootAdmin* (ligne 18), ce qui lui donne par héritage les permissions des deux rôles juniors, *voipAdmin* (ligne 25) et *bgpAdmin* (ligne 29). Le rôle *voipAdmin* dispose de la permission 9 (ligne 41) qui permet de lire et d'écrire (l'attribut *op* a la valeur *rw*) dans les sous-arbres dont les racines sont les nœuds sélectionnés par l'expression XPath `/ygp :netconf/ygp :security/ac :rbac/ac :roles`. Cela signifie que cette permission autorise la modification de la définition des rôles eux-mêmes.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <rbac xmlns="urn:loria:madyne:ensuite:yencap:module:RBAC:1.0"
3     xmlns:ycp="urn:loria:madyne:ensuite:yencap:1.0"
4     xmlns:ifs="urn:loria:madyne:ensuite:yencap:module:Interfaces:1.0"
5     xmlns:bgp="urn:loria:madyne:ensuite:yencap:module:BGP:1.0"
6     xmlns:ac="urn:loria:madyne:ensuite:yencap:module:RBAC:1.0"
7     xmlns:rt="urn:loria:madyne:ensuite:yencap:module:Route:1.0"
8     xmlns:s="urn:loria:madyne:ensuite:yencap:module:System:1.0">
9   <users>
10    <user id="15">
11      <login>netconf</login>
12      <password>netconf</password>
13      <public-key keytype="rsa">AAAAB3Nz...bPOCDbc35ORfDJ6M=</public-key>
14    </user>
15    <!-- Other users -->
16  </users>
17  <roles>
18    <role id="1">
19      <name>rootAdmin</name>
20      <junior-roles>
21        <junior-role roleRef="2"/>
22        <junior-role roleRef="3"/>
23      </junior-roles>
24    </role>
25    <role id="2">
26      <name>voipAdmin</name>
27      <junior-roles/>
28    </role>
29    <role id="3">
30      <name>bgpAdmin</name>
31      <junior-roles/>
32    </role>
33  </roles>
34  <permissions>
35    <permission id="1" op="rw">
36      <scope>/ycp:netconf/ycp:security/ac:rbac/ac:permissions/ac:permission[@id='5']</scope>
37    </permission>
38    <permission id="3" op="rw">
39      <scope>/ycp:netconf/ycp:routing</scope>
40    </permission>
41    <permission id="9" op="rw">
42      <scope>/ycp:netconf/ycp:security/ac:rbac/ac:roles</scope>
43    </permission>
44  </permissions>
45  <user-assignments>
46    <user-assignment roleRef="1" userRef="15" id="1"/>
47    <!-- Other User-Role Assignements -->
48  </user-assignments>
49  <permission-assignments>
50    <permission-assignment roleRef="1" permRef="1" id="1"/>
51    <permission-assignment roleRef="2" permRef="9" id="2"/>
52    <!-- Other Permission-Role Assignements -->
53  </permission-assignments>
54 </rbac>

```

FIG. 6.1 – Représentation XML de la politique RBAC dans l'agent Netconf

6.1.4 Un gendarme pour le filtrage des données XML

La Figure 6.2 illustre l'implantation du système RBAC. La complexité du système RBAC est cachée grâce à un mécanisme de *Facade*. Le module *RbacManager* est responsable du contrôle d'accès sur les opérations et les données.

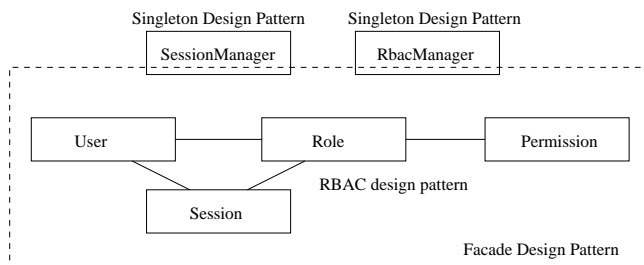


FIG. 6.2 – Implantation du système de contrôle d'accès

Un module de contrôle d'accès intégré à Netconf doit être capable de filtrer les données en fonction de la politique de sécurité de l'agent. Deux algorithmes ont été réalisés pour cela : le premier pour les opérations de type *get-config* (ou *get*) et le second pour les opérations de type *edit-config*. Les opérations *edit-config*, *get-config* et *get* font appel au système de contrôle d'accès *rbacManager* qui est en charge de prendre la décision d'accès ou de filtrer les données.

get-config Tout d'abord, un document résultat est construit en fonction de la requête reçue, que ce soit du subtree filtering ou XPath. Ensuite, les permissions en lecture de l'utilisateur sont rassemblées. Les expressions XPath de ces permissions sont appliquées au résultat et fournissent en sortie une liste de nœuds sélectionnés. On sélectionne également par propagation leurs nœuds parents et enfants pour aboutir au document XML final. Tous les nœuds non sélectionnés sont élagués du document comme illustré par la Figure 6.3.

edit-config L'algorithme comporte plusieurs étapes résumées ici. Tout d'abord, l'ensemble des permissions en écriture de l'utilisateur sont rassemblées. Ces expressions XPath sont appliquées sur la requête reçue et les nœuds sélectionnés sont placés dans une liste. Puis, la liste des parents et celle des enfants sont créées. La liste des parents ne doit pas contenir d'opérations *create*, *delete* ou *replace* car l'utilisateur n'a pas les droits sur les parents. Enfin, il faut vérifier qu'il n'y a pas de nœuds qui n'appartiennent ni à la liste des parents, ni à la liste des enfants, ni aux nœuds sélectionnés par XPath. Si tel était le cas, cela signifierait que des opérations pourraient être effectuées sur des nœuds non couverts par la politique de sécurité.

6.2 Proposition d'extension pour Netconf : "#rbac capability"

6.2.1 Une nouvelle opération Netconf

Nous avons étendu Netconf avec une nouvelle *capability* "urn:madyne:params:xml:ns:netconf:capability:rbac:1.0". Celle-ci, comme toutes les autres fonctionnalités optionnelles de Netconf, est annoncée dans le message *hello*. Elle indique que l'agent ou le manager Netconf supporte le contrôle d'accès basé sur les rôles et notamment une nouvelle opération Netconf que l'on nomme *rbac*. Cette *capability* permet de gérer la partie dynamique du modèle RBAC, que sont les concepts d'activation et désactivation de rôles. A distance, un utilisateur Netconf

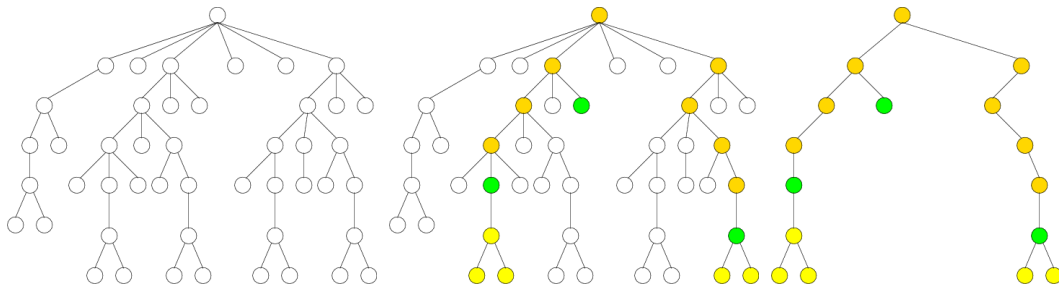


FIG. 6.3 – Filtrage par sous-arbre (subtree filtering) et par XPath

sera donc capable d'endosser un certain nombre de rôles pour obtenir plus de privilèges et ainsi réaliser des opérations de configuration.

En interne dans l'agent de supervision, un assemblage des permissions spécifiées et héritées par la hiérarchie de rôles est effectué. A chaque activation ou désactivation, l'ensemble des privilèges de l'utilisateur est recalculé en fonction des permissions propres aux rôles actifs mais également des permissions héritées des rôles juniors.

Le grand avantage, par rapport à CLI, vient du fait que la hiérarchie de rôles n'est pas linéaire. La différence est en tout point comparable à l'héritage simple ou multiple dans les langages de programmation orientés objets. D'un point de vue pratique, une activation de rôle est très similaire à la commande CLI suivante, `enable 5`, qui permet de passer au niveau de privilège 5.

6.2.2 Format des messages

Cette section définit le format des messages nécessaires à l'activation et la désactivation des rôles. L'opération `rbac` se place au même niveau que n'importe quelle autre opération Netconf, c'est-à-dire directement au-dessous de l'élément `rpc`. Deux éléments enfants de `rbac` sont définis : `activate` et `deactivate`. Chacun de ces éléments contient un unique élément `roles` qui contient l'ensemble des rôles à activer ou désactiver. La Figure 6.4 illustre une opération d'activation. Celle-ci concerne deux rôles, `netAdmin` et `sysAdmin`. A la réception d'un message d'activation, l'agent Netconf vérifie la politique de contrôle d'accès pour s'assurer qu'il existe une assignation entre l'utilisateur courant et les rôles demandés. Si c'est le cas, les rôles sont activés et les privilèges de l'utilisateur sont augmentés en fonction des permissions associées à ces rôles. Un message `<ok/>` est renvoyé au manager Netconf en cas de succès. Si l'utilisateur n'est pas autorisé à activer ce rôle ou si l'utilisateur a déjà activé ce rôle, un message d'erreur est envoyé. De la même façon, la Figure 6.4 illustre une opération de désactivation.

6.3 Scenario d'utilisation

Dans cette section, nous illustrons l'intérêt de notre modèle de contrôle d'accès dans un contexte concret de supervision de réseau et en particulier de politique de routage. Considérons un fournisseur d'accès disposant d'un système autonome (AS) connecté à plusieurs autres AS. Nous supposons que ce fournisseur d'accès a mis en place une politique de routage interne basée sur RIP et une politique de routage externe basée sur BGP. Nous nous plaçons sur un routeur de bordure qui utilise donc à la fois le protocole interne IGP et le protocole externe EGP.

```
1 <rpc message-id="101"
2   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
3   <rbac xmlns="urn:loria:madynes:ensuite:yencap:rbac:capability">
4     <activate>
5       <roles>
6         <role>netAdmin</role>
7         <role>sysAdmin</role>
8       </roles>
9     </activate>
10  </rbac>
11 </rpc>
```

Message Netconf pour l'activation de rôles

```
1 <rpc message-id="101"
2   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
3   <rbac xmlns="urn:loria:madynes:ensuite:yencap:rbac:capability">
4     <deactivate>
5       <roles>
6         <role>sysAdmin</role>
7       </roles>
8     </deactivate>
9   </rbac>
10 </rpc>
```

Message Netconf pour la désactivation de rôles

FIG. 6.4 – Nouveaux messages Netconf pour RBAC

Ce fournisseur emploie des administrateurs ayant diverses compétences en réseau. Certains, comme Alice, sont capables de mettre en oeuvre le protocole RIP mais ne sont pas des spécialistes de BGP. D'autres, comme Bob, connaissent mieux BGP que RIP et d'autres (Tom) ont la double compétence et sont ainsi capables d'avoir une vision d'ensemble de la politique de routage, c'est-à-dire de prendre en charge les redistributions de routes apprises dynamiquement entre les protocoles.

L'intérêt du modèle RBAC ici est que nous allons définir des rôles associés à chacune de ces fonctions de supervision et y affecter des administrateurs en fonction de leurs compétences. Nous définissons tout d'abord un rôle générique de gestionnaire du routage : *ManagerRoutage*. Ce rôle a des permissions mineures qui permettent notamment de consulter la configuration liée au routage mais ne permettent pas de la modifier. Ensuite, nous définissons deux rôles senior du rôle précédant et qui sont dédiés l'un à la configuration du routage externe (*ManagerRoutageExterne*) et l'autre à la configuration du routage interne (*ManagerRoutageInterne*). Le premier a le privilège de modifier la configuration BGP alors que le second le privilège de modifier la configuration RIP. Comme ce sont des rôles seniors du rôle *ManagerRoutage*, ils sont également capables de lire la configuration de routage en entier. Par exemple, un utilisateur, par l'intermédiaire du rôle *ManagerRoutageExterne* peut lire non seulement la configuration de routage externe mais aussi la configuration du routage interne à titre informatif ou pour l'aider dans ses choix de configuration.

Enfin, nous définissons les rôles *ManagerSécurité* et *ManagerRoot*. Le premier gère la politique de sécurité via les ACL pour empêcher le routeur d'apprendre des routes issues de routeurs ayant un niveau de confiance insuffisant. Le second, *SuperManager*, est défini comme rôle sénior des trois rôles, *ManagerSécurité*, *ManagerRoutageExterne* et *ManagerRoutageInterne* et hérite ainsi de tous leurs privilèges. *SuperManager* n'est utilisé qu'en cas de problème grave, nécessitant par exemple une remise à plat de toute la configuration.

La figure 6.5 illustre la hiérarchie des rôles définis précédemment. Notons que, dans notre exemple, le graphe des rôles est connexe mais ce n'est pas forcément le cas dans le modèle RBAC. De plus, chaque utilisateur ne sera associé directement qu'à un seul rôle. Cette propriété n'est pas forcément vraie dans le cas général. Les cinq rôles, ainsi que tous les paramètres relatifs à RBAC sont décrits dans un fichier `rbac.xml`.

Spécifions un peu plus finement les privilèges de chacun des rôles. *ManagerRoutage* a le privilège de lire la configuration donc a la permission d'exécuter la commande `show running-config` dans un contexte CLI. Dans notre contexte Netconf, nous pouvons spécifier la permission en lecture sur la ressource identifiée par l'expression XPath `/ycp :netconf/ycp :routing`, `ycp` étant le préfixe de l'espace de noms XML. Le rôle *ManagerRoutageExterne* a la permission de modifier la configuration BGP, identifiée par l'expression XPath `/ycp :netconf/ycp :routing/bgp :bgp`, `bgp` étant le préfixe de l'espace de noms du module de MIB de BGP. Entre parenthèses, on pourrait aussi écrire `//bgp :bgp` ou même

```
//bgp :bgp/bgp :bgprouter//bgp :neighbor[@ip = '100.100.100.100']
```

pour que ce rôle ne puisse modifier que la définition du voisin BGP ayant l'adresse IP spécifiée.

De façon similaire, le rôle *ManagerRoutageInterne* (ligne 24 à 29, Figure 6.7) a la permission (ligne 8 à 10, Figure 6.7) en écriture sur les sous-arbres dont les racines sont sélectionnées par l'expression

```
/ycp :netconf/ycp :routing/rip :rip
```

Le rôle *ManagerSécurité* a toutes les permissions sur les ACLs (ligne 11 à 13, Figure 6.7) identifiées par l'expression

```
/ycp :netconf/ycp :routing/sec :acl
```

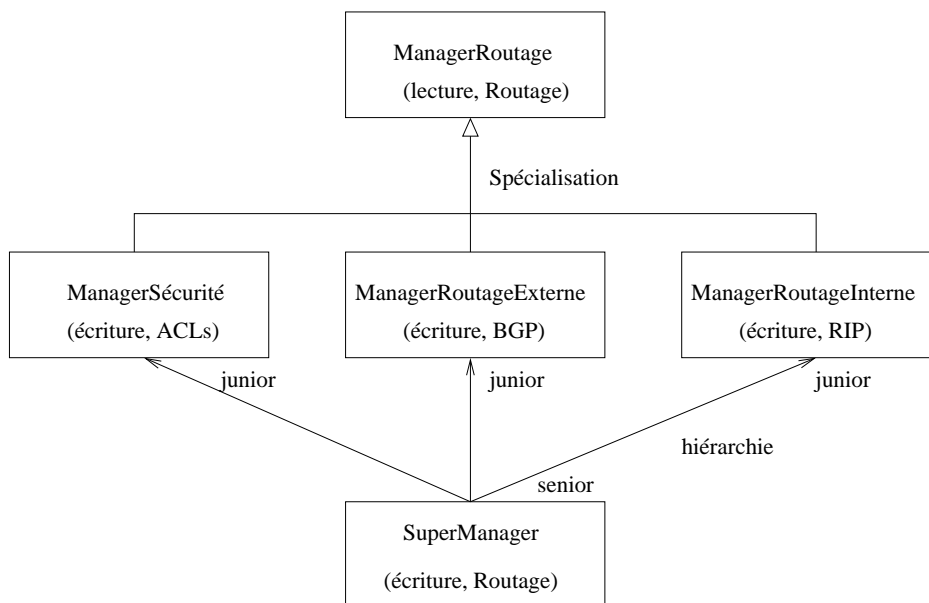


FIG. 6.5 – Définition des rôles pour le routeur de bordure

Enfin, le rôle *SuperManager* a la permission en écriture sur toute la partie routage :

```
/ycp : netconf / ycp : routing
```

A titre indicatif, toutes les expressions XPath données ici correspondent au modèle de données que nous avons défini dans la plateforme EnSuite et sont donc spécifiques à cette plateforme.

Les affectations des utilisateurs aux différents rôles sont définies comme suit : Alice est associée au rôle *ManagerRoutageInterne*, Bob est associé au rôle *ManagerRoutageExterne*, Tom est associé au rôle *SuperManager*. Par héritage, Tom est le seul à être aussi *ManagerSécurité*.

Les associations entre les privilèges et les rôles sont définies de la ligne 23 à 29 (Figure 6.7). Des références (ou pointeurs) sont utilisées pour identifier les permissions et les rôles définis par ailleurs.

6.4 Synthèse et perspectives

Pour conclure, Netconf a potentiellement un champ d'action suffisamment grand (BGP, VoIP, etc...) pour justifier un tel découpage en rôle des responsabilités. Il nous semble être une bonne pratique d'adopter une démarche similaire à celle de Webmin en commençant par définir une permission par module de MIB, ce qui revient à un modèle à grain moyen puis de spécialiser ces permissions plus finement lorsque cela est nécessaire. Ces permissions plus fines portent sur des sous-parties d'un même module. Cela crée une double granularité probablement suffisante pour la majorité des cas. Cette façon de faire ne change en rien le modèle décrit jusqu'ici puisque le nœud représentant un module s'exprime toujours par une expression XPath.

La meilleure réponse au choix entre une définition des permissions par module ou par expressions XPath libres ne peut venir que par l'expérience d'un déploiement à grande échelle des agents Netconf, chacun mettant à disposition un nombre important de modules. Dans un tel contexte, il est probable que la définition des permissions penche vers l'option *module* plutôt que vers l'option *XPath*.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <rbac xmlns="urn:loria:madynes:ensuite:yencap:module:RBAC:1.0"
3     xmlns:ycp="urn:loria:madynes:ensuite:yencap:1.0"
4     xmlns:bgp="urn:loria:madynes:ensuite:yencap:module:BGP:1.0"
5     xmlns:rip="urn:loria:madynes:ensuite:yencap:module:RIP:1.0">
6   <users>
7     <user id="1">
8       <login>Alice</login>
9       <password>Alice</password>
10    </user>
11    <user id="2">
12      <login>Bob</login>
13      <password>Bob</password>
14    </user>
15    <user id="3">
16      <login>Tom</login>
17      <password>Tom</password>
18    </user>
19  </users>
20  <roles>
21    <role id="1">
22      <name>ManagerRoutage</name>
23    </role>
24    <role id="2">
25      <name>ManagerRoutageInterne</name>
26      <junior-roles>
27        <junior-role roleRef="1"/>
28      </junior-roles>
29    </role>
30    <role id="3">
31      <name>ManagerRoutageExterne</name>
32      <junior-roles>
33        <junior-role roleRef="1"/>
34      </junior-roles>
35    </role>
36    <role id="4">
37      <name>ManagerSécurité</name>
38      <junior-roles>
39        <junior-role roleRef="1"/>
40      </junior-roles>
41    </role>
42    <role id="5">
43      <name>SuperManager</name>
44      <junior-roles>
45        <junior-role roleRef="2"/>
46        <junior-role roleRef="3"/>
47        <junior-role roleRef="4"/>
48      </junior-roles>
49    </role>
50  </roles>

```

FIG. 6.6 – Représentation XML de la politique RBAC dans l'agent Netconf (début)

```
1 <permissions>
2   <permission id="1" op="r">
3     <scope>/ycp:netconf/ycp:routing</scope>
4   </permission>
5   <permission id="2" op="w">
6     <scope>/ycp:netconf/ycp:routing/bgp:bgp</scope>
7   </permission>
8   <permission id="3" op="w">
9     <scope>/ycp:netconf/ycp:routing/rip:rip</scope>
10  </permission>
11  <permission id="4" op="w">
12    <scope>/ycp:netconf/ycp:routing/sec:acl</scope>
13  </permission>
14  <permission id="5" op="w">
15    <scope>/ycp:netconf/ycp:routing</scope>
16  </permission>
17 </permissions>
18 <user-assignments>
19   <user-assignment roleRef="2" userRef="1" id="1"/>
20   <user-assignment roleRef="3" userRef="2" id="2"/>
21   <user-assignment roleRef="5" userRef="3" id="3"/>
22 </user-assignments>
23 <permission-assignments>
24   <permission-assignment roleRef="1" permRef="1" id="1"/>
25   <permission-assignment roleRef="2" permRef="3" id="2"/>
26   <permission-assignment roleRef="3" permRef="2" id="3"/>
27   <permission-assignment roleRef="4" permRef="4" id="4"/>
28   <permission-assignment roleRef="5" permRef="5" id="5"/>
29 </permission-assignments>
30 </rbac>
```

FIG. 6.7 – Représentation XML de la politique RBAC dans l'agent Netconf (fin)

Il est prudent de se poser la question de la définition des permissions en fonction des seules opérations de lecture ("r") et d'écriture ("w"). Le privilège de verrouiller/déverrouiller une configuration devrait aussi être pris en considération car il empêche toute intervention durant la durée de possession de ce verrou. Dans le futur, d'autres types d'opération pourraient apparaître. Par exemple, nous avons développé un mécanisme de déploiement et de chargement à chaud de modules sous la forme d'archives compressées. Ce type d'opération n'entre pas dans le cadre d'une opération de type "r" ou "w". Cette remarque s'applique aussi pour les notifications qui sont en cours de spécification par le groupe de travail Netconf. Donc, notre approche doit rester souple pour supporter les évolutions futures du protocole Netconf.

Les principales différences, par rapport au modèle de contrôle d'accès basé sur les vues de SNMPv3, sont les suivantes : il n'y a pas de possibilité de spécifier l'inclusion ou l'exclusion d'une ressource car on ne spécifie que des permissions positives, d'où la disparition des conflits. On ne fait plus intervenir le modèle de sécurité sous-jacent, ce qui divise la complexité par le nombre de modèles de sécurité possibles. On ne fait plus intervenir non plus les services de sécurité utilisés, ce qui divise la complexité par trois, car il y a trois niveaux possibles (noauthnopriv, authnopriv et authpriv). Ici, on est favorisé par le contexte, car on suppose que tous les protocoles sous-jacents fournissent les mêmes services de sécurité (SOAP/HTTPS, SSH, BEEP/SSL). Une vue (ensemble de ressources protégées) est exprimée avec une seule expression XPath au lieu d'un ensemble de couple (OID, masque). Les rôles sont organisés dans une hiérarchie alors que les groupes ne l'étaient pas. Tous ces changements font que la politique de sécurité est plus simple à gérer, tout en étant plus puissante.

Troisième partie

Expérimentations et évaluations

Passerelle XML/SNMP sécurisée

Sommaire

7.1	Introduction	89
7.1.1	Les passerelles XML/SNMP	89
7.1.2	Motivation	90
7.1.3	Qualités requises pour la gestion de la politique de sécurité	90
7.2	Architecture fonctionnelle	90
7.3	Définition d'une politique d'autorisation	92
7.3.1	Lien entre RBAC et les données de gestion	92
7.3.2	Algorithme de translation unidirectionnel RBAC/VACM	92
7.4	Scénario d'utilisation	95
7.4.1	Processus de contrôle d'accès	95
7.4.2	Prise en comptes de nouveaux agents	96
7.4.3	Une interface pour l'administration de la passerelle	96
7.4.4	Un continuum d'authentification et de confidentialité	96
7.5	Synthèse	96

7.1 Introduction

7.1.1 Les passerelles XML/SNMP

Avec l'essor des technologies XML, le monde de la gestion de réseau, comme de nombreux autres domaines, a observé avec intérêt les possibilités offertes par cette technologie. Le langage XML est, en effet particulièrement adapté aux systèmes fortement distribués comme l'activité de gestion : encodage standard des données, outils de définition de modèles de données ou de format de message et de protocoles, outils de validation automatique de structure de messages, librairie de traitement des données (parsing, transformation).

Avant de faire une migration totale vers XML comme, par exemple, Netconf, les passerelles XML/SNMP constituent une approche intermédiaire qui font un pont entre les mondes XML et SNMP. Fondamentalement, il n'y a pas une très grande différence entre la modélisation de données avec un schéma XML ou ASN.1 ou l'encodage des messages avec XML ou BER. Les grandes différences, probablement, sont que le langage XML est lisible facilement même lorsqu'il transite sur le réseau, qu'il est apparu en même temps que de nombreux langages populaires (Java,

.NET, ...) et qu'il est utilisé dans presque tous les domaines. Ces facteurs font que les coûts de développement diminuent fortement et de nombreux ingénieurs sont formés pour l'XML.

Une passerelle XML/SNMP est un élément qui se place entre un ou plusieurs gestionnaires utilisant XML et un ou plusieurs agents SNMP. Elle est capable de traduire les requêtes XML en requêtes SNMP puis de transformer la réponse SNMP en réponse XML. En quelque sorte, c'est un outil de traduction entre XML et SNMP. Pour réaliser cette traduction, il est nécessaire au préalable de générer un schéma XML compatible avec les MIBs SNMP existantes et standardisées au cours des 20 dernières années. Des algorithmes ont été proposés pour traduire automatiquement les MIBs en schémas XML afin de ne pas perdre cet effort de standardisation. D'ailleurs des travaux similaires sont menés dans le cadre de Netconf.

7.1.2 Motivation

Les passerelles XML/SNMP existantes [73] se sont principalement intéressées à la partie opérationnelle de l'activité de gestion de réseau, sans vraiment aborder les problèmes de sécurité. Or le déploiement de passerelle XML/SNMP dans un environnement opérationnel réel nécessite l'ajout de mécanismes de sécurité. La motivation principale de notre travail est de fournir une plateforme sécurisée et cohérente dans le cadre des passerelles XML/SNMP, en réalisant un *continuum* de sécurité, c'est-à-dire en assurant que la politique mise en œuvre est la même quel que soit le modèle de gestion de réseau utilisé : SNMP natif, passerelles XML/SNMP, ou XML natif.

Nous présentons, dans ce chapitre, une extension de sécurité pour les passerelles XML/SNMP pour permettre les services d'authentification, de confidentialité et d'autorisation tout au long de la chaîne de gestion. Ces services de sécurité visent à fournir la capacité à réaliser des opérations de gestion sensibles, ayant un impact potentiellement important sur le réseau, ce qui manque dans les premières versions de SNMP. Un tel *continuum* de sécurité est essentiel pour la configuration sûre de nombreux équipements et peut apporter un plus dans un contexte multi-domaine. Nous proposons, dans ce chapitre, une passerelle XML/SNMP étendue avec un mécanisme de contrôle d'accès qui répond à cet objectif.

7.1.3 Qualités requises pour la gestion de la politique de sécurité

La passerelle fournit une politique de sécurité unifiée basée sur le modèle de contrôle d'accès basé sur les rôles (RBAC [52]). Le module de contrôle d'accès doit être capable de créer des sessions utilisateurs et de distribuer les droits d'accès de chaque utilisateur sur les agents SNMP à la volée. Cela simplifie grandement la gestion des droits d'accès puisqu'il suffit de gérer une unique politique centralisée et de la déployer dynamiquement. De plus, éditer et maintenir les politiques d'autorisation RBAC est simple, passe à l'échelle et moins sujet aux erreurs, ceci par construction. L'avantage de notre approche est qu'elle ne nécessite pas de modification du modèle VACM de SNMPv3 puisque sa distribution s'adapte au modèle cible.

7.2 Architecture fonctionnelle

La passerelle XML/SNMP sécurisée, illustrée par la Figure 7.1, embarque une couche SSL pour permettre des communications sécurisées entre les managers et la passerelle. Cette couche est nécessaire pour permettre l'identification du manager, qui, à son tour, est nécessaire pour le processus de contrôle d'accès. Tout d'abord, le manager initie une session chiffrée SSL avec la

passerelle. Ensuite, le manager est identifié et authentifié en fournissant ses login/password via un formulaire HTML.

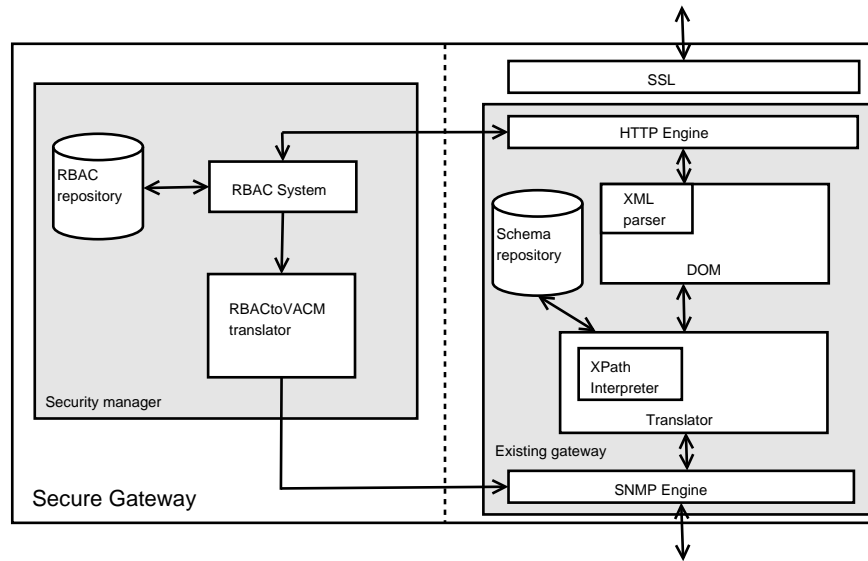


FIG. 7.1 – Architecture fonctionnelle de la passerelle sécurisée

De plus, les nouvelles fonctions de sécurité de SNMPv3 sont difficiles à utiliser et souffrent d'un manque de capacité à résister au facteur d'échelle [55]. C'est pourquoi l'architecture proposée ici utilise un modèle de contrôle d'accès différent et donc indépendant de celui défini dans SNMPv3. Par conséquent, notre architecture embarque un *manager RBAC* pour le processus de décision d'autorisation qui est constitué du *système RBAC* et de la base de données RBAC. Pour pouvoir continuer à gérer les équipements SNMPv3 en natif, c'est-à-dire entre un gestionnaire SNMP et un agent SNMP, il doit être possible de déployer les politiques de contrôle d'accès sur les agents. Comme les modèles VACM et RBAC sont différents, un module de traduction, *RBACToVACM translator*, est également nécessaire. La base de données RBAC possède sa propre politique d'autorisation. La politique décrit :

- l'ensemble des utilisateurs autorisés à interagir avec le système,
- l'ensemble des rôles qui peuvent être endossés par les utilisateurs,
- l'ensemble des objets (ou ressources) sur lesquels des permissions sont définies,
- l'ensemble des permissions qui consistent en une opération sur une ressource,
- l'ensemble des UAs (user assignment) qui décrivent les relations entre utilisateurs et rôles et l'ensemble des PAs (permission assignment).

Un exemple d'une telle politique est illustrée sur la Figure 7.2.c. Le *système RBAC* implante toutes les fonctionnalités nécessaires à la manipulation des données d'autorisation :

- gestion (création, modification, effacement) des composants RBAC (utilisateurs, rôles, permissions) selon les besoins,
- gestion (ouverture, fermeture) des sessions utilisateurs,
- gestion (activation, désactivation) de rôles dans une session,
- évaluation des requêtes d'accès par rapport à la politique.

De plus, quand un nouveau rôle est activé, le *système RBAC* appelle le *RBACToVACM translator* pour mettre à jour les politiques VACM des agents. La plupart des fonctionnalités décrites ici sont détaillées dans [52]. Le *RBACToVACM translator* est responsable de traduire

la politique RBAC de la passerelle sur la politique VACM de l'agent. Différents documents XML sont nécessaires pour réaliser cette traduction. Nous les décrivons dans la section suivante.

7.3 Définition d'une politique d'autorisation

7.3.1 Lien entre RBAC et les données de gestion

Le document XML de la Figure 7.2.c décrit un scénario de politique RBAC. Différents langages existent comme le profil XACML [61] pour RBAC ou encore le langage d'autorisation de .NET MyServices [60]. Tous deux peuvent modéliser une politique RBAC. Bien que ces deux langages soient reconnus conformes avec le standard NIST RBAC, nous proposons notre propre langage XML qui se veut être le plus proche possible du modèle et de la terminologie NIST RBAC, tout en restant suffisamment simple pour les besoins d'un prototype de passerelle XML/SNMP sécurisée. Quoi qu'il en soit, la représentation XML du modèle est d'une importance toute relative car nous proposons ici de traduire un modèle et ses concepts, et non sa représentation XML.

Chaque composant RBAC est décrit indépendamment et possède un identifiant unique (l'attribut XML *id*) de façon à pouvoir y faire référence dans les relations Utilisateur-Rôle (URA) et Permission-Rôle (PRA). Chaque utilisateur (ici, bob et alice) ont un login et un mot de passe. Les rôles sont décrits et identifiés par leur nom. Les portées référencent, en utilisant le langage XPath, les données de gestion XML de la Figure 7.2. XPath a l'avantage d'être simple et compact et d'autoriser un adressage puissant d'ensembles d'éléments d'un document XML. Cette relation est mise en valeur par la première flèche. Chaque permission a une référence vers une portée et une opération. Seul un sous-ensemble des expressions XPath possibles est permis dans notre politique de contrôle d'accès puisque la table *vacmViewTreeFamilyTable* de VACM ne peut contenir que des OIDs associés à des masques. Par exemple, une expression XPath spécifiant une valeur d'attribut spécifique ne peut être convertie vers VACM.

Notons que les ressources XML illustrées par la Figure 7.2.a sont conformes au schéma XML généré à partir de la MIB IF-MIB illustrée partiellement par la Figure 7.2.b. Par exemple, la structure *ifPhysAddress* est décrite dans le schéma XML (cf. la deuxième flèche). Notons également que chaque élément du schéma contient l'OID correspondant pour permettre une traduction inverse. Ces données seront utilisées lors de la traduction de la politique RBAC vers les tables de VACM comme illustré par la troisième flèche.

7.3.2 Algorithme de translation unidirectionnel RBAC/VACM

Pour traduire dynamiquement la politique RBAC sur les agents SNMP, il est nécessaire de traduire la politique XML RBAC dans les tables SNMP VACM. Ce paragraphe fournit un algorithme de traduction des utilisateurs RBAC vers les utilisateurs USM, les rôles RBAC vers les groupes VACM, les portées vers les vues et les permissions vers les entrées de la table *vacmAccessTable*. Le contrôle d'accès ainsi traduit doit refléter le comportement attendu par la politique initiale. Bien que les sessions ne peuvent être exprimées dans VACM, elles peuvent être simulées en traduisant seulement les rôles actifs pour un utilisateur donné. Les tables VACM reflètent les rôles activés, les permissions et les utilisateurs du modèle RBAC. Le modèle RBAC n'est pas complètement traduit vers les tables VACM des agents, mais les permissions sont chargées à la demande sur les agents.

Les différentes étapes de l'algorithme de traduction sont les suivantes : tout d'abord, un groupe est créé pour chaque utilisateur dans la table *vacmSecurityToGroupTable* ; puis, une

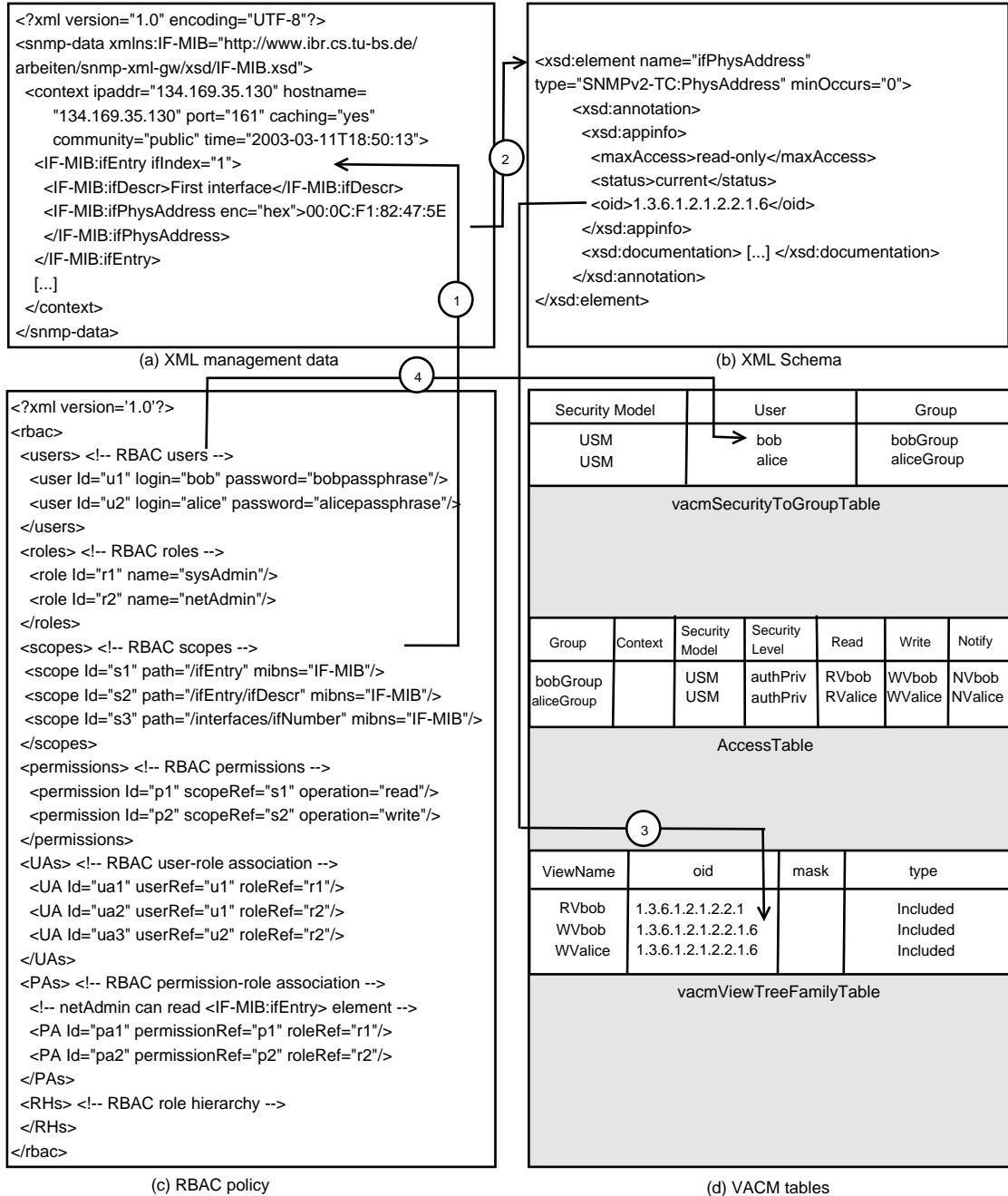


FIG. 7.2 – Schéma général de la traduction RBAC/VACM

entrée comportant trois vues est ajoutée dans la table *vacmAccessTable*. Ces vues sont définies dans la table *vacmViewTreeFamilyTable* comme des ensembles de nœuds de la MIB SNMP. Dans la table *vacmAccessTable*, chaque vue correspond à l'une des opérations *read*, *write* et *notify*.

Considérons l'exemple développé par la Figure 7.2.c. Notre approche consiste à collecter toutes les permissions associées à tous les rôles actifs d'un utilisateur donné. Un groupe spécifique à l'utilisateur et rassemblant toutes ses permissions est créé. La quatrième flèche de la Figure 7.2.d illustre le fait qu'une nouvelle entrée est créée pour chaque utilisateur RBAC (par exemple, bob) dans la table *vacmSecurityToGroupTable*, avec USM comme modèle de sécurité par défaut et un nom de groupe reflétant le nom de l'utilisateur (bobGroup). La Figure 7.3 montre l'algorithme en pseudo-code d'une activation de rôle immédiatement après la phase de login, c'est-à-dire sans aucun rôle encore activé.

```
addActiveRole(user u, role r){
  // Example: ("USM", "Bob", "Bobgroup")
  AddSecurityToGroupTableEntry(USM, u.securityName,
                              u.securityName+"group");

  // Example: (Bob, "", USM, authpriv, "", "RVBob", "WVBob", "NVBob")
  AddVacmAccessEntry(u.securityName, "", USM, authpriv, "",
                    "RV"+u.securityName, "WV"+u.securityName,
                    "NV"+u.securityName);

  Foreach permission p of r{
    // Example: ("R"+"V"+"Bob", "1.3.6.1.2.1.2.2.1.6", "FF", "included");
    addVacmViewTreeFamilyEntry(p.operation+"V"+u.securityName,
                              p.getOid, mask, included);
  }
}
```

FIG. 7.3 – Algorithme de traduction de RBAC vers VACM

Les permissions sont triées par opération, de façon à construire les vues disponibles pour cet utilisateur. L'algorithme utilise le schéma XML du dépôt (figure 7.2.b) pour retrouver les OIDs correspondants à cette portée. Remarquons qu'une portée contient une expression XPath et l'attribut *mibns* relatif à une MIB particulière. La table *vacmAccessTable* illustrée Figure 7.2.d va contenir une entrée pour chaque groupe. Par exemple, pour construire la vue en lecture de bob, il faut :

1. rassembler toutes les permissions actives de bob dont l'opération est *read*,
2. retrouver leur OID à l'aide du schéma XML,
3. ajouter les entrées correspondantes dans la table *vacmViewTreeFamilyTable* pour construire une vue complexe.

BobGroup est la réunion des deux rôles *SysAdmin* et *NetAdmin*. Les rôles peuvent être associés à plusieurs permissions communes. Sans contraintes de séparation des pouvoirs, *Bob* peut activer ses deux rôles disponibles en même temps dans une session. Par conséquent, *Bob* peut lire *1.3.6.1.2.1.2.2.1* et écrire dans le sous-arbre *1.3.6.1.2.1.2.2.1.6*. La table *vacmViewTreeFamilyTable* résultante aura les entrées montrées sur la Figure 7.2.d. Les permissions sont mises à jour dynamiquement sur les agents SNMP seulement lorsqu'un utilisateur demande l'activation

ou la désactivation d'un rôle. Il est toujours possible de mettre en place des contraintes de séparation des pouvoirs contrôlées du côté de la passerelle : comme la passerelle est la seule entité qui peut configurer les permissions sur les agents SNMP, la passerelle peut empêcher un utilisateur d'activer un rôle en fonction des contraintes précédemment citées.

C'est le fonctionnement de base de l'algorithme. Il est possible que plusieurs entrées identiques soient définies dans la table *vacmViewTreeFamilyTable*. Le nombre d'entrées peut être optimisé lorsque, pour une vue donnée, l'accès est donné à une branche *b* et à une sous-branche de *b*. Comme nous utilisons le modèle RBAC NIST standard qui ne définit pas de contraintes de contexte, toutes les entrées utilisent le modèle de sécurité USM associé au niveau de sécurité *AuthPriv*. Mais il est toujours envisageable de rajouter ces contraintes et donc de créer des vues différentes en fonction du contexte. Ce contexte RBAC devrait être défini en utilisant le modèle RBAC comme décrit dans [63]. Cet algorithme de traduction sert de base pour l'ajout du module RBAC dans la passerelle XML/SNMP.

Pour améliorer la facilité d'usage du manager RBAC, il peut être très intéressant d'utiliser un adressage des objets basé sur XPath [16]. XML DOM [81] peut alors être utilisé pour traduire une expression XPath vers des OIDs et des vues VACM. Cela permet également de faire un contrôle d'accès de niveau XML.

7.4 Scénario d'utilisation

7.4.1 Processus de contrôle d'accès

Chaque manager (utilisateur dans la terminologie RBAC) possède un login/password. Ce login/password est un secret partagé entre le manager et la passerelle. Ces tokens de sécurité sont utilisés pour générer les clés SNMP d'authentification et de confidentialité. Par conséquent, le manager et la passerelle sont tous deux capables de faire de requêtes sécurisées puisqu'ils connaissent le mot de passe du manager. Ainsi, un manager peut gérer des agents SNMP sans passer par la passerelle, à condition que les permissions soient déployées sur les agents gérés.

Les managers peuvent utiliser une session RBAC permanente. Un manager doit explicitement fermer une session RBAC. Lorsque cet événement se produit, les permissions de tous les agents gérés par ce manager doivent être effacées. Un manager peut également se déconnecter de la passerelle sans fermer sa session RBAC, pour que ses droits d'accès restent déployés sur les agents SNMP. Les managers peuvent activer et désactiver des rôles quand c'est nécessaire. Si un manager se déconnecte de la passerelle sans fermer sa session RBAC, la passerelle n'efface pas les droits d'accès sur l'agent de façon à ce que le manager puisse continuer à faire des requêtes SNMPv3 sur l'agent sans passer par la passerelle. Dans cette approche, les sessions RBAC peuvent être vues comme persistantes ou à durée de vie prolongée. Ceci évite une traduction de RBAC vers VACM et une remise à plat des droits d'accès trop fréquents, c'est-à-dire à chaque login ou logout du manager. D'où une réduction du nombre de requête VACM.

Lorsqu'un manager active un rôle sur la passerelle, la passerelle met à jour les permissions disponibles dans les tables VACM de l'agent. Pour mettre à jour les permissions d'un agent, le système RBAC traduit les permissions de tous les rôles actifs de cet utilisateur.

Quand la passerelle reçoit une requête d'un manager, elle utilise le login/password de ce manager pour générer la requête SNMP. Ainsi, l'agent sait qui est à l'origine de la requête et effectue le contrôle d'accès en conséquence. Tous ces mécanismes sont donc transparents pour l'agent. Le login/password de chaque manager est stocké dans le modèle RBAC dans la partie utilisateur. Pour simplifier la gestion login/password, la passerelle utilise le même login/password pour l'identification du manager sur la passerelle et sur les agents avec SNMPv3. Cependant, ce n'est

pas une contrainte forte et cela permet à la passerelle de ne stocker qu'un couple login/password par manager.

7.4.2 Prise en comptes de nouveaux agents

Quand un manager veut accéder à un nouvel équipement via la passerelle, ce dernier crée un nouveau compte en clonant son propre compte. Le manager active un rôle sur la passerelle, qui traduit les permissions associées dans les tables VACM de l'agent. Ensuite, le manager peut commencer à émettre des opérations de gestion, soit via la passerelle, soit directement vers l'agent SNMPv3. Ainsi, les équipements tout juste déployés sont auto-configurés pour la partie contrôle d'accès. La seule contrainte d'initialisation est que la passerelle doit disposer d'un compte super-utilisateur sur les agents SNMPv3.

7.4.3 Une interface pour l'administration de la passerelle

Au niveau de la passerelle, un utilisateur spécial a le droit particulier de modifier la politique de contrôle d'accès présente sur la passerelle, c'est-à-dire de réaliser des opérations de maintenance (création d'utilisateurs, de rôles, ...). Lorsque cet utilisateur définit des permissions avec XPath, il n'a pas à se soucier de la traduction vers VACM. Cette opération est transparente pour lui. Tous les autres utilisateurs de la passerelle ne peuvent qu'ouvrir et fermer des sessions RBAC, et activer ou désactiver des rôles.

7.4.4 Un continuum d'authentification et de confidentialité

Pour permettre l'identification des managers sur la passerelle, un message contenant les tokens de sécurité (login/password) doit être envoyé vers la passerelle. Ce message devrait être chiffré pour éviter l'interception de ces tokens par un tiers. De plus, il faut que la passerelle soit authentifiée par les managers avant que ceux-ci n'envoient leurs login/password. C'est pourquoi la passerelle sécurisée utilise SSL (Secure Socket Layer [35]) pour l'authentification de la passerelle via un certificat X509 et l'établissement d'une session chiffrée. Le certificat X509 doit être délivré par une autorité de certification en laquelle les managers ont confiance.

Le password du manager est également utilisé pour générer les clés secrètes SNMPv3. Comme la passerelle et le manager ont la connaissance de cette clé, ils sont tous deux capables de générer les clés nécessaires à l'authentification et au chiffrement des messages SNMPv3. L'avantage est qu'un manager peut continuer à gérer les agents SNMPv3 même si la passerelle est en panne.

Le prototype réalisé utilise le framework net-snmp⁶, disponible librement sur la plupart des distributions Linux.

7.5 Synthèse

L'approche proposée dans ce chapitre a trois avantages par rapport à l'existant. Tout d'abord, elle facilite le déploiement des politiques de sécurité pour SNMPv3, en automatisant la distribution de droits d'accès. Ceci a pour conséquence d'accélérer la configuration des nouveaux agents, et d'harmoniser les politiques de sécurités qui sont éparpillées à travers le réseau sur un nombre potentiellement grands d'équipements.

Ensuite, la conséquence de l'utilisation de RBAC est que seuls les droits strictement nécessaires sont déployés à un instant donné au niveau d'un agent. Cela contraint fortement dans le

⁶<http://www.net-snmp.org>

temps les possibilités d'usurper l'identité d'un manager et de bénéficier de ses privilèges. Dans l'espace également, puisque seuls les agents voulus sont configurés. Cela limite également les risques d'erreur de configuration puisqu'on limite le champs des opérations autorisées.

Enfin, cette approche masque la complexité du modèle VACM en proposant l'utilisation d'un modèle de contrôle d'accès éprouvé et de technologies connues pour l'adressage des ressources protégées.

Implantation d'une suite logicielle pour Netconf

Sommaire

8.1	EnSuite : une plateforme Netconf de configuration de réseau . . .	99
8.1.1	La plateforme EnSuite	99
8.1.2	Le côté agent de la plateforme EnSuite	100
8.1.3	Manager	104
8.2	Impact de la sécurité sur les performances	104
8.2.1	Performances générales	104
8.2.2	De l'utilisation de la compression	107
8.2.3	Intégrité et confidentialité	108
8.2.4	Combiner compression et chiffrement	109
8.2.5	Performance du filtrage XPath comparée au subtree filtering	109
8.2.6	Impact de la politique de contrôle d'accès	110
8.2.7	Sécurité XML intégrée vs SSH	110
8.3	Synthèse	111

Nous avons implanté le protocole Netconf dans une suite logicielle baptisée EnSuite⁷. Les raisons de ce développement sont multiples. Tout d'abord, au démarrage de nos travaux, il n'existait pas encore d'implantation open source de Netconf, donc ce prototype apporte à la communauté une base de travail pour tester les fonctionnalités et les performances du protocole par rapport à l'existant et ceci dans différents cas d'échelle de réseau ou de sécurité. Dans ce chapitre, nous menons une étude de performance avec un intérêt particulier pour l'aspect sécurité. Une implantation est un excellent moyen de comprendre les problématiques de la configuration de réseau et de confronter les acquis théoriques avec la réalité bas niveau d'un protocole tel que Netconf.

8.1 EnSuite : une plateforme Netconf de configuration de réseau

8.1.1 La plateforme EnSuite

EnSuite implante l'agent (YencaP) et le manager (YencaPManager) du protocole Netconf. Typiquement, un gestionnaire ouvre une session HTTPS (via un navigateur web) sur le serveur

⁷<http://libresource.inria.fr/projects/ensuite>

web de YencaPManager. Au cours de cette session HTTPS, un gestionnaire peut ouvrir plusieurs sessions Netconf avec des agents différents. Les connexions sont sécurisées des deux côtés de YencaPManager avec SSL pour la partie HTTP et SSH pour Netconf. La Figure 8.1 illustre la plateforme.

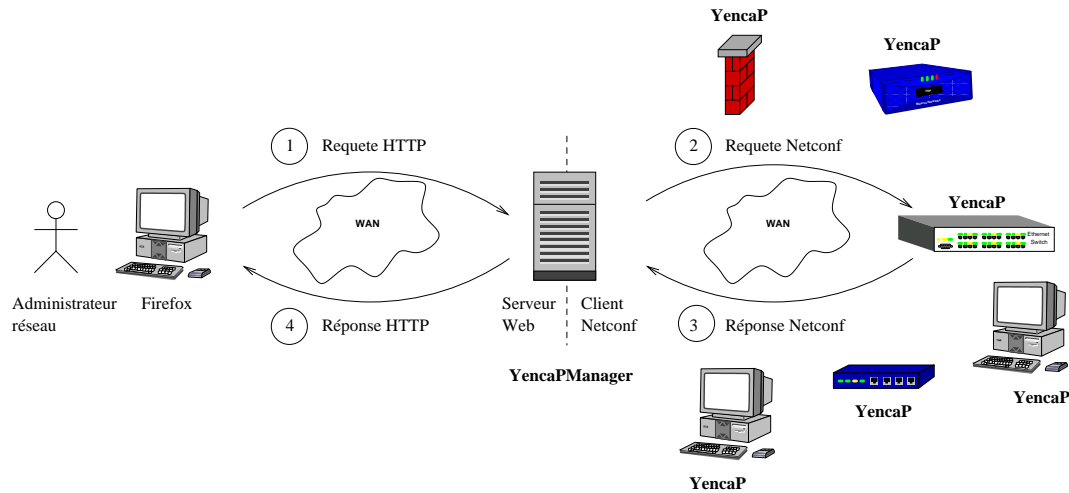


FIG. 8.1 – Vue globale de la plateforme EnSuite

Le modèle d'information de la plateforme est modulaire et extensible et permet de gérer notamment des routeurs BGP et RIP. L'ensemble des opérations supportées peut également être étendu sans modification interne du code. Nous avons illustré cette capacité en introduisant une nouvelle opération pour le contrôle d'accès.

8.1.2 Le côté agent de la plateforme EnSuite

Une architecture en couches calquée sur le protocole

L'architecture de l'agent (YencaP), illustrée par la Figure 8.2, suit à la lettre les sous-couches protocolaires de Netconf. Elle définit une classe standard pour chacune des couches *Content*, *Operation*, *RPC* et *Transport Protocol*. Chacune de ces classes communique par l'intermédiaire d'API bien définies. Par exemple, tous les modules ont une interface de programmation commune héritée de la classe générique *Module*. De même, toutes les opérations Netconf sont implantées dans des classes respectives et héritent de la classe générique *Operation*. En ce qui concerne les réponses aux appels de méthode précédemment cités, nous avons définis des classes génériques qui servent à tout module et toute opération. Par exemple, les modules YencaP répondent à la couche *Operation* avec des objets *ModuleReply*. Ainsi, il est beaucoup plus simple d'implanter un nouveau module puisqu'il suffit d'utiliser les classes existantes. C'est également beaucoup plus propre pour la gestion des erreurs définies dans le standard Netconf car, toutes les erreurs sont instanciables facilement grâce à la classe *ModuleReply*.

Aujourd'hui, grâce à Netconf et XML, il est possible d'aboutir à un niveau de sûreté du logiciel très important par rapport aux autres protocoles de gestion de réseau existants. En effet, le protocole Netconf étant décrit par une grammaire de type XML schema, il est possible de vérifier la validité syntaxique et grammaticale de tous les messages Netconf entrant et sortant. De plus, les configurations elles-mêmes peuvent être vérifiées à condition qu'un schéma XML ou autre ait été fourni avec le modèle de données. Une telle validation permet de détecter très simplement un

très grand pourcentage des erreurs potentielles involontaires ou volontaires de programmation. Des outils externes à l'implantation du protocole sont disponibles pour réaliser l'ensemble de ces vérifications, ce qui est un gage de sécurité et permet de profiter des améliorations futures des bibliothèques de validation utilisées.

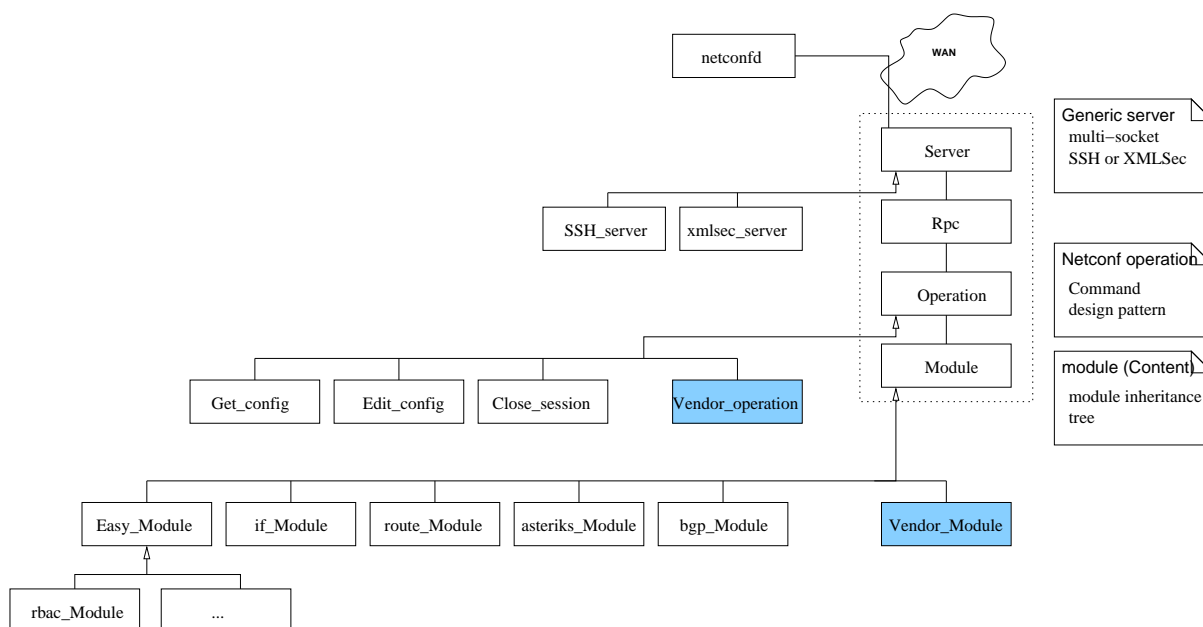


FIG. 8.2 – Structure logicielle de EnSuite

Une modularité à tous les niveaux

Serveur Yencap fonctionne par défaut sur SSH. Mais il peut également utiliser un serveur XMLSec qui chiffre tous les messages en respectant la norme WS-Encryption. Cette norme permet de chiffrer les échanges tout en conservant un codage XML. WS-Encryption est une des briques de sécurité qui a été développée dans le cadre des Web services. En combinaison avec XMLSec, il est possible de compresser les messages ce qui a deux avantages : d'une part, la réduction de la taille des messages et d'autre part, de meilleures propriétés cryptographiques car la compression des données rend beaucoup plus difficile la reconnaissance de patterns.

Bien que nous ne l'ayons pas implanté, la couche Server est suffisamment générique pour être étendue aux protocoles SOAP et BEEP qui sont les deux autres protocoles de transport possibles de Netconf.

Opération EnSuite étant un prototype destiné à fournir une plateforme de tests pour les chercheurs ou même les industriels, nous avons organisé les opérations Netconf de façon à pouvoir ajouter de nouvelles commandes sans changer quoi que ce soit dans le code de la pile Netconf. Le design pattern *Command* répond exactement à cette problématique car il découple la logique de traitement de la classe concernée. La conséquence est que l'ajout de nouvelles commandes ne modifie pas la classe de départ qui, elle, ne contient plus que l'état de la classe (les attributs). Les commandes ont été classifiées en deux grands ensembles : les commandes de bases qui sont standardisées dans la spécification de Netconf et les commandes d'extension que tout développeur

souhaite ajouter. Un gestionnaire de commandes a été implanté pour répondre à nos besoins. Il est capable de charger de nouvelles commandes à l'exécution simplement en connaissant le nom de la classe. Toutefois, un fichier de configuration est nécessaire pour connaître un minimum d'information sur les opérations Netconf disponibles.

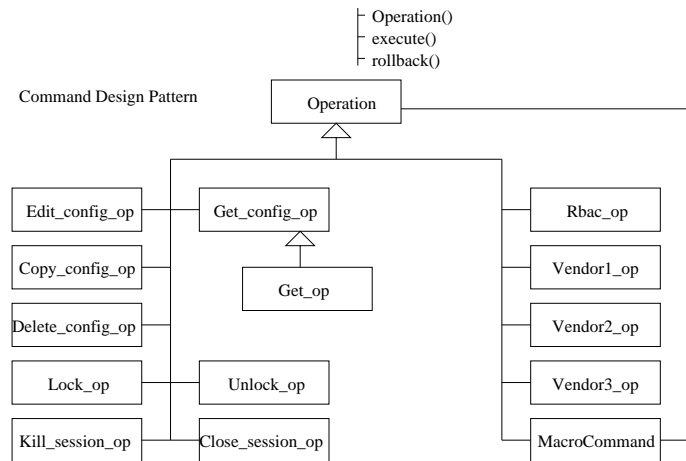


FIG. 8.3 – Design pattern Command

Modules Une approche identique a été suivie pour la partie *Content* de la pile Netconf. Le modèle de données de YencaP est divisé en sous-modèles, chacun étant géré par un module. L'approche est comparable aux modules de MIBs du protocole SNMP. L'extensibilité du modèle de données est assurée par l'ajout de nouveaux sous-modèles qui viennent se greffer dans l'arborescence générale représentant le modèle de données en entier. Pour être interopérable et être intégré avec succès dans YencaP, un module doit hériter de la classe Module qui définit un certain nombre de méthodes communes à tous les modules et implanter un certain nombre de méthode. Chaque module a son propre espace de noms [11] ce qui permet d'accéder aux données sans ambiguïté vis à vis d'un éventuel double emploi d'un nom de balise XML dans deux modules distincts. Les modules peuvent être déployés dans l'agent et chargés à chaud, c'est-à-dire sans nécessiter de redémarrage de l'agent.

Les modules d'ores et déjà disponibles dans YencaP sont :

- BGP pour la configuration de routeurs BGP,
- RIP pour la configuration de routeurs RIP,
- Asterisk pour la configuration du logiciel de VoIP Asterisk,
- RBAC pour l'autoconfiguration de la politique de sécurité de Yencap,
- Route pour la configuration des routes statiques,
- Interfaces pour la configuration des interfaces réseau.

Un cas d'utilisation : RIP

Les routeurs RIP de l'Internet sont usuellement configurés avec l'interface CLI. La figure 8.4 illustre un exemple de configuration de routeur RIP en environnement CLI. La deuxième partie de la figure illustre l'équivalent que nous avons proposé en langage XML. La syntaxe choisie reste le plus proche possible de la syntaxe originale pour faciliter le mapping. En effet, le module RIP pour YencaP s'appuie sur CLI pour une raison simple : cela permet un gain de temps important

puisque l'interface existe donc nous évitons de devoir modifier les logiciels existants et nous pouvons utiliser directement des logiciels de routage comme GNU/Zebra.

```
1 !
2 router rip
3   version 2
4   redistribute kernel metric 1 route-map 1
5   redistribute ospf route-map 4
6   redistribute bgp metric 5
7   network 10.0.0.0/8
8   network eth0
9   neighbor 10.0.0.1
10  neighbor 10.0.0.2
11  passive-interface eth3
12  passive-interface eth8
13  distribute-list private in eth0
14 !
```

```
1 <rip version="2" default-metric="1">
2   <redistribute>
3     <kernel metric="1" route-map="1"/>
4     <ospf route-map="4"/>
5     <bgp metric="5"/>
6   </redistribute>
7   <networks>
8     <network>10.0.0.0/8</network>
9     <network>11.0.0.0/8</network>
10    <network>eth0</network>
11  </networks>
12  <neighbors>
13    <neighbor>10.0.0.1</neighbor>
14    <neighbor>10.0.0.2</neighbor>
15  </neighbors>
16  <passive-interfaces>
17    <passive-interface>eth3</passive-interface>
18    <passive-interface>eth8</passive-interface>
19  </passive-interfaces>
20  <distribute-lists>
21    <distribute-list direct="in" name="private">eth0</distribute-list>
22  </distribute-lists>
23 </rip>
```

FIG. 8.4 – Extrait de configuration relative au protocole RIP

La figure 8.5 illustre l'implantation interne du module RIP. Chaque commande CLI fait l'objet d'une classe ce qui permet de sérialiser facilement les objets en commandes texte ou en document XML. Cela a aussi le grand avantage de pouvoir implanter par la suite les rollbacks de Netconf simplement en complétant les méthodes `do()` et `undo()` typiques du Design Pattern Command.

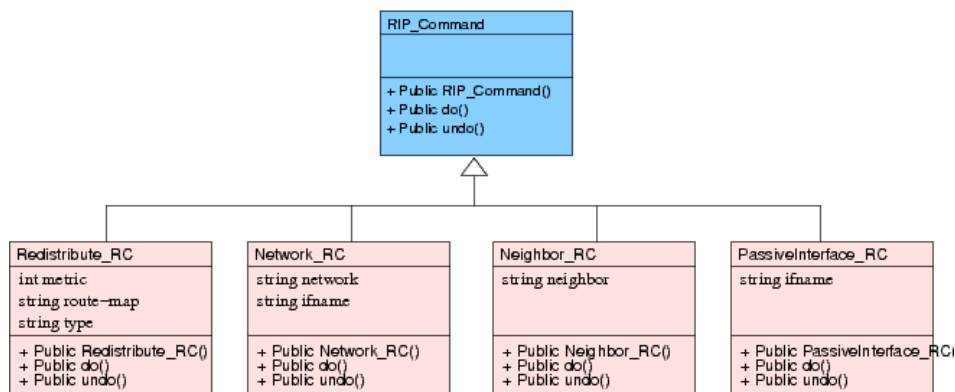


FIG. 8.5 – Diagramme UML des commandes RIP

8.1.3 Manager

Une architecture modulaire

Le logiciel EnSuite ayant vocation de prototype et de permettre la découverte du protocole Netconf, nous avons divisé l'interface web en deux parties. La première cache la logique protocolaire à l'utilisateur. Il s'agit de formulaires HTML standards qui répondent aux besoins d'un modèle de données connu, c'est-à-dire celui que nous avons développé pour EnSuite (BGP, RIP, RBAC, ...). En interne, le gestionnaire web génère de requêtes Netconf en fonction des données remplies dans les formulaires HTML. La seconde partie est beaucoup plus technique et à but éducatif car elle demande au gestionnaire de générer à la main une partie de la requête Netconf : un filtre XML, une requête XPath, une requête edit-config. Cela permet d'interagir plus finement avec Netconf et de découvrir la syntaxe du protocole et ses différents paramètres.

Extensions pour plus d'ergonomie Comme dans l'architecture logicielle de l'agent, il est possible d'étendre le gestionnaire web avec des modules. Ces modules permettent d'interagir avec un modèle de données particulier comme BGP ou RIP. Pour cela, il suffit de fournir une feuille XSLT capable de transformer le document XML contenu dans une réponse Netconf en un formulaire HTML. C'est beaucoup plus confortable pour un administrateur de gérer la configuration avec des formulaires. La deuxième chose à fournir est la génération des requêtes Netconf. En fonction des actions réalisées sur l'interface web, le module doit être capable de générer les requêtes Netconf correspondantes.

8.2 Impact de la sécurité sur les performances

8.2.1 Performances générales

Pour valider et évaluer l'architecture de sécurité proposée et le module de compression, un prototype [23] a été implémenté. Les tests de performance sont réalisés localement sur une seule machine ayant un Pentium 4 3.2GHz avec 2 Go RAM. Les spécifications détaillées sont présentées dans la Table 8.1.

Pour donner un ordre d'idée sur les performances globales, l'agent est capable de traiter jusqu'à 17 opérations get-config par seconde quand il ne journalise pas les opérations dans un fichier et qu'il n'applique aucun mécanisme de sécurité. Lorsque les mécanismes de sécurité de

Description	Caractéristiques
Mémoire RAM	2 Gigabytes
Processeur	Pentium 4, 3.2 GHz
OS	Linux FC4, kernel 2.6.12
python	2.4
librairie sécurité python	PyXMLSec, paramiko
longueur des clés crypto	128 bits

TAB. 8.1 – Caractéristiques de l’environnement de test

confidentialité et de contrôle d’accès (RBAC) sont activés, le nombre d’opérations traitées par seconde passe à 8. La sécurité introduit donc un facteur de ralentissement de 2.

Comme nous nous intéressons aux problématiques de temps de réponse et de sécurité, nous considérons dans la suite le pire cas. Celui-ci correspond à l’utilisation de requête de type *get-config* car, comme c’est le type de requête qui correspond au traitement interne à l’agent le plus rapide, cela maximise l’impact de la sécurité sur le temps de calcul global. Si au contraire, nous effectuons les tests sur des requêtes qui prennent beaucoup de temps à s’exécuter, le temps de calcul consacré aux mécanismes de sécurité devient négligeable et donc difficilement observable. Avec des requêtes rapides, il est possible de mettre en valeur l’influence des mécanismes de sécurité sur le temps de calcul global. C’est pourquoi, dans la suite de cette étude de performances, nous nous limitons aux opérations *get-config*. Cependant, nous comparons les performances des temps de compression et de chiffrement non seulement pour des documents de petite taille mais aussi de grande taille.

La Figure 8.6 illustre différents cas d’utilisation : (1) chiffrement + compression, (2) chiffrement, (3) compression, (4) rien. Les mesures sont réalisées après instrumentation de l’agent. Comme c’est souvent le cas, l’instrumentation a un léger impact sur les performances car des accès disques sont effectués pour stocker les résultats. Dans la suite, nous considérons cette instrumentation comme négligeable. Pour ce test, 7 requêtes différentes de type *get-config* sont effectuées, soit utilisant la méthode de sélection *subtree filtering* soit *XPath*. Chacune est réalisée 64 fois et une moyenne est calculée à partir de ces résultats. La moyenne est représentée sur la Figure 8.6. Chaque colonne représente le temps total du traitement de la requête et détaille le pourcentage d’occupation de chaque mécanisme.

Les résultats montrent que le temps de compression et de décompression est négligeable par rapport au temps global de traitement de la requête. Ceci est probablement dû au fait que ces bibliothèques sont très matures et sont souvent des wrappers de bibliothèques C, donc rapides par rapport à l’exécution classique de code python. Le mécanisme de contrôle d’accès RBAC occupe environ 6% du temps global, ce qui est acceptable. Le chiffrement représente 9.5% alors que le déchiffrement représente 3.6% du temps global. La concaténation de tous ces mécanismes d’extension par rapport au protocole *Netconf* standard (mais sans protocole *SSH*) crée un overhead de 20% du temps total. Evidemment, l’utilisation de *SSH* introduit également un surcout au même titre que *XMLSec*.

Notons que le temps de calcul total augmente par un facteur 60 si l’agent procède à la validation syntaxique systématique des messages reçus avec le schéma XML du protocole. Le temps total moyen passe, en effet, de 11 ms à 650 ms. Evidemment, dans de telles conditions, le pourcentage du temps total dédié aux mécanismes de compression et de chiffrement diminue fortement. Il est important de noter que l’utilisation CPU et mémoire induite par la technologie *XPath* est toute relative par rapport à un processus de validation XML. Pour illustrer ces propos,

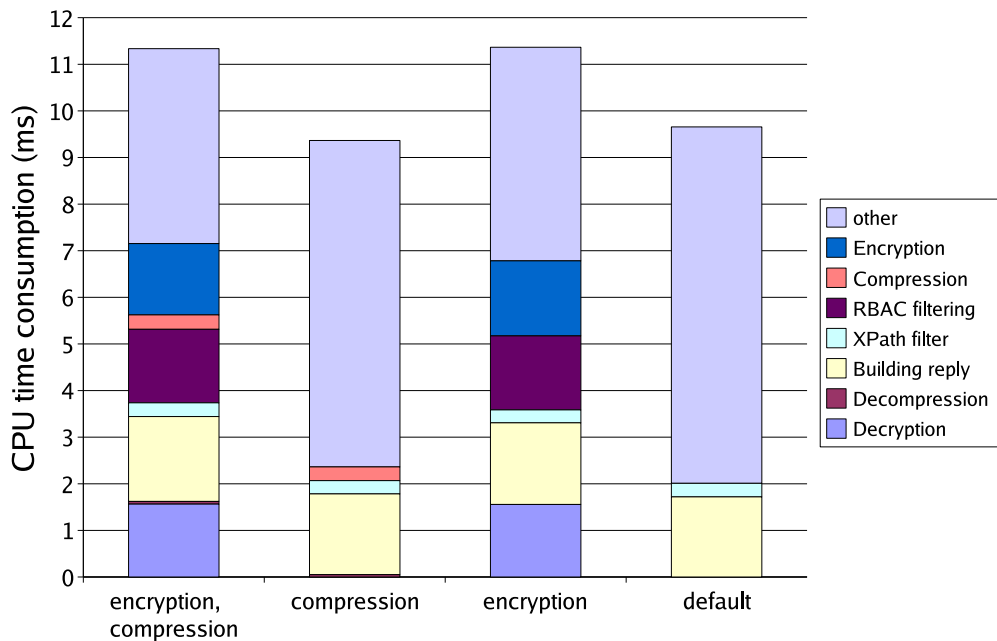


FIG. 8.6 – Temps de traitement CPU global d'une requête

nous avons évalué le filtrage d'un document XML avec XPath à 1ms, alors que la validation d'un message Netconf prend plus de 600 ms. Ces résultats apportent un éclairage sur les performances réelles de XPath ainsi que sur les temps de calcul respectifs des différents mécanismes.

La Table 8.2 donne le temps de réponse de plusieurs requêtes typiques qui consistent à récupérer des données de configuration. L'agent et le manager sont localisés sur deux équipements connectés par un hub et communiquent via SSH. Dans le cadre de ce test uniquement, l'agent tourne sur un laptop centrino 1.6GHz avec 1 Gb de RAM et où est installée une distribution Linux (Fedora Core 4). Le temps de traitement global d'une requête par agent reste le même que précédemment, mais du côté manager, on observe les résultats suivants. Le manager envoie 1000 fois une série de requêtes get-config avec XPath comme méthode de filtrage puis, à l'issue du test, le temps de réponse est calculé pour chacune des requêtes. Les résultats montrent que le module BGP utilisé est suffisamment performant pour fonctionner sur un vrai réseau. Quand le test porte sur une configuration BGP très simple, le temps de réponse est de 58.4 ms, alors que avec une configuration plus proche de la réalité en terme de taille, la configuration est chargée en 200 ms en moyenne. Ces résultats montrent que Netconf est assez performant même pour des configurations complexes de type BGP.

Données de gestion	Temps de réponse
Interfaces réseau	38.73
Routes	49.71
Configuration BGP	200.08
Politique RBAC	43.05

TAB. 8.2 – Temps de réponse moyen en ms

8.2.2 De l'utilisation de la compression

La compression apporte plusieurs avantages. D'une part, elle améliore les propriétés de sécurité en éliminant sensiblement la possibilité de reconnaissance par patterns. D'autre part, elle permet de réduire la bande passante utilisée lors de la transmission de configurations XML qui sont susceptibles d'atteindre des tailles importantes. Enfin, la compression améliore les performances du chiffrement en réduisant la taille des données à chiffrer, sachant que le temps de compression est négligeable par rapport au chiffrement.

Pour illustrer l'intérêt de la compression, la Figure 8.7 prend en considération différents exemples de messages Netconf ou de configuration au format XML. Chaque type de document XML est placé sur l'échelle pseudo-logarithmique en fonction de sa taille. Du fait de la distribution hétérogène de la taille des messages, une échelle uniforme n'aurait pas pu représenter ces messages. La taille de nombreux messages Netconf varie entre 100 et 300 octets : `get-config` avec XPath, `ok`, `rpc-reply`, `copy-config`, `lock/unlock`, etc... L'utilisation d'un processus de compression pour ces messages n'a pas un grand intérêt. Nous évaluons à approximativement 50% le pourcentage de ces messages Netconf.

Comparativement, la taille d'un message hello contenant 6 capacités est de 500 octets. La taille d'une configuration RBAC extrêmement simple ou encore d'une configuration des interfaces réseaux ne dépasse pas les 2500 octets. Une configuration BGP, quant à elle, peut varier de 5000 à 10000 octets dans notre modèle. La taille d'un document XML contenant la liste entière des rpm installés sur notre machine est de 25000 octets. La taille globale d'une configuration peut facilement atteindre plusieurs centaines de kilo-octets. La Figure 8.7 donne un ordre d'idée sur la taille des messages susceptibles d'être rencontrés mais ces tailles peuvent exploser en fonction du modèle de données. Par exemple, une requête `edit-config` pourrait tout à fait atteindre les 30 kilo-octets.

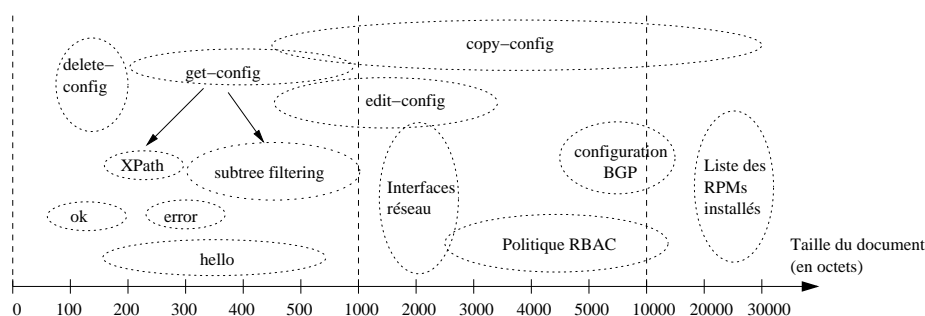


FIG. 8.7 – Aperçu des tailles de messages Netconf

La Figure 8.8 donne les résultats liés à la compression de documents XML arbitraires. Nous avons limité nos mesures à une taille de document avant compression de 25 kilo-octets. Pour cette taille de document, le taux de compression atteint 65%. Ces mesures prouvent le grand intérêt de l'utilisation de mécanismes de compression, car ceux-ci réduisent grandement l'overhead sur la bande passante tout en ayant une consommation CPU très limitée.

La Figure 8.8 illustre la consommation CPU pour la compression et la décompression en fonction de la taille du document. Le temps moyen de compression est 5 fois plus grand que le temps de décompression et augmente très lentement avec la taille du document en entrée, ce qui est très encourageant si la taille des configurations vient à augmenter de façon importante. Dans le pire cas, qui correspond à la situation où le temps de traitement global du message par l'agent

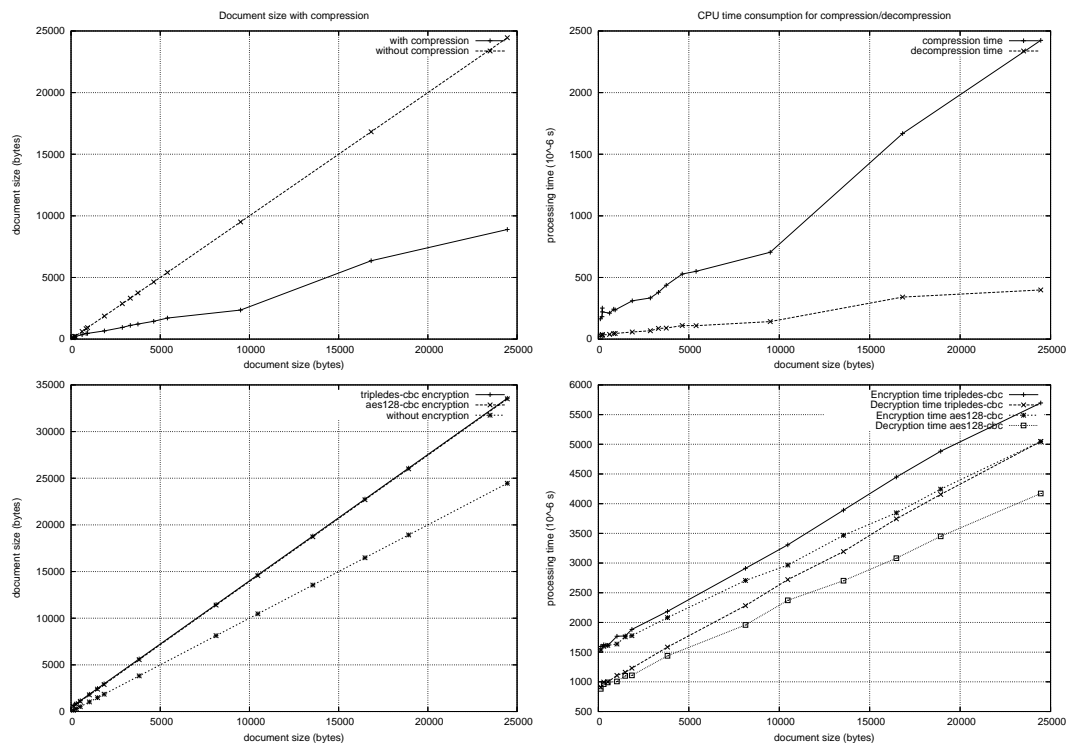


FIG. 8.8 – Résultat de compression et de chiffrement

est très court, les temps de compression et décompression ne prennent quand même que très peu de temps proportionnellement au temps global : moins de 1%.

Pour conclure, il y a un grand intérêt à utiliser la compression dans le cadre de Netconf.

8.2.3 Intégrité et confidentialité

La Figure 8.8 est un graphe représentant les tailles des requêtes Netconf chiffrées ou non puis le temps de calcul, en fonction de la taille originale du document. Les algorithmes de chiffrement sont le *tripleDES-cbc* et *aes128-cbc* utilisant tous les deux une clé de 128 bits. Dans le cadre de ce test, nous considérons un échantillon de documents XML ayant des tailles variables et pour obtenir des résultats cohérents, chaque algorithme de chiffrement est appliqué 100 fois pour chaque document. Ensuite, nous calculons la moyenne pour chaque document. Les résultats montrent que la taille du document chiffré augmente de façon linéaire avec la taille du document original. Il est à noter que la taille des documents chiffrés varie très peu en fonction de l'algorithme de cryptage utilisé ; en revanche, les temps CPU varient d'un algorithme à l'autre. En effet, *aes128-cbc* consomme moins de cycles d'horloge que *tripleDES-cbc*, que ce soit pour le cryptage ou le décryptage. C'est pourquoi nous pensons que *aes128-cbc*, qui est l'algorithme actuellement recommandé par la NSA, est le meilleur choix.

Les processus de chiffrement et déchiffrement prennent moins de 3 ms pour des documents de 10 kilo octets. Dans le pire cas, ces deux processus représentent seulement 13% du temps global du traitement de la requête Netconf. Rappelons que le pire cas se produit lorsque ce temps global est le plus faible, ce qui accroît la proportion du temps de chiffrement par rapport au temps global et lorsque la taille du document résultat est grande. Dans cette situation, la proportion de temps

CPU pour le cryptage/décryptage est maximale.

8.2.4 Combiner compression et chiffrement

Une expérience visant à comparer les temps de calcul dans différentes situations, compression, chiffrement, compression+chiffrement, nous a conduit aux conclusions suivantes. Lorsque la taille du document atteint un certain seuil, il devient plus rapide de compresser et crypter le document que de seulement le crypter. Cela signifie que, à partir de ce seuil, utiliser à la fois la compression et le chiffrement, permet non seulement d'économiser de la bande passante mais aussi de réduire le temps d'occupation CPU de l'agent.

La Figure 8.9 montre que la courbe **compression+encryption** croise la courbe **encryption**. Cet évènement se produit lorsque la taille de document atteint environ 4 ko, ce qui correspond à la taille d'une configuration BGP simple dans notre modèle.

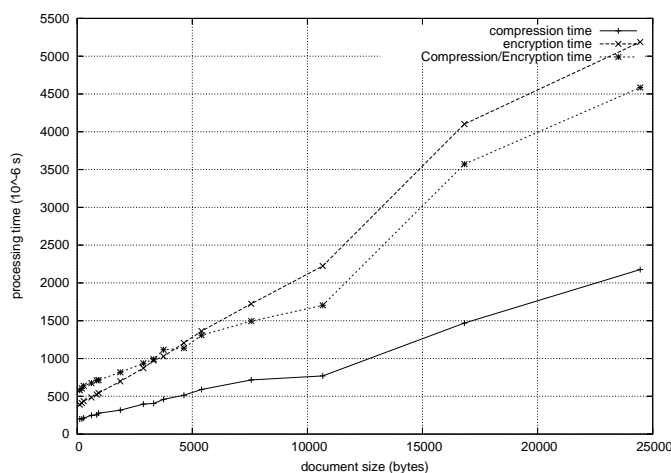


FIG. 8.9 – Temps d'occupation CPU avec différentes options

8.2.5 Performance du filtrage XPath comparée au subtree filtering

De nombreuses affirmations ont été vues sur la liste de diffusion du groupe de travail Netconf à l'IETF, au sujet de la lourdeur de XPath et des mauvaises performances qui en résulteraient. Nous avons souhaité vérifier ces affirmations dans le cadre de notre prototype. A première vue, ce problème n'est pas directement en rapport avec le problème de la sécurité dans Netconf. En réalité, dans notre prototype, les performances du filtrage XPath sont importantes car la politique de sécurité utilise XPath pour décrire les ressources protégées et pour ensuite filtrer les données. Donc, nous faisons d'une pierre deux coups en menant cette étude comparative.

Pour comparer les performances respectives des deux méthodes de filtrage, XPath et subtree filtering, nous avons préparé 8 requêtes de type get-config basées sur le subtree filtering et leurs équivalents avec XPath. Il est toujours possible de convertir une requête de subtree filtering vers une requête XPath. Le contraire est faux car certaines fonctions de XPath ne peuvent être exprimées dans le subtree filtering. Les requêtes sont disponibles à l'url suivante : <http://www.loria.fr/~cridligv/xpath.Subtree.html>. Un maximum de cas de sélection a été inclus dans les requêtes utilisant le subtree filtering : nœuds de sélection, nœuds de sélection par rapport au contenu, nœuds éléments. Ces trois types de sélection sont décrits dans le draft. Chaque

requête est exécutée 1000 fois et la moyenne est ensuite calculée. Au delà des performances, ces expériences nous ont permis de mieux appréhender la complémentarité de ces deux approches de filtrage. Le subtree filtering est adapté pour visualiser le document avant de le peupler avec les données à la manière d'un template, alors que XPath est plus adapté pour sélectionner des nœuds très précisément, à l'aide de critères plus complexes. On peut voir le subtree filtering comme un intermédiaire entre XPath et XSLT : il permet de sélectionner des nœuds mais aussi d'appliquer un masque à la configuration. Pour assembler des sous-arbres complètement différents avec XPath il faut utiliser l'opérateur |, ce qui permet de réaliser une jointure des sous-arbres. XPath a l'avantage de pouvoir sélectionner les nœuds avec un adressage relatif ou absolu, alors que le subtree filtering est forcément absolu.

Les résultats illustrés par la Table 8.3 montrent que les deux approches ont des temps de calcul très semblables. Il est difficile de conclure que l'une ou l'autre de ces méthodes est meilleure. Cela dépend des besoins du manager. Malgré tout, on peut dire que XPath est moins consommateur en bande passante car sa syntaxe est beaucoup plus compacte que le subtree filtering.

Pour pondérer l'importance de ces résultats, il est important de noter que le temps de filtrage est bien inférieur au temps global du traitement de la requête, et ce, quelle que soit la méthode de sélection. En conclusion, il est faux de dire que XPath est très consommateur de ressources. Les deux approches sont complémentaires et ont des performances similaires.

Requête \ Filtrage	1	2	3	4	5	6	7	8	9	10	11	12
Subtree	1171	1482	694	463	477	1503	2114	1371	919	775	1733	863
Xpath	1173	1039	1219	194	687	1286	2828	1973	852	916	1332	2368

TAB. 8.3 – Performances des méthodes de filtrage en μs

8.2.6 Impact de la politique de contrôle d'accès

Nous avons aussi souhaité montrer l'impact du contrôle d'accès sur le temps d'occupation CPU au niveau de l'agent. Pour cela, une politique de contrôle d'accès basée sur les rôles (RBAC) a été écrite. Celle-ci repose sur un schéma d'adressage XPath. Un superviseur a été implanté dans l'agent pour filtrer les documents résultats en fonction de la politique de sécurité et ceci, pour les deux grandes classes d'opération *get* et *set*.

Dans notre modèle, une requête Netconf s'exécute dans un contexte de sécurité. Un utilisateur a activé un certain nombre de rôles à l'instant t , ce qui lui procure des permissions. L'agent est en mesure de construire cette liste des permissions dynamiquement en prenant en compte la hiérarchie de rôles de la politique de sécurité. Les résultats, illustrés par la Figure 8.10, donnent le temps de calcul supplémentaire introduit par le mécanisme de contrôle d'accès. Les valeurs sont données en fonction du nombre de permissions actives. Dans cette expérience, nous nous limitons à un nombre de permissions variant de 0 à 9. Le temps CPU augmente linéairement avec le nombre de règles, ce qui est normal car les expressions XPath sont appliquées en série.

8.2.7 Sécurité XML intégrée vs SSH

A un autre niveau du protocole Netconf, il peut être intéressant de comparer différents protocoles de transport. Pour étudier les performances de deux différents protocoles de transport sécurisés, en l'occurrence notre framework de sécurité basé sur XML et SSH, l'environnement

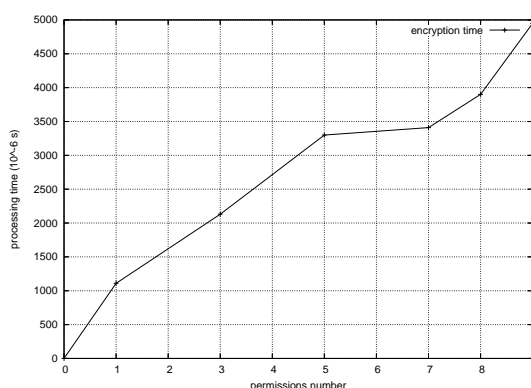


FIG. 8.10 – Impact du nombre de permissions sur le temps d'évaluation

suivant a été testé. Un gestionnaire Netconf utilise tour à tour ces deux protocoles pour envoyer 1000 fois une requête de type get-config avec XPath. Cette requête est la 8^{ème} de la Table 8.3.

Le temps total entre l'instant de l'émission de la première requête et l'instant de réception de la dernière réponse s'élève, pour SSH, à 16.77 secondes et pour XMLSec à 15.62 secondes. Un très léger avantage est à mettre au compte de SSH. Comme nos mesures sont relatives à une requête, l'établissement de la session SSH n'est pas pris en compte, de même que la distribution des clés dans XMLSec.

8.3 Synthèse

Dans ce chapitre, nous avons prouvé qu'il était réalisable d'associer à chaque agent Netconf une politique de contrôle d'accès basée sur les rôles. Les temps de décision relatifs au contrôle d'accès sont améliorés du fait de la notion d'activation de rôles. En effet, seules les permissions actives sont évaluées donc le rôle introduit comme pour le nombre d'utilisateurs un facteur de passage à l'échelle.

Nous avons également réalisé une série de tests avec différentes options, respectivement le chiffrement et la compression, tout en considérant une plage de tailles de message Netconf large pour observer le comportement du protocole dans différentes situations. Pour cela, nous avons fourni à titre indicatif des estimations sur la taille des messages Netconf usuels.

Il est toutefois important de prendre les résultats donnés avec prudence car ceux-ci sont dépendants de notre plateforme de test : processeur, mémoire, langage de programmation Python, design de l'implantation.

Conclusion

Conclusions et perspectives des recherches

Sommaire

9.1	Conclusions	115
9.2	Perspectives	116
9.2.1	Performances des modèles de sécurité dans le cadre de la supervision de réseau	116
9.2.2	Contrôle d'intégrité des configurations	116
9.2.3	Une approche de fédération des identités	117
9.2.4	Couche logicielle de convergence bas niveau	117
9.2.5	Représentation des données	118

9.1 Conclusions

Dans cette thèse, nous nous sommes intéressés à l'environnement fortement distribué et hétérogène composé des plateformes de supervision de réseau IP. Nous avons rappelé les services de sécurité existants et les différents choix de modélisation de sécurité de ces plateformes. Nous avons extrait leurs divergences vis à vis de chacun des services de sécurité pour mettre en valeur le phénomène d'hétérogénéité.

Nous avons identifié les conflits potentiels qui peuvent naître de cette hétérogénéité en définissant deux niveaux de cohérence de politiques de sécurité et plus particulièrement relatives au contrôle d'accès : cohérence locale et cohérence globale. La première concerne la prise en compte des politiques de sécurité de plusieurs plateformes de supervision sur un même équipement. Des incohérences sont susceptibles d'apparaître lorsque il y a recouvrement des modèles de données de plusieurs plateformes auquel cas les politiques de sécurité peuvent donner lieu à des prises de décision contradictoires. Ce problème de cohérence locale est en partie un problème de fédération des identités car un manager peut avoir accès à plusieurs plateformes avec des noms différents. C'est également un problème de définitions des privilèges et des différences de structure du modèle de données d'une plateforme à l'autre. La seconde est en grande partie liée à la capacité du protocole à déployer sa propre politique de sécurité à grande échelle. La cohérence globale consiste à observer les mêmes règles de sécurité sur des équipements de même nature.

Pour résoudre ces deux problèmes de cohérence, nous avons proposé différentes approches essentiellement basées sur le modèle de contrôle d'accès basé sur les rôles. Une approche pour la

cohérence locale consiste à être capable soit de convertir une politique vers plusieurs politiques de destination exprimés dans différents référentiels ou espace d'adressage, soit à l'inverse de vérifier d'être capable de convertir les politiques déployées vers un référentiel unique. La première approche peut être qualifiée de *top-down* ou active alors que la deuxième est plutôt *bottom-up* et peut s'associer à des actions correctives si des conflits sont détectés. Nous avons exploré les deux directions avec d'une part des algorithmes de conversion du modèle RBAC vers le modèle de sécurité de CLI ou celui de SNMP, et d'autre part une conversion formelle inverse pour exprimer les politiques déployées dans un modèle et une syntaxe communs et ensuite comparer les droits des différentes entités.

Pour améliorer la cohérence globale, nous avons proposé une extension de sécurité pour les passerelles XML/SNMP qui permet de déployer dynamiquement et à la demande une même politique de contrôle d'accès sur un ensemble de machines. Nous avons également étendu le protocole Netconf pour que l'agent puisse contrôler l'accès aux données qu'il gère. Ces règles d'accès sont administrables via Netconf lui-même ce qui autorise un déploiement à grande échelle et donc favorise la cohérence globale.

Nous avons réalisé des tests de performance de notre implantation du protocole de configuration Netconf en nous focalisant sur les aspects sécurité. Les résultats montrent que le contrôle d'accès basé sur XPath est faisable avec un pourcentage de temps dédié au contrôle d'accès d'environ 20% par rapport au temps de traitement complet d'une requête par un agent. Les résultats sont très dépendants du nombre de permissions applicables en fonction des rôles activés ce qui est un argument supplémentaire en faveur de RBAC. En effet, seules les permissions actives sont évaluées donc cela accélère énormément les temps d'évaluation de la politique de contrôle d'accès. Nous avons également redémontré l'intérêt d'utiliser un algorithme de compression en plus d'un algorithme de chiffrement dans le contexte de Netconf.

9.2 Perspectives

9.2.1 Performances des modèles de sécurité dans le cadre de la supervision de réseau

Dans cette thèse, nous avons analysé les performances de notre implantation de Netconf. Il serait intéressant d'étendre cette étude en incluant les autres protocoles de supervision et en jouant sur les options de sécurité (confidentialité et/ou authentification) et sur la complexité des politiques de contrôle d'accès. [17] a proposé une analyse de performance de SNMPv3 par rapport à SNMPv2 en prenant en compte les aspects sécurité.

Il est tout à fait envisageable de réutiliser ces travaux et ceux de cette thèse pour généraliser cette étude de performances orientée sécurité. Pour cela, il suffirait de générer des politiques de sécurité équivalentes et de les déployer sur un équipement. Le fait que la cohérence locale soit assurée garantirait la validité de la comparaison inter-protocoles car on pourrait affirmer qu'avec une même politique, l'un ou l'autre des protocoles, en tout cas son implantation, est plus performant.

9.2.2 Contrôle d'intégrité des configurations

Une des attaques difficile à détecter car elle n'affecte pas le fonctionnement du réseau est la reconfiguration silencieuse des équipements en vue de préparer des attaques ultérieures. Une solution à ce problème serait de signer les configurations des équipements et de vérifier périodiquement que celles-ci sont toujours valides. Il se trouve que ce mécanisme est extrêmement

simple à mettre en œuvre dans le cas de Netconf car il existe des spécifications [4] pour créer des signatures de documents XML, qui dans notre cas serait la configuration de l'équipement. Une extension du protocole Netconf permettrait alors d'interroger l'agent pour vérifier que la configuration est toujours celle qui a été déployée de façon licite. Cette solution peut également être appliquée avec les autres plateformes de supervision.

9.2.3 Une approche de fédération des identités

Parallèlement à nos travaux visant à uniformiser les modèles de sécurité de plateformes hétérogènes, des progrès importants ont été faits avec SAML (Security Assertion Markup Language [57]) pour la fédération des identités dans le contexte des services webs. L'objectif recherché est l'authentification unique (Single Sign-On) de l'utilisateur qui accède à plusieurs sites coopérants : site de réservation en ligne de billets d'avion, de location de voiture, d'hôtel. L'idée est que les sites se transmettent mutuellement des assertions de sécurité contenant les preuves d'authentification, les attributs et les privilèges du consommateur. Des alias sont utilisés pour fédérer les différentes identités de l'utilisateur sur chacun des sites.

On pourrait tout à fait transposer cette vision de l'uniformisation dans le cadre de la supervision de réseau car le contexte est similaire : plusieurs sites (les agents), plusieurs utilisateurs (les managers) ayant des comptes sur plusieurs sites et même plusieurs comptes utilisateurs par site. Les assertions d'autorisation seraient alors exprimées dans un référentiel commun (représentation Meta-CLI compact), ce qui faciliterait grandement la prise de décision pour le contrôle d'accès.

9.2.4 Couche logicielle de convergence bas niveau

Une approche totalement différente pour assurer la cohérence locale serait de développer une couche logicielle de convergence bas niveau entre les protocoles et le système, puis placer le contrôle d'accès à cet endroit. Ainsi on assurerait que le contrôle d'accès est le même quelque soit le protocole. Cependant, la faisabilité et le coût d'une telle approche sont prohibitifs, car cela suppose que toutes les implantations reposent sur cet API commune. Cette approche n'est pas vraiment envisageable, compte tenu de l'évolution incessante des matériels réseaux et des constructeurs qui ont leurs propres outils de gestion. Elle serait possible à condition de maîtriser la chaîne de production (hardware et logicielle) de bout en bout.

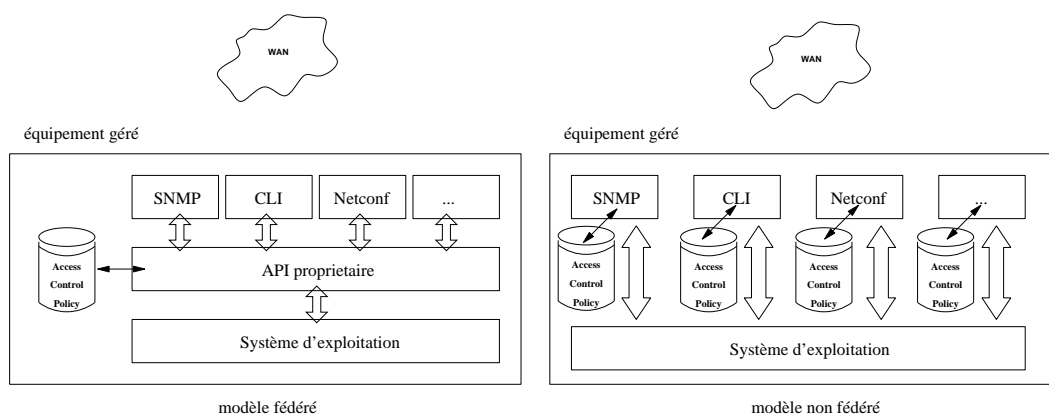


FIG. 9.1 – Deux approches pour le déploiement de politiques de contrôle d'accès

La Figure 9.1 illustre la différence entre l'approche suivie dans ce manuscrit et l'approche

présentée dans cette section. On remarque qu'il n'y a qu'une occurrence de politique de contrôle d'accès dans l'approche fédérée alors qu'il y en avait plusieurs, c'est-à-dire une pour chaque plateforme de gestion, dans les approches précédemment exposées, non fédérées.

9.2.5 Représentation des données

Les limites énoncées à la fin du chapitre 5 font état d'un manque au niveau de la comparaison des données au travers de leur représentation. Il serait intéressant à plus long terme de développer des modèles et des outils pour faciliter la comparaison de ressources décrites dans des représentations incompatibles. Comme les jeux de données sont de taille relativement importante dans le domaine de la supervision des réseaux, des techniques d'apprentissage automatique et progressif pourraient être mises en œuvre.

Publications

Journaux internationaux

XBGP-MAN : an XML management architecture for BGP V. Cridlig, H. Abdelnur, R. State, O. Festor, IJNM : International Journal of Network Management, Volume 16 , Issue 4 (July 2006), Pages : 295 - 309, ISSN :1099-1190, John Wiley & Sons, Inc., New York, NY, USA

Role-Based Access Control for XML Enabled Multi-Protocol Management Gateways V. Cridlig, R. State, O. Festor, eTNSM : eTransactions on Network and Service Management (IEEE), Vol.3 No.2 Second Quarter 2006

Conférences internationales avec comité de lecture

An Integrated Security Framework for XML based Management V. Cridlig, O. Festor, R. State, IM 2005 : 9th IFIP/IEEE International Symposium on Integrated Network Management, 15-19 May 2005, Nice-Acropolis, Exhibition Hall, Nice, France

RADIUS-Based SNMP Authorization V. Cridlig, O. Festor, R. State, IM Application Session 2005 : 9th IFIP/IEEE International Symposium on Integrated Network Management, 15-19 May 2005, Nice-Acropolis, Exhibition Hall, Nice, France.

Role based Access Control for XML based Management Gateways *Best student paper award DSOM 2004*, V. Cridlig, O. Festor, R. State, DSOM 2004 : IFIP/IEEE Distributed Systems Operations and Management 2004, Nov 15-17, 2004, Davis, CA, USA

Workshops internationaux avec comité de lecture

A NetConf Network Management Suite : ENSUITE V. Cridlig, H. Abdelnur, J. Bourdellon, R. State, IPOM 2005 : 5th IEEE International Workshop on IP Operations and Management, Barcelona, Spain, 26-28 October 2005, titre volume : Operations and Management in IP-Based Networks

Conférences nationales avec comité de lecture

Architecture de sécurité fondée sur Radius pour le plan de gestion de réseau V. Cridlig, O. Festor, R. State, SAR 2005 : 4ème Conférence sur la Sécurité et Architectures Réseaux, Batz sur Mer, France, 6-10 Juin 2005

Sécurité du plan de gestion. Besoins, modèles et outils V. Cridlig, O. Festor, R. State, SAR 2004 : Sécurité des architectures et des réseaux 2004 Papier invité à SAR'2004, La Londe, Cote d'Azur (France), Juin 21-25, 2004.

Revue de vulgarisation scientifique

EnSuite, une plateforme libre de configuration de réseau V. Cridlig, R. State, O. Festor, Teching : Techniques de l'ingénieur A paraître en 2006.

Rapports de recherche

Assessment of security extended XML-based Management V. Cridlig, H. Abdelnur, R. State, O. Festor, Rapport de recherche INRIA, Juillet 2006.

A

Résultats de la conversion des modèles de contrôle d'accès

A.1 Exemple de conversion de VACM vers RBAC

A.1.1 Politique VACM en entrée

La Figure A.1 illustre le résultat partiellement tronqué d'une opération `snmpwalk` sur la politique de contrôle d'accès VACM d'un agent SNMP. On voit en particulier de la ligne 11 à la ligne 17 les relations entre les groupes et les vues de la table `vacmAccessTable`. A partir de la ligne 27 se trouve la définition des vues. Pour chacune d'elle, la sortie affiche le masque (ligne 27 à 30), le type (31 à 34) qui peut être *included* ou *excluded*.

Le document présenté par cette figure est passé en paramètre d'entrée de l'algorithme de conversion vers le modèle RBAC.

A.1.2 Résultat de la translation vers RBAC de la politique VACM

La Figure A.2 illustre un résultat obtenu en sortie de l'algorithme de conversion. Ce document est volontairement tronqué pour des besoins de mise en page. Ce document textuel contient les rôles (ligne 2 à 8), les utilisateurs (ligne 9 à 11), les permissions (lignes 12 à 21) et leurs associations (lignes 22 à 44). La syntaxe de ce document est conforme au langage proposé dans le rapport introduisant Meta-CLI.

A.2 Exemple de conversion de CLI vers RBAC

A.2.1 Politique CLI en entrée

La Figure A.3 est un document textuel dérivant la politique de sécurité d'un équipement. Elle indique principalement quel est le niveau de privilège maximum d'un utilisateur et réaffecte éventuellement le niveau de sécurité requis par certaines commandes. Par exemple, la ligne 1 signifie que l'utilisateur *john* a le niveau de privilège maximum *9* et a le mot de passe (non chiffré *0*) *doe*. La ligne 8 indique que la commande *aaa* ne peut être exécutée que si le niveau de privilège courant est 15. *john* ne peut donc pas exécuter cette commande.

```

1 VACM-MIB::vacmContextName." " = STRING:
2 VACM-MIB::vacmGroupName.1."notConfigUser" = STRING: notConfigGroup
3 VACM-MIB::vacmGroupName.2."notConfigUser" = STRING: notConfigGroup
4 VACM-MIB::vacmSecurityToGroupStorageType.1."notConfigUser" = INTEGER: permanent(4)
5 VACM-MIB::vacmSecurityToGroupStorageType.2."notConfigUser" = INTEGER: permanent(4)
6 VACM-MIB::vacmSecurityToGroupStatus.1."notConfigUser" = INTEGER: active(1)
7 VACM-MIB::vacmSecurityToGroupStatus.2."notConfigUser" = INTEGER: active(1)
8 VACM-MIB::vacmAccessContextMatch."superAdmin"."".0.noAuthNoPriv = INTEGER: exact(1)
9 VACM-MIB::vacmAccessContextMatch."notConfigGroup"."".0.noAuthNoPriv = INTEGER: exact(1)
10 VACM-MIB::vacmAccessContextMatch."securityManager"."".0.noAuthNoPriv = INTEGER: exact(1)
11 VACM-MIB::vacmAccessReadViewName."superAdmin"."".0.noAuthNoPriv = STRING: allview
12 VACM-MIB::vacmAccessReadViewName."notConfigGroup"."".0.noAuthNoPriv = STRING: allview
13 VACM-MIB::vacmAccessReadViewName."securityManager"."".0.noAuthNoPriv = STRING: securityview
14 VACM-MIB::vacmAccessWriteViewName."superAdmin"."".0.noAuthNoPriv = STRING: none
15 VACM-MIB::vacmAccessWriteViewName."notConfigGroup"."".0.noAuthNoPriv = STRING: none
16 VACM-MIB::vacmAccessWriteViewName."securityManager"."".0.noAuthNoPriv = STRING: none
17 VACM-MIB::vacmAccessNotifyViewName."superAdmin"."".0.noAuthNoPriv = STRING: none
18 VACM-MIB::vacmAccessNotifyViewName."notConfigGroup"."".0.noAuthNoPriv = STRING: none
19 VACM-MIB::vacmAccessNotifyViewName."securityManager"."".0.noAuthNoPriv = STRING: none
20 VACM-MIB::vacmAccessStorageType."superAdmin"."".0.noAuthNoPriv = INTEGER: permanent(4)
21 VACM-MIB::vacmAccessStorageType."notConfigGroup"."".0.noAuthNoPriv = INTEGER: permanent(4)
22 VACM-MIB::vacmAccessStorageType."securityManager"."".0.noAuthNoPriv = INTEGER: permanent(4)
23 VACM-MIB::vacmAccessStatus."superAdmin"."".0.noAuthNoPriv = INTEGER: active(1)
24 VACM-MIB::vacmAccessStatus."notConfigGroup"."".0.noAuthNoPriv = INTEGER: active(1)
25 VACM-MIB::vacmAccessStatus."securityManager"."".0.noAuthNoPriv = INTEGER: active(1)
26 VACM-MIB::vacmViewSpinLock.0 = INTEGER: 0
27 VACM-MIB::vacmViewTreeFamilyMask."allview".3.1.3.6 = Hex-STRING: FF
28 VACM-MIB::vacmViewTreeFamilyMask."mib2view".6.1.3.6.1.2.1 = Hex-STRING: FF
29 VACM-MIB::vacmViewTreeFamilyMask."systemview".9.1.3.6.1.2.1.25.1.1 = Hex-STRING: FF FF
30 VACM-MIB::vacmViewTreeFamilyMask."securityview".6.1.3.6.1.6.3 = Hex-STRING: FF
31 VACM-MIB::vacmViewTreeFamilyType."allview".3.1.3.6 = INTEGER: included(1)
32 VACM-MIB::vacmViewTreeFamilyType."mib2view".6.1.3.6.1.2.1 = INTEGER: included(1)
33 VACM-MIB::vacmViewTreeFamilyType."systemview".9.1.3.6.1.2.1.25.1.1 = INTEGER: included(1)
34 VACM-MIB::vacmViewTreeFamilyType."securityview".6.1.3.6.1.6.3 = INTEGER: included(1)
35 VACM-MIB::vacmViewTreeFamilyStorageType."allview".3.1.3.6 = INTEGER: permanent(4)
36 VACM-MIB::vacmViewTreeFamilyStorageType."mib2view".6.1.3.6.1.2.1 = INTEGER: permanent(4)
37 VACM-MIB::vacmViewTreeFamilyStorageType."securityview".6.1.3.6.1.6.3 = INTEGER: permanent(4)
38 VACM-MIB::vacmViewTreeFamilyStatus."allview".3.1.3.6 = INTEGER: active(1)
39 VACM-MIB::vacmViewTreeFamilyStatus."mib2view".6.1.3.6.1.2.1 = INTEGER: active(1)
40 VACM-MIB::vacmViewTreeFamilyStatus."systemview".9.1.3.6.1.2.1.25.1.1 = INTEGER: active(1)
41 VACM-MIB::vacmViewTreeFamilyStatus."securityview".6.1.3.6.1.6.3 = INTEGER: active(1)

```

FIG. A.1 – Configuration de sécurité originale de SNMP VACM

```
1 - rbac:
2   -roles:
3     -role: '1'
4       -name: 'notConfigGroup'
5     -role: '2'
6       -name: 'superAdmin'
7     -role: '3'
8       -name: 'securityManager'
9   -users:
10    -user: '1'
11      -name: 'notConfigUser'
12  -permissions:
13    -permission: '1'
14      -scope: '/iso/org/dod'
15      -operation: 'r'
16    -permission: '2'
17      -scope: '/iso/org/dod/internet/snmpV2/snmpModules'
18      -operation: 'r'
19    -permission: '3'
20      -scope: 'none'
21      -operation: 'w'
22  -user-role-assignments:
23    -user-role-assignment: '0'
24      -user: '1'
25      -role: '1'
26  -permission-role-assignments:
27    -permission-role-assignment: '0'
28      -permission: '1'
29      -role: '1'
30    -permission-role-assignment: '1'
31      -permission: '3'
32      -role: '1'
33    -permission-role-assignment: '2'
34      -permission: '1'
35      -role: '2'
36    -permission-role-assignment: '3'
37      -permission: '3'
38      -role: '2'
39    -permission-role-assignment: '4'
40      -permission: '2'
41      -role: '3'
42    -permission-role-assignment: '5'
43      -permission: '3'
44      -role: '3'
```

FIG. A.2 – Configuration de sécurité SNMP VACM convertie

```
1 username john privilege 9 password 0 doe
2 username six privilege 6 password 0 six
3 username poweruser privilege 15 password poweruser
4 username inout password inout
5 privilege configure level 8 snmp-server community
6 privilege configure level 3 command ping
7 privilege configure level 5 mode enable command configure
8 privilege configure level 15 command aaa
9 privilege configure level 15 command aaa-server
10 privilege configure level 15 command access-group
11 privilege configure level 15 command activation-key
```

FIG. A.3 – Configuration de sécurité originale de CLI

A.2.2 Résultat de la translation vers RBAC de la politique CLI

La Figure A.4 présente la politique de sécurité obtenue après application de l'algorithme de conversion. Le document respecte la même syntaxe et la même sémantique que celui illustré dans la section précédente avec VACM. L'expression des permissions se fait à l'aide de la représentation compacte introduite dans [29]. Un exemple d'une telle expression est : */configure/command/aaa*.

A.3 Modèle RBAC utilisé dans EnSuite

A.3.1 Politique RBAC en entrée

Le document de la Figure A.5 est un échantillon de politique de contrôle d'accès telle que nous l'avons définie pour Netconf. Elle s'exprime en langage XML et elle est construite nativement sur le modèle RBAC ce qui rend la conversion ultérieure évidente.

A.3.2 Résultat de la translation vers RBAC de la politique RBAC de EnSuite

La Figure A.6 contient le document obtenu après conversion de la politique RBAC exprimée en langage XML. Seule la syntaxe change par rapport au document original de la Figure A.5.

```

1 - rbac:
2   -roles:
3     -role: '1'
4       -name: 'role9'
5     -role: '2'
6       -name: 'role6'
7     -role: '3'
8       -name: 'role15'
9     -role: '4'
10      -name: 'role0'
11  -users:
12    -user: '1'
13      -name: 'john'
14    -user: '2'
15      -name: 'six'
16    -user: '3'
17      -name: 'poweruser'
18  -permissions:
19    -permission: '1'
20      -scope: '/configure/snmp-server/community'
21      -operation: 'w'
22    -permission: '2'
23      -scope: '/configure/command/ping'
24      -operation: 'w'
25    -permission: '3'
26      -scope: '/configure/mode/enable/command/configure'
27      -operation: 'w'
28    -permission: '4'
29      -scope: '/configure/command/aaa'
30      -operation: 'w'
31    -permission: '5'
32      -scope: '/configure/command/aaa-server'
33      -operation: 'w'
34  -user-role-assignments:
35    -user-role-assignement: '1'
36      -user: '1'
37      -role: '1'
38    -user-role-assignement: '2'
39      -user: '2'
40      -role: '2'
41    -user-role-assignement: '3'
42      -user: '3'
43      -role: '3'
44  -permission-role-assignments:
45    -permission-role-assignement: '1'
46      -permission: '4'
47      -role: '3'
48    -permission-role-assignement: '2'
49      -permission: '5'
50      -role: '3'
51    -permission-role-assignement: '3'
52      -permission: '6'
53      -role: '3'

```

FIG. A.4 – Configuration de sécurité CLI convertie

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <rbac xmlns="urn:loria:madyne:ensuite:yencap:module:RBAC:1.0"
3     xmlns:ycp="urn:loria:madyne:ensuite:yencap:1.0"
4     xmlns:ifs="urn:loria:madyne:ensuite:yencap:module:Interfaces:1.0"
5     xmlns:bgp="urn:loria:madyne:ensuite:yencap:module:BGP:1.0"
6     xmlns:ac="urn:loria:madyne:ensuite:yencap:module:RBAC:1.0">
7   <users>
8     <user id="9">
9       <login>neuf</login>
10      <password>telecom</password>
11    </user>
12    <!-- ... -->
13  </users>
14  <roles>
15    <role id="1">
16      <name>sysAdmin</name>
17      <junior-roles>
18        <junior-role roleRef="2"/>
19        <junior-role roleRef="3"/>
20      </junior-roles>
21    </role>
22    <role id="2">
23      <name>netAdmin</name>
24      <junior-roles>
25        <junior-role roleRef="4"/>
26      </junior-roles>
27    </role>
28    <!-- ... -->
29  </roles>
30  <permissions>
31    <permission id="1" op="rw">
32      <scope>/ycp:netconf/ycp:security/ac:rbac/ac:permissions/ac:permission[@id='5']</scope>
33    </permission>
34    <permission id="2" op="rw">
35      <scope>/ycp:netconf/ycp:network/ifs:interfaces</scope>
36    </permission>
37    <permission id="3" op="rw">
38      <scope>/ycp:netconf/ycp:routing/bgp:bgp</scope>
39    </permission>
40    <!-- ... -->
41  </permissions>
42  <user-assignments>
43    <user-assignment roleRef="1" userRef="13" id="1"/>
44    <user-assignment roleRef="2" userRef="14" id="2"/>
45    <user-assignment roleRef="6" userRef="9" id="5"/>
46    <!-- ... -->
47  </user-assignments>
48  <permission-assignments>
49    <permission-assignment roleRef="1" permRef="1" id="1"/>
50    <permission-assignment roleRef="2" permRef="2" id="2"/>
51    <permission-assignment roleRef="2" permRef="3" id="3"/>
52    <!-- ... -->
53  </permission-assignments>
54 </rbac>

```

FIG. A.5 – Configuration de sécurité originale de Netconf

```

1 - rbac:
2   - roles:
3     - role: '1'
4       - name: 'sysAdmin'
5       - junior-roles:
6         - junior-role: '2'
7         - junior-role: '3'
8     - role: '2'
9       - name: 'netAdmin'
10      - junior-roles:
11        - junior-role: '4'
12   - users:
13     - user: '9'
14       - name: 'neuf'
15     - user: '13'
16       - name: 'cridligv'
17     - user: '14'
18       - name: 'alice'
19   - permissions:
20     - permission: '1'
21       - scope: '/ycp:netconf/ycp:security/ac:rbac/ac:permissions/ac:permission[@id='5']'
22       - operation: 'r w'
23     - permission: '2'
24       - scope: '/ycp:netconf/ycp:network/ifs:interfaces'
25       - operation: 'r w'
26     - permission: '3'
27       - scope: '/ycp:netconf/ycp:routing/bgp:bgp'
28       - operation: 'r w'
29     - permission: '4'
30       - scope: '/ycp:netconf/s:system'
31       - operation: 'r w'
32   - user-role-assignments:
33     - user-role-assignment: '1'
34       - user: '13'
35       - role: '1'
36     - user-role-assignment: '2'
37       - user: '14'
38       - role: '2'
39   - permission-role-assignments:
40     - permission-role-assignment: '1'
41       - user: '1'
42       - role: '1'
43     - permission-role-assignment: '2'
44       - user: '2'
45       - role: '2'
46     - permission-role-assignment: '3'
47       - user: '3'
48       - role: '2'
49     - permission-role-assignment: '4'
50       - user: '4'
51       - role: '3'
52     - permission-role-assignment: '5'
53       - user: '5'
54       - role: '4'

```

FIG. A.6 – Configuration de sécurité de Netconf (fonctionnalité RBAC) convertie

B

Les modèles et technologies XML émergents

B.1 Sécurité des web services

B.1.1 XML-Encryption

XML-Encryption [44] est la spécification d'un langage XML qui permet de créer des documents XML cryptés. Cette spécification est un gage d'interopérabilité car elle assure que les applications respectant ce langage pourront chiffrer et déchiffrer les données. La structure contenant les données XML est auto-descriptive puisqu'elle contient la méthode de chiffrement, les informations nécessaires pour retrouver la clé et enfin, les données chiffrées. La Figure B.1 est un exemple de nœud XML respectant cette spécification proposée par le W3C⁸. L'élément principal est *EncryptedData* et contient trois éléments enfants : *EncryptionMethod* qui est la méthode de chiffrement (ici aes128-cbc), *KeyInfo* qui contient des informations sur la clé utilisée et *CipherData* qui est la donnée cryptée.

```

1 <EncryptedData Id='ED' xmlns='http://www.w3.org/2001/04/xmlenc#'>
2   <EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc' />
3   <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
4     <ds:KeyName>Sally Doe</ds:KeyName>
5   </ds:KeyInfo>
6   <CipherData><CipherValue>DEADBEEF</CipherValue></CipherData>
7 </EncryptedData>

```

FIG. B.1 – Exemple d'élément XML chiffré respectant la syntaxe XML-Encryption

Cette spécification est particulièrement adaptée au contrôle d'accès distribué puisqu'il est possible de crypter différentes parties d'un même document. Un cas d'utilisation typique consiste à chiffrer différentes parties d'un document à l'aide de différentes clés. Ainsi, le même document peut être envoyé à différentes entités mais seules les entités possédant les clés peuvent accéder à l'information. Chaque entité a une vue différente du document car elle peut décrypter une partie du document chiffré en fonction de son niveau de privilège.

⁸World Wide Web Consortium, <http://www.w3.org>

B.1.2 XML-DigitalSignature

Le W3C a proposé en 2002 un langage XML pour la signature numérique [4]. La spécification satisfait aux services d'intégrité, d'authentification du message et d'authentification du signataire. Elle repose sur les outils habituels de signature numérique : infrastructure à clés publiques, certificats, algorithme de hachage.

La signature de documents XML utilise le mécanisme de canonisation [46] pour éviter les problèmes inhérents à un document XML. Il est possible que deux documents XML soient identiques par leur contenu excepté les espaces, les tabulations ou les déclarations d'espaces de noms et préfixes. En effet, il y a plusieurs façon de déclarer un espace de noms. Ces différences légères et leur gestion par les librairies XML sont problématiques lorsqu'on souhaite calculer un haché du document, utile pour signer électroniquement le document. Le problème survient très souvent pour des documents transmis via le réseau et manipulés (sérialisation et désérialisation) par différentes librairies. La moindre différence syntaxique dans le document original à signer modifie complètement le haché. La canonisation consiste principalement à normaliser les espaces et les déclarations d'espaces de noms. Ainsi, deux documents XML égaux par leurs valeurs deviennent également égaux dans leur forme, ce qui est très important pour le calcul du haché.

La Figure B.2 illustre un exemple partiel de signature électronique détachée. Cet exemple indique que la méthode de canonisation qui a été utilisée est *c14n*, la méthode de signature est *dsa-sha1* et la méthode de hachage est *sha1*. Utilisant ces méthodes, un haché contenu dans l'élément *DigestValue* a été calculé et la signature est stockée dans l'élément *SignatureValue*. A la fin de la signature, on trouve des informations qui identifient la clé utilisée.

```

1 <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
2   <SignedInfo>
3     <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
4     <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
5     <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
6       <Transforms>
7         <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
8       </Transforms>
9       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
10      <DigestValue>j6lwx3rvEP00vKtMup4NbeVu8nk=</DigestValue>
11    </Reference>
12  </SignedInfo>
13  <SignatureValue>MCOCFFrVLtRlk=...</SignatureValue>
14  <KeyInfo>
15    <KeyValue>
16      <DSAKeyValue>
17        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
18      </DSAKeyValue>
19    </KeyValue>
20  </KeyInfo>
21 </Signature>

```

FIG. B.2 – Exemple de signature XML-DSIG

Etant donné l'intérêt actuel de la communauté de gestion de réseau pour les technologies XML (cf. web services, passerelle XML/SNMP, Netconf), il est intéressant de considérer les solutions de sécurité XML qui apportent plus de flexibilité que, par exemple, un tunnel SSH qui va authentifier et crypter l'ensemble des données. On peut vouloir signer seulement certaines

parties de documents XML, ce qui est tout à fait envisageable avec l'approche [4]. Enfin, il est possible de combiner cryptage et signature électronique avec ces deux technologies ([44] et [4]).

B.2 SAML et XACML

B.2.1 Autorisation distribuée et Single-Sign On : SAML

Security Assertion Markup Language (SAML [57]) est une initiative d'OASIS⁹ pour fédérer les identités des internautes, et, entre autres, permettre qu'un utilisateur n'ait besoin de s'authentifier qu'une seule fois et accéder cependant à plusieurs sites web. C'est ce qu'on appelle le Single Sign On (SSO). Les sites commerciaux qui supportent SAML ont besoin de communiquer entre eux pour se transmettre des assertions de sécurité relatives à un utilisateur. Ces communications permettent de s'assurer qu'un visiteur a été authentifié même si c'était sur un autre système, de découvrir son identité, ses caractéristiques (attributs dans le langage SAML) et ses privilèges. SAML définit des assertions, des protocoles, des bindings et des profils.

Une assertion SAML peut contenir trois types de rapport (statement) :

- Un rapport d'authentification : ce type de rapport est délivré par une autorité qui a authentifié un utilisateur. Elle précise diverses informations comme l'entité qui a délivré le rapport, l'utilisateur authentifié ou encore la durée de validité du rapport. La figure B.3 illustre une assertion comportant un rapport d'authentification,
- Un rapport d'attribut : ce deuxième type de rapport contient des informations relatives à un utilisateur, et qui peuvent servir à un service donné,
- Un rapport d'autorisation : ce dernier rapport contient des privilèges de l'utilisateur.

```

1 <saml:Assertion xmlns:saml=?urn:oasis:names:tc:SAML:2.0:assertion? Version="2.0"
2   IssueInstant="2005-01-31T12:00:00Z">
3   <saml:Issuer>www.acompany.com</saml:Issuer>
4   <saml:Subject>
5     <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
6       j.doe@acompany.com
7     </saml:NameID>
8   </saml:Subject>
9   <saml:Conditions NotBefore="2005-01-31T12:00:00Z" NotOnOrAfter="2005-01-31T12:00:00Z">
10    </saml:Conditions>
11   <saml:AuthnStatement AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
12     <saml:AuthnContext>
13       <saml:AuthnContextClassRef>
14         urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport
15       </saml:AuthnContextClassRef>
16     </saml:AuthnContext>
17   </saml:AuthnStatement>
18 </saml:Assertion>

```

FIG. B.3 – Exemple d'assertion SAML (Source : [57])

Les protocoles SAML sont de plusieurs types : protocole de requête et réponse d'assertion, demande d'authentification, résolution d'artifact, fédération des utilisateurs par utilisation d'alias. Les bindings correspondent aux différentes options dans le choix des couches protocolaires sur lesquelles SAML peut reposer. Par exemple, la façon d'intégrer les messages SAML dans SOAP

⁹Organization for the Advancement of Structured Information Standards, <http://www.oasis-open.org/>

correspond à un binding. Il en existe d'autres comme les bindings HTTP Redirect, HTTP POST ou HTTP Artifact. Enfin, les combinaisons possibles entre des assertions, des protocoles et des bindings SAML sont appelées des profils SAML. Ils permettent de répondre de façon très adaptée à des cas d'utilisation précis comme le profil SSO via un navigateur web qui se base sur le protocole de requête d'authentification cité précédemment et différents bindings HTTP.

L'intérêt de SAML dans le cadre de la gestion de réseau est multiple. Comme de nombreux équipements réseaux utilisent maintenant des interfaces web, il pourrait être intéressant pour un utilisateur de logger sur un équipement une fois pour toutes. Pour cela, les équipements pourraient mettre en place une architecture SAML qui leur permettrait d'échanger des informations sur un utilisateur qui interagirait avec plusieurs équipements. Ainsi, de façon totalement transparente, l'utilisateur pourrait passer d'un équipement à l'autre sans avoir à se logger systématiquement, d'où un gain de temps important et une meilleure gestion des identités. Par extension, cette approche est d'autant plus envisageable dans le cas du binding du protocole Netconf sur SOAP [36]. Les équipements sont déjà capables de traiter des requêtes SOAP donc il n'y aurait pas de coût supplémentaire du point de vue du protocole sous-jacent.

B.2.2 Contrôle d'accès par politiques : XACML

OASIS eXtensible Access Control Markup Language (XACML [61]) est un langage XML permettant d'exprimer des politiques de contrôle d'accès. Le principal objectif d'XACML est d'un part de centraliser la gestion du contrôle d'accès de façon à faciliter sa maintenance et d'autre part de standardiser le langage d'expression de politique et les formats de requêtes d'accès de façon à rendreinteropéables les services qui utilisaient précédemment des langages propriétaires. XACML, qui est à la fois un protocole et un langage, suit l'approche basée sur les politiques en définissant les entités suivantes :

- Policy Enforcement Point (PEP) : un PEP est une entité cliente des décisions de contrôle d'accès. Elle a besoin d'interroger une entité PDP pour savoir si un utilisateur du service fourni par PEP a les privilèges suffisants pour réaliser telle ou telle opération,
- Policy Decision Point (PDP) : un PDP est une entité capable de répondre à des requêtes d'accès venant d'un PEP. Pour cela, il utilise une politique de contrôle d'accès dans un ou plusieurs fichiers XML en utilisant le langage XACML,
- Policy Information Point (PIP) : un PIP est une entité qui stocke des informations sur les utilisateurs, les ressources

Une politique XACML est composée de quatre éléments principaux :

- Target : Cet élément permet de déterminer si la politique s'applique ou non pour la requête à évaluer, en fonction du sujet émetteur de la requête, de la ressource et de l'action. Si la politique est applicable, les règles sont évaluées,
- Rules : Une règle est elle-même composée de trois éléments : les *conditions* à vérifier pour que la règle soit appliquée (un peu à la manière d'une règle de firewall), l'*effet* qui peut être soit *Permit* soit *Deny*, et à nouveau un élément Target qui permet de savoir si la règle est applicable ou non. Si la condition est évaluée à faux, la règle n'est pas applicable, sinon le résultat est le contenu de *Effect*.
- Rule-combining algorithm : comme plusieurs règles peuvent s'appliquer et générer des résultats différents, il est possible de créer des situations où les décisions sont contradictoires. Pour résoudre ces conflits, des algorithmes de priorité (*Deny* ou *Permit* est prioritaire) et d'ordre (la première règle applicable est prioritaire) sont mis en œuvre.
- Obligations : les obligations sont des actions à exécuter par le PEP, lorsque le PDP renvoie la réponse d'autorisation. En plus de la décision, le message contient une liste d'actions à

effectuer par le PEP.

Le document [3] décrit comment représenter une politique RBAC à l'aide du langage XACML. Il fournit des directives à suivre pour représenter les composants RBAC que sont les rôles et les différentes associations entre utilisateurs et rôles ou entre permissions et rôles. La Figure B.4 montre comment définir les permissions pour le rôle *employee*. Tout n'est pas spécifié dans cet exemple, en particulier le lien entre le rôle *employee* et ces permissions et le lien entre un utilisateur et ce rôle.

```

1 <PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
2   PolicySetId="PPS:employee:role" PolicyCombiningAlgId="&policy-combine;permit-overrides">
3   <!-- Permissions specifically for the employee role -->
4   <Policy PolicyId="Permissions:specifically:for:the:employee:role"
5     RuleCombiningAlgId="&rule-combine;permit-overrides">
6     <!-- Permission to create a purchase order -->
7     <Rule RuleId="Permission:to:create:a:purchase:order" Effect="Permit">
8       <Target>
9         <Resources>
10          <Resource>
11            <ResourceMatch MatchId="&function;string-equal">
12              <AttributeValue DataType="&xml:string">purchase order</AttributeValue>
13              <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml:string"/>
14            </ResourceMatch>
15          </Resource>
16        </Resources>
17        <Actions>
18          <Action>
19            <ActionMatch MatchId="&function;string-equal">
20              <AttributeValue DataType="&xml:string">create</AttributeValue>
21              <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml:string"/>
22            </ActionMatch>
23          </Action>
24        </Actions>
25      </Target>
26    </Rule>
27  </Policy>
28 </PolicySet>

```

FIG. B.4 – Exemple de politique XACML (Source : [3])

L'inconvénient majeur de la version actuelle de ce binding est l'absence de la notion d'activation de rôles pourtant très importante dans le modèle RBAC. Ici, on considère que tous les rôles affectés (*enabled*) à un utilisateur sont actifs.

B.3 Contrôle d'accès sur le contenu XML

L'intérêt pour le contrôle d'accès sur les documents XML [51, 43, 15] a suivi la progression de ce langage et notamment dans le cadre des web services [6]. Il existe deux types de contrôle d'accès complémentaires :

- basé sur les types : chaque type d'élément sera concerné par la règle d'accès. Par exemple, toutes les routes d'un routeur. La règle s'applique en fait sur le modèle de données. Une telle approche a été proposée dans [84]. Les règles d'accès sont définies au niveau du schéma XML et ne sont donc pas relatives aux valeurs des paramètres du modèle de donné instancié,

- basé sur les instances : un ensemble d'instances est concerné par la règle d'accès. Par exemple, certaines routes d'un routeur. La règle s'applique sur l'instanciation du modèle de données.

Bibliographie

- [1] C. Adams and S. Lloyd. *Understanding PKI : Concepts, Standards, and Deployment Considerations, Second Edition*. Addison-Wesley Pub Co, November 2002.
- [2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23, October 2005.
- [3] A. Anderson. XACML Profile for Role Based Access Control (RBAC). OASIS Standard, February 2005.
- [4] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing. W3C Recommendation, <http://www.w3.org/TR/xmlsig-core>, February 2002.
- [5] E. Bertino and P. A. Bonatti. TRBAC : A Temporal Role-Based Access Control Model. *ACM Transactions on Information and System Security*, 4(3) :191–223, August 2001.
- [6] R. Bhatti, E. Bertino, A. Ghafoor, and J. B.D. Joshi. Xml-based specification for web services document security. *Computer, Putting the Web to Work*, 37(4) :41–49, April 2004.
- [7] R. Bhatti, A. Ghafoor, E. Bertino, and J. B. D. Joshi. X-gtrbac : an xml-based policy specification framework and architecture for enterprise-wide access control. *ACM Trans. Inf. Syst. Secur.*, 8(2) :187–227, 2005.
- [8] R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. B. D. Joshi. X-gtrbac admin : A decentralized administration model for enterprise-wide access control. *ACM Trans. Inf. Syst. Secur.*, 8(4) :388–423, 2005.
- [9] M. Bishop. *Computer Security : art and science*. Addison Wesley, 2003.
- [10] U. Blumenthal and B. Wijnen. User-based Security Model for version 3 of the Simple Network Management Protocol (SNMPv3). STD 62, <http://www.ietf.org/rfc/rfc3414.txt>, December 2002.
- [11] T. Bray, D. Hollander, and A. Layman. Namespaces in XML. W3C Recommendation, <http://www.w3.org/TR/REC-xml-names>, January 1999.
- [12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, <http://www.w3.org/TR/REC-xml>, February 2004.
- [13] B. Briscoe. MARKS : Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In *Proceedings of the First International COST264 Workshop on Networked Group Communication*, pages 301–320. Springer-Verlag, 1999.
- [14] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). STD 62, <http://www.ietf.org/rfc/rfc3412.txt>, December 2002.

- [15] R. Chandramouli. Application of xml tools for enterprise-wide rbac implementation tasks. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 11–18. ACM Press, 2000.
- [16] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, <http://www.w3.org/TR/xpath>, November 1999.
- [17] A. Corrente and L. Tura. Security performance analysis of snmpv3 with respect to snmpv2c. In Raouf Boutaba and Seong-Beom Kim, editors, *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, pages 729–742, Seoul, Korea, April 2004.
- [18] V. Cridlig, H. Abdelnur, Jérôme Bourdellon, and R. State. A NetConf Network Management Suite : ENSUITE. In *Proceedings of the 5th IEEE International Workshop on IP Operations and Management, IPOM 2005*, volume 3751 of *Lecture Notes in Computer Science*, pages 152–161. Springer, October 2005.
- [19] V. Cridlig, H. Abdelnur, R. State, and O. Festor. XBGP-MAN : A XML management architecture for BGP. *International Journal of Network Management*, 2006.
- [20] V. Cridlig, R. State, and O. Festor. Role based access control for XML based management gateway. In *Proceedings of the 15th IFIP/IEEE Distributed Systems : Operations and Management, DSOM 2004*, December 2004.
- [21] V. Cridlig, R. State, and O. Festor. An Integrated Security Framework for XML based Management. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management, IM 2005*, IFIP Conference Proceedings, May 2005.
- [22] V. Cridlig, R. State, and O. Festor. Ensuite, une plateforme libre de configuration de réseau. *Techniques de l'ingénieur*, 2006.
- [23] V. Cridlig, R. State, and O. Festor. Extended Netconf Suite : EnSuite. <http://libre-source.inria.fr/projects/ensuite>, May 2006.
- [24] V. Cridlig, R. State, and O. Festor. Role-Based Access Control for XML Enabled Multi-Protocol Management Gateways. *eTransactions on Network and Service Management*, 3(1), April 2006.
- [25] V. Cridlig, R. State, O. Festor, and J.-F. Leroy. RADIUS-Based SNMP Authorization. In *Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management (IM Application Session 2005)*, May 2005.
- [26] F. Cuppens, Nora Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the or-bac model and application in a network environment. In *Foundations of Computer Security (FCS'04)*, July 2004.
- [27] F. Cuppens and A. Miège. Modelling contexts in the or-bac model. In *19th Applied Computer Security Associates Conference (ACSAC 2003)*, December 2003.
- [28] F. Cuppens and A. Miège. Administration model for or-bac. *International Journal of Computer Systems Science and Engineering (CSSE)*, 19(4), May 2004.
- [29] R. Deca. Meta-cli configuration model for network device management. Master's thesis, Concordia University, January 2003.
- [30] T. Dierks and C. Allen. The TLS Protocol Version 1.0. STD 62, <http://www.ietf.org/rfc/rfc2246.txt>, January 1999.
- [31] D. Eastlake and P. Jones. Us secure hash algorithm 1 (sha1). STD 62, <http://www.ietf.org/rfc/rfc3174.txt>, September 2001.

-
- [32] R. Enns. NETCONF Configuration Protocol. Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt>, February 2006.
- [33] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3) :224–274, 2001.
- [34] C. Finseth. An Access Control Protocol, Sometimes Called TACACS. <http://www.ietf.org/rfc/rfc1492.txt>, July 1993.
- [35] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol Version 3.0. Technical report, Netscape, 1996.
- [36] T. Goddard. Using the Network Configuration Protocol (NETCONF) Over the Simple Object Access Protocol (SOAP). Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-netconf-soap-08.txt>, March 2006.
- [37] DSLHome-Technical Working Group. CPE WAN Management Protocol, May 2004.
- [38] H. Hamed and E. Al-Shaer. Taxonomy of Conflicts in Network Security Policies. *IEEE Communications Magazine*, February 2006.
- [39] W. Hardaker and D. Perkins. A Session-Based Security Model (SBSM) for version 3 of the Simple Network Management Protocol (SNMPv3). Internet Draft, February 2004.
- [40] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing SNMP Management Frameworks. STD 62, <http://www.ietf.org/rfc/rfc2571.txt>, April 1999.
- [41] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. STD 62, <http://www.ietf.org/rfc/rfc3411.txt>, December 2002.
- [42] C. Hedrick. Routing Information Protocol. <http://www.ietf.org/rfc/rfc1058.txt>, June 1988.
- [43] M. Hitchens and V. Varadharajan. Rbac for xml document stores. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *Proceedings of the 3rd IFIP/IEEE International Conference on Information and Communications Security, ICICS 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 131–143. Springer, November 2001.
- [44] T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing. W3C Recommendation, <http://www.w3.org/TR/xmlenc-core>, December 2002.
- [45] A. K. Jain and R. C. Dubes. *Algorithm form Clustering Data*. Prentice Hall, 1988.
- [46] J. Boyer, D. E. Eastlake, and J. Reagle. Exclusive XML Canonicalization Version 1.0. W3C Recommendation, <http://www.w3.org/TR/xml-exc-c14n>, July 2002.
- [47] A. A. El Kalam, S. Benferhat, A. Miège, R. El Baida, F. Cuppens, and C. Saurel. Orbac : un modèle de contrôle d'accès basé sur les organisations. *Cahiers francophones de la recherche en sécurité de l'information*, (2) :30–43, 2003.
- [48] A. Kern. Advanced features for enterprise-wide role-based access control. In *ACSAC '02 : Proceedings of the 18th Annual Computer Security Applications Conference*, page 333, Washington, DC, USA, 2002. IEEE Computer Society.
- [49] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *SACMAT '03 : Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 3–11, New York, NY, USA, 2003. ACM Press.
- [50] H. Krawczyk, M. Bellare, and R. Canetti. HMAC : Keyed-Hashing for Message Authentication. STD 62, <http://www.ietf.org/rfc/rfc2104.txt>, February 1997.

- [51] M. Kudo and S. Hada. Xml document security based on provisional authorization. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 87–96. ACM Press, 2000.
- [52] R. Kuhn. Role Based Access Control. NIST Standard Draft, April 2003.
- [53] U. Latif. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1) :4–23, 2005. James B. D. Joshi and Fellow-Elisa Bertino and Fellow-Arif Ghafoor.
- [54] E. Lear and K. Crozier. Using the NETCONF Protocol over Blocks Extensible Exchange Protocol (BEEP). Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-netconf-beep-10.txt>, March 2006.
- [55] H. Lee and B. Noh. Design and Analysis of Role-Based Security Model in SNMPv3 for Policy-Based Security Management. In *Proceedings of the International Conference on Wireless Communications Technologies and Network Applications, ICOIN 2002*, volume 2344 of *Lecture Notes in Computer Science*, pages 430–441. Springer, January 2002.
- [56] D. Levi, P. Meyer, and B. Stewart. Simple network management protocol (snmp) applications. STD 62, <http://www.ietf.org/rfc/rfc3413.txt>, December 2002.
- [57] H. Lockhart, T. Wisniewski, P. Mishra, and N. Ragouzis. Security Assertion Markup Language (SAML) V2.0 Technical Overview. OASIS Working Draft, September 2005.
- [58] E. Lupu, Z. Milosevic, and M. Sloman. Use of Roles and Policies for Specifying, and Managing a Virtual Enterprise. In *Ninth IEEE International Workshop on Research Issues on Data Engineering : Information Technology for Virtual Enterprises (RIDE-VE'99)*, March 1999.
- [59] E. Lupu and M. Sloman. Towards a Role Based Framework for Distributed Systems Management. *Journal of Network and Systems Management*, 5(1) :5–30, 1997.
- [60] Microsoft. Ws-authorization. <http://msdn.microsoft.com/ws-security/>.
- [61] T. Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, February 2005.
- [62] R. Neisse, R. L. Vianna, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco. Implementation and bandwidth consumption evaluation of snmp to web services gateways. In Raouf Boutaba and Seong-Beom Kim, editors, *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, pages 715–728, Seoul, Korea, April 2004.
- [63] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 65–79. ACM Press, 2003.
- [64] Y.-J. Oh, H.-T. Ju, M.-J. Choi, and J. W.-K. Hong. Interaction Translation Methods for XML/SNMP Gateway. In Metin Feridun, Peter G. Kropf, and Gilbert Babin, editors, *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management, DSOM 2002*, volume 2506 of *Lecture Notes in Computer Science*, pages 54–65. Springer, October 2002.
- [65] A. Pras, J. Schönwälder, and O. Festor. XML-Based Management of Networks and Services. *IEEE Communications Magazine*, 42(7) :56–57, July 2004.
- [66] R. Presuhn, J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Management information base (mib) for the simple network management protocol (snmp). STD 62, <http://www.ietf.org/rfc/rfc3418.txt>, December 2002.

-
- [67] R. Presuhn, J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport mappings for the simple network management protocol (snmp). STD 62, <http://www.ietf.org/rfc/rfc3417.txt>, December 2002.
- [68] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). <http://www.ietf.org/rfc/rfc1771.txt>, March 1995.
- [69] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). <http://www.ietf.org/rfc/rfc2865.txt>, June 2000.
- [70] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 1996.
- [71] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Third Edition*. Addison Wesley, 1999.
- [72] R. State, O. Festor, and I. Chrisment. Context Driven Access Control to SNMP MIB objects in multi-homed environments. In Marcus Brunner and Alexander Keller, editors, *Proceedings of the 14th IFIP/IEEE International Workshop on Distributed Systems : Operations and Management, DSOM 2003*, volume 2867 of *Lecture Notes in Computer Science*, pages 169–180. Springer, October 2003.
- [73] F. Strauß and T. Klie. Towards XML Oriented Internet Management. In Germán S. Goldszmidt and Jürgen Schönwälder, editors, *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, volume 246 of *IFIP Conference Proceedings*, pages 505–518. Kluwer, March 2003.
- [74] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1 : Structures. W3C Recommendation, <http://www.w3.org/TR/xmlschema-1>, May 2001.
- [75] Vijay Varadharajan. Distributed authorization : Principles and practice.
- [76] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey Framework : Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications*, 17(9) :1614–1631, September 1999.
- [77] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast : Issues and Architectures. RFC 2627 (informational), June 1999.
- [78] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure Shell (SSH). Internet Draft, <http://www.ietf.org/internet-drafts/draft-ietf-netconf-ssh-06.txt>, March 2006.
- [79] B. Wijnen and U. Blumenthal. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). STD 62, <http://www.ietf.org/rfc/rfc3415.txt>, December 2002.
- [80] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Trans. Netw.*, 8(1) :16–30, 2000.
- [81] DOM working group. Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation, <http://www.w3.org/TR/DOM-Level-3-Core>, April 2004.
- [82] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. STD 62, <http://www.ietf.org/rfc/rfc4252.txt>, January 2006.
- [83] J.-H. Yoon, H.-T. Ju, and J. W. Hong. Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management. *International Journal of Network Management*, 13(4) :259–276, 2003.

- [84] X. Zhang, J. Park, and R. Sandhu. Schema based xml security : Rbac approach. In S. di Vimercati and Indrakshi Ray, editors, *Proceedings of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, Estes Park, Colorado, USA, August 2003. Kluwer.

Résumé

Au cours des dernières années, l'évolution rapide des réseaux a provoqué une explosion de la diversité et du nombre des équipements gérés et, par conséquent, du nombre de plateformes de supervision de réseaux. Chacune de ces plateformes induit un environnement de gestion à la fois clos de part la spécificité de son architecture et de son protocole de communication mais également non isolé de part la nature des ressources gérées. En effet, bien qu'hétérogènes, les modèles de données de ces plateformes se recouvrent au moins partiellement.

Il s'ensuit un fort problème de sécurité puisque chacune de ces plateformes de supervision met en œuvre sa propre architecture de sécurité avec ses paramètres associés. Le problème apparaît également au sein d'une même plateforme qui autoriserait plusieurs modèles de sécurité ou plusieurs protocoles sous-jacents. Jusqu'à maintenant, il n'y a pas eu de véritable réflexion sur les conséquences de cet environnement et les solutions envisageables.

Dans cette thèse, nous proposons des architectures inter-plateformes de distribution automatique des droits d'accès de façon à limiter les différences de permissions et aboutir à une meilleure cohérence des politiques de sécurité. Nous définissons également un modèle de vérification des droits d'accès pour assurer une cohérence locale au sein d'un même équipement. Ce modèle exprime dans un référentiel commun des politiques hétérogènes de façon à en extraire les différences de permissions. Nous avons également étendu le protocole Netconf avec un modèle de contrôle d'accès basé sur les rôles. Cette proposition a été implantée et a donné lieu à étude de l'impact des règles d'accès et des services de sécurité mis en œuvre sur les performances.

Mots-clés: Supervision des réseaux IP, sécurité, approches de fédération, contrôle d'accès

Abstract

Over the past decades, the fast evolution of networks led to an increasing diversity and number of managed devices, and consequently to many network management platforms. Each of these platforms creates a close environment delimited by its management model, protocol and data model but also a non-isolated environment because of the shared managed resources within a device. Indeed, although the data models are heterogeneous across platform, they may recover partially.

This environment is responsible for a security gap since each network management platform has its own security architecture along with its own associated security parameters. The problem also appears within a single platform allowing many security models or underlying protocols. Up to now, there has been no real work regarding the consequences of such an heterogeneous environment and the possible solutions.

In this thesis, we propose some multi-platform architectures that are able to distribute access rights automatically and on-the-fly in order to limit the privilege discrepancies between platforms and to improve security policies consistency. We also define a model for checking the access rights in order to guarantee the local consistency within one device. This model translates heterogeneous policies in a convergent representation in order to extract their privilege differences. We have extended the Netconf configuration protocol with a role-based access control framework. This proposal has been implemented and a series of benchmarks showed that the impact of XPath-based access control rules on the global processing time of a Netconf agent was acceptable.

Keywords: IP network management, security, federation approaches, access control.

